Diss. ETH No. 14013

Multiresolution Representation and Compression of Surfaces and Volumes

A dissertation submitted to the
Swiss Federal Institute of Technology Zurich

for the degree of
Doctor of Technical Sciences

presented by
OLIVER STAADT
Diplom-Informatiker
Darmstadt University of Technology, Germany
born 30 July 1968
citizen of Germany

accepted on the recommendation of
Prof. M. H. Gross, examiner
Prof. P. Widmayer, co-examiner

2001

“All this time the Guard was looking at her,
first through a telescope, then through a microscope,
and then through an opera-glass.”

Lewis Carroll
Through the Looking-Glass

ABSTRACT

In this thesis we present a wavelet-based geometry compression pipeline in the context of hierarchical surface and volume representations. Due to the increasing complexity of geometric models used in a vast number of different application fields, new methods have to be devised that enable one to store, transmit and manipulate large amounts of data.

Based on a multi-resolution wavelet representation, we have developed a complete compression pipeline suitable for geometric data on uniform grids in two and three dimensions. Local and global oracles in wavelet space are employed to control the approximation error in lossy compression settings. Two novel geometry simplification schemes, which are able to build hierarchical mesh representations, are an essential part of the pipeline.

The first method, a bottom-up vertex removal scheme, analyzes the detail information of the data at different levels after reconstruction from wavelet space. The resulting hierarchical quadtree data structure is triangulated subsequently using a look-up-table that stores the necessary connectivity information. The second method implements a top-down vertex insertion strategy that is capable of progressively reconstructing the model. Vertex connectivity is derived using Delaunay triangulation. This approach provides high flexibility for the construction of adaptive surface and volume approximations. Furthermore, it is possible to extract high quality iso-contours and to compress texture attributes along with the geometric surface model.

In contrast to the two wavelet-based approximation schemes, we have devised the *progressive tetrahedralization* method, an extension of the popular *progressive meshes* into volumetric settings. These strategies can be used to transform unstructured input meshes into special representations which enable a model to be reconstructed progressively. In the volumetric setting, using tetrahedral mesh approximations, we have to account for potential mesh inconsistencies arising frequently during the course of the transformation.

All methods developed in this dissertation have been implemented utilizing a software component-based approach. Several hundred components at different abstraction levels can be combined to build powerful prototype applications with high flexibility.

We compare the three approximation schemes with each other using several two- and three-dimensional geometric models and provide an extensive error and performance analysis. These results emphasize the individual strengths of each of the introduced methods and concepts.

ZUSAMMENFASSUNG

Diese Dissertation stellt ein Wavelet basiertes Verfahren zur Kompression und zur hierarchischen Repräsentation von Oberflächen- und Volumendaten vor. Bedingt durch die zunehmende Komplexität geometrischer Modelle, die in einer Vielzahl von Anwendungsgebieten benötigt werden, müssen neue Methoden zur effizienten Speicherung, Übertragung und Manipulation solcher Modelle entwickelt werden.

Basierend auf der Wavelettheorie wurde eine vollständige Pipeline zur Verarbeitung zwei- und dreidimensionaler geometrischer Modelle auf regulären Gittern entwickelt. Lokale und globale "Orakel" analysieren die Daten im Waveletraum und kontrollieren das Approximationsverhalten bei verlustbehafteter Kompression. Zwei neuartige Methoden zur Vereinfachung von geometrischen Modellen sind essentieller Bestandteil der Verarbeitungspipeline.

Das erste Verfahren, welches einzelne Knoten aus einem Netz entfernt, analysiert die Detailinformationen der Daten nach der Rekonstruktion aus dem Waveletraum. Auf diese Weise wird eine "Quadtree" Datenstruktur erzeugt, die mit Hilfe einer Tabelle sehr effizient trianguliert werden kann. Das zweite vorgestellte Verfahren fügt sukzessive Knoten in ein zunächst nur dünn besetztes Netz ein. Dies erlaubt eine progressive Verfeinerung der Approximation. Die Verbindung zwischen den Knoten wird durch ein Delaunay Triangulationsverfahren bestimmt. Dieser Ansatz gewährleistet eine hohe Flexibilität bei der adaptiven Rekonstruktion von Oberflächen- und Volumenmodellen. Es ist weiterhin möglich qualitativ hochwertige Isokonturen aus den Daten zu extrahieren und zusätzliche Texturinformation zusammen mit der Geometrie zu komprimieren.

Zusätzlich zu den beiden Wavelet basierten Verfahren wurde eine dritte Methode, die *Progressive Tetraedisierung*, entwickelt. Dies ist eine neuartige Erweiterung des verbreiteten *Progressive Mesh* Verfahrens, welche die Umwandlung eines unstrukturierten Tetraedernetzes in eine spezielle Repräsentation erlaubt. Somit wird eine schnelle und flexible Rekonstruktion räumlicher Datensätze ermöglicht. Bei der Verarbeitung von Volumendaten mit Hilfe dieser Methode muss besonderes Augenmerk auf mögliche Inkonsistenzen in der Tetraedisierung gerichtet werden, die im Verlauf der Transformation entstehen können.

Alle Methoden und Algorithmen, die im Verlauf dieser Arbeit entwickelt wurden, sind in Form von Softwarekomponenten implementiert worden. Mehrere Hundert dieser Komponenten können zu flexiblen und leistungsfähigen Anwendungen kombiniert werden.

Die drei vorgestellten Approximationsverfahren wurden Anhand unterschiedlicher Testdaten miteinander verglichen. Eine ausführliche Analyse der resultierenden Approximationsfehler hebt die Leistungsfähigkeit der einzelnen Verfahren hervor.

ACKNOWLEDGMENTS

At this place, I would like to take the opportunity to express my gratitude to all friends and colleagues who have helped and supported me in the course of this project.

I am especially grateful to my advisor Professor Markus Gross. Without his guidance and help, this thesis could have never been realized. Furthermore, I thank Markus for stimulating my interest in computer graphics almost a decade ago and for providing me with an excellent research environment ever since. I enjoyed the many fruitful discussions, not only over this research thesis.

My special thanks go to my co-examiner Professor Peter Widmayer. This work has greatly benefited from his support, comments and advice.

I thank all my fellow colleagues at the Computer Graphics Group who have helped me along the way, especially Lars Lippert, Martin Roth, Rolf Koch, Thomas Sprenger, Reto Lütolf, Daniel Bielser, Andreas Hubeli, Mathias Zwicker, Ronny Peikert, and all the others not explicitly named here. Working with them has always been a great pleasure.

Many thanks to all the former students who have contributed to this work: Roger Gatti, Roland Kisseleff, Roger Weber, Jean-Paul Hofstetter, Marcelo Vetter, Urs Hengartner, and Philip Ruser. They have implemented many parts of the methods presented in this thesis.

Last but not least, I want to thank my family for generously providing me with the greatest support all the way through the last years. Without their help, I could not have achieved the goal to finish this thesis.

This work was supported by the ETH research council under grant No. 41-2642.5.

CONTENTS

FIGURES

TABLES

1

INTRODUCTION

During the past decade, the complexity of geometric models in various application fields has increased steadily. With the wide availability of powerful computer systems, many conventional design and production processes are either simulated or carried out entirely on computers. Good examples include the wide field of product design in almost any engineering discipline. The design cycle for many consumer products including cars, toys, appliances, and so on, typically incorporates highly detailed geometric models. Moreover, the simulation of natural phenomena requires the creation of very large and complex geometric models for further computational processing and scientific visualization.

Even though the performance of state-of-the-art computer systems is still increasing in terms of computational power and memory availability, the complexity of geometric models, driven by the demand of applications, is at least increasing at the same rate.

Several solutions have been proposed to attack these problems, which can more or less be classified into the following overlapping areas: Geometry simplification, geometry compression and hierarchical mesh representations.

Non-hierarchical geometry simplification schemes have successfully been employed for many years to reduce the number of elements in geometric models. The use of simplified models can speed-up visualization and further numerical processing. Those method allow for the generation of high-quality approximations at a single level of resolution. If the user requests a mesh at a different resolution, however, the whole simplification process has to be repeated from the start.

A completely different approach is to optimize the memory requirements for storing and transmitting large models. Geometry compression methods reduce the memory footprint of a model without compromising its geometrical or topological properties. We distinguish between lossy and lossless compression methods. A certain loss of precision can usually be tolerated for compressing positional information of individual vertices constituting the model. Connectivity information, which specifies the relation between these vertices and defines the model's topology, should be compressed without loss of information.

Although non-hierarchical simplification methods and geometry compression schemes often allow for progressive reconstruction and transmission over networks, they do not inherently define a hierarchical representation of the model. For very large data sets, however, it might be advantageous to inspect or even edit the data at different levels, and hierarchical representations provide build-in support for such applications. Note the difference between "linear" progressive schemes, which also enable the reconstruction of different levels, and fully hierarchical methods: The latter contain additional information about dependencies between successive levels, thus allowing to switch consistently between them. This is especially desirable in multi-resolution editing applications, where changes applied at one approximation level are automatically propagated throughout the hierarchy.

1.1 RELATED WORK

This section summarizes important work in each of the research areas described above that is relevant to the new methods which will be introduced in the course of this thesis.

1.1.1 Geometry Simplification

There are numerous schemes for simplifying geometric meshes in computer graphics and scientific visualization, of which Garland and Heckbert provide an extensive overview in [49].

Previous work closely related to our method can be broadly categorized into two main classes. Firstly, vertex removal methods and secondly, methods based on edge collapse and vertex split transformations.

Vertex removal methods. Even before the use of surface simplification methods, curve simplification algorithms have been employed in cartography, computer graphics, computer vision, and a variety of other fields. The most widely used method is that of Douglas and Peuker [29]. A sequence of points is approximated by a line connecting the two endpoints of a curve. The point farthest from this line is found and the algorithm is applied recursively to the two sub-sequences before and after the new point.

Various decimation algorithms for surface simplification have been published which remove single vertices from the mesh and retriangulate the resulting hole. One of the most popular methods in that category was introduced by Schroeder and others [90]. They describe a multi-pass method that takes a triangulated surface as input. On each pass, all vertices, whose distance to the approximating plane of the surrounding vertices is below a user-specified threshold, are deleted from the mesh. The resulting holes are retriangulated and the process is repeated until no more vertices can be found for this threshold.

In addition to [90], Cohen and others [21] were able to bound the maximum error of the approximation by restricting it to lie between two offset surfaces.

Turk [105] also starts with a triangulated surface and sprinkles a set of points on the mesh, distributed according to estimates of local curvature. The mesh is retriangulated based on those points.

The special case of digital terrain data was addressed, for instance, by Lindstrom and others [65] and Pajarola [75].

Methods based on edge collapsing. Hoppe and others [54] proposed a triangular mesh simplification scheme based on edge collapse and vertex split transformations. A set

of energy functions is used to decide on the sequence of edge collapse transformations, and a sophisticated optimization method, which determines the position of the new vertex, is provided. Hoppe [52] extended this work to support selective refinement and progressive transmission, as well as lossy geometric compression [26] and the possibility to interpolate smoothly between different levels of approximation.

Hoppe's method has been the basis for several other extensions. Xia and others [116] and Hoppe [53] focused on fast, hierarchical representations, that enable one to efficiently reconstruct selected regions of a triangular progressive mesh. All of these algorithms preserve the topology of the original data, whereas others [80, 35, 89, 84] allow topological changes, such as filling of holes or connecting disconnected parts of a mesh. Popovic and Hoppe [80] devised a general approach for arbitrary dimensional simplicial complexes, which also allows topological changes of the data to further reduce the complexity of the mesh. Unfortunately, they only provide an implementation and examples for the triangular case. Additional important types of simplices, such as tetrahedra, are not addressed in detail.

Other work [84, 45, 89] differs from [54] and [52] in the ordering of edge collapse transformations and the optimization method which specifies the new vertex position. Garland and Heckbert [35] propose a very elegant scheme using quadrics for both tasks.

Independent form our work presented in this thesis, Trotts and others [104] have presented a tetrahedral mesh simplification scheme based on edge collapsing. Their method, however, does not take into account possible topological inconsistencies, which are likely to arise in tetrahedral settings.

1.1.2 Geometry Compression

We can generally distinguish between lossy and lossless geometry compression methods. Typical lossless methods try to find a bit-efficient coding of the cell-vertex incidence which is represented as connectivity graph.

Deering [26] introduced a scheme for efficiently encoding generalized triangle strips. His method was designed for decompression by a graphics hardware adapter with limited memory. It is non-trivial, however, to identify sufficiently long strips efficiently, which helps to improve the performance of the compression method. For that reason, a triangle-spanning tree—whose nodes correspond to all the triangles and whose edges correspond to some of the mesh's interior edges—is often used instead of triangle strips.

Unfortunately, a triangle-spanning tree does not capture the entire topology of the triangle-vertex incidence. Taubin and Rossignac [101] employ a triangle-spanning tree and its dual, a vertex-spanning tree. Both trees are encoded efficiently with three bits per triangle in worst case. Rossignac [85] further improved the encoding of the triangle-vertex incidence graphs and his scheme requires only 1.5 bits per triangle for large meshes while guaranteeing an upper bound of two bits. Pajarola and Rossignac [76] devised an efficient method for compressing triangle meshes in progressive settings with less than four bits per triangle. An extension for progressive tetrahedral meshes has recently been introduced in [77].

The vertex locations can either be compressed lossless or lossy. Predictor-based compression of the vertex locations is often employed in lossless settings. Here, corrections between the predicted and the actual position of a vertex are encoded. Only short corrective vectors are necessary if the predictors are accurate. Taubin and Rossignac [101] use predictor-based compression in combination with their encoding of the triangle-vertex

incidence graphs. Hoppe [52] applied a similar scheme in a progressive mesh setting. Touma and Gotsman [103] construct a parallelogram to estimate the location of the free vertex of a new triangle which is adjacent to an already known triangle.

Compression of vertex location can further be improved by using lossy compression. Typically, vertex locations are quantized by expressing the coordinate vector a k-bit integer in a normalized coordinate system. The coordinate system may be derived from a minimum-axis bounding box [26, 80, 101].

Note that geometry simplification methods can also be considered lossy compression schemes.

1.1.3 Hierarchical Mesh Representations

Some of the work discussed above supports hierarchical setting [53, 116]. Lounsbery and others [69] generalize the concept of multi-resolution analysis (MRA) to surfaces of arbitrary topological type by employing linear wavelet bases. Eck and others [31] extend this work by devising an approximation method for meshes with arbitrary connectivity making them suitable to serve as input for the method described in [69]. Certain and others [11] further allow to capture color attributes and present a multi-resolution viewer supporting progressive transmission over networks.

Rossignac and Borrel [86] merge vertices of a model using spatial binning. An important aspect of their approach is that the topology of the model may change in the process. Although this method is very fast, it ignores important geometric qualities such as curvature, and the resulting approximations can be far from optimal.

Fewer methods have been devised that decimate higher dimensional simplicial complexes. Grosso and others [44] use finite element computations to represent triangular and tetrahedral meshes at multiple levels. Cignoni and others [18] propose a framework which is based on a decimation method and allows one to represent tetrahedral meshes at arbitrary resolution. In the context of direct volume rendering, Lippert [66] presented several new methods based on wavelet representations.

1.2 CONTRIBUTION

Much effort has been spent on finding appropriate mesh simplification and representation methods which allow for fast and progressive transmission and rendering of complex scenes. However, little attention has been spent on the following issues:

- In practice, many technical applications, such as medical imaging systems and digital terrain modelling, produce raw data sampled over uniform grids. Due to their complexity, these data sets have to be compressed and stored in a remote database server. Thus, visual inspection and browsing requires computation of piecewise linear geometric reconstructions from the compressed data set.

- Up to now compression was mostly considered a mesh representation problem. The manifold aspects of a full compression pipeline such as precoding, global and local oracles, quantization, and optimal bit allocation have rarely been discussed in full detail.

- Compression and reconstruction should be embedded in a framework which provides an interface for the client and offers a testbed for individual methods. In particular, both lossy and lossless compression must be combined to satisfy demands arising from geometric reconstructions.

The research presented in this thesis is stimulated by the issues discussed above. Our goal is to provide an efficient and versatile compression and reconstruction pipeline which accounts for client-server setups. The framework is hybrid in the sense that it combines both lossy and lossless compression. The primary contribution of this thesis addresses:

- *Wavelet-based geometry compression:* The new compression pipeline comprises higher-order B-spline wavelet bases for uniform data in two and three dimensions. Global and local oracles are applied to govern the process. In addition, texture attributes can be compressed along with the geometry.

- *Bottom-up vertex removal:* A novel method for generating hierarchical approximations of triangle meshes. Based on the analysis of wavelet-transformed data sets, a quadtree data structure is constructed and subsequently triangulated using a fast and elegant look-up-table approach.

- *Top-down vertex insertion:* Based on our wavelet-based compression pipeline, we devise a progressive vertex insertion algorithm for one-, two- and three-dimensional data sampled on uniform grids. Delaunay triangulation is used to derive connectivity information for hierarchical mesh representations. Furthermore, higher-order extraction of implicit structures with superior quality compared to standard methods is supported.

- *Progressive tetrahedralizations:* An alternative approach to wavelet-based multi-resolution approximations is presented. The popular progressive mesh method is extended to volumetric settings. Special emphasis is put on topological fallacies arising when dealing with tetrahedral meshes. Results from this method are compared to both wavelet-based schemes.

We have published research results from our bottom-up vertex removal method in [41, 43]. The wavelet-based geometry compression pipeline and the top-down vertex insertion scheme have been published in [95, 42]. Progressive tetrahedralizations have first been presented in [94].

1.3 OUTLINE OF THE THESIS

The remainder of the thesis is outlined as follows:

- *Chapter 2* gives an introduction to wavelet theory, the underlying mathematical framework. After describing the general concepts of wavelets and multi-resolution analysis, we focus on specific basis functions and oracles which are employed in subsequent chapters.

- *Chapter 3* illustrates some basic concepts from geometry and piecewise-linear topology. In this chapter, important properties of geometric representations and algorithms define the nature of a variety of data which are being represented with the novel methods introduced in Chapter 5 and Chapter 6. In addition, adequate quantitative error metrics for a variety of data are defined.

- *Chapter 4* discusses the problem of data compression and illustrates some important methods, which can be employed in different applications. The two general concepts of lossless and lossy compression are introduced on the basis of specific coding methods, many of which are used in the compression pipeline in Chapter 5.
- *Chapter 5* describes a versatile geometry representation pipeline comprising efficient data compression and different mesh reconstruction schemes. This pipeline applies to a variety of data including non-parametric and parametric surface data, scalar volume data and iso-contours, as well as surface texture representations.
- *Chapter 6* contains the description of a flexible surface and volume representation, which contrasts with the methods introduced in Chapter 5. Based on the powerful concept of progressive meshes, we introduce the extension to tetrahedral settings while focusing on topological problems arising in volumetric representations.
- *Chapter 7* contains experimental results from the algorithms introduced in the previous chapters. In addition to results for individual methods, different aspects of the wavelet-based schemes and progressive tetrahedralizations are compared, while specific advantages and disadvantages are emphasized.
- *Chapter 8* summarizes the conclusions of this thesis and points out future directions.
- The appendix includes a brief description of the software components of our algorithms as well as a list of technical terms and symbols that are used in this text.

2

WAVELETS

This chapter gives a brief introduction to wavelet theory, which is the underlying mathematical framework used in our approach. Firstly, the multi-resolution analysis (MRA), an efficient representation of hierarchical bases, is introduced. Secondly, an important set of basis functions, the cardinal B-spline bases are presented. Consideration of boundary problems is highly important for practical applications and leads to special constructions of bases and transform schemes. Finally, the use of oracles, which are means of omitting unimportant basis functions, is summarized.

Throughout this chapter, the functions of interest are restricted to the Hilbert space of square-integrable functions, $L^2(\subsetneq)$. It is defined as the space of Lebesgue-measurable functions for which

$$\|f\|^2_{L^2(\subsetneq)} = \int_{-\infty}^{\infty} |f(x)|^2 dx < \infty . \tag{2.1}$$

Additional notation of terms used in the following sections is given in Appendix B.

2.1 MULTI-RESOLUTION ANALYSIS

With the introduction of the multi-resolution analysis (MRA) by Mallat [70], wavelets have become an important mathematical tool over the last decade. The MRA, originally designed for image analysis, defines an operator which approximates signals at different resolutions. The key idea is to decompose a signal into a low-resolution part and a sequence of detail information in increasing order.

2.1.1 Definition

Formally, an MRA of $L^2(\subsetneq)$ is defined as a sequence of closed subspaces V_m of $L^2(\subsetneq)$, $m \in \subsetneq$, with the following properties:

1. *Nestedness from fine-to-coarse:* $L^2(\mathbb{R}) \supset \ldots \supset V_0 \supset V_1 \supset \ldots \supset V_m \supset \ldots \supset \{0\}$
2. *Scale-invariance:* $f(x) \in V_m \Leftrightarrow f(2^m x) \in V_0$
3. *Shift-invariance:* $f(x) \in V_0 \Rightarrow f(x-p) \in V_0, \quad p \in \mathbb{Z}$
4. *Completeness:* $\bigcup_{m=\infty}^{\infty} V_m$ is dense in $L^2(\mathbb{R})$ and $\bigcap_{-\infty}^{\infty} V_m = \{0\}$
5. *Riesz basis:* $\exists \phi(x) \in V_0$: $\{\phi(x-p) | p \in \mathbb{Z}\}$ is a Riesz basis of V_0

This implies that a set of functions $\{\phi(\cdot - p)\}_{p \in \mathbb{Z}}$ spans V_0, and

$$\forall \{c_p\}_{p \in \mathbb{Z}} \in l^2(\mathbb{Z}): \quad A \cdot \|c_p\|^2_{l^2(\mathbb{Z})} \leq \left\| \sum_{p \in \mathbb{Z}} c_p \cdot \phi(\cdot - p) \right\| \leq B \cdot \|c_p\|^2_{l^2(\mathbb{Z})} \tag{2.2}$$

with $A > 0$ and $B < \infty$ ($l^2(\mathbb{Z})$ is the space of square-summable sequences).
Two other important characteristics of an MRA are the two-scale relation

$$\phi(x) = \sqrt{2} \sum_{p \in \mathbb{Z}} h_p \cdot \phi(2x - p), \tag{2.3}$$

and the partition of unity

$$\sum_{p \in \mathbb{Z}} \phi(\cdot - p) = 1. \tag{2.4}$$

The sequence $\{h_p\}_{p \in \mathbb{Z}}$ is required for the implementation of the fast wavelet transform (FWT), which will be described in Section 2.2.3.

In addition to the nested set of subspaces $\{V_m\}_{m \in \mathbb{Z}}$, an inner product of two functions $f, g \in V_m$ is defined as

$$\langle f, g \rangle = \int_{-\infty}^{\infty} f(x) \overline{g(x)} dx \ . \tag{2.5}$$

If, and only if these conditions are satisfied, $\phi(x)$ is a valid scaling function of $L^2(\mathbb{R})$. The set of functions $\{\phi_{mp}\}_{p \in \mathbb{Z}}$ with

$$\phi_{mp}(x) = 2^{-\frac{m}{2}} \phi(2^{-m} x - p) \tag{2.6}$$

constitutes a Riesz basis for subspace V_m. Therefore, each function $f \in V_{m_0}$ can be expressed as the weighted sum of scaling functions $\phi_{m_0 p}$ of iteration level m_0:

$$f(x) = \sum_{p \in \mathbb{Z}} c_p \cdot \phi_{m_0 p}(x). \tag{2.7}$$

Detail information of subspace V_m can be captured in an additional subspace W_{m+1}, such that

$$V_m = V_{m+1} \oplus W_{m+1}. \tag{2.8}$$

The subspaces $\{W_m\}_{m \in \mathbb{Z}}$ are spanned by wavelet functions ψ_{mp} that are constructed from a mother wavelet ψ by binary dilation and dyadic translation:

$$\psi_{mp}(x) = 2^{-\frac{m}{2}} \psi(2^{-m}x - p), \quad m, p \in \mathbb{Z}. \tag{2.9}$$

Note that the other properties mentioned above for the scaling function $\phi(x)$ also hold for the wavelet function $\psi(x)$.

It is now possible to decompose a function f into a sequence of projections into V_m and $\{W_m\}_{0 < m < M}$:

$$\begin{array}{ccccccc} f \in V_0 & \longrightarrow & \cdots & \longrightarrow & f_m \in V_m & \longrightarrow & \cdots \longrightarrow f_M \in V_M \\ & \searrow & \cdots & \searrow & f_{m-1} - f_m \in W_m & \searrow & \cdots \searrow f_{M-1} - f_M \in W_M \end{array} \tag{2.10}$$

Thus, a hierarchical representation of function f_{m_0} at level m_0 can now be written as the linear combination of its coarse shape at level $M > m_0$ and the detail information of all intermediate levels $m_0 < m \leq M$:

$$f(x) = \sum_{p \in \mathbb{Z}} c_{Mp} \phi_{Mp}(x) + \sum_{m=m_0}^{M} \sum_{p \in \mathbb{Z}} d_{mp} \psi(x). \tag{2.11}$$

The coefficients c_{mp} and d_{mp} are determined by the inner products of f with ϕ_{mp}, and ψ_{mp}, respectively:

$$c_{mp} = \langle f, \phi_{mp} \rangle \qquad d_{mp} = \langle f, \psi_{mp} \rangle \ . \tag{2.12}$$

Finally, the the two-scale relation for the wavelet bases can be written as

$$\psi(x) = \sum_{p \in \mathbb{Z}} g_p \cdot \phi(2x - p). \tag{2.13}$$

2.1.2 Properties of Wavelets

After the definition of the essential parts of multi-resolution analysis, some important properties of wavelets are listed [57]:

- *Compact support.* The support of a function is defined as the size of the interval over which the function is nonzero:

$$\text{supp}(f) = \{x \in \mathbb{Z} | (f(x) \neq 0)\}. \tag{2.14}$$

  Compactly supported wavelets can be represented by finite impulse response (FIR) filters. This has an important influence on efficient implementations of fast wavelet transform algorithms by allowing the use of sparse matrix operators (see Section 2.2.3).

- *Localization.* The size of the interval of a compactly supported function plays a significant role for its localization quality. If the interval is sufficiently small one may speak of functions with *small local support*. This property ensures high positional accuracy in spatial domain. In contrast to many other bases (e.g., trigonometric basis functions of

the Fourier transform) wavelets are also well localized in frequency domain. The relation between the localization of a function $f(x)$ in spatial domain, Δ_x^f, and the localization of its Fourier transform $F(\omega)$ in frequency domain, Δ_ω^f, depends on the *Heisenberg uncertainty principle* and is defined by the *Heisenberg window:*

$$\Delta_x^f \cdot \Delta_\omega^f \geq \frac{1}{4\pi}. \tag{2.15}$$

For that reason, there is always a trade-off between good localization in spatial and in frequency domain. The Heisenberg window of the Gabor function, a modulated Gaussian function, is optimal in spatial/frequency localization since it is equal to the lower boundary in Equation 2.15 [107]. B-Spline functions, which will be introduced in Section 2.2, asymtotically reach this boundary with increasing order.

- *Smoothness.* The smoothness of wavelets is an important property for compression applications. Compression is achieved by setting small d_{mp} to zero, and thus leaving out the component $d_{mp}\psi(x)$ from the expansion in Equation 2.11. When smooth functions are approximated by truncated expansions, smooth basis functions usually give better results. More details on truncation and related oracles can be found in Section 2.4. The order of smoothness can be measured by analyzing the number of continuous derivates of the basis function. Unfortunately, a higher order of smoothness results in a larger support and vice versa.

- *Number of vanishing moments.* The k-moment of a wavelet $\psi(x)$ is defined as the inner product of ψ with it's variable x raised to the power of k. ψ has n *vanishing* moments if this inner product is equal to zero for all $k < n$:

$$\left\langle \psi, \cdot^k \right\rangle = \int \psi(x) \cdot x^k dx = 0, \quad 0 \leq k < n \quad . \tag{2.16}$$

If a signal is well-represented by a low-order polynomial of degree $p < k$, the resulting wavelet coefficients d are close to zero. Consequently, the number of vanishing moments is a very important indicator of the compression ability of a given wavelet. Note that for any wavelet the number of vanishing moments must at least be one.

- *Interpolation.* For some applications it might be required to not only approximate the initial samples with the scaling function, but to interpolate the samples. In that case, the scaling function has to satisfy the following condition:

$$\phi = \delta_{p0}, \quad p \in \subsetneq. \tag{2.17}$$

In this case it is trivial to find a function of V_m that interpolates data sampled on a grid with spacing 2^m, since the coefficients are equal to the samples.

- *Orthogonality.* In many situations it is very convenient to have an orthogonal MRA, because the L^2-norm of a function is directly linked to the norm of the wavelet coefficients. The projection operators of an orthogonal MRA into the different subspaces yield optimal approximation in the L^2 sense.

  Orthogonality and the more general formulations of semi- and bi-orthogonality are discussed in detail in Section 2.1.3.

Unfortunately, it is not possible to construct an MRA that is optimal with respect to all properties listed above. Hence, there is always a trade-off between them.

2.1.3 Orthogonality

The class orthogonal wavelets is particularly interesting for several reasons. In this section, the concept of orthogonal wavelets and the more general classes of semi- and bi-orthogonal wavelets are introduced.

Orthogonal wavelets. A multi-resolution analysis is orthogonal, when the wavelet spaces W_j are defined as the orthogonal complement of V_m in V_{m-1}. Thus, the spaces $\{W_m\}_{m \in \subsetneq}$ are all mutually orthogonal. A sufficient condition for a multi-resolution analysis to be orthogonal is

$$W_0 \perp V_0 \tag{2.18}$$

or

$$\langle \psi, \phi(\cdot - p) \rangle = 0, \quad p \in \boldsymbol{Z}. \tag{2.19}$$

A scaling function ϕ is orthogonal if the set $\{\phi(x-p) | p \in \subsetneq\}$ is an orthonormal basis, or

$$\langle \phi, \phi(\cdot - p) \rangle = \delta_{p0}, \quad p \in \subsetneq, \tag{2.20}$$

and the collection of functions $\{\phi_{mp}\}_{p \in \subsetneq}$ is an orthonormal basis of V_m. δ_{pq} denotes the Kronecker-delta function.

Likewise, an orthogonal wavelet ψ is a function such that the set of functions $\{\psi(x-p) | p \in \subsetneq\}$ is an orthonormal basis of W_0. This is obviously the case if

$$\langle \psi, \psi(\cdot - p) \rangle = \delta_{p0}, \quad p \in \subsetneq. \tag{2.21}$$

Since the spaces W_j are mutually orthogonal, the collection of wavelets $\{\psi_{mp}\}_{m,\, p \in \subsetneq}$ is an orthonormal basis of $L^2(\subsetneq)$.

Let $\wp_U$ be the projection operator which projects into a vector space U. We then can project the function $f(x) \in L^2(\subsetneq)$ into V_m and W_m with the operators $\wp_{V_m}$ and $\wp_{W_m}$:

$$\wp_{V_m} f(x) = \sum_p \langle f, \phi_{mp} \rangle \cdot \phi_{mp}(x) \quad \text{and} \quad \wp_{W_m} f(x) = \sum_p \langle f, \psi_{mp} \rangle \cdot \psi_{mp}(x). \tag{2.22}$$

These operators yield the best L^2-approximation of f in V_m and W_m, respectively. Thus, the orthogonal expansion of $f \in L^2(\subsetneq)$ is

$$f(x) = \sum_{m,\, p \in \subsetneq} d_{mp} \cdot \psi_{mp}(x), \quad \text{with} \quad d_{mp} = \langle f, \psi_{mp} \rangle. \tag{2.23}$$

Fast and efficient algorithms for the computation of the coefficients c_{mp} and d_{mp} will be described in Section 2.2.3.

Widely used orthogonal wavelets are the family of Daubechies wavelets [25].

As already stated in Section 2.1.2, one of the most important properties of orthogonal wavelets is that the energy norm of a function is directly linked to the energy norm of the wavelet coefficients:

$$\|f\|_{L^2} = \sqrt{\sum_p (c_{m_0 p})^2} = \sqrt{\sum_p (c_{Mp})^2 + \sum_{m=m_0}^{M} \sum_p (d_{mp})^2}, \quad \forall f \in V_{m_0}. \tag{2.24}$$

Bi-orthogonal wavelets. Orthogonality puts hard constraints on the construction of wavelet basis functions. The Haar wavelet, which will be described in Section 2.2.1, is the only real-valued wavelet that is compactly supported, symmetric, and orthogonal [24]. To gain more flexibility in constructing wavelets, orthogonal wavelets can be generalized to bi-orthogonal wavelets. In addition to ϕ and ψ, dual scaling functions $\tilde{\phi}$ and wavelets ψ exist which generate a dual MRA with subspaces $\tilde{V}_m$ and $\tilde{W}_m$, such that

$$\tilde{V}_m \perp W_m \quad \text{and} \quad V_m \perp \tilde{W}_m, \tag{2.25}$$

and

$$\tilde{W}_m \perp W_{m'} \quad \text{for} \quad m \neq m'. \tag{2.26}$$

Moreover, the following properties are true for the dual functions:

$$\left\langle \tilde{\phi}_{mp}, \phi_{mp'} \right\rangle = \delta_{pp'} \quad \text{and} \quad \left\langle \tilde{\psi}_{mp}, \psi_{m'p'} \right\rangle = \delta_{mm'} \cdot \delta_{pp'}. \tag{2.27}$$

Consequently, the projection operators are different from those in the orthogonal case and take the form [57]

$$\wp_{V_m} f(x) = \sum_p \left\langle f, \tilde{\phi}_{mp} \right\rangle \cdot \phi_{mp}(x) \quad \text{and} \quad \wp_{W_m} f(x) = \sum_p \left\langle f, \tilde{\psi}_{mp} \right\rangle \cdot \psi_{mp}(x) \tag{2.28}$$

and

$$f = \sum_{m,p} \left\langle f, \tilde{\psi}_{mp} \right\rangle \cdot \psi_{mp}. \tag{2.29}$$

The introduction of a dual MRA allows constructing a larger number of different scaling functions and wavelets. See [13, 19,] for more details about the construction of bi-orthogonal wavelets.

Semi-orthogonal wavelets. For many practical applications, using a full bi-orthogonal setting is inconvenient since the dual basis functions are usually not compactly supported. This prohibits the design of fast wavelet transform schemes with linear time complexity. Semi-orthogonal wavelets are a special case of bi-orthogonal wavelets, which provide more flexibility than orthogonal wavelets while enabling one to construct linear-time transform algorithms.

A bi-orthogonal scaling function and wavelet are semi-orthogonal if they generate an orthogonal MRA [13]. Mutual orthogonality of the subspaces W_m implies that

$$W_m \perp \tilde{W}_{m'} \quad \text{and} \quad W_m \perp W_{m'} \quad m \neq m'. \tag{2.30}$$

Therefore, $W_m = \tilde{W}_m$, which further implies that $V_m = \tilde{V}_m$. Hence, the primary and the dual basis functions generate the same multi-resolution analysis, which is in fact orthogonal. The projection operators from Equation 2.28 now take the simplified form

$$\wp_{V_m} f(x) = \sum_p c_{mp} \cdot \phi_{mp}(x) \quad \text{and} \quad \wp_{W_m} f(x) = \sum_p d_{mp} \cdot \psi_{mp}(x), \tag{2.31}$$

and finally, the expansion of f takes the form

$$f = \sum_{m,\,p} d_{mp} \cdot \psi_{mp}(x). \tag{2.32}$$

Note that semi-orthogonality does not necessarily imply dual functions with compact support. However, since the primary and dual functions span the same subspaces V_m and W_m, it is possible to construct efficient transform schemes with linear complexity (see Section 2.2.3).

An important example of semi-orthogonal wavelets is the class of B-spline wavelets [13, 108], which will be described in Section 2.2.

2.2 CARDINAL B-SPLINE BASES

In most computer graphics applications [39, 34] bases with strict local support along with an appropriately smooth shape, symmetry, and fast decay in frequency domain are required. An important class of semi-orthogonal bases which meets these requirements is the set of cardinal B-spline wavelets. These bases were developed independently by Chui [15, 13] and Unser [108] and are widely used in computer graphics and modelling [97].

The cardinal B-splines N^j of order j are piecewise-polynomial spline functions with equally spaced simple knots [13]. They can be defined as a recurrence relation and are assumed to be the cardinal B-spline bases:

$$\begin{aligned} N^j(x) &= (N^{j-1} * N^j)(x) \\ &= \int_0^1 N^{j-1}(x-t)\,dt \\ &= \frac{x}{j-1} N^{j-1}(x) + \frac{j-x}{j-1} N^{j-1}(x-1), \quad j \geq 2 \end{aligned} \quad , \tag{2.33}$$

where * denotes the convolution operator

That is, the bases are derived from each other by self-convolution of an initial basis of order 1, where:

$$N^1(x) := \begin{cases} 1: & 0 \leq x \leq 1 \\ 0: & \text{otherwise} \end{cases}. \tag{2.34}$$

2.2.1 Haar Wavelets

Equation 2.34 is the scaling function of the well known Haar basis, the simplest multiresolution basis in one dimension [48]. This first order B-spline scaling function of degree 0 is only of theoretical interest. Although compactly supported, it lacks most of the other properties listed in Section 2.1.2. Since it is not continuous itself, it is not suited for anal-

ysis and approximation of continuous functions. Moreover, it has bad localization properties in frequency domain since its Fourier transform decays like $1/|\omega|$.

The Haar basis $\phi^{[1]} = N^1$ spans the space V_m of piecewise-constant functions and its complement space W_m is spanned by piecewise-constant wavelets $\psi^{[1]}$ defined as

$$\psi^{[1]}(x) := \begin{cases} 1: & 0 \le x < 0.5 \\ -1: & 0.5 \le x < 1 \\ 0: & \text{otherwise} \end{cases}. \tag{2.35}$$

From Equation 2.8 we obtain that any wavelet can be constructed as the linear combination of scaling functions from the next lower sub-space:

$$\psi^{[1]}_{m,p} = \phi^{[1]}_{m+1,2p} - \phi^{[1]}_{m+1,2p+1}.$$

The Haar scaling function and the Haar wavelet are depicted in Figure 2.1.

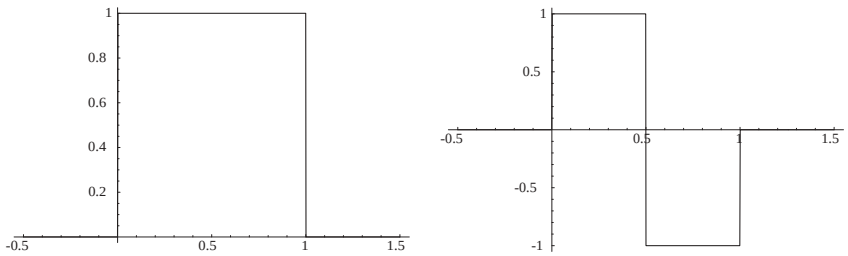


Figure 2.1 The Haar basis: **a:** scaling function. **b:** wavelet function.

2.2.2 Higher-Order Bases

Higher-order B-spline basis $\phi^{[j]}_{m,p}(x) := N^j(2^m x - p)$ can be constructed by recursively evaluating Equation 2.33. The support $\text{supp}(\phi^{[j]})$ of a B-spline scaling function $\phi^{[j]}$ is always bound by $[0, j]$. Furthermore, the scaling functions are symmetric with respect to the center of support. Figure 2.2 illustrates a set of different B-spline scaling functions of orders $j = 2, 3, 4$.

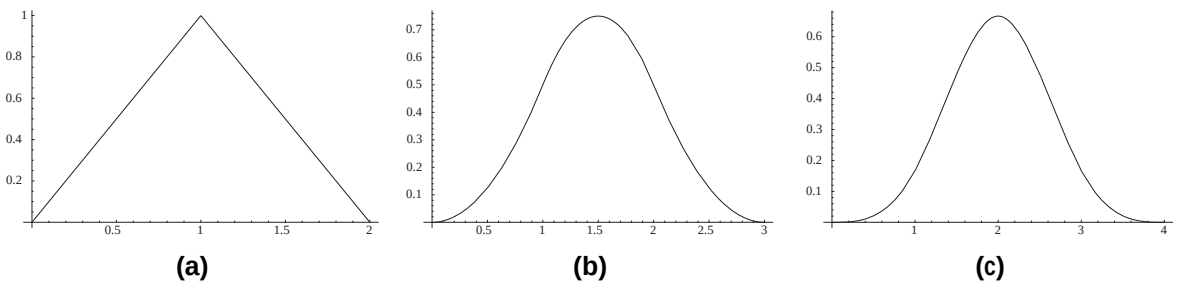


Figure 2.2 Cardinal B-spline scaling functions of increasing order j: **a:** $j = 2$. **b:** $j = 3$. **c:** $j = 4$.

The construction of a corresponding B-wavelet $\psi^{[j]}$ of order j, which spans the orthogonal complement W_m between two approximation spaces of scaling functions V_m and V_{m-1} of different resolution, can be achieved by defining

$$\psi^{[j]}(x) = \sum_{k=0}^{3j-2} q_{j,k}\phi^{[j]}(2x-k) \tag{2.36}$$

with

$$q_{j,k} = \frac{(-1)^k}{2^{j-1}} \sum_{l=0}^{j} \binom{j}{l} \phi^{[2j]}(k+1-l) \text{ [13].}$$

It follows from Equation 2.36 that the support $\text{supp}(\psi^{[j]})$ of $\psi^{[j]}$ is bound by $[0, 2j-1]$. As opposed to the scaling functions $\phi^{[j]}$, the wavelets $\psi^{[j]}$ are only symmetric for even j, but anti-symmetric for odd j. Figure 2.3 depicts the wavelets corresponding to the scaling functions shown in Figure 2.1.

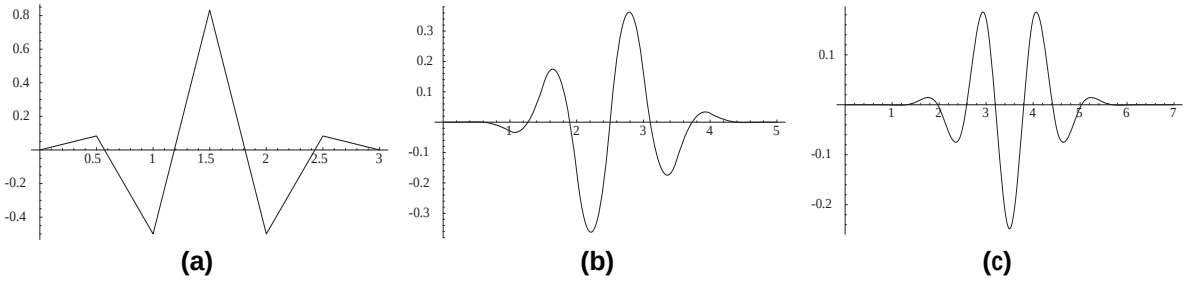


Figure 2.3 B-wavelets of increasing order j: **a:** $j = 2$. **b:** $j = 3$. **c:** $j = 4$.

An important property of B-wavelets is that they approach optimal time-frequency localization with increasing order j. The proof of this property is given in [107]. Values of $\Delta_x^f \cdot \Delta_\omega^f$ for different orders in comparison to the optimal bound of the Gabor function are given in Table 2.1.

Table 2.1 Values of $\Delta_x^f \cdot \Delta_\omega^f$ for B-wavelets of increasing order.

j	$\Delta_x^f \cdot \Delta_\omega^f$
2	0.154653
3	0.085159
4	0.080348
5	0.079725
6	0.079636
∞ (Gabor function)	0.079577 = 1/4π

The construction of the dual functions $\tilde{\phi}^{[j]}$ and $\tilde{\psi}^{[j]}$ is discussed in detail in [13]. Note, however, that the duals do not have compact support, but fast exponential decay. Therefore, the dual functions are usually used for decomposition and the primal functions for reconstruction. The design of a fast wavelet transform algorithm is described in Section 2.2.3.

The number of vanishing moments of B-wavelets is equal to $j - 1$, which makes them an attractive choice for compression applications.

2.2.3 Fast Wavelet Transform

In this section, the construction of the fast wavelet transform for the B-spline basis is explained. A more general description of the fast wavelet transform is given in [70, 25, 13].

For reasons of efficiency, the projection operators defined in Equation 2.31 on page 13 are implemented as matrix operations. A data vector in $\mathbf{c}_m$ is decomposed into scaling function coefficients $\mathbf{c}_{m+1}$ by

$$\mathbf{c}_{m+1} = \mathbf{A}_{m+1}\mathbf{c}_m \tag{2.37}$$

and wavelet coefficients $\mathbf{d}_{m+1}$ by

$$\mathbf{d}_{m+1} = \mathbf{B}_{m+1}\mathbf{c}_m . \tag{2.38}$$

The decomposition scheme is depicted in Figure 2.4.

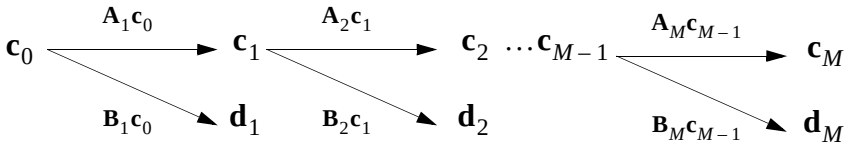


Figure 2.4 Fast wavelet transform: Decomposition (analysis).

Reconstruction is performed by reversing the above process applying the matrix operators $\mathbf{P}_m$ and $\mathbf{Q}_m$:

$$\mathbf{c}_m = \mathbf{P}_{m+1}\mathbf{c}_{m+1} + \mathbf{Q}_{m+1}\mathbf{d}_{m+1}, \tag{2.39}$$

which is depicted in Figure 2.5.

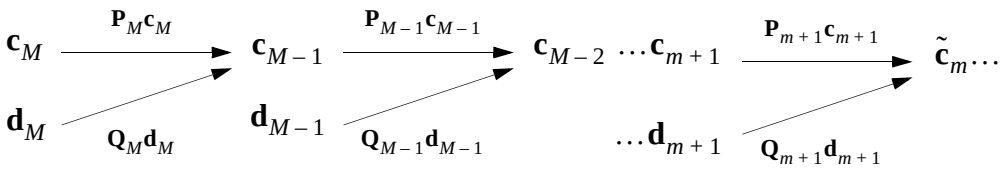


Figure 2.5 Fast wavelet transform: Reconstruction (synthesis).

Given the B-spline and the B-wavelet functions, the operators can be constructed easily. $\mathbf{P}_m$ is determined by

$$\Phi_{m+1} = \Phi_m\mathbf{P}_m \tag{2.40}$$

with Φ_m representing the column vector of the B-splines $[\phi_{m,1}, \ldots, \phi_{m,2^m}]$. It follows directly from Equation 2.6 that a solution for this system exists. Similarly, $\mathbf{Q}_m$ is determined by

$$\Psi_{m+1} = \Phi_m \mathbf{Q}_m, \tag{2.41}$$

with $\Psi_m = [\psi_{m,1}, \ldots, \psi_{m,2^m}]$. Equation 2.40 and Equation 2.41 can be combined to yield the following:

$$[\Phi_{m+1} | \Psi_{m+1}] = \Phi_m [\mathbf{P}_m | \mathbf{Q}_m]. \tag{2.42}$$

Accordingly, the decomposition operators $\mathbf{A}_m$ and $\mathbf{B}_m$ fulfill the following relation:

$$[\Phi_{m+1} | \Psi_{m+1}] \left[\frac{\mathbf{A}_m}{\mathbf{B}_m}\right] = \Phi_m. \tag{2.43}$$

$[\mathbf{A}_m | \mathbf{B}_m]^{\mathrm{T}}$ and $[\mathbf{P}_m | \mathbf{Q}_m]$ are square matrices. It follows directly from Equation 2.42 and Equation 2.43 that $\mathbf{A}_m$ and $\mathbf{B}_m$ can also be calculated by

$$\left[\frac{\mathbf{A}_m}{\mathbf{B}_m}\right] = [\mathbf{P}_m | \mathbf{Q}_m]^{-1}. \tag{2.44}$$

The operators $\mathbf{A}_m, \mathbf{B}_m, \mathbf{P}_m, \mathbf{Q}_m$ are band matrices if the primal and dual functions have compact support. Since the duals of the B-spline basis, represented by $\mathbf{A}_m$ and $\mathbf{B}_m$, are not compactly supported, we need to modify the decomposition scheme. Quak and Weyrich [30] propose an efficient and elegant scheme which does not explicitly use the dual operators.

Due to semi-orthogonality, both ϕ_{mp} and $\tilde{\phi}_{mp}$ are bases of V_m, as well as ψ_{mp} and ψ_{mp} are bases of W_m:

$$f^m(x) = \sum_{p=-j+1}^{2^m-1} c_{mp}\phi_{mp}(x) = \sum_{p=-j+1}^{2^m-1} \tilde{c}_{mp}\tilde{\phi}_{mp}(x). \tag{2.45}$$

Instead of using $\mathbf{A}_m$ and $\mathbf{B}_m$ for analysis, it is possible to carry out a basis transform from the primal into the dual basis. Taking the inner products with ϕ_{mp} and ψ_{mp}, respectively, yields

$$\tilde{\mathbf{c}}_m = (\langle \phi_{mp}, \phi_{m\tilde{p}} \rangle)_{p,\tilde{p}=-j+1}^{2^m-1} \mathbf{c}_m = \mathbf{C}_m \mathbf{c}_m \tag{2.46}$$

and

$$\tilde{\mathbf{d}}_m = (\langle \psi_{mp}, \psi_{m\tilde{p}} \rangle)_{p,\tilde{p}=-j+1}^{2^m-j} \mathbf{d}_m = \mathbf{D}_m \mathbf{d}_m. \tag{2.47}$$

The modified decomposition scheme is depicted in Figure 2.6.

Since the matrices $\mathbf{C}_m$ and $\mathbf{D}_m$ are banded it is possible to carry out the decomposition in $O(2^m)$. Note that it is not necessary to compute the inner products of the wavelets to derive $\mathbf{D}_m$. It follows directly from the two-scale relation of the wavelets that

$$\mathbf{D}_m = (\langle \psi_{mp}, \psi_{mq} \rangle)_{p,q=-j+1}^{2^m-j} = \mathbf{Q}_m^{\mathrm{T}} \mathbf{C}_{m-1} \mathbf{Q}_m. \tag{2.48}$$

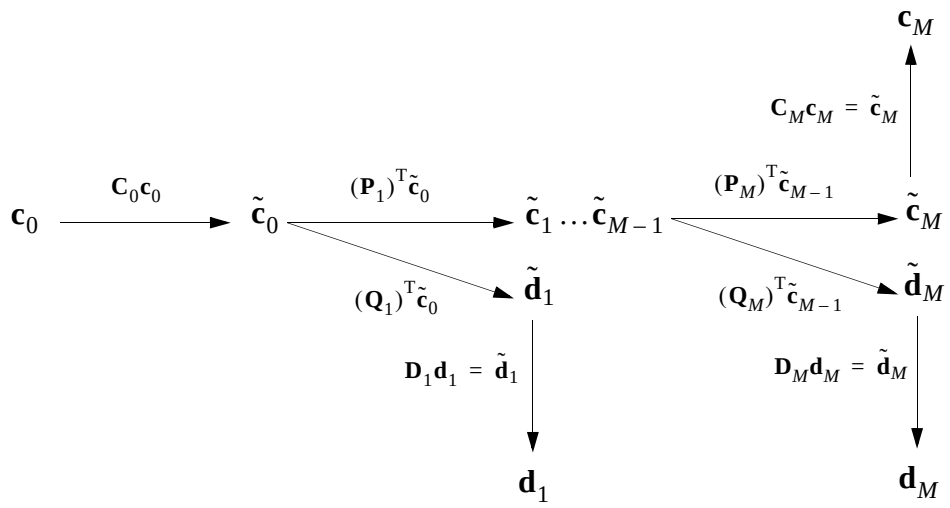


Figure 2.6 Modified decomposition scheme.

The matrices $\mathbf{C}_m$ and $\mathbf{D}_m$ are well conditioned which guarantees numerically stable solutions of the linear systems.

2.2.4 B-Spline Interpolation

Recalling the approximation properties of B–splines it is recommended to precompute an interpolation problem to get the appropriate coefficients related to the data samples to be interpolated. These types of interpolations are extensively investigated and relate tightly to inverse B–spline filtering problems which perform in linear time [106].

Interpolating B-spline curves and surfaces are necessary to solve the implicit interpolation problem using the B-spline bases introduced above (see Section 5.5 for details on implicit interpolation for iso-surface extraction).

2.2.5 Multi-Dimensional Extensions

So far, we have focused on the one-dimensional setting. For many practical applications, such as surface and volume representations, it is essential to extend the bases to two and three dimensions.

Basically, there are two different possibilities for the construction of multi-dimensional extension [98]:

1. *Separable:* Products of one-dimensional bases (tensor product).
2. *Non-separable:* Genuinely multi-dimensional bases (e.g., Gabor functions).

We will derive separable two- and three-dimensional bases from the one-dimensional B-spline basis introduced in the preceding section. Given a basis ϕ_{mp} for V_m, the corresponding basis for the bi-variate space $V_m^2 = V_m \times V_m$ is the separable tensor product basis

$$\phi^2_{mpq}(x, y) = \phi_{mp}(x)\phi_{mq}(y)\,. \tag{2.49}$$

There are two different approaches for the construction of the complement spaces, which both will be explained briefly for the two-dimensional setting.

Standard decomposition. To obtain the standard decomposition [6, 97]], the one-dimensional wavelet transform is applied to all rows of the data. Next, the one-dimensional transform is applied to each column of the data resulting from the previous transforms. The resulting values of this process are all detail coefficients except for one single overall average coefficient. Therefore, a function $f(x, y) \in L^2(\subsetneq^2)$ at the initial resolution m_0 can be represented as

$$\begin{aligned} f(x,y) &= \sum_{p,q \in \subsetneq} c_{m_0} \phi_{m_0 p}(x) \phi_{m_0 q}(y) \\ &= \sum_{p,q \in \subsetneq} c_{m_0} \phi^2_{m_0 pq}(x,y) \\ &= \sum_{m_1 = m_0}^{M} \sum_{m_2 = m_0}^{M} \sum_{\text{type}=1}^{3} \sum_{p,q \in \subsetneq} d^{\text{type}}_{M m_1 m_2 pq} \psi^{2,\text{type}}_{M m_1 m_2 pq}(x,y) + \sum_{p,q \in \subsetneq} c_{Mpq} \phi^2_{Mpq}(x,y) \end{aligned} \tag{2.50}$$

with the four types of basis functions

$$\begin{aligned} \phi^2_{Mpq}(x,y) &= \phi_{Mp}(x) \phi_{Mq} \\ \psi^{2,1}_{M m_1 m_2 pq}(x,y) &= \psi_{m_1 p}(x) \phi_{Mq}(y) \\ \psi^{2,2}_{M m_1 m_2 pq}(x,y) &= \phi_{Mp}(x) \psi_{m_2 q}(y) \\ \psi^{2,3}_{M m_1 m_2 pq}(x,y) &= \psi_{m_1 p}(x) \psi_{m_2 q}(y) \end{aligned} . \tag{2.51}$$

One sees immediately that wavelets from different resolution levels $m_1, m_2 \in \subsetneq^2$ have to be combined. This does not allow direct control of resolution level m. The non-standard decomposition overcomes this problem.

Nonstandard decomposition. Instead of performing a full 1-D transform first for all rows and then for all columns, it is possible to alternate between row and column transformation at each level. Accordingly, a function $f(x, y) \in L^2(\subsetneq^2)$ can be approximated as follows:

$$\begin{aligned} f(x,y) &= \sum_{p,q} c_{m_0 pq} \phi^2_{m_0 pq}(x,y) \\ &= \sum_{m=m_0+1}^{M} \sum_{\text{type}=1}^{3} \sum_{p,q \in \subsetneq} d^{\text{type}}_{mpq} \psi^{2,\text{type}}_{mpq}(x,y) + \sum_{p,q \in \subsetneq} c_{Mpq} \phi^2_{Mpq}(x,y) \end{aligned} \tag{2.52}$$

with

$$\begin{aligned} \phi^2_{mpq}(x,y) &= \phi_{Mp}(x) \phi_{Mq} \\ \psi^{2,1}_{mpq}(x,y) &= \psi_{mp}(x) \phi_{mq}(y) \\ \psi^{2,2}_{mpq}(x,y) &= \phi_{mp}(x) \psi_{mq}(y) \\ \psi^{2,3}_{mpq}(x,y) &= \psi_{mp}(x) \psi_{mq}(y) \end{aligned} . \tag{2.53}$$

In contrast to the standard decomposition, there are no combinations of basis functions of different levels. Another consideration is the support of the basis functions. All of the nonstandard basis functions have square support, whereas some of the standard basis functions have non-square support. Furthermore, the computation of the standard decomposition is slightly less efficient than that of the nonstandard decomposition. Hence, the latter is used throughout this thesis, because of the better control of the resolution level, higher efficientcy and square support.

Figure 2.7 depicts the nonstandard decomposition for two resolution steps in two dimensions.

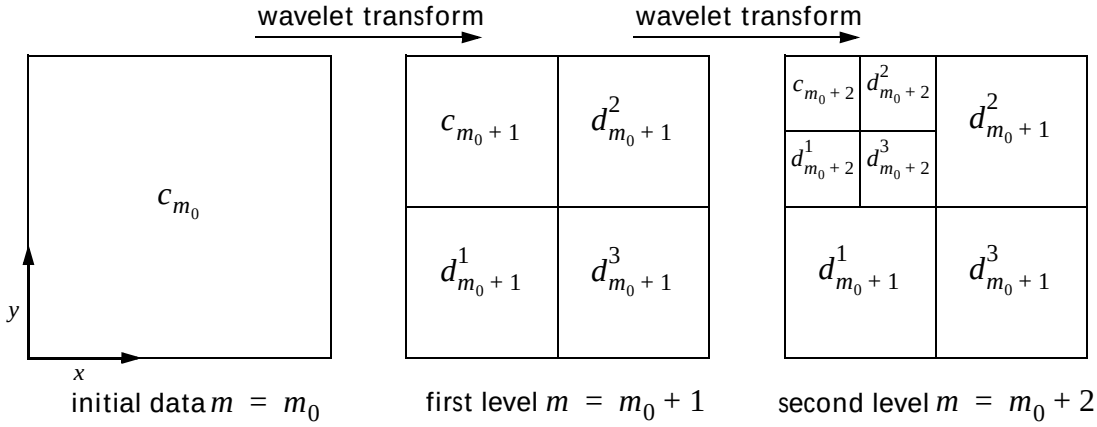


Figure 2.7 Two-dimensional wavelet transform with nonstandard decomposition.

The two-dimensional wavelet transform described in this section can be extended to three or more dimensions straightforwardly and will not be discussed here explicitly for that reason.

A two-dimensional B-spline and B-wavelet are depicted in Figure 2.8. Note that both, scaling function and wavelet have square support.

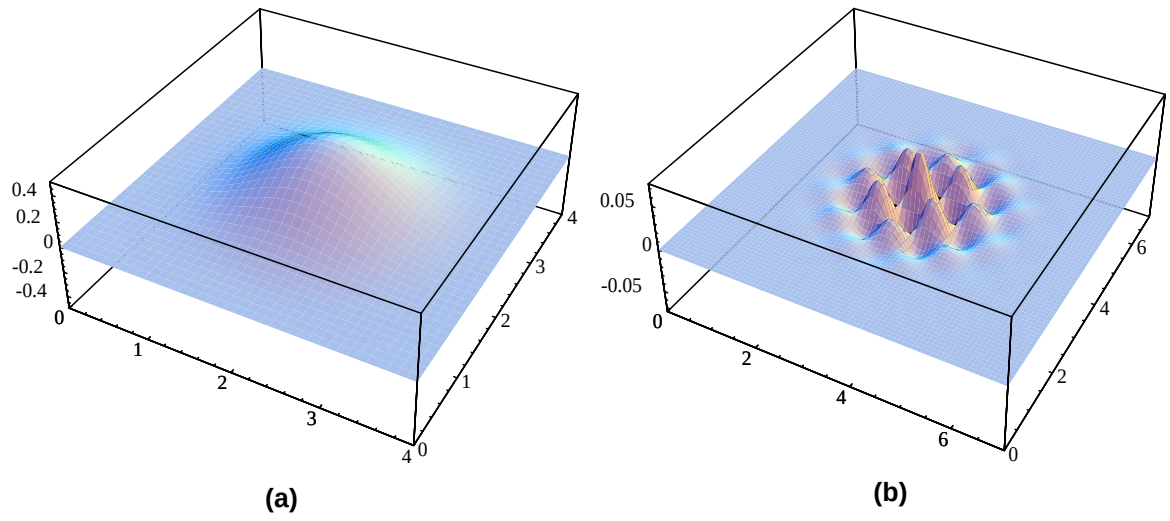


Figure 2.8 Two-dimensional B-spline basis: **a**: scaling function. **b**: wavelet.

2.3 BOUNDARY DISTORTION

All definitions in this chapter hold for functions in $L^2(\subsetneq)$, functions defined on the real axis and its higher dimensional analogs. For the application of surface and volume representations, strategies for dealing with functions on bounded intervals, squares, and cubes have to be derived. Usually, basis functions with support greater than one overlap interval boundaries as illustrated in Figure 2.9.

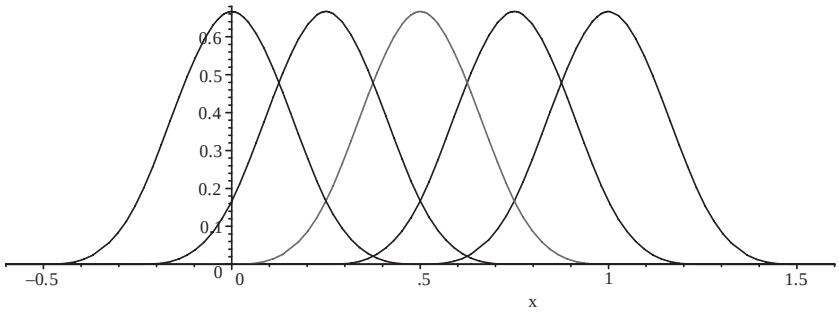


Figure 2.9 Cubic B-splines overlapping the interval [0,1].

Possible solutions to avoid overlapping include a suitable extension of a given function to the real axis or its periodic version [57]. Another possibility is the construction of special boundary elements, which define—in conjunction with the usual wavelets and scaling functions for the interior—a multi-resolution analysis on the bounded interval [30]. See [20] for a detailed discussion of these and other approaches.

2.3.1 Simple Methods

Let us restrict the investigation to the unit interval $[0, 1]$. A very simple and intuitive solution is to set $f(x) = 0$ outside $[0, 1]$ and then use the wavelet theory on the real axis. Unfortunately, this "zero padding" introduces discontinuities at the endpoints 0 and 1. This can be seen easily if one considers the simple function $f(x) = 1$ with $x \in [0, 1]$. Since wavelets are very effective in detecting singularities in data, such artificial singularities are likely to introduce severe errors.

Another approach, which is widely used in practice, is periodization of the function $f(x)$ with period 1, so $f(x + 1) = f(x)$. In other words, f is defined on the torus with $[0, 1]$. Wavelet theory on the torus parallels that on the real axis. This allows one to construct decomposition and reconstructions operators by wrapping the overlapping rows of the matrix. which is illustrated in Figure 2.10. This approach is, for example, used in [82].

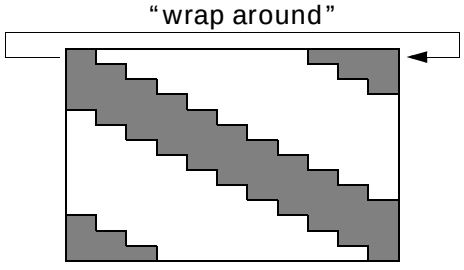


Figure 2.10 Periodization of the operator.

This "wrap around" is satisfactory in many situations and it is obviously successful for functions like $f(x) = 1,\ x \in [0, 1]$. However, unless the behavior of f at 0 matches that at 1, the periodization of f has singularities there. A good example is the function $f(x) = x,\ x \in [0, 1]$.

These approaches are not satisfactory for applications in geometric surface and volume representations. The use of oracles for data compression, which are introduced in Section 2.4.1, leads to lossy data approximation where only a subset of the wavelets are used for reconstruction. This might amplify the discontinuities and might introduce severe artefacts.

2.3.2 Endpoint-Interpolating Bases

B-spline wavelets intrinsically defined on the bounded interval $L^2[0, 1]$ were first introduced by [14] and several decomposition and reconstruction algorithms based on these bases have been proposed by [30]. The efficient decomposition scheme from [30] has already been discussed in Section 2.2.3. This section will define the special boundary elements that will extend the wavelet transform from Section 2.2.3 to an MRA in $L^2[0, 1]$.

Let us take a closer look at the B-spline scaling functions defined in Equation 2.33 on page 13.

Let $\boldsymbol{t}_m^j := \{t_m^k\}_{k=-j+1}^{2^m+j-1}$, with

$$t_m^{-j+1} = t_m^{-j+2} = \ldots = t_m^0 = 0,$$

$$t_m^k = k2^{-m}, \quad k = 1, \ldots 2^m - 1,$$

$$t_m^{2^m} = t_m^{2^m+1} = \ldots = t_m^{2^m+j-1} = 1$$

be a knot sequence on $[0, 1]$. The corresponding B-splines are defined as

$$B_{mp}^j(x) = N^j(2^m x - p), \quad p = 0, \ldots 2^m - j,$$

where $N^j(x)$ is the cardinal B-spline of order j. For $p = -j+1, \ldots, -1$, B_{mp}^j contains a multiple knot at 0 and for $p = 2^m - j + 1, \ldots, 2^m - 1$, a multiple knot at 1. Chui and Quak show in [14] that $\phi_{mp}^{[j]} = B_{mp}^j$ span V_m. Endpoint-interpolating cubic B-splines are depicted in Figure 2.11.

To complete the definition of an endpoint-interpolating B-spline MRA, appropriate basis functions for W_m have to be found. After defining the scaling functions, the wavelets are constrained, though not completely determined. For the wavelets to be orthogonal to the scaling functions, the columns of $\mathbf{Q}_m$ must form a basis for the null space of $\mathbf{M}_m := (\mathbf{P}_m)^T[\langle \Phi_m, \Phi_m \rangle]$. There are many possible bases for the null space of a matrix. To determine $\mathbf{Q}_m$, one can decide what properties the wavelets should have. In this thesis, the approach of Finkelstein and Salesin [33], who construct wavelets with minimal support, is taken. See Figure 2.12 for the endpoint-interpolating cubic B-wavelets.

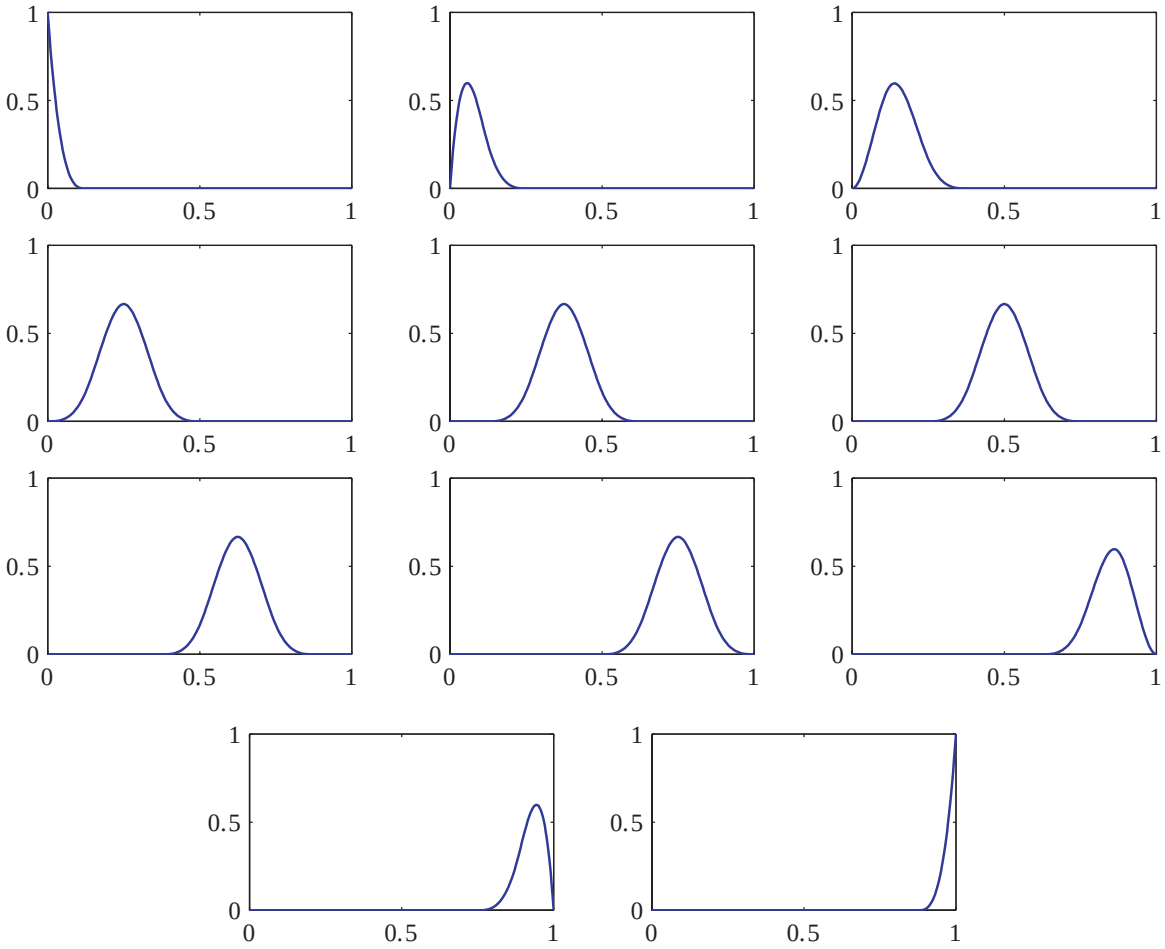


Figure 2.11 Sequence of endpoint-interpolating cubic B-splines at level $m = 3$ in the interval $[0, 1]$ with multiple knots at 0 and 1.

2.4 ORACLES

One of the most important applications of wavelets is data compression. Instead of using all wavelets ψ_{mp} for reconstruction, it is possible to truncate the approximation. This method belongs to the class of lossy compression schemes, which will be described in depth in Section 4.1. An oracle is used to determine the set of wavelets that are omitted in the truncated approximation.

2.4.1 Global Oracles

A global oracle rejects unimportant wavelet coefficients from the approximation while minimizing a given error norm.

First, let us recall finite dimensional orthogonal approximations for $L^2(\subsetneq)$. The basis $\{\phi_i(x)\}_{i=1\ldots N}$ expands a function $f(x)$ as

$$f(x) = \sum_{i=1}^{N} c_i \phi_i(x), \tag{2.54}$$

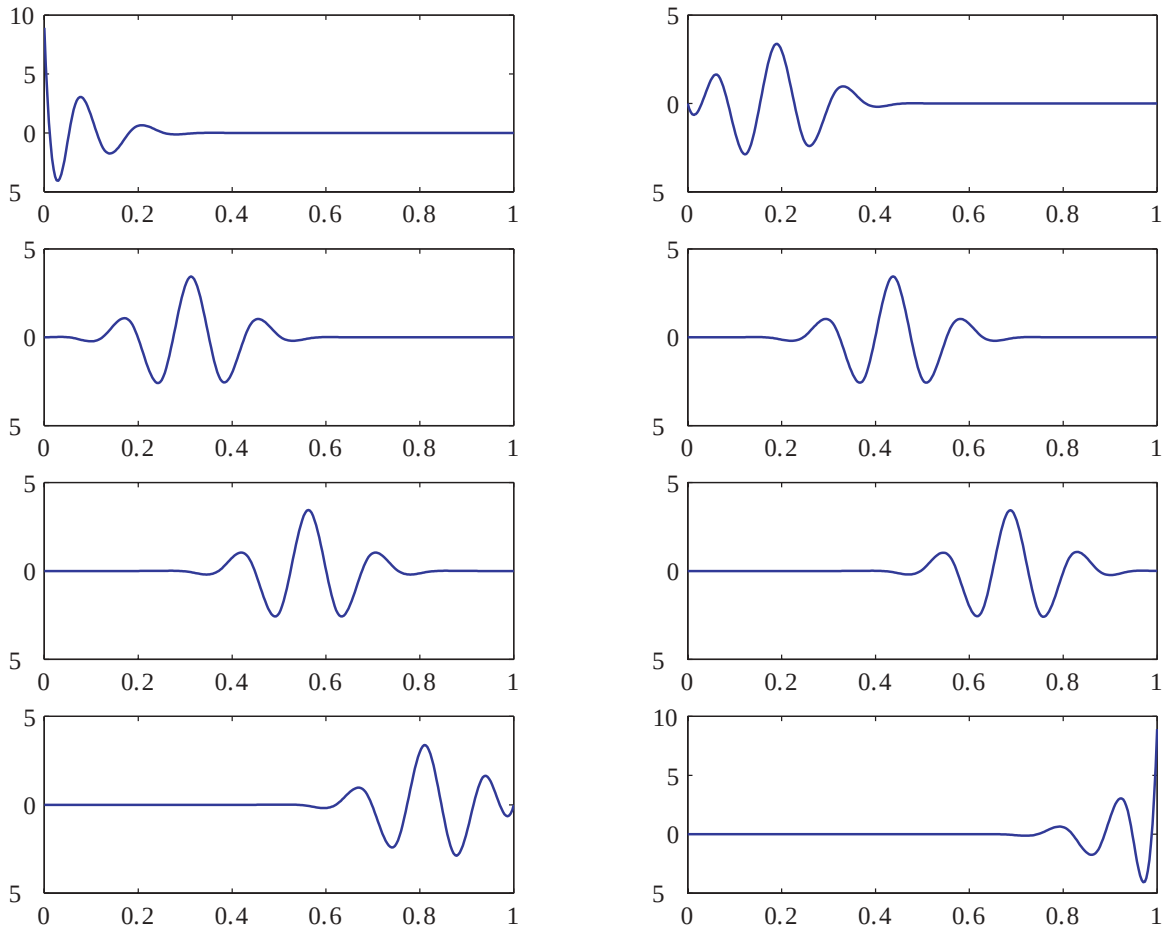


Figure 2.12 Sequence of endpoint-interpolating cubic B-wavelets at level $m = 3$ with multiple knots at 0 and 1.

with coefficients $c_i \in V_{m_0}$. Given an integer $1 \le K < N$, we try to find the best approximation of f with the truncated approximation f'. The L^2 error of $f'(x)$ is determined by the linear combination of the rejected basis functions

$$\begin{aligned} \|f(x) - f'(x)\|_{L^2}^2 &= \left\| \sum_{i=K+1}^{N} c_i \phi_i(x) \right\|_{L^2}^2 \\ &= \left\langle \sum_{i=K+1}^{N} c_i \phi_i(x), \sum_{i=K+1}^{N} c_i \phi_i(x) \right\rangle. \\ &= \sum_{i=K+1}^{N} |c_i|^2 \end{aligned} \tag{2.55}$$

Note, that the orthogonality of $\langle \phi_i, \phi_j \rangle = \delta_{ij}$ cancels out all intermediate terms in Equation 2.55, which simplifies the relation. In other words, the magnitude of the coefficient c_i corresponds exactly to the fraction of energy provided by the corresponding basis ϕ_i. Hence, the L^2-energy of a function $f(x)$ is computed by

$$\|f(x)\|_{L^2}^2 = \langle f(x), f(x) \rangle = \int_{-\infty}^{\infty} |f(x)|^2 dx \quad . \tag{2.56}$$

Finding the "best" truncated approximation can be formulated in terms of two variants [40]:

1. Find a subset of K out of N coefficients which minimizes the overall approximation error induced by rejection of the corresponding basis functions. These hold for optimal compression of a predefined ratio $N/(N-K)$.

2. Reject a maximum number of coefficients while keeping the approximation error below a predefined bound E_b.

A globally optimal compression can thus be achieved by sorting the coefficients by their magnitude and by rejecting the K smallest ones [97]. Obviously, this strategy yields an L^2-optimal oracle for orthogonal bases and can be computed in $O(N \log N)$ using a fast sorting algorithm [22].

Unfortunately, in the semi-orthogonal case the intermediate terms in Equation 2.55 are not canceled out, which complicates the construction of an oracle. Maximum distance norm oracles have been proposed by Stollnitz and others [96] for bi-orthogonal wavelets.

Gross [40] and Staadt et al. [95] propose a global oracle for the semi-orthogonal setting. The computational scheme for the global oracle is based on the observation that the energy of a function expanded by semi-orthogonal wavelets is obtained by the following sum of scalar products:

$$\|f(x)\|_{L^2}^2 = \left\langle \mathbf{c}^M, \tilde{\mathbf{c}}^M \right\rangle + \sum_{m=1}^{M} \left\langle \mathbf{d}^m, \tilde{\mathbf{d}}^m \right\rangle . \tag{2.57}$$

Due to the orthogonality of different complement spaces it is sufficient to analyze the error norm of a single space W_m. In order to derive an incremental method we assume K out of $N/2^m + j - 1$ coefficients in this space to vanish. The approximation error is then determined by the following relation:

$$\begin{aligned} &\left\| \Delta f^m(x) - \Delta f'^m(x) \right\|_{L^2}^2 \Big|_{\mathrm{Rej}(K)} \\ &= \left\langle \sum_{i \in \mathrm{Rej}(K)} d_{mi} \psi_{mi}(x), \sum_{i \in \mathrm{Rej}(K)} d_{mi} \psi_{mi}(x) \right\rangle \\ &= \sum_{i \in \mathrm{Rej}(K)} \sum_{j \in \mathrm{Rej}(K)} d_{mi} d_{mj} \cdot \langle \psi_{mi}(x), \psi_{mj}(x) \rangle \end{aligned} \tag{2.58}$$

where $\Delta f'^m(x)$ represents the residual approximation and $\mathrm{Rej}(K)$ denotes the set of all coefficients being rejected from the initial approximation. In a subsequent step, it is computed how the upper error behaves when rejecting an arbitrary $d_{mk} \neq 0$, assuming K coefficients to have already been rejected in previous procedures. That is, the expression for the error increment generated by a single coefficient is computed.

$$\left\|\Delta f^m(x) - \Delta f'^m(x)\right\|_{L^2}^2\Big|_{\mathrm{Rej}(K,\,k)} \tag{2.59}$$

$$= \left\|\Delta f^m(x) - \Delta f'^m(x)\right\|_{L^2}^2\Big|_{\mathrm{Rej}(K)} + (d_{mk})^2$$

$$+ \; 2 \cdot \sum_{i \,\in\, \mathrm{Rej}(K)} d_{mk} d_{mi} \cdot \left\langle \psi_{mk}(x), \psi_{mi}(x) \right\rangle$$

Equation 2.59 expresses the dependence of the error on an increment of the rejection set. It will be referred to as the *conditional approximation error* in all subsequent discussions. The factor of 2 follows immediately from the symmetry of the inner product matrix. Apparently, the conditional increment is computed by adding one row and one column to the matrix type structure representing the double summation in Equation 2.58, such as depicted in Figure 2.13.

To sum up, the error can be updated by adding the products of the coefficient d_{mk} and the elements of the rejection set multiplied by the associated entry of the inner product matrix. Note that this error can be considered a score which reflects the conditional importance of an individual coefficient.

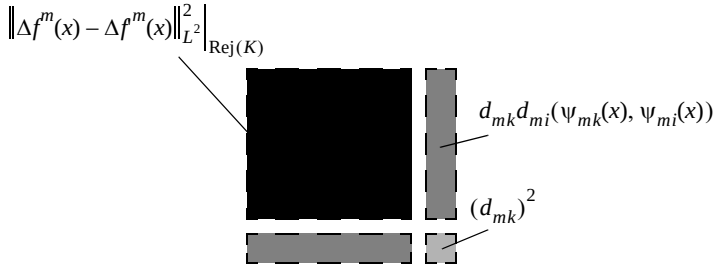


Figure 2.13 Illustration of the conditional approximation error increment.

The relations derived above represent an essential step towards the development of an oracle. They allow one to predict how the approximation error changes when rejecting an individual wavelet coefficient under the precondition that K other coefficients have been rejected earlier. Based on this fundamental relationship, it is possible to develop a greedy rejection algorithm which optimizes locally and computes a minimum error rejection set of coefficients. In essence, the greedy oracle operates as follows: It first assigns an initial score to all wavelet coefficients of all iterations m. The score is defined by the overall conditional approximation error, which governs the oracle. In a second step, the oracle iteratively selects the coefficient with the minimum score, rejects it, and recomputes the scores of all other coefficients. The iteration loop runs up to a predefined number of cycles K or up to a predefined error bound E_b. As with Equation 2.59, the score can be recomputed by an appropriate increment after each iteration. Thus we end up with a simple reject–and–update scheme for our oracle. The pseudocode is provided below:

```
Initialize: score[i,m] ← d[i,m]·d[i,m]; old_score ← 0
for i ← 1 to K
  for m ← 1 to M do
    Search: coefficient[i_rej,m_rej]|score[irej,mrej] = min ≠ 0;
      Reject: d[i_rej,m_rej] ← 0;
      if m = m_rej && score[i,m] ≠ 0 then
        increment[i,m] ← 2·d[i,m]·d[i_rej,m_rej]·ψ[i,i_rej,m]
                + score[i_rej,m_rej] - old_score
```

```
      else if score[i,m] ≠ 0
        increment[i,m] ← score[i_rej,m_rej] - old_score;
      Update: score[i,m] ← score[i,m] + increment[i,m];
  end;
```

After each step, the `score[i,m]` of a coefficient `d[i,m]` represents the overall conditional approximation error when rejecting `d[i,m]`. Note that the time-complexity of the oracle is equal to $O(N^2)$ and applies only on forward compression.

2.4.2 Local Oracles

A local oracle allows one to control the approximation locally in interesting regions. Here, the spatial localization of the basis functions enables one to accentuate particular wavelets while suppressing the influence of others. In this understanding, a straightforward local oracle consists of a weighting function which operates on the coefficients of the transform. A first approach to this is given in Gross and others [43] who employed a Gaussian weighting. The basic idea is to assume some ellipsoidal weighting area as a local region of interest in the spatial domain. Localization of the wavelet transform allows for the projection of scaled and translated versions of it into wavelet space, where individual coefficients are influenced. The initial version presented in [43], however, did not consider the support regions of individual basis functions, and can lead to some artifacts by rejecting wavelets ranging into the region of interest. Therefore, the method has been extended in [95] by computing the support of the basis functions and by using endpoint-interpolating B-wavelets as described in Section 2.3.2.

The use of local oracles corresponds to a filter operation in wavelet space. It can be defined in analogy with the well known filters in frequency domain. Since the filter affects the local frequency characteristics of the signal it is called *wavelet space filter*.

In the two-dimensional case, let $g(x, y)$ be a Gaussian weighting function, centered at $(x_0 y_0)$, scaled by (σ_x, σ_y), and rotated by Θ, which quantifies the local level-of-detail around some location in space (x_0, y_0) and whose elliptical shape is depicted in Figure 2.14 (a).

Note that to compute its transformation into wavelet space, any point (x_0, y_0) in spatial domain can only be located within the Heisenberg bound in wavelet domain. Furthermore, the spatial localization decreases with increasing iteration depth m.

With the dyadic scale of the two-dimensional wavelet transform, however, the Gaussian splits into all frequency channels and its centers in wavelet space are computed according to

$$x_0^m := \frac{x_0}{2^m} \qquad y_0^m := \frac{y_0}{2^m}. \tag{2.60}$$

The rotation angle Θ is invariant to the transform. The set of Gaussian weighting functions $\{g^m(x^m, y^m)\}$ in wavelet space can be described elegantly by using homogeneous coordinates:

$$g^m(x^m, y^m) = e^{-(\mathbf{R}^m \cdot \mathbf{p}^m)^T (\mathbf{R}^m \cdot \mathbf{p}^m) + 1}. \tag{2.61}$$

The matrix $\mathbf{R}^m$ represents the affine transform of the Gaussian:

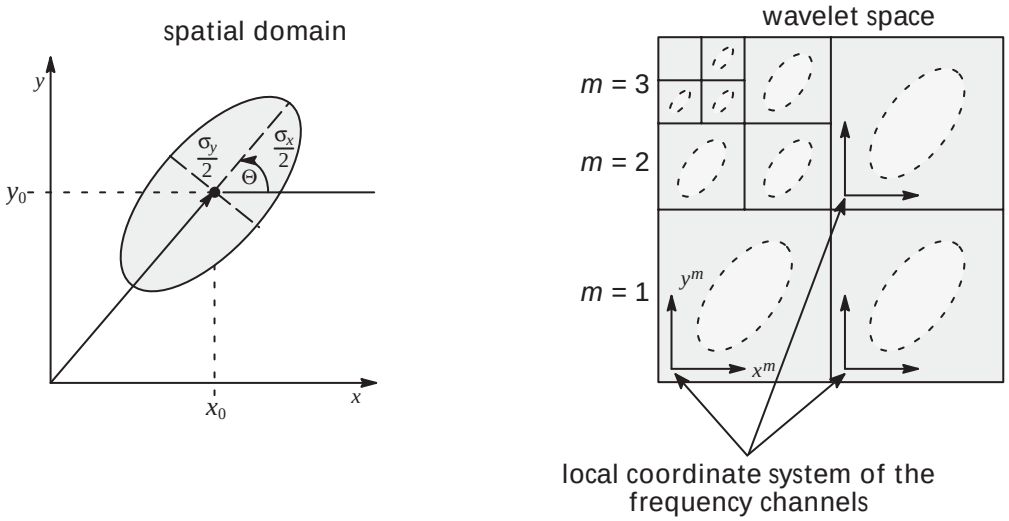


Figure 2.14 Filtering in wavelet space. **a**: Rotated, translated, and scaled two-dimensional Gaussian weighting function in spatial domain. **b**: Transformation of the filter into wavelet space results in multiple Gaussians located in each frequency channel.

$$\mathbf{R}^m = \begin{bmatrix} \frac{\cos\Theta}{\sigma_x^m} & \frac{\sin\Theta}{\sigma_x^m} & \frac{-x_0\cos\Theta + y_0\sin\Theta}{\sigma_x^m} \\ \frac{-\sin\Theta}{\sigma_y^m} & \frac{\cos\Theta}{\sigma_y^m} & \frac{x_0\sin\Theta - y_0\cos\Theta}{\sigma_y^m} \\ 0 & 0 & 1 \end{bmatrix} \tag{2.62}$$

and $\mathbf{p}^m = (x^m, y^m, 1)^{\mathrm{T}}$ denotes a position in homogeneous coordinates.

Examples of the use of global and local oracles will be given in Chapter 7.

2.5 IMPLEMENTATION OF THE FAST WAVELET TRANSFORM

We have seen in this chapter that the complexity of the fast wavelet transform is generally of order $O(N)$, where N denotes the length of the data vector to be transformed. In the special case of bi- or semi-orthogonal wavelets this is not necessarily true. The B-spline wavelet basis, however, allows one to construct an algorithm with linear complexity. Let us first consider the one-dimensional case.

2.5.1 One-dimensional Decomposition

Given a data vector $\mathbf{c}_m$ of length

$$N = 2^{M-m} + d,$$

where d is the degree of the B-spline—in the case of cubic B-splines d is equal to three. Note that the original data vector has to be expanded by repeating the last data value to yield the required length of $2^M + d$. The decomposition according to Figure 2.6 on

page 18 transforms $\mathbf{c}_m$ into dual space by multiplication with the inner-product matrix $\mathbf{C}_m$

$$\tilde{\mathbf{c}}_m = \mathbf{C}_m \mathbf{c}_m . \tag{2.63}$$

Since $\mathbf{C}_m$ is a band diagonal system, Equation 2.63 can be computed in $O(N)$. The actual transform is performed by applying the following operations:

$$\begin{aligned} (\mathbf{P}_{m+1})^{\mathrm{T}} \tilde{\mathbf{c}}_m &= \tilde{\mathbf{c}}_{m+1} \\ (\mathbf{Q}_{m+1})^{\mathrm{T}} \tilde{\mathbf{c}}_m &= \tilde{\mathbf{d}}_{m+1} \end{aligned} . \tag{2.64}$$

The length of the vectors $\tilde{\mathbf{c}}_{m+1}$ and $\tilde{\mathbf{d}}_{m+1}$ is equal to $2^{M-m+1} + d$ and 2^{M-m+1}, respectively. Hence, the total length of the decomposition is equal to the length of the original vector, which allows for space-efficient in-place implementation.

To recover the detail signals $\mathbf{d}_{m+1}$, $\tilde{\mathbf{d}}_{m+1}$ has to be transformed back into primal space by solving

$$\mathbf{D}_{m+1} \mathbf{d}_{m+1} = \tilde{\mathbf{d}}_{m+1} . \tag{2.65}$$

Although $\mathbf{D}_{m+1}$ is band diagonal as well, its inverse is dense and ill conditioned. Therefore, brute force matrix inversions cannot be applied. The application of an LU-decomposition, however, allows us to find $\mathbf{d}_{m+1}$ in linear time [82]. $\mathbf{D}_{m+1}$ is diagonal dominant and well conditioned. Thus, the LU-decomposition is numerically stable. Note that $\mathbf{D}_{m+1}$—as well as $\mathbf{P}_{m+1}$, $\mathbf{Q}_{m+1}$, and $\mathbf{C}_{m+1}$—can be precomputed and stored implicitly.

At level M, $\mathbf{c}_M$ can be computed in the same fashion as the detail signal at the previous levels.

2.5.2 Decomposition in Higher Dimensions

The decomposition scheme for the one-dimension case needs to be adapted for higher dimensions. Whereas in one dimension primal and dual coefficients are fully separated, in higher dimensions we encounter mixed bands comprising both, primal and dual coefficients. Figure 2.15 illustrates the different steps which are required to perform a single iteration of a two-dimensional decomposition.

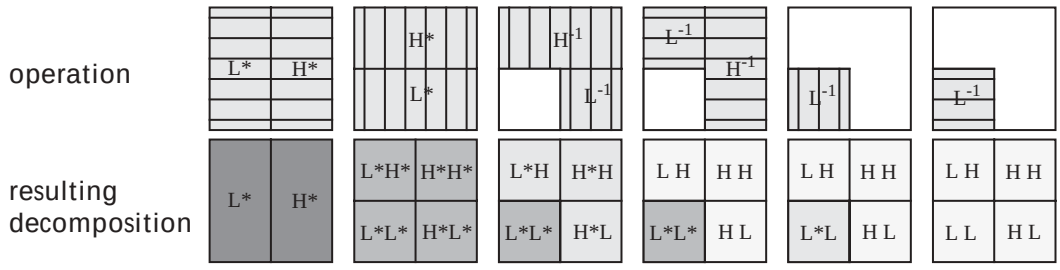


Figure 2.15 Required steps for a single iteration of a two-dimensional decomposition: The upper row shows the single operations and the lower row the resolution decomposition. L^* denotes scaling function coefficients in dual space, L denotes the scaling functions in primal space, L^{-1} denotes the inverse operator which projects back into primal space. H^*, H, and H^{-1} are defined accordingly.

Note that the backprojection of the pure scaling function subband into primal space has to be carried out separately.

It is straightforward to see that the complexity of the reconstruction scheme of Figure 2.5 on page 16, which only involves the operators $\mathbf{P}_m$ and $\mathbf{Q}_m$, is $O(N)$.

To sum up, the overall complexity of our implementation of the fast wavelet transform is $O(N)$.

A quantitative analysis of the fast wavelet transform yields the following number of multiplications for each data value is listed in Table 2.2.

Table 2.2 Quantitative analysis of the fast wavelet transform.

Operation	# multiplications
Projection of c_i into dual space: $\mathbf{C}_m\mathbf{c}_m = \tilde{\mathbf{c}}_m$	4
Projection into scaling functions: $(\mathbf{P}_{m+1})^T\tilde{\mathbf{c}}_m = \tilde{\mathbf{c}}_{m+1}$	3/2
Projection into wavelet functions: $(\mathbf{Q}_{m+1})^T\tilde{\mathbf{c}}_m = \tilde{\mathbf{d}}_{m+1}$	5/2
Backprojection of scaling function into primal space	7/2
Backprojection of wavelet function into primal space	13/2
Reconstruction of scaling function	(2+1)/2
Reconstruction of wavelet function	(3+2/2
Total	**22**

The total number of multiplications can be further reduced, since the initial projections into dual space and the projection of the scaling function coefficients back into primal space have to be carried out only once. Hence, full decomposition and reconstruction of a one-dimensional data vector does only require 30.25 multiplications per data value. Further details on the implementation can be found in [59].

3

GEOMETRY & TOPOLOGY

In this chapter, some basic concepts of discrete geometry and piecewise-linear topology are introduced. Furthermore, some practical issues of mesh representations and related error metrics are being discussed.

3.1 PRINCIPLES OF PIECEWISE-LINEAR TOPOLOGY

To define both triangulated and tetrahedral mesh representations, some abstractions from piecewise-linear topology [87, 55] are very convenient.

A geometric model is represented as a tuple (K, V). K is a combinatorial structure specifying the connectivity of vertices, edges, triangles, tetrahedra, etc. It is an *simplicial complex*, which realizes the *microtopology* of the model.

The *geometry* V is a set of vertex positions specifying the shape of the model in $\mathbb{R}^n$.

Topological properties of a model that are independent of its discrete representation are regarded a *macrotopology*. Those properties include the definition of an n-manifold and its *genus* (i.e., the number of "holes" in the model).

3.1.1 Complexes

Simplicial complexes are the principal building blocks in piecewise linear topology. Basically, they comprise simplices, cells, and cell complexes.

Simplices. The points $\{\mathbf{v}_0, \mathbf{v}_1, \ldots, \mathbf{v}_n\} \subset \mathbb{R}^m$ are called *independent* if the vectors $\{\mathbf{v}_i - \mathbf{v}_0\}$ are linearly independent. They define an *n-simplex* $A \subset \mathbb{R}^m$. The points $\mathbf{v}_i$ are called *vertices* and they *span* simplex A. A simplex spanned by a subset of the vertices is called a *face* of A. The set of faces of A is denoted with $\text{faces}(A)$. If B is a face of A we can write $B < A$. Vertices are 0-simplices and they are also regarded as faces. The empty set is the (-1)-simplex, which has no vertices and is thus a face of all simplices. Usually, 1-simplices are called *edges*, 2-simplices are called *triangles*, and 3-simplices are called *tet-*

rahedra. The *star* $\text{star}(A)$ of a simplex A is defined as the set of simplices of which A is a face.

Cells. A subset $C \subset \subsetneq^m$ is convex if for each pair of points $\mathbf{a}, \mathbf{b} \in C$, the straight-line segment $\overline{\mathbf{ab}} \subset C$. A compact convex polyhedron which spans a subspace of dimension n is called *n-cell.* Note that each n-simplex is an n-cell, but not vice versa.

Cell complexes. A *cell complex* K is a finite collection of cells in $\subsetneq^n$ satisfying the following conditions [87]:

1. From $C \in K$ and $B < C$ follows $B \in K$.
2. If $B, C \in K$ then $B \cap C$ is a face of both B and C.

The definition of simplicial complexes follows immediately from the definitions of simplices and cell complexes.

Simplicial complexes. A cell complex K is a *simplicial complex* if each cell $C \in K$ is a simplex. The relation between a simplicial complex and its simplices and faces is depicted in Figure 3.1. The *children* of an n-simplex A are the $(n-1)$-simplices in $\text{faces}(A)$, and its *parents* are the $(n+1)$ simplices in $\text{star}(A)$. A is a *boundary simplex* if it only has one parent, and its a *principal simplex* if it has no parents at all. If all principal simplices of the simplicial complex K are of the same dimensions, it is said to be *regular*.

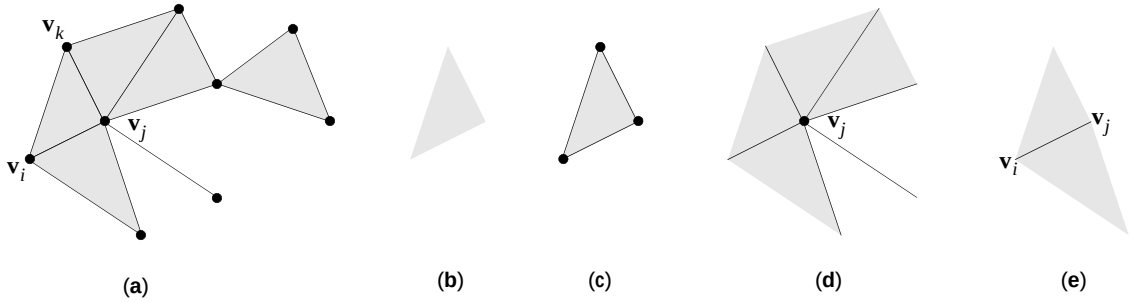


Figure 3.1 **a:** Simplicial complex K. **b:** A 2-simplex A spanned by $\{\mathbf{v}_i, \mathbf{v}_j, \mathbf{v}_k\}$. **c:** The faces of A. **d:** $\text{star}(\{\mathbf{v}_j\})$. **e:** $\text{star}(\{\mathbf{v}_i, \mathbf{v}_j\})$. [80]

3.1.2 Manifolds

A polyhedron M is an *unbounded piecewise-linear manifold of dimension* n, or short n-manifold, if each point $x \in M$ has a neighborhood in M which is homeomorphic to an open set in R^n [87]. A homeomorphism is a continuous one-to-one mapping between two sets whose inverse is also continuous.

M is an *n-manifold with boundary* if each point has a neighborhood in either $\subsetneq^n$ or $\subsetneq^n_+$. The subset $\subsetneq^n_+ \subset \subsetneq^n$ is defined by $\{x \in \subsetneq^n : \ x_n \geq 0\}$.

2-manifold surfaces are a very powerful tool to model a variety of geometric models. Intuitively, a 2-manifold can be interpreted as a surface without self intersections.

3.1.3 Non-Manifolds and Singularities

Consider the simplicial complex K depicted in Figure 3.2. Apparently, this is a non-manifold surface, since the vertex $\mathbf{v}$ does not have a neighborhood homeomorphic to either $\subsetneq^2$ or $\subsetneq^2_+$. Although K does not satisfy the manifold condition, it is a pseudo-manifold

as defined in [71]. An n-dimensional *pseudo-manifold* P is a an n-dimensional regular complex which satisfies the following conditions:

1. Every face in P is a face of some n-cell.
2. Every $(n-1)$-dimensional cell is the face of exactly two n-cells.
3. Given two n-cells C and C' there exists a sequence of n-cells $\{C_0, \ldots, C_k\}$ where $C = C_0$, $C' = C_k$ and each pair $\{C_{i-1}, C_i\}$ has a common $n-1$-dimensional face.

Singular vertices in pseudo-manifolds are very important, since their removal may inflict changes of the macrotopology of the complex. The removal of vertex v in Figure 3.2 would split the surface into two disconnected simplicial complexes. This property should be kept in mind when designing mesh simplification methods, which should preserve the macrotopology of the model.

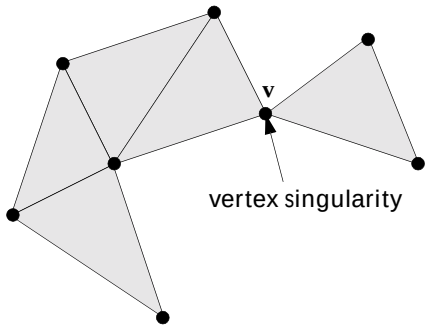


Figure 3.2 Pseudo-manifold with singular vertex v.

3.1.4 Collapsing and Shelling

Collapsing is a very useful tool in algebraic topology [87]. It forms the theoretical foundation on which many powerful surface and volume representation schemes are based on. Those schemes includes many popular methods which employ edge-collapse and vertex-split operations (see Chapter 6)[52, 35, 80, 54, 94].

Given two polyhedra $X \supset Y$, where $X = Y \cup B^n$ and $Y \cap B^n = B^{n-1}$. B^{n-1} is a face of X. We can say that there is an *elementary collapse* of X on Y *across* B^n *onto* B^{n-1} from the complementary face $C^{n-1} = \text{clos}(\partial B^n - B^{n-1})$ and write $X \downarrow Y$. ∂B^n denotes the boundary of B^n. The collapse operation is illustrated in Figure 3.3.

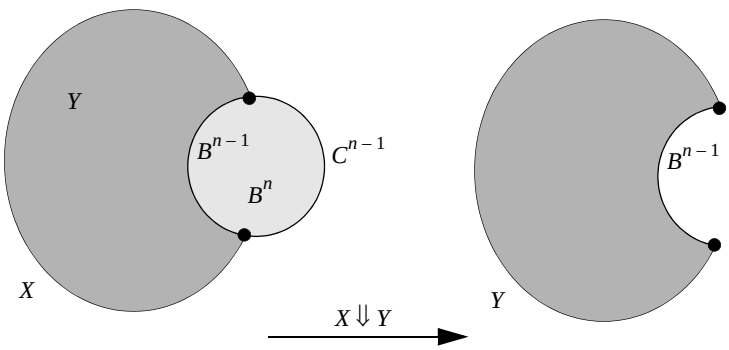


Figure 3.3 Collapse of X to Y from C^{n-1} onto B^{n-1} across B^n.

We say X *collapses* on Y and write $X \Downarrow Y$ if there is a sequence of elementary collapses $X = X_0 \downarrow X_1 \downarrow \ldots \downarrow X_n = Y$. If Y is a point, X is *collapsible* and we write $X \Downarrow 0$. Note that collapses are invertible operations.

A special case of collapsing is shelling. Suppose that $M_1 \subset M$ are n-manifolds and $M \downarrow M_1$ across B^n from C^{n-1} onto B^{n-1}. If $B^{n-1} \subset \partial M_1$ and $C^{n-1} \subset \partial M$ this collapse is called *elementary shelling* and a sequence of such operations is a *shelling.*

An example of a shelling of a triangulated surface mesh equivalent to Figure 3.3 is given in Figure 3.4.

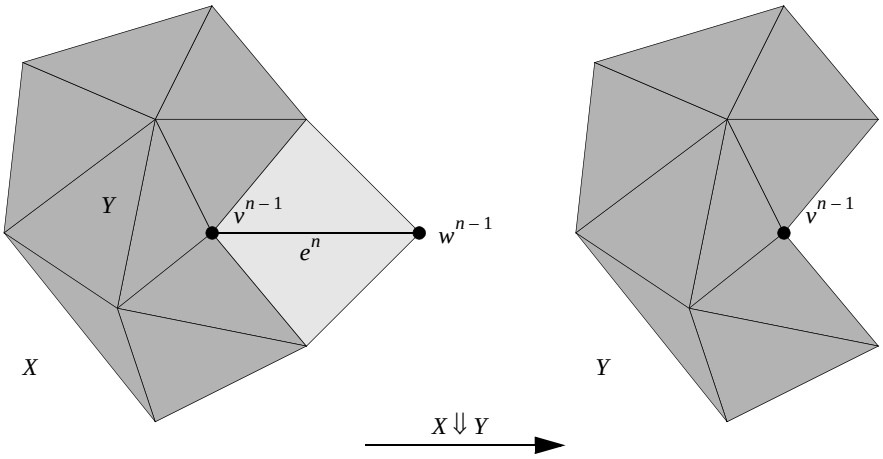


Figure 3.4 Collapse of triangulation X to Y from vertex w^{n-1} onto vertex v^{n-1} across edge $e^n = \{v^{n-1}, w^{n-1}\}$.

3.2 GEOMETRIC SURFACE AND VOLUME REPRESENTATIONS

After introducing some important concepts of algebraic topology, we give a brief overview of piecewise-linear surface and volume representations, which are widely used in practical applications.

3.2.1 Surface and Volume Meshes

Typically, mesh types differ in the amount of information needed to specify vertex positions and connectivity:

- *Structured:* For structured meshes, which include uniform meshes, geometry and micro-topology is given implicitly. Hence, it can be derived from the structure of the mesh.
- *Unstructured:* In the case of structured meshes, geometry and microtopology have to be specified explicitly.

See Table 3.1 for a summary of the different mesh types.

Uniform meshes. A uniform mesh comprises vertices with uniform spacing along a given axis. The geometry is specified by the positions of the minimum and maximum vertices and the number of vertices along each axis. See Figure 3.5 for an illustration of a uniform mesh.

Table 3.1 Information which needs to be specified for different mesh types.

Type	Geometry	Micro-topology
Uniform	min. and max vertices only	implicit
Structured	all vertices	implicit
Unstructured	all vertices	explicit

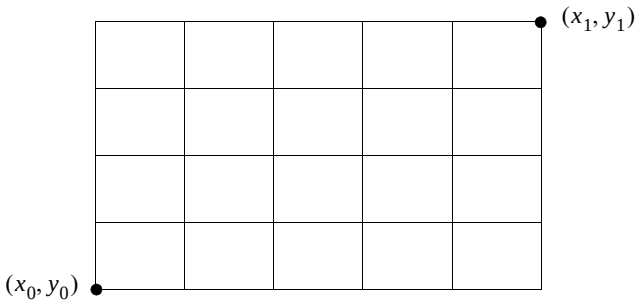


Figure 3.5 A uniform mesh defined by minimum vertex (x_0, y_0, z_0), maximum vertex (x_1, y_1, z_1), x-extension $\text{ext}(x) = 6$, and y-extension $\text{ext}(y) = 5$.

Given this information, the computation of the coordinates of the remaining vertices as well as the connectivity of the quadrilateral cells is trivial.

Uniform meshes are widely used in practical applications where data is sampled on a uniform grid. Important examples are digital terrain models and medical data including *computer tomography* (CT) and *magnetic resonance imaging* (MRI). In these cases additional data attributes are associated with each vertex (see Section 3.2.2).

Structured meshes. Structured meshes specify the geometry of the mesh explicitly. Due to the higher degree of freedom of the vertex placement, structured meshes allow for greater flexibility of the mesh in comparison to uniform meshes. Figure 3.6 depicts the *NASA bluntfin* data set, a structured mesh, which is widely used as a test data set for *computational fluid dynamics* (CFD) applications. Note, however, that connectivity is still implicitly contained in the mesh. The number of vertices along each axis is fixed as it is in the case of uniform meshes. Since vertex positions on the boundary are fully specified, however, regular spacing of vertices is no longer required.

Structured meshes are common in application areas such as CFD and *finite element analysis* (FEA).

The use of structured meshes simplifies the compression of the mesh, since only geometric information has to be taken into account. See Section 4.5 for further details on mesh compression.

Unstructured meshes. The most general representation is the unstructured mesh. Before the availability of mesh compression schemes, the use of unstructured meshes has usually been limited to small or medium-sized data sets. The additional specification of microtopology imposes a significantly increased demand on application memory. Nevertheless, many real-word problems are not defined on regular grids. Thus, the demand for efficient representations of unstructured meshes is steadily increasing.

The data structures which are typically used to represent unstructured meshes, comprise a list of vertex positions and a list of n-tuples of indices into this vertex list. The n-

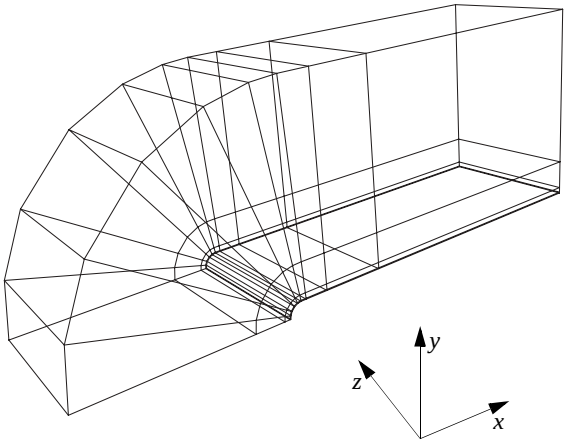


Figure 3.6 The NASA bluntfin data set: An example of a structured mesh with $\text{ext}(x) = 10$, $\text{ext}(y) = 5$, and $\text{ext}(z) = 2$.

tuples constitute n-cells, which are the basic building blocks of unstructured meshes. In other words, unstructured meshes are used in practice to represent any kind of cell-complexes and, in the case of purely simplicial cells, simplicial complexes (see Section 3.1.1).

Typical cell types are depicted in Figure 3.7.

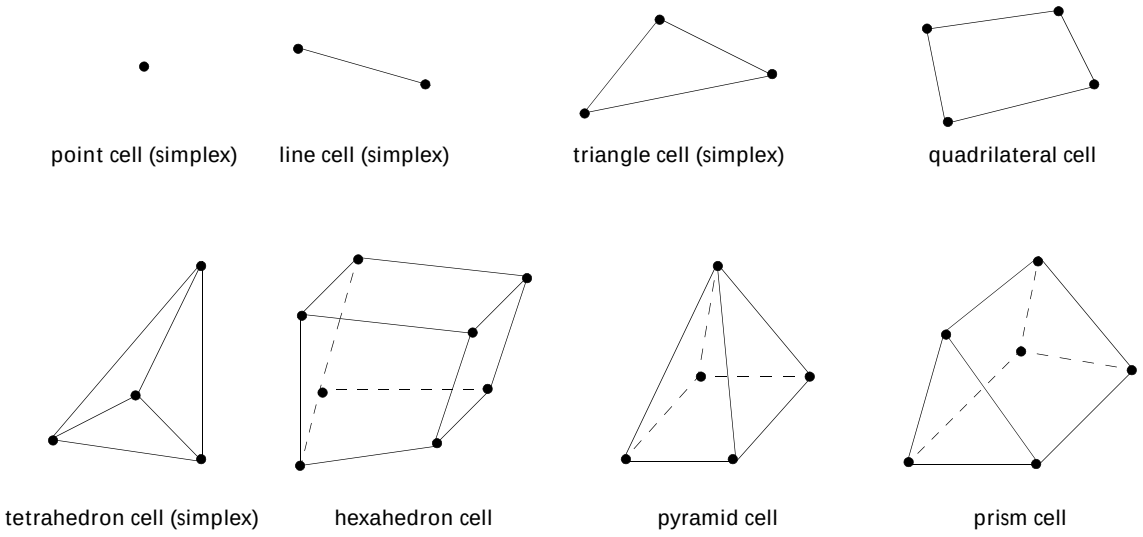


Figure 3.7 Common cell types used with unstructured meshes.

It might be advantageous for specific applications to employ non-simplicial cells [61]. Whenever possible, however, simplices are preferred as basic building blocks of the mesh. They provide for the greatest possible flexibility. Other cells of the same dimension can always be decomposed into simplices, usually with additional computational and memory overhead, though.

Table 3.2 is an example of a typical unstructured mesh representation comprising a vertex list and a list of tetrahedron cells.

Assuming 32-bit floating-point coordinate values for vertices and 32-bit integer values for cells, the typical memory consumption of an unstructured mesh is as follows [102]:

Table 3.2 Tetrahedral mesh representation.

Geometry	Microtopology
vertex 1 (x_1, y_1, z_1)	tetrahedron 1 (1, 2, 3, 4)
vertex 2 (x_2, y_2, z_2)	tetrahedron 2 (3, 2, 4, 6)
vertex 3 (x_3, y_3, z_3)	tetrahedron 3 (4, 2, 5, 8)
vertex 4 (x_4, y_4, z_4)	tetrahedron 4 (6, 5, 8, 4)
vertex 5 (x_5, y_5, z_5)	...
vertex 6 (x_6, y_6, z_6)	
vertex 7 (x_7, y_7, z_7)	
vertex 8 (x_8, y_8, z_8)	
...	

- *Vertices:* 3×32 bits/vertex
- *Triangle per vertex incidence:* $3 \times \log_2(\bar{v})$ bits/triangle
- *Tetrahedron per vertex incidence:* $4 \times \log_2(\bar{v})$ bits/tetrahedron

where $\bar{v}$ denotes the total number of vertices in the mesh.

Mesh characteristics. So far, we did not differentiate the dimensionality of uniform, structured, or unstructured meshes. In general, we can distinguish between the computational and the physical dimensions of a mesh. The computational dimension is the dimension of the parameter domain of the object. The physical dimension denotes the dimensionality of the vertex coordinates. Consider a surface mesh on the plane as an example of a 2-D mesh in two-space. Figure 3.8 depicts some further examples of n-dimensional meshes in m-space.

Note that a 3-D object, by definition, must exist in three-space (or higher), whereas a 2-D mesh can exist either in two-space or in three-space.

3.2.2 Mesh Attributes

In addition to geometry and connectivity, a variety of different attributes can be associated with meshes. Attributes can either be scalar- or vector-valued and they are associated with vertices, cells or even whole meshes. Table 3.3 lists possible mesh attributes commonly used in a range of application areas.

3.3 MESH GENERATION AND SIMPLIFICATION

If only the positions of a set of points in n-space is known, mesh generation methods have to be employed to determine the connectivity. In the general case of arbitrarily positioned points in two-space, this problem is not trivial. If it is known that the points actually define a surface, it is possible to use surface fitting algorithms to find an optimal surface representation [54]. Otherwise, methods from computational geometry, such as Delaunay triangulation, can be applied to generate the connectivity of points in two-space or three-space.

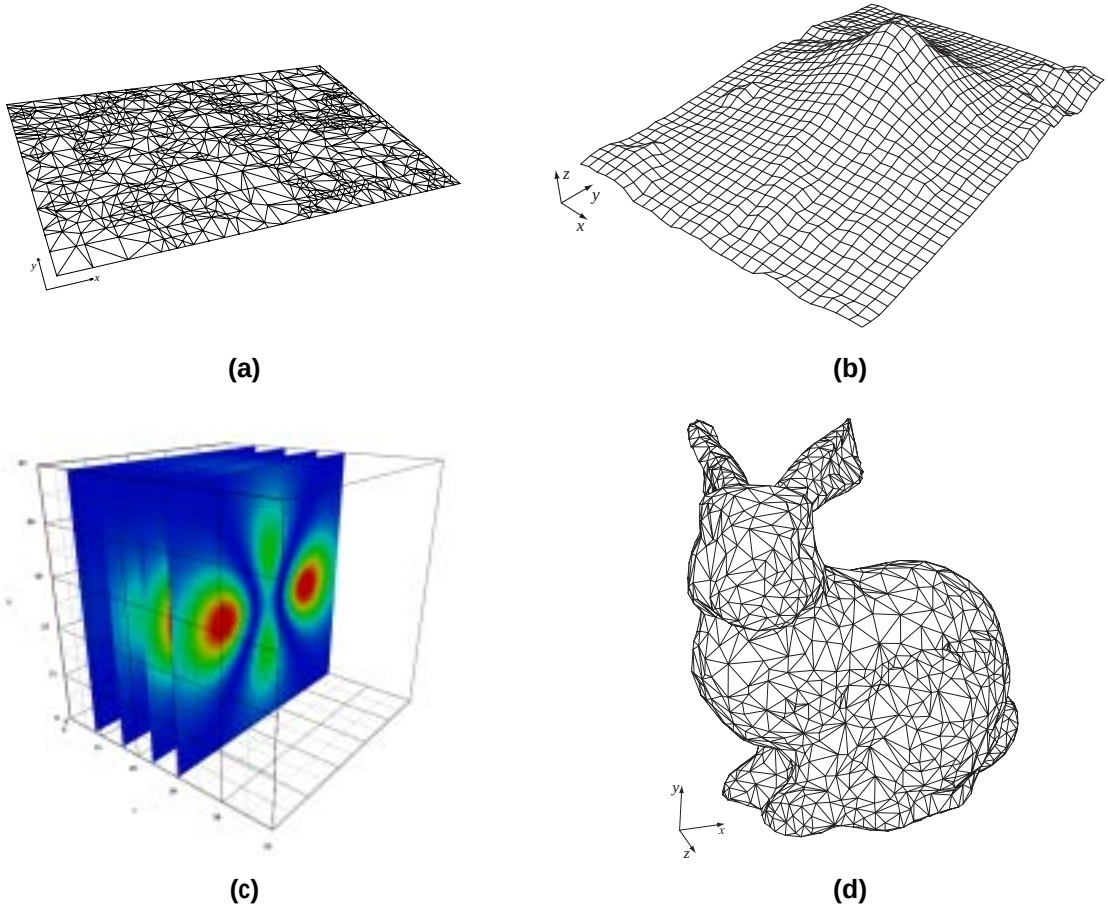


Figure 3.8 Examples of meshes of different dimensionality: **a:** Plane (2-D, 2-space). **b:** Digital terrain model (2-D, 3-space). **c:** CT volume data set (3-D, 3-space). **d:** Geometric model (2-D, 3-space). [AVS©], [Stanford©]

Table 3.3 Typical mesh attributes in different application fields.

Mesh type	Attributes
Digital terrain models	height values, GIS data, etc.
CAD models	color, normals, gradients, texture coordinates, curvature, etc.
CT/MRI data	intensity, segmentation classes
CFD/FEA models	velocity, vorticity, momentum, pressure, stagnation, pressure, elasticity, etc.
Radiosity meshes	radiosity, etc.

3.3.1 Delaunay Triangulation

Let S be a set of n points in the plane. No two points in S are connected with a straight line and no four points are cocircular. The perpendicular bisector b_{ij} of $\mathbf{p}_i, \mathbf{p}_j \in S$ is defined as

$$b_{ij} := \{\mathbf{x} \in \subsetneq^2 \,|\, d(\mathbf{x}, \mathbf{p}_i) = d(\mathbf{x}, \mathbf{p}_j)\} \tag{3.1}$$

where $d(\mathbf{x}, \mathbf{y})$ denotes the Euclidian distance between two points. Let further be

$$H_{ij} := \{\mathbf{x} \in \subsetneq^2 \,\big|\, d(\mathbf{x}, \mathbf{p}_i) \le d(\mathbf{x}, \mathbf{p}_j)\} \tag{3.2}$$

the *half-plane* containing $\mathbf{p}_i$ that is defined by b_{ij}. The intersection of $n-1$ half-planes constitutes the Voronoi polygon V_i [111]:

$$V_i = \bigcap_{i \neq j} H_{ij}. \tag{3.3}$$

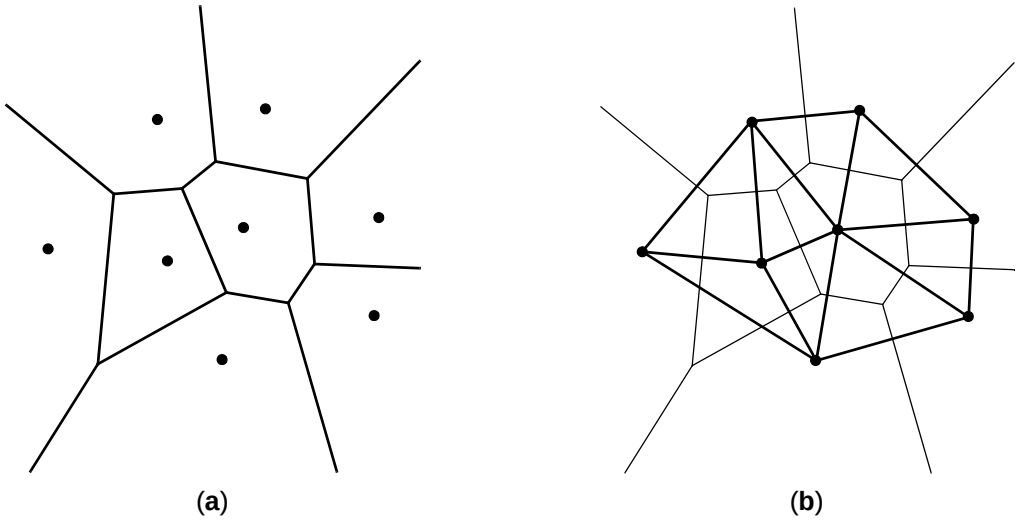
V_i defines the locus of points closer to $\mathbf{p}_i$ than to any other point in S. It is a convex region with no more than $n-1$ sides. The Voronoi diagram is the unity of all Voronoi polygons associated with points in S:

$$\mathrm{Vor}(S) = \bigcup_{i=1}^{n} \partial V_i. \tag{3.4}$$

The Delaunay triangulation $D(S)$ is the straight-line dual of $\mathrm{Vor}(S)$, with the following properties [27]:

1. Vertices of $D(S)$ are the points in S.
2. Two points $\mathbf{p}_i$ and $\mathbf{p}_j$ are connected with a straight line if and only if the Voronoi polygons V_i and V_j share a common edge.
3. Each edge in $D(S)$ is associated with a circle which touches the endpoints of the edge and which does not contain any other points in S.
4. A Delaunay triangulation $D(S)$ is planar.
5. The circumcircle of each triangle in $D(S)$ does not contain any other points in S.

The duality of Voronoi diagrams and Delaunay triangulations is illustrated in Figure 3.9.



(**a**) (**b**)

Figure 3.9 Duality: **a:** Voronoi diagram. **b:** Delaunay triangulation.

Delaunay triangulations define optimal meshes for a given set of points with respect to the following criterion: A planar mesh meets the Delaunay criterion if the minimum angle

of all triangles in $D(S)$ is globally maximized. A Delaunay triangulation of a set of n points in the plane can be constructed in $O(n\log n)$ time, which is optimal [81].

Brown [9] has shown a direct relation between the d-dimensional Delaunay triangulation and the $d+1$–dimensional convex hull of a set of points. This provides for an elegant way of constructing Delaunay triangulations of degree $d \geq 2$. The relation is illustrated for the two-dimensional case, but can be extended into higher dimensions in a straightforward manner: Let S be a set of n points in the plane. This plane is embedded into the hyperplane $H = \{(x, y, z) | z = 1\}$ in E^3, the three-dimensional Euclidian space. The analogous process is illustrated in Figure 3.10 for one less dimension. H is now projected onto the sphere C with radius $1/2$, centered at $(0, 0, 1/2)$, using a stereographic projection. The edges of the convex hull of the transformed points S' on C are, after back transformation, exactly the edges of the Delaunay triangulation.

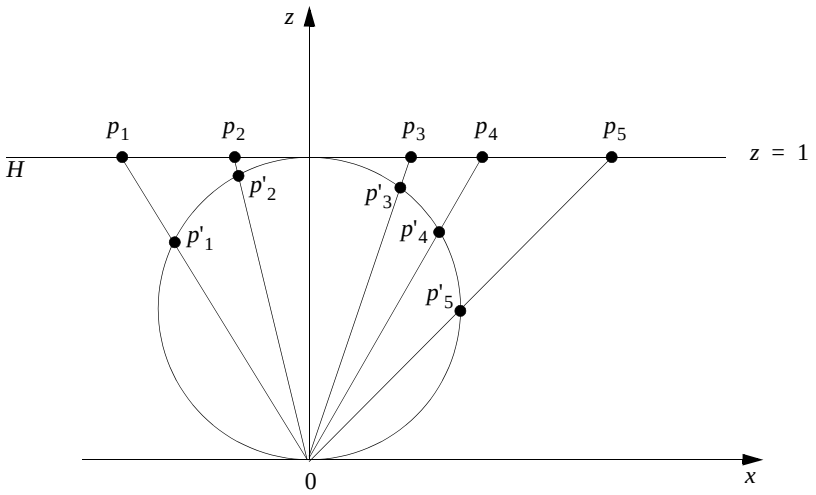


Figure 3.10 Projection of a point set S on the plane $z = 1$ onto the sphere C [81].

Hence, fast convex hull methods can be employed to find Delaunay triangulations of higher dimensions. The *quick hull* algorithm [4] is used to determine Delaunay triangulations in two and three dimensions in Section 5.4.

3.3.2 Mesh Simplification

Due to the increasing complexity of geometric meshes, mesh simplification methods are devised to reduce the number of mesh primitives.

Let us start with a brief look at curve simplification, a technique that has long been used in fields such as cartography, computer graphics, and computer vision. The basic idea is to take a polygonized curve with n vertices as input and to generate a curve with $n' < n$ vertices as output. The Douglas-Peuker algorithm [29] is probably the most widely used high-quality curve simplification algorithm. This recursive algorithm attempts to approximate a sequence of points with a straight-line segment from the first to the last point. The point with maximum distance to this approximation is found. The approximation is accepted if the distance between the new point an the line segment is below a given threshold. Otherwise, the algorithm is applied to the two sub-segments before and after the chosen point. Even though this algorithm is not optimal, it has been found that it produces quality approximations as compared to many other heuristic methods [72, 114]. An extension of the Douglas-Peuker algorithm and its generalization to surface and volume

mesh simplification will be introduced in Section 5.4. A thorough evaluation of the Douglas-Peucker scheme has recently been given in [3].

Surface and volume simplification methods are usually more sophisticated than curve simplification methods. This is due to the large variety of different surface types as described in Section 3.2.1. Nevertheless, many surface simplification algorithms have been introduced over the past decade, most of which are specialized to a either height fields, manifold surfaces, or non-manifold surfaces. For a detailed overview of simplification methods, the reader is referred to Heckbert and Garland [49], who give a concise survey of many methods found in literature.

3.4 ERROR MEASURES

Whenever dealing with mesh approximations, as described in Section 3.3.2 and in Section 4.5.2, means to quantify the approximation error are required. Very common error metrics are based on the L^2 and the L^∞ norm.

The L^2-error between two d-dimensional vectors $\mathbf{u}$ and $\mathbf{v}$ is defined as

$$\|\mathbf{u}-\mathbf{v}\|_2 = \sqrt{\sum_{i=1}^{d}(u_i-v_i)^2}, \tag{3.5}$$

and the L^∞-error, which is also known as *maximum error*, is defined as

$$\|\mathbf{u}-\mathbf{v}\|_\infty = \max_{i=0}^{d}|u_i-v_i|. \tag{3.6}$$

The *squared error* is defined as the square of the L^2-error and the *root mean square* (RMS) error is defined as

$$\mathrm{RMS}(\mathbf{u},\mathbf{v}) = \frac{1}{\sqrt{d}}\sqrt{\sum_{i=1}^{d}(u_i-v_i)^2}. \tag{3.7}$$

Optimization with respect to the L^2 and L^∞ metrics are called *least squares* and *min-max* optimization, respectively.

3.4.1 Surface Mesh Errors

These metrics are usually not sufficient to quantify the difference between a surface mesh and its approximation. The approximation error between two surfaces may be defined as the distance between corresponding regions of the mesh. Let us first define the distance $e(\mathbf{p}, S)$ between a point $\mathbf{p}$ and a surface S as

$$e(\mathbf{p},S) = \min_{\mathbf{p}'\in S} d(\mathbf{p},\mathbf{p}'). \tag{3.8}$$

The one-sided distance $E(S_1, S_2)$ between two surfaces S_1 and S_2 is then defined as

$$E(S_1,S_2) = \max_{\mathbf{p}\in S_1} e(\mathbf{p},S_2). \tag{3.9}$$

The definition of Equation 3.9 is not symmetric as there exist surfaces where $E(S_1, S_2) \neq E(S_2, S_1)$. We yield symmetry by taking the maximum of $E(S_1, S_2)$ and $E(S_2, S_1)$. This two-sided distance is known as *Hausdorff distance*. In other words, the single-sided Hausdorff distance is the maximum distance of S_1 to the nearest point in S_2.

Cignoni and others [17] define additional metrics for measuring surface approximation errors based on uniformly sampled distances. The *volume* between two closed surfaces S_1 and S_2 is defined as the surface integral of the distance function over S_1:

$$E_v(S_1, S_2) = \int_{S_1} e(\mathbf{p}, S_2) ds \tag{3.10}$$

and the *mean distance* between two surfaces is defined as the surface integral of the distance divided by the area of S_1:

$$E_m(S_1, S_2) = \frac{1}{|S_1|} \int_{S_1} e(\mathbf{p}, S_2) ds\,. \tag{3.11}$$

Orientable surfaces allow for an additional extension of the error metrics to signed mesh distances: Let $\mathbf{N_p}$ be the surface normal at $\mathbf{p} \in S_1$ and $\mathbf{p}' \in S_2$ the point which lies closest to $\mathbf{p}$, then the sign of the *signed distance* e' is the sign of the inner product $\langle \mathbf{N_p}, (\mathbf{p}' - \mathbf{p}) \rangle$. With the notion of signed distances it is possible to differentiate between positive and negative distances of two surfaces as

$$E^+(S_1, S_2) = \max_{\mathbf{p} \in S_1} e'(\mathbf{p}, S_2) \tag{3.12}$$

and

$$E^-(S_1, S_2) = \left| \min_{\mathbf{p} \in S_1} e'(\mathbf{p}, S_2) \right|, \tag{3.13}$$

respectively.

The error metrics defined in this section will be used to evaluate the quality of the surface approximations in Chapter 7. Further evaluation criteria introduced in [17] include surface area and totals length of the feature edges.

3.4.2 Volume Mesh Errors

In the case of volume meshes, a distance error is usually not applicable. In contrast to surface meshes, volume mesh approximations are frequently not optimized for maintaining geometric shape. Shape is only important at the boundary of the object, which should approximate the original as close as possible. The quality of the interior of a simplified volume mesh is largely determined by the approximation of mesh attributes associated with single vertices or cells (see Section 3.2.2).

One possible error measure is to sample both volume meshes using, for example, Monte Carlo sampling methods. The RMS error between the approximated data attributes at the sample positions and the corresponding signal-to-noise ratio deliver quantitative results for the quality of the approximation.

4

COMPRESSION

Data compression methods are used to represent information in a compact form. Knowledge about the internal structure of data is exploited to analyze and reduce redundancies and thereby to achieve a more memory efficient representation. This chapter summarizes some important properties of data compression schemes in general and highlights two important fields in the context of this thesis: wavelet-based subband coding and geometry compression. A detailed introduction to data compression can be found in [88].

4.1 PRINCIPLES OF COMPRESSION

Methods for efficient data compression have been developed for various application fields including text compression, audio and video compression, and more recently geometry compression. Since the beginning of electronic communication, the limited bandwidth of communication channels urged a need for simple compression schemes to speed up transmission time. An early example is the Morse code, where single letters are encoded with sequences of dots and dashes. Since certain letters, such as *e* (·) and *a* (· –), occur more frequently than, for example, *j* (· – – –) and *q* (– – · –), they are assigned shorter sequences. The basic idea of using shorter codes for frequent symbols is the basis for many compression methods, including Huffman coding, which will be described in Section 4.3.

Generally, a compression technique comprises two algorithms. The *compression* algorithm takes a sequence $\mathbf{X}$ and generates a compressed representation $\overline{\mathbf{X}}$. The *decompression* algorithm takes $\overline{\mathbf{X}}$ and reconstructs a sequence $\mathbf{Y}$. Usually, the two algorithms together are referred to as compression algorithm or *codec*.

Depending on application requirements, compression algorithms can be divided into two different classes: *lossless* compression schemes, where $\mathbf{X}$ is equal to $\mathbf{Y}$. and *lossy* compression schemes, where $\mathbf{X}$ and $\mathbf{Y}$ may differ.

For in-depth treatment of the mathematical preliminaries of compression, the reader is referred to [88]

4.2 PERFORMANCE MEASURES

When dealing with compression algorithms, measures for evaluating and comparing the performance of different methods have to be defined. Common measures, such as algorithmic complexity, memory consumption, and execution speed can be used as well, but in the context of compression we need some additional criteria:

- *Compression ratio:* A simple way of measuring the performance is to look at the ratio of the number of bits used to represent the original data and the number of bits used to represent the compressed data. This ratio is referred to as *compression ratio.* Consider a volumetric data set made up of a cubic array of $64 \times 64 \times 64$ voxels. The amount of memory which is required to store this data is 262,144 bytes. Suppose a compressed version of the data requires only 32,768 bytes. In this case the compression ratio is 8:1.
- *Rate:* An alternative measure is to provide an average number of bits required to represent a singe sample. This measure is known as bitrate or short *rate.* In the above example, assuming one byte or eight bits per voxel, the average number of bits in the compressed representation is one. Hence, the rate is one bit per voxel.
- *Distortion:* In lossy compression settings (see Section 4.4) where the reconstruction differs from the original data, we need a measure to quantify this difference or the *distortion.* In this case, one can distinct between a pure mathematical difference and a perceptual difference. The latter is important for applications where the human perceptual system is involved (e.g., image or audio compression).

The *peak signal to noise ratio* (PSNR) is the most commonly used distortion metric in image and video compression literature. In general, the higher the PSNR, the higher the quality of the compressed data. If two data sets are identical, the PSNR would be infinite. Although the PSNR is a very popular measure, it has only limited, approximate relationship with perceived errors noticed by the human visual system. Before we can define the PSNR, we need the notion of the mean square error.

The *mean square error* (MSE) is defined as the sum of the squares of the differences between the values of data elements in the reference data set $\mathbf{X}$ and compressed data set $\mathbf{Y}$:

$$\mathrm{MSE}(\mathbf{X}, \mathbf{Y}) = \frac{1}{n} \sum_{i=1}^{n} \left| X_i - Y_i \right|^2 . \tag{4.1}$$

The *root mean squared error,* $\mathrm{RMSE}(\mathbf{X}, \mathbf{Y}) = \sqrt{\mathrm{MSE}(\mathbf{X}, \mathbf{Y})}$, is essentially the average change of a data value caused by the compression and decompression process.

Given a definition for the RMSE, the PSNR is defined as

$$\mathrm{PSNR}(\mathbf{X}, \mathbf{Y}) = 20 \log\left(\frac{p}{\mathrm{RMSE}}\right) = 10 \log\left(\frac{p^2}{\mathrm{MSE}}\right), \tag{4.2}$$

where p is the peak for data value (e.g., 255 for eight bit pixels). PSNR is usually quoted in decibels (dB), a logarithmic unit. It is technically a fidelity metric. It measures how close a data set resembles an original uncorrupted data set, often know as the reference data set. It is assumed that perceived quality is related to fidelity.

Equation 4.1 defines the MSE for scalar data sets, only. For multi-channel data, such as color images and textures, the definition has to be modified to take all channels into

account [117]. The MSE on RGB color images is usually computed as point-by-point vector length of the RGB difference image between a reference image and its reproduction:

$$\mathrm{MSE}_{\mathrm{RGB}}(\mathbf{X}_{\mathrm{RGB}}, \mathbf{Y}_{\mathrm{RGB}}) = \frac{1}{n}\sum_{i=1}^{n}\left(\left|X_{i,\mathrm{R}} - Y_{i,\mathrm{R}}\right|^2 + \left|X_{i,\mathrm{G}} - Y_{i,\mathrm{G}}\right|^2 + \left|X_{i,\mathrm{B}} - Y_{i,\mathrm{B}}\right|^2\right). \tag{4.3}$$

This definition of the mean squared error for RGB images does not include any information about the device used to present the images. Hence, the PSNR value computed using $\mathrm{MSE}_{\mathrm{RGB}}$ is uncalibrated. Using uncalibrated image values to measure perceptual differences is often considered bad practice. Since, however, the definition of powerful color image fidelity metrics is still an area of active research [117], the PSNR based on $\mathrm{MSE}_{\mathrm{RGB}}$ will be used as an error metric in this thesis where appropriate. For the time being, it is still one of the most common metrics used in literature.

4.3 LOSSLESS COMPRESSION

As indicated by their name, lossless compression algorithms involve no loss of information. This property is of vital importance for applications where no reconstruction error can be tolerated. Consider, for example, the compression of a binary executable program. The change of a single bit might already make the program unusable. Further examples are compression of text and certain kinds of image and video compression. In general, lossless techniques are used to compress “discrete” data.

In the remainder of this section, two important and widely-used classes of compression algorithms are described briefly.

4.3.1 Huffman Coding

Huffman coding [56] is one of the most popular lossless compression schemes. A basic law of information theory states that the lower bound of compression for any sequence of symbols $\mathbf{X}$ is equal to its entropy $H(\mathbf{X})$:

$$H(\mathbf{X}) = -\sum_{i} P(X_i)\log P(X_i). \tag{4.4}$$

$P(X_i)$ denotes the probability of occurrence of symbol $X_i \in \mathbf{X}$. If the rate of a code is equal to the entropy, this code is the best possible (lossless) code for a given source. Huffman codes are guarantied to perform within one bit of the entropy. Hence, they are optimal for a given set of probabilities. Huffman codes are based on the observation that symbols that occur more frequently (i.e., symbols whose probability of occurrence is higher) will have shorter code words than symbols that occur less frequently.

The following example will demonstrate the design of a Huffman code. Given a set of symbols $\mathbf{X} = \{X_1, X_2, X_3, X_4\}$ with $P(X_1) = 0.4$, $P(X_2) = 0.2$, $P(X_3) = 0.1$, and $P(X_4) = 0.3$. The entropy $H(\mathbf{X})$ is 1.8464 bits/symbol. The list of symbols sorted in descendent order of probability is shown in Table 4.1.

The two symbols with the lowest probability, X_2 and X_3, are assigned the two code words $c(X_2) = \alpha_1 * 0$ and $c(X_3) = \alpha_1 * 1$, respectively. α_1 is a binary string and * denotes string concatenation. In the second step, a new set of symbols

Table 4.1 Initial set of symbols.

Symbol	Probability	Code word
X_1	0.4	$c(X_1)$
X_4	0.3	$c(X_4)$
X_2	0.2	$c(X_2)$
X_3	0.1	$c(X_3)$

$\mathbf{X}' = \{X_1, X'_2, X_4\}$ is used, where X'_2 is composed of X_2 and X_3 with probability $P(X'_2) = P(X_2) + P(X_3) = 0.3$. The new list of symbols is given in Table 4.2.

Table 4.2 Set of symbols after step one.

Symbol	Probability	Code word
X_1	0.4	$c(X_1)$
X_4	0.3	$c(X_4)$
X'_2	0.3	α_1

The resulting code words for X_4 and X'_2 are $c(X_4) = \alpha_2 * 0$ and $c(X'_2) = \alpha_2 * 1$, respectively. Hence, $\alpha_1 = \alpha_2 * 1$, which implies that $c(X_2) = \alpha_2 * 10$ and $c(X_3) = \alpha_2 * 11$. In the last step, X_4 and X'_2 are combined to X'_4 with probability $P(X'_4) = 0.6$. With only two symbols X_1 and X'_4 left, we assign $c(X_1) = 1$ and $c(X'_4) = 0$, which in turn means $\alpha_2 = 0$. Accordingly, the binary values of α_i can be substituted top-down and the resulting code words are listed in Table 4.3.

Table 4.3 Resulting code words for the initial set of symbols.

Symbol	Probability	Code word
X_1	0.4	1
X_4	0.3	00
X_2	0.2	010
X_3	0.1	011

Note that none of the code words in Table 4.3 is the prefix of any other code word. A code with this property is called *prefix code* and it can be decoded unambiguously. Thus, decompression of prefix codes is straightforward.

Huffman codes are only optimal in the case where the symbols are generated for each individual input. Therefore, the symbol table has to be stored with the compressed data. For a small number of symbols, this does usually not influence the performance of the compression scheme significantly. If, however, this additional overhead cannot be tolerated, and it is further possible to collect relevant heuristic information about the probability of symbols in the source data, the symbol table can be precomputed and stored separately. This approach is, for example, taken in the JPEG image compression scheme [112].

4.3.2 Arithmetic Coding

According to Equation 4.4, an optimal entropy codec requires $L_i = -\log P(X_i)$ bits to encode symbol X_i. As opposed to Huffman coding, which rounds L_i to the nearest integer, arithmetic coding [115] does manage to encode symbols using a noninteger number of bits.

An input string of arbitrary length is represented as real number R in the interval $0 \le R < 1$. The precision of R adapts to the length of the string. Let us consider the example from the previous section with symbols $\{X_1, X_2, X_3, X_4\}$. Figure 4.1 shows a sequence of three symbols $X_1X_2X_3$ being encoded. The initial interval $[0, 1)$ is divided into four segments whose length is proportional to the probabilities of the symbols in the alphabet. The first symbol, X_1, limits R to the interval $[0.6, 1.0)$, which is in turn subdivided into four segments with lengths proportional to the $P(X_i)$'s. The second symbol, X_2, narrows the interval to $[0.76, 0.84)$, and the final symbol further narrows the interval to $[0.784, 0.792)$. Note that *any* value of R in this interval now represents the sequence $X_1X_2X_3$. In particular, the binary fraction 0.1100101 represents the value 0.7890625 with seven bits, which is the lower bound in this setting.

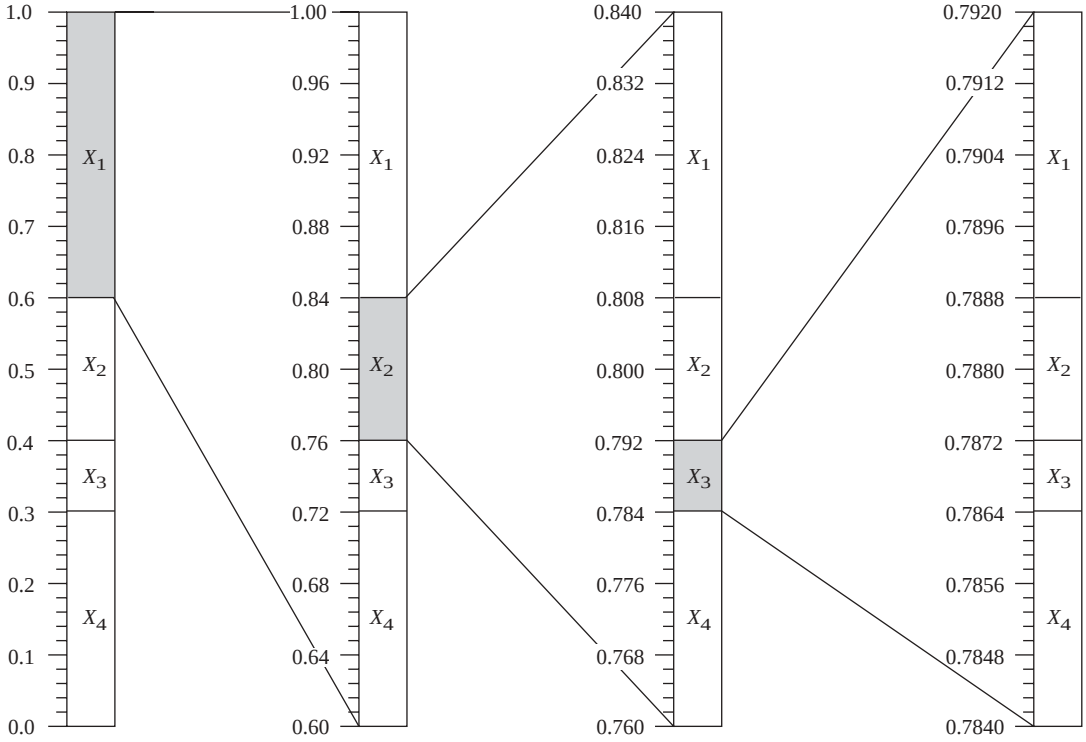


Figure 4.1 Arithmetic coding of the sequence $X_1X_2X_3$: Successive symbols further subdivide the initial interval $[0, 1)$. The lengths of the subintervals allocated for each symbol are proportional to its probability. The final value can be chosen arbitrarily in the interval $[0.784, 0.792)$.

Obviously, Huffman coding only requires six bits to encode the same sequence. Hence, arithmetic coding does not always outperform Huffman coding. Furthermore, arithmetic coding does not generate prefix codes. In other words, the above representation encodes any sequence of symbols $X_1X_2X_3\ldots$, where the ellipsis represents an infinite number of

subsequent symbols. We need to specify an additional symbol which marks the end of the sequence.

4.3.3 Run-Length Coding

Run-length coding is one of the simplest lossless compression schemes. It is based on the Capon model [10] used for compression of black and white images. The Capon model is a two-state Markov model with states S_w and S_b, which correspond to the respective cases where a black and a white pixel had just been drawn. If the state probabilities $P(w|w)$ and $P(b|b)$ are significantly higher than the transition probabilities $P(b|w)$ and $P(w|b)$, which is usually the case for binary images, than it is more efficient to encode the length of the run of a single color instead of the color of each individual pixel.

Run-length coding is sometimes used in conjunction with Huffman coding. In the case of codes with long runs of Huffman symbols this provides for an easy, yet efficient improvement of the overall performance. The compression scheme which is introduced in Section 5.2 is an example of such a combination.

4.4 LOSSY COMPRESSION

The schemes introduced in the previous section deliver good results for lossless compression tasks. If, however, some loss of information can be accepted, much better compression schemes can be designed. For many practical applications, such as image, video or speech compression, the lack of exact reconstruction of the original data is not a problem.

4.4.1 Quantization

Quantization is one of the most general ideas in lossy compression. One can distinguish between two classes of quantization: If source and output data are scalar, it is called scalar quantization, otherwise it is called vector quantization.

In general, a quantizer consists of two mappings: The encoder and the decoder. The encoder divides the range of values of the source data into a number of intervals, each of which is represented by a unique code word. The decoder generates a reconstruction value for every code word generated by the encoder. Note that, since the mapping of the encoder is irreversible, the reconstructed value differs from the source value. Hence, quantization is a lossy compression scheme. The mapping of a quantizer can be represented by input-output maps as depicted in Figure 4.2.

Scalar quantization. The simplest type of scalar quantizers is a uniform quantizer as depicted in Figure 4.2. Uniform quantizers feature a fixed step size Δ, both for encoding and decoding. One can further distinguish between two types of uniform quantizers. If the quantizer does not have zero as one of its representation levels it is called *midrise* quantizer, otherwise it is called *midtread* quantizer. Midtread quantizers are especially important in applications where zero values have to be represented accurately. Usually, the design of a uniform quantizer involves finding a step size Δ with minimum distortion for a given source and number of decision levels. See Section 5.2 for the description of the uniform quantizer used in this thesis.

If the input is not distributed uniformly, however, uniform quantization may lead to suboptimal results. Nonuniform quantization provides for a solution to this problem. Smaller intervals are used in regions where symbols occur with higher probability. In order

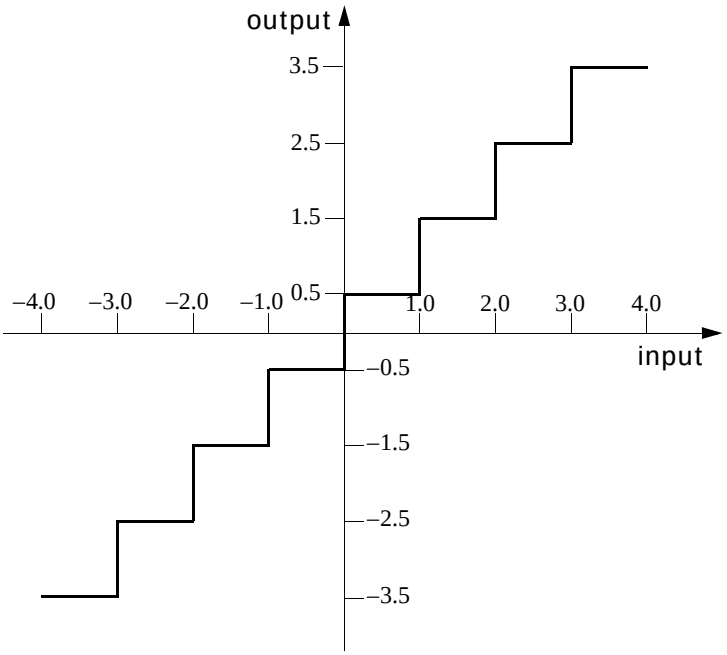


Figure 4.2 A simple midrise quantizer input-output map with uniform intervals. For example, values in the interval $[2.0, 3.0)$ are mapped to 2.5

to design an optimum nonuniform quantizer, we need a good probability model for the input distribution. An iterative algorithm for designing optimum nonuniform quantizers can be found in [67]

An important step in the compression pipeline is code word selection. An efficient way to create variable-length code words is to entropy code the quantizer outputs. For a large number of reconstruction levels, nonuniform quantizers typically outperform uniform quantizers. For optimum high-rate entropy-coded quantizers, however, uniform quantizers are superior [38].

Vector quantization. Scalar quantizers treat each input sample individually. If the samples are correlated, however, consecutive samples can be grouped into blocks or vectors to make use of inherent structures. Thus, a vector of samples forms the input to a vector quantizer. For both, the encoder and the decoder, there exists a set of vectors called *code book*. The code book contains the *code-vectors* which represent the vectors from the input samples. Each code-vector is assigned a binary index.

For encoding, the input vector is compared to each code-vector to find the closest match. The elements of this code-vector are the quantized values of the input. The binary index of the code-vector is stored or transmitted for subsequent decoding. The decoder can retrieve the quantized values from its own copy of the code book. Figure 4.3 illustrates this process.

Compared to scalar quantization, the encoding step requires a considerably higher amount of computation. This is because the closest reproduction vector has to be found in the code book (note, that this is not a simple table lookup!). Beforehand, the code book has to be constructed, which is the most important step in the design of a vector quantizer. One can either construct and store a code book for each source data set, or find a more general code book for a class of similar input data. The most common algorithm for code

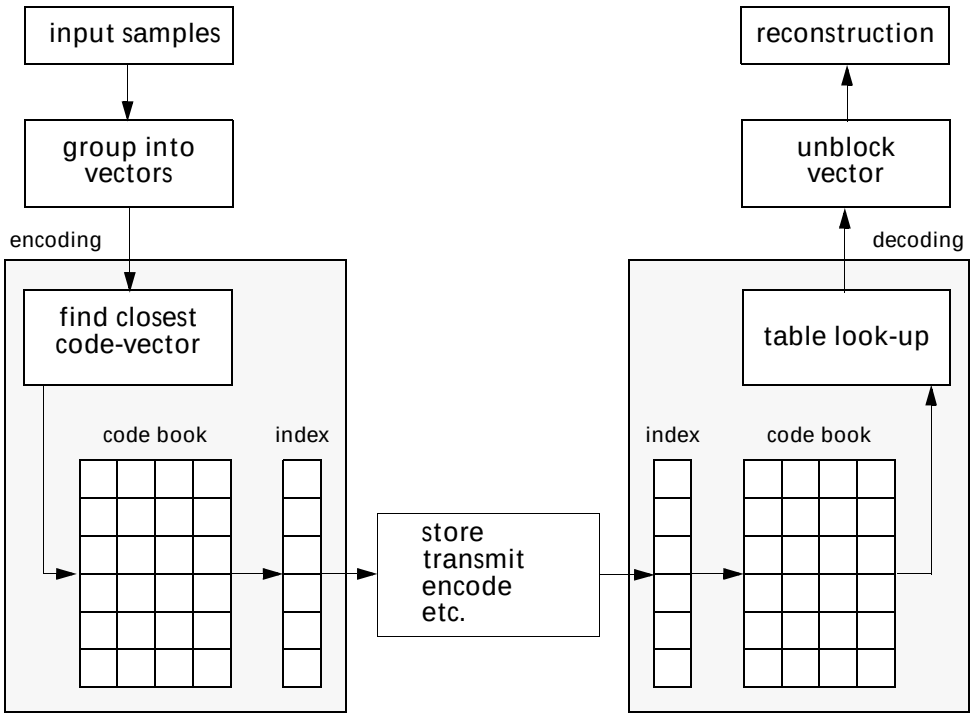


Figure 4.3 Illustration of the vector quantization process.

book design is the Linde-Buzo-Gray algorithm [63]. A general introduction to vector quantization can be found in [37].

4.4.2 Subband Coding

The compression schemes described so far operate directly on the input data. A different approach is to decompose the data into its constituents. Each constituent part is then encoded using one of the techniques that have been described previously. Generally, subband coding relies on separating the input into different bands of frequency using a pair of low-pass and high-pass filters. The filters are cascaded in stages as illustrated in Figure 4.4.

The most popular filters used in this approach are the *quadrature mirror filters* (QMF) proposed by Croisier and colleagues [23]. QMF filters share the property that the high-pass impulse response $\{g_n\}$ is directly related to the low-pass impulse response $\{h_n\}$:

$$\{g_n\} := \{(-1)^n h_{N-1-n}\}\,. \tag{4.5}$$

The general subband coding scheme is depicted in Figure 4.5. Before encoding, the input is passed through an *analysis filter bank* and is subsampled. For reconstruction, the decoded data is upsampled and passed through a *synthesis filter bank*, which inverts the analysis step. The subsampling is justified by a generalization of the Nyquist rule: Sampling data at critical level results in twice as many samples per second as the range of frequencies. Since the range of frequencies is decimated by half in each step of the analysis filter bank, we only need half of the samples in each step for an accurate representation of the data.

Wavelets, as introduced in Chapter 2, cannot only be seen from the mathematical point of view of functional analysis, but also from a signal processing point of view. In the

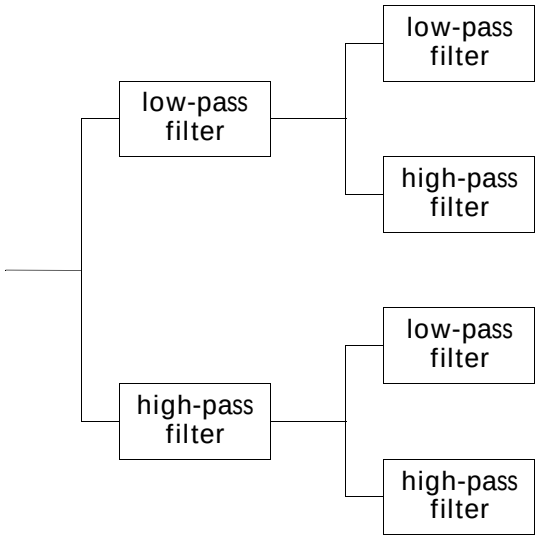


Figure 4.4 A four-band filter bank used for subband coding.

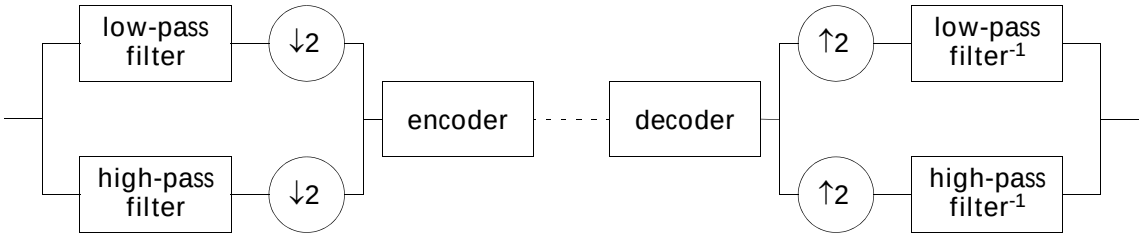


Figure 4.5 Subband coding scheme for a two-band filter bank.

latter case, they can be expressed by QMF filter banks. Hence, the scaling function operator acts as the low-pass filter $\mathbf{H} = \{h_n\}$ and the wavelets acts as the high-pass filter $\mathbf{G} = \{g_n\}$.

4.4.3 Transform Coding

Another technique of breaking down the input data into its components is transform coding. Instead of directly decomposing the data sequence $\{x_n\}$ into its frequency bands, we build small blocks of size N. Each of these blocks is then mapped into a transform sequence $\{\theta_n\}$ using a reversible mapping function. The mapping function ideally decorelates the input samples. In other words, the sample-to-sample correlation of $\{\theta_n\}$ is equal to zero. The signal energy within a transformed block of data is concentrated in a small number of samples. Similar to subband coding, the transformed samples can further be quantized and entropy encoded.

Widely used mapping functions include the discrete Karhunen-Loeve transform [2], which consists of the eigenvectors of the autocorrelation matrix of the input samples, and the discrete cosine transform [98], which is closely related to the discrete Fourier transform.

An example for the use of transform coding is the JPEG still image compression standard [112, 78].

4.5 COMPRESSION OF GEOMETRIC MODELS

As opposed to still image, video and audio compression, mesh compression, especially in combination with progressive transmission, is a relatively new area of active research. The basic idea is to convert the representation of a geometric model into a binary bitstream. Geometric models, as defined in Section 3.2, comprise geometry, the vertex positions in 3-space, and micro-topology, the connectivity between vertices. Most compression schemes are restricted to triangular meshes [26, 101]. Only recently, compression schemes for tetrahedral meshes have been proposed [95, 100, 77, 46].

In general, two separate problems have to be solved when designing compression schemes for geometric models. Firstly, the compression of the micro-topology, which usually involves a lossless codec. If, however, traditional mesh simplification techniques [90, 49] are used to reduce mesh complexity, we can also speak of lossy compression, because the original connectivity cannot be recovered. Secondly, the compression of the vertex positions, which can either be lossy or lossless.

4.5.1 Compression of Micro-topology

A well known method of compressing micro-topology is a representation based on triangle strips, as it is supported by the OpenGL graphics standard [74] and other graphics libraries. Here, the number of times a single vertex is transferred and processed by the graphics subsystem is reduced by combining a new vertex description with the description of the two previously processed vertices, which are temporarily buffered. Each new triangle, C, shares an edge with the previous triangle in the strip. By using a convention to orient the mesh, the "new" edges of C can be labeled as the *left* and the *right* edge. Only one bit per triangle is sufficient to indicate whether the triangle is incident upon the left or the right edge of the previous triangle. Since the first two vertices of each strip are an overhead, it is desirable to construct long strips. The efficient construction of optimal triangle strips is still a challenging problem [32].

In OpenGL [74], an alternating sequence between left and right edges throughout the strip is required instead of using a left/right bit. Thus, two consecutive right or left "moves" have to be implemented by encoding a vertex twice without breaking the strip. Figure 4.6 depicts an example of encoding the triangulated boundary of a cube using a single triangle strip.

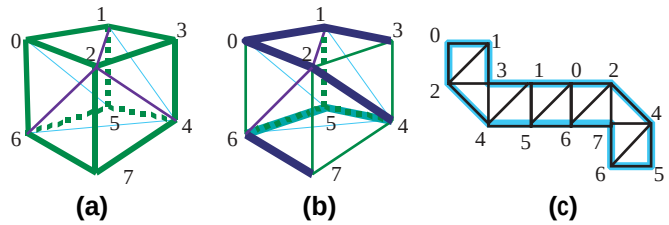


Figure 4.6 Triangle strips: **a:** Triangulated boundary of a cube. **b:** The boundary is cut along the thick edges. **c:** A flat triangulated polygon without interior edges as a result of the cut. (image courtesy of [85])

All triangles in the triangle strip shown in Figure 4.6c are attached to free edges of the previous triangle. As mentioned before, OpenGL requires to alternate between left and right free edges. With an exception for vertices 0, 1, 6, and 5 at either end of the strip, each vertex has three incident edges. Note that each vertex, except for vertices 3 and 7, is

encoded twice. Even this simple example shows that the encoding strategy used by OpenGL is sub-optimal in most cases. By avoiding vertex replication and using left/right bits for each triangle, the triangle strip format can be improved significantly: One bit per triangle encodes whether the next triangle is attached to either the left or the right free edge and an additional bit indicates whether the next vertex has already been encoded. Thus, the micro-topology cost becomes two bits per triangle plus $\log(v)$ for half the triangles which reference a previously decoded vertex. The total cost per triangle is

$$2 + \frac{1}{2}\log(v), \tag{4.6}$$

where v denotes the total number of vertices in the triangulation. Various other schemes, including [26] and [12], have been proposed which further reduce this cost.

Apparently, a major challenge is to find a suitable triangle strip beforehand. A key observation is that any genus-0 manifold is topologically equivalent to a planar graph. This relationship, which is depicted in Figure 4.7, can be exploited to code a winding path within that graph.

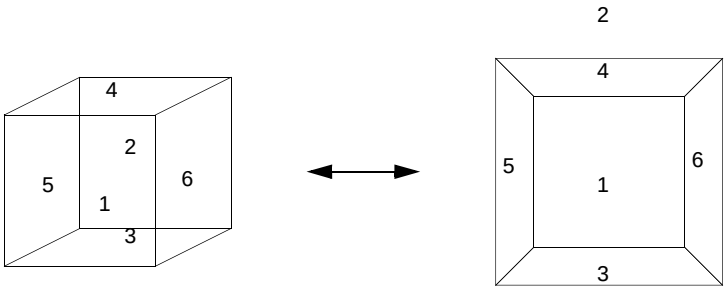


Figure 4.7 Relationship between a genus-0 2-manifold and a planar graph.

Most efforts in recent research have concentrated on the compression of that connectivity graph and various different approaches have been proposed [26, 101, 47, 85, 58].

4.5.2 Compression of Geometry

Using IEEE-754 floating point numbers to represent the vertex positions of a geometric model allows one to choose the positions within many orders of magnitude. For a given object, however, the full eight bit exponent is not needed to accurately represent the vertices. The usage of the remaining 23 bit fixed-point mantissa allows a geometric representation which is still too detailed for most applications in practice. Quantization methods, as explained in Section 4.4.1, can be employed to further reduce the number of bits per vertex position. The vertex coordinates can therefore be represented as integer values relative to the bounding box of the model. A uniform quantizer for a planar graph is depicted in Figure 4.8.

Instead of quantizing the coordinate vectors independently, one can exploit the coherence of neighboring vertices. At compression and decompression, the position of the next vertex can be predicted from already known vertices. The prediction errors can than be encoded in corrective vectors (i.e., the positional delta) using a variable-length entropy coder (e.g., Huffman coding). The size of the corrective coordinates depends largely on the quality of the prediction.

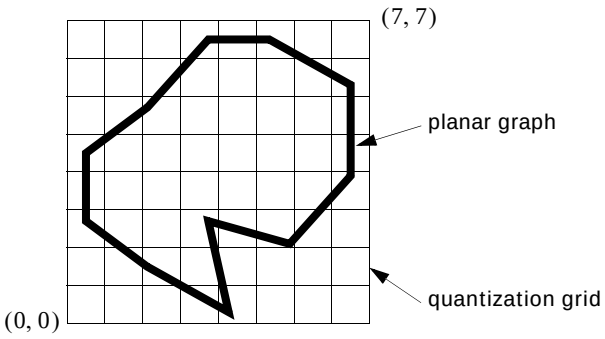


Figure 4.8 Quantization of vertex positions in a planar graph using 3-bit integers.

A straight-forward approach is to store the corrective vector relative to the first or the previous vertex [26]. A more efficient scheme is used in [101]. A spanning tree of vertices is constructed from the mesh. The position of a vertex $P_{\mathbf{v}_i}$ is predicted using a linear combination of its K ancestors $\mathbf{v}_{i-1}, \ldots, \mathbf{v}_{i-K}$ in the spanning tree:

$$P_{v_i}(\lambda, \mathbf{v}_{i-1}, \ldots, \mathbf{v}_{i-K}) = \sum_{k=1}^{K} \lambda_k \mathbf{v}_{i-k},$$

where λ is a vector of integer weights. For $K = 1$ and $\lambda_i = 1$, this is equal to the predictor used in [26]. The choice of K and λ determines the quality of the predictor. Taubin and Rossignac [101] estimate λ by minimizing the least-square error

$$\sum_{i \geq K} \|\varepsilon_i\|^2.$$

A different predictor is used by Touma and Gotsman [103]. As a result of their connectivity scheme, each new vertex $\mathbf{v}_i$ is neighboring a triangle $\{\mathbf{v}_j, \mathbf{v}_k, \mathbf{v}_l\}$ whose vertices have already been decoded. Hence, $\mathbf{v}_i$, together with two of the other vertices, forms a second triangle $\{\mathbf{v}_i, \mathbf{v}_j, \mathbf{v}_k\}$. These two triangles form a parallelogram $\{\mathbf{v}_i, \mathbf{v}_j, \mathbf{v}_k, \mathbf{v}_l\}$. Accordingly, the position of $\mathbf{v}_i$ can be predicted using the following parallelogram rule:

$$\mathbf{v}_i = \mathbf{v}_k + \mathbf{v}_j - \mathbf{v}_l.$$

Since the parallelogram is not necessarily planar, the prediction can further be improved by estimating the crease angle along the edge $\{\mathbf{v}_j, \mathbf{v}_k\}$. The average of the crease angles between two sets of adjacent triangles proves to be a suitable estimate.

As a result, geometry compressed with one of the above methods requires between four and six bits per coordinate [102].

5

WAVELET-BASED SURFACE AND VOLUME COMPRESSION

In the previous chapters, basic concepts from wavelet theory, geometric representations and data compression have been introduced and discussed. In this chapter, we will employ some of these methods to build a novel surface and volume compression framework. This framework comprises a wavelet-based compression pipeline and two mesh simplification schemes.

The first scheme is a bottom-up vertex removal method, generating a quadtree representation of the mesh. We traverse the full quadtree bottom-up, subsequently removing unimportant vertices. A fast look-up-table approach is employed to determine connectivity information for the quadtree on the fly.

The second scheme is a top-down vertex insertion technique, which is combined with Delaunay triangulation methods. We start at the root node of the quadtree/octree and insert relevant vertices. This allows for progressive transmission and reconstruction of surface and volume meshes. Furthermore, the special problem of implicit interpolation for iso-lines and -surfaces in adaptive multi-resolution representations is addressed.

The novel methods presented in this chapter are well suited for a variety of data sets defined on regular grids. The use of wavelet functions provide for a flexible, fast, and elegant solution for compression of geometric information including attributes.

5.1 PRINCIPLES

Figure 5.1 depicts an overview of the framework. The individual components can be combined according to requirements of the application. A remote server performs data compression and is governed by parameter settings for global and local oracles, and a bitstream received by a client is produced. It is at this step where geometric reconstruction and interactive visualization are computed. The quality of the geometric reconstruction computed

by the client can be controlled depending on network performance, computational and storage capabilities of the client or on the data themselves.

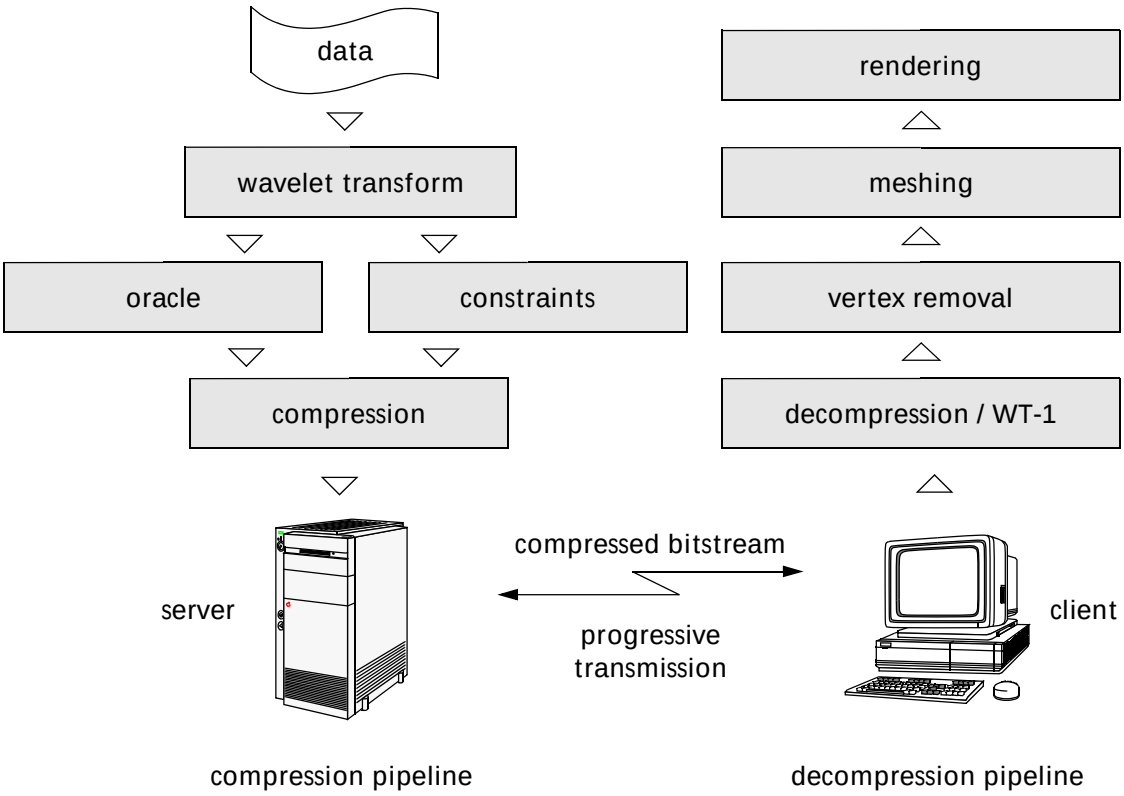


Figure 5.1 Illustration of the conceptual components of the compression and reconstruction framework embedded into a client-server setup

The restriction to uniformly sampled data might be considered a major drawback. The rich variety of applications covered by this approach, however, justifies the presented research.

5.2 GEOMETRY COMPRESSION

This section explains in detail all essential processing steps associated with the definition of appropriate geometry compression strategies. Firstly, an overview of the compression and decompression pipelines, which are hybrid in the sense that they combine both lossy and lossless methods depending on the type of feature to encode, is given.

Some important and widely-used data compression methods have been discussed in Chapter 4. However, none of these methods directly applies to geometry compression. Hence, the individual requirements of a geometry-based approach encouraged us to design the pipeline explained subsequently. For instance, in the context of lossy compression, issues of floating point data handling and quantization must be adapted to our needs where the structure of the wavelet representation plays an important role. Furthermore, additional effort has to be spent on progressive settings. Since the preservation of constraints, such as iso-values, boundary-values, or lines, is desirable in many applications, we propose a lossless compression strategy for these features.

5.2.1 Overview

Employing a wavelet-based sub-band codec, we designed a compression/decompression pipeline as depicted in Figure 5.2. The forward compression proceeds as follows: After extraction of constraints (see Section 5.2.3), the data set is normalized, transformed into wavelet space and both local and global approximation errors are controlled by the oracles introduced in Section 2.4.1 and Section 2.4.2. Sorting of the individual channels of the WT transforms the multi-dimensional array into a 1-D data vector, which is quantized and encoded subsequently. Line constraints, as extracted earlier, are fed into a lossless compression scheme. Conversely, the decompression pipeline inverts the procedure and prepares the data for subsequent geometric reconstruction.

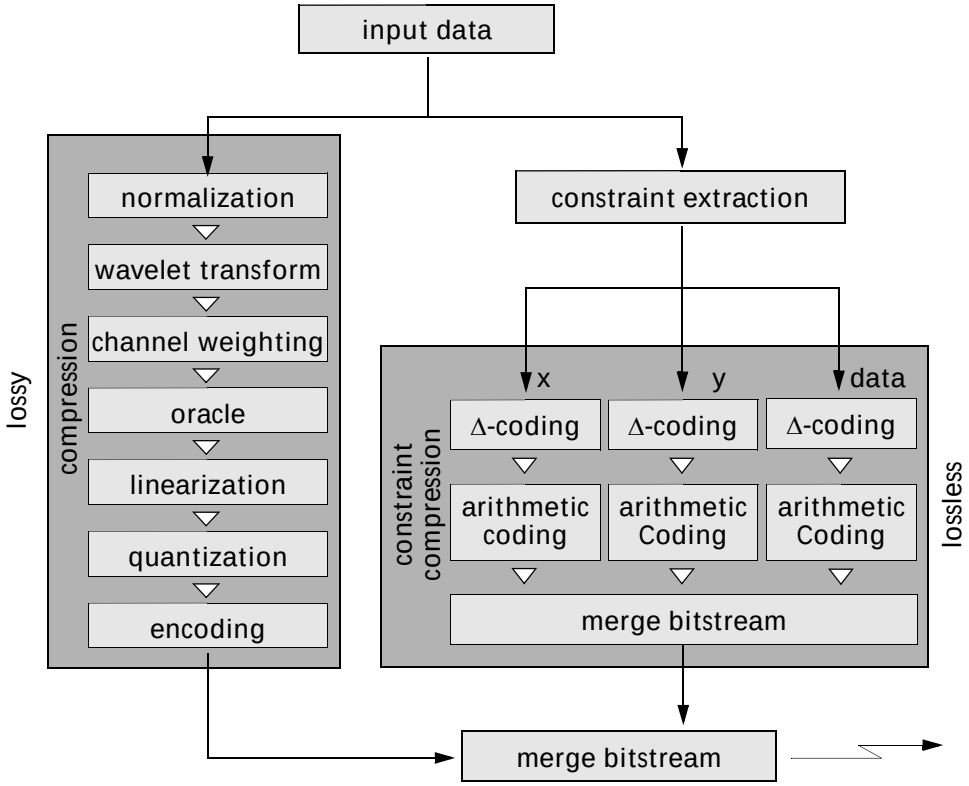


Figure 5.2 Compression pipeline including both lossless and lossy data compression. For decompression, all of the above steps have to be reversed.

5.2.2 Progressive Lossy Compression

Normalization. As opposed to applications such as image compression, the order of magnitude for geometric data is usually not known in advance, but may vary between data sets and application areas. For that reason, the data are normalized, that is the original values c_{orig} are mapped into the interval $[0,\ldots,1]$ using a simple mapping function:

$$c_{\text{norm}} = \frac{c_{\text{orig}} - c_{\text{max}}}{c_{\text{max}} - c_{\text{min}}}. \tag{5.1}$$

The normalized value is denoted with c_{norm} while c_{max} and c_{min} represent the maximum and minimum values of the data, respectively.

Normalization is carried out *before* transformation, because post–normalization maps an offset onto small wavelet coefficients and is more difficult to handle upon compression.

For reconstruction, it is straightforward to de-normalize the coefficients as follows:

$$c_{\text{orig}} = c_{\text{norm}}(c_{\max} - c_{\min}) + c_{\min}. \tag{5.2}$$

Wavelet transform. The subband codec used in this approach employs cardinal B-spline wavelets as introduced in Section 2.2. These basis functions are very well suited for the application of geometry compression, since they share some very important properties:

- *Endpoint-interpolation* is of special importance. Serious artefacts may arise in wavelet-based mesh simplification methods when the boundary problems of the wavelet transform are not compensated or avoided [43].
- *A high number of vanishing moments* is closely related to the compression performance of the codec [88, 110]. For B-spline wavelets, the number of vanishing moments is proportional to the order of the basis functions. See Section 2.2.2 for details.
- *Compact support* is especially important for local oracle operations, where the region of influence of the basis functions should be as small as possible. Again, the support of the B-spline bases is proportional to their order as defined in Section 2.2.2.
- *An efficient implementation* of the fast wavelet transform is necessary to yield a good overall performance of the codec. The complexity of the fast B-spline wavelet transform is equal to $O(n)$, where n denotes the number of discrete samples.

Hence, the B-spline wavelet basis is an appropriate choice for a high-performance subband codec.

Oracle. Local and global oracles, which have already been introduced in Section 2.4.1 and Section 2.4.2 are used to derive a truncated series of basis functions. In spite of the fact that semi-orthogonal B-spline wavelets are used for this codec, orthogonal oracles based on the L^2-norm are employed. They can only approximate semi-orthogonal L^2-oracles. Gross[40], however, presents a comparison between orthogonal and semi-orthogonal L^2-oracles, which provides for a justification of this approximation.

Linearization. To prepare the data for bandwise progressive transmission, the multidimensional coefficient array is mapped into an 1-D vector as displayed in Figure 5.3. Here, the array is traversed from the most significant scaling function coefficients to the high frequency bands representing fine grained detail.

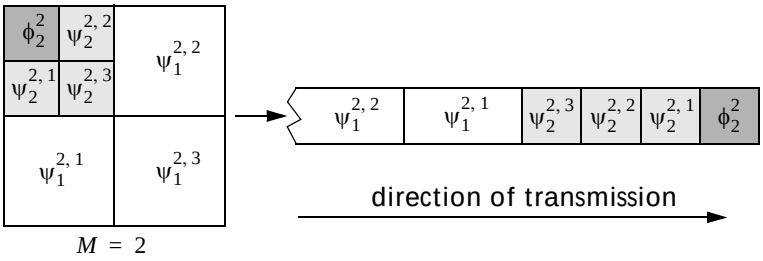


Figure 5.3 Conversion of the multidimensional array into a 1-D coefficient vector depicted for a 2-D WT.

This linearization step is carried out similar to the “zig-zag”-traversal of the DCT coefficients in the JPEG image compression standard [112]. The DCT coefficients are spa-

tially clustered according to their frequency, where the DC part, representing the average value in the image and thus the lowest frequency, is located in one corner of the decomposition, and the coefficients with the highest frequency in the opposite corner. For that reason, a "zig-zag"-traversal sorts all coefficients with increasing frequency.

The wavelet transform, which is localized in both spatial and frequency domain, however, does not reorder coefficients according to their frequency, but preserves the original ordering. Hence, "zig-zag"-traversal of wavelet and scaling functions coefficients would not necessarily improve the compression performance for subsequent run-length encoding. Due to the high number of vanishing moments of the B-spline wavelets and the use of global and local oracles, clusters of zero-valued coefficients are spread throughout the decomposed array, howbeit the position, size, and orientation of these clusters depends largely on the data and is usually not known in advance. Therefore, the linearization as introduced above delivers good results for the proposed compression pipeline.

Note that the linearized vector contains floating point values that have to be converted into an array of integers.

Another common concept for the ordering and compression of wavelet coefficients is the embedded zerotree wavelet algorithm (EZW) [91]. Originally designed for progressive compression and transmission of 2-D images, this method could easily be extended to higher dimensions. The order of progression of the EZW is the numerical precision of the coefficients. This does not seem to be advantageous in our setting, however, where the vertex insertion scheme is based on the coefficients which are further processed after being decoded progressively. If significant coefficients are not send at high precesion, either expensive re-computes of vertices already inserted or discarded, or loss of precision would be the consequence.

Quantization. The quantization step comprises a multiplication of the initial floating point coefficients with a factor of 2^{n-1}, where n represents the number of bits to be assigned for each coefficient. Subsequent rounding operations transform the floating point value into signed integer formats of size n. Let c_{float} be a coefficient. Its quantized version c_{quant} is obtained by applying a scalar midtreat quantizer of the form

$$c_{\text{quant}} = \text{round}(2^{n-1} \cdot c_{\text{float}}). \tag{5.3}$$

A similar quantizer is depicted in Figure 4.2.

Note that n strongly affects the quantization error and appears as noise after reconstruction. Lossless quantization would typically require 23 bits on a 32 bit machine for single precision due to the IEEE–754 floating point format [51].

On first thought, it would appear naturally to employ vector quantization instead of scalar quantization. Separate code books could be designed for each subband, reflecting the characteristics of that particular subband. Unfortunately, the low-low subband requires a larger number of bits per sample and vector quantizers perform badly at high rates [88]. Consider the following example: The code book size for a 10-dimensional 4-bits-per-sample vector quantizer would be $10^{10 \times 4} = 10^{40}$. If the input data was represented with 32 bit floating point values, a rate of 4-bits-per-sample corresponds to a compression ratio of 8:1. The memory requirements to store this code book is equal to $2^{40} \times 10 \times 4$ bytes—approximately 42 terabytes—and the number of comparisons to find the closest vector is more than one trillion. For these reasons, scalar quantizing is generally used in conjunction with subband coding [88].

Encoding and Bit Allocation. An important task in the proposed compression pipeline is to convert the quantized integer vector into a bitstream of data. Therefore, an entropy coding scheme in the spirit of JPEG [112] is employed. Assuming that many of the coefficients will be equal to zero, encoding is carried out as follows: All non-zero coefficients are represented by two-tuples, where the first element represents the number of bits required to encode the second one. The second element contains the data value itself. All negative numbers are thus replaced by their absolute values, where in the case of a positive number the first bit is cleared. This enables one to encode of the sign elegantly. See Table 5.1 for some examples.

Table 5.1 Integer coefficients and associated 2-tuples.

Positive coefficient	2-tuple	Negative coefficient	2-tuple
3	(2, 01)	-3	(2, 11)
17	(5, 00001)	-17	(5, 10001)
5	(3, 001)	-5	(3, 101)
30	(5, 01110)	-30	(5, 11110)

Note specifically that since the number of bits is known in advance, the representation is unique and the additional encoding of the sign bit in the most significant bit is possible.

Zero-valued coefficients are encoded differently. Here, a runlength coding up to a length of $2^5 = 32$ is applied, which generates a set of 32 new symbols. These symbols, together with the first part of the two-tuples, are stored in a Huffman–table which has essentially 64 entries. The Huffman symbols are chosen as follows:

- *Symbols 0–30:* First element of a 2-tuple minus 1
- *Symbol 31:* EOB (End Of Bitstream)
- *Symbols 32–63:* Runlength of zero-coefficients

The scheme proposed here compromises the complexity of the Huffman–table with the maximum number of zero coefficients (32) to be encoded in one symbol. The *EOB* (end of block) symbol usually allows for the encoding of long sequences of zero-coefficients in the least significant positions of the data vector. However, it is only used when the Huffman table has not been built individually. The following pseudocode illustrates the procedural flow of the scheme:

```
// N:total number of integer coefficients
// d_i: coefficient i
// hufflen_i: length of Huffman–code for symbol i
// huffcode_i: Huffman-code for symbol i
// WriteBits(l,i):
//       appends the last l bits of i to bitsream
// Make2Tupel(i,first,second):
//       converts integer into 2-tuple
i ← 0;
while i < N do
  if d_i = 0 then
    j ← 0;
    while j<32 && d_i ← 0 do inc(i); inc(j); end;
    WriteBits(hufflen_j+31, huffcode_j+31);
  else
    Make2Tupel(d_i,first,second);
    WriteBits(hufflen_first-1,huffcode_first-1);
```

```
    WriteBits(first,second);
    inc(i);
  end;
end;
WriteBits(hufflen31,huffcode31);
```

The construction of the Huffman table is straightforward and is not explained in detail (see Section 4.3.1). In this codec, however, the Huffman table is generated individually for each data set upon compression and is transmitted along with the data and header information, which is presented in Table 5.2. Since the size of the table is fixed to 64 entries, this does not introduce a notable overhead. Another solution would be the employment of a generic table, such as in image compression which, however, drops the compression gain and, due to the variety of geometric data, is much more difficult to construct. An example of encoding a sequence of coefficients is given in Figure 5.4.

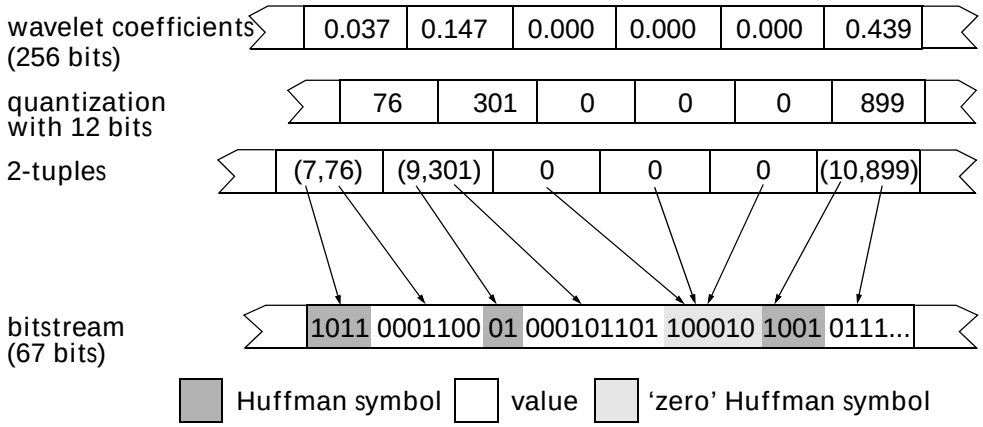


Figure 5.4 Example of encoding a sequence of coefficients and the resulting bitstream.

It should be stated that progression is achieved channel by channel (i.e., subband by subband). That is, the low frequency scaling function coefficients are transmitted first, followed by the wavelet coefficient channels in order of ascending frequency.

5.2.3 Compression of Constraints

In some applications, it is desirable to compress spatially interesting features, such as boundary- or iso-lines and individual vertices in a lossless manner. These data are commonly referred to as *constraints*, since they usually constrain subsequent geometric reconstruction.

A good example of those constraints are material boundary interfaces within volumetric data sets. Those data sets contain volume fractions with different material properties, which are bounded by interface surfaces [8]. Those interfaces can be extracted from the data and can thus constrain the simplification of the model.

In the proposed pipeline, constraints are represented as polylines or polygons. Figure 5.5 illustrates the use of constraints in a digital terrain data set of the Swiss Alps. Here, the geometric reconstruction (i.e., triangulation of the surface) was simplified up to a given bound. The constraints invoked by the polygon force the reconstruction to keep the triangulation dense, however. The constraint is imposed in terms of a terrain following polyline of a given extent.

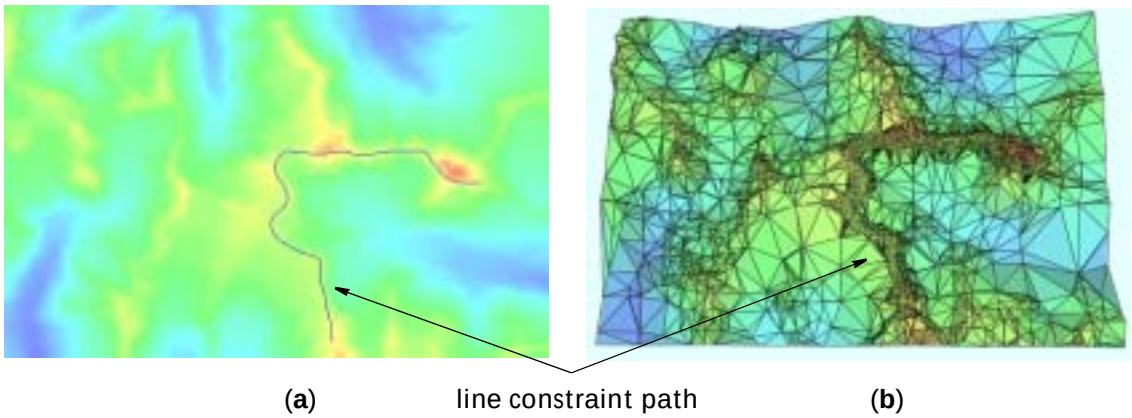


Figure 5.5 Illustration of constraints in a digital terrain data set. **a:** Interactive specification of the constraint path. **b:** Mesh after constraint insertion. [DHM©]

Assuming the polyline constraint is represented as a stream of vertices of type (x, y, data), a lossless compression strategy, as shown in Figure 5.2, is employed.

The position (x, y) and the data value are encoded separately using both delta and higher order arithmetic compression algorithms [88].

The resulting bitstream format is presented below in Figure 5.6, where two headers are followed by the x-stream, the y-stream, and the data-stream.

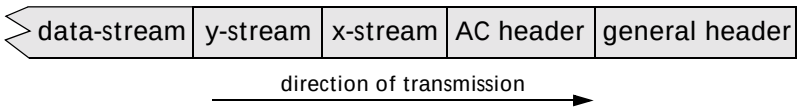


Figure 5.6 Data format of the bitstream for constraint compression. The individual header formats are given in Table 5.2.

Table 5.2 Header formats of bitstream.

	Name	Type	Description
General Header	magic_number	byte	ASCII '67'
	stream_size	integer	total size
	xValues_size	integer	size of x-stream
	yValues_size	integer	size of y-stream
	info	byte	misc info
	width	float	constraint width
	field_dims	integer[2]	mesh dimension
	npoints	integer	# points of constraint
	ndata	integer	# extracted data values
	xFirstValue	float	first x-coordinate
	yFirstValue	float	first y-coordinate

Table 5.2 Header formats of bitstream. (Continued)

	Name	Type	Description
Header for Arithmettic Coding	arithFirstValue	float	first extracted data value
	maxValue	float	maximal value
	minValue	float	minimal value
	nIntBits	integer	multiplication factor
	huffFirstValue	integer	1st integer value
Header for Encode	iterationDepth	short	iteration depth
	weightArr	float[]	array of weights
	huffTable	integer[64]	Huffman–table
	quantBits	short	quantization (# of bits)
	degree	short	degree of B–spline bases
	minValue	float	minimum coefficient value
	maxValue	float	maximum coefficient value
	nDim	short	# of dimensions
	dimArr	short[]	dimension array

5.3 QUADTREE-BASED VERTEX REMOVAL

The new quadtree-based vertex removal scheme is a very fast and efficient method for further reducing the complexity of the data.

This bottom-up strategy starts with a dense set of vertices and subsequently removes nodes according to a user-specified error threshold. This process eventually constructs a quadtree data structure which is triangulated by employing an efficient table look-up. The importance of a vertex is determined by analyzing detail coefficients at dyadic points reconstructed from wavelet space. For that reason, a modified inverse wavelet transform has to be designed, which enables one to reconstruct the different sub-bands in wavelet space individually.

A modified compression pipeline, which contains this *single-step inverse wavelet transform* is depicted in Figure 5.7. Forward transform and compression are carried out in the same fashion as proposed in Section 5.2. Note that in order to focus on the quadtree-based vertex removal process, some compression steps have been omitted from the illustration of the pipeline in Figure 5.7.

5.3.1 Single-Step Inverse Wavelet Transform

One issue regarding the fast implementation of the wavelet transform as explained in Section 2.2.3 is, that access to the difference signal is required in each iteration step m of the reconstruction. This is necessary because the detail signal at a particular mesh vertex and ins neighbors finally decides on whether or not it can be removed. For this purpose, the reconstruction pyramid has to be modified, as indicated in Figure 5.8. The procedure

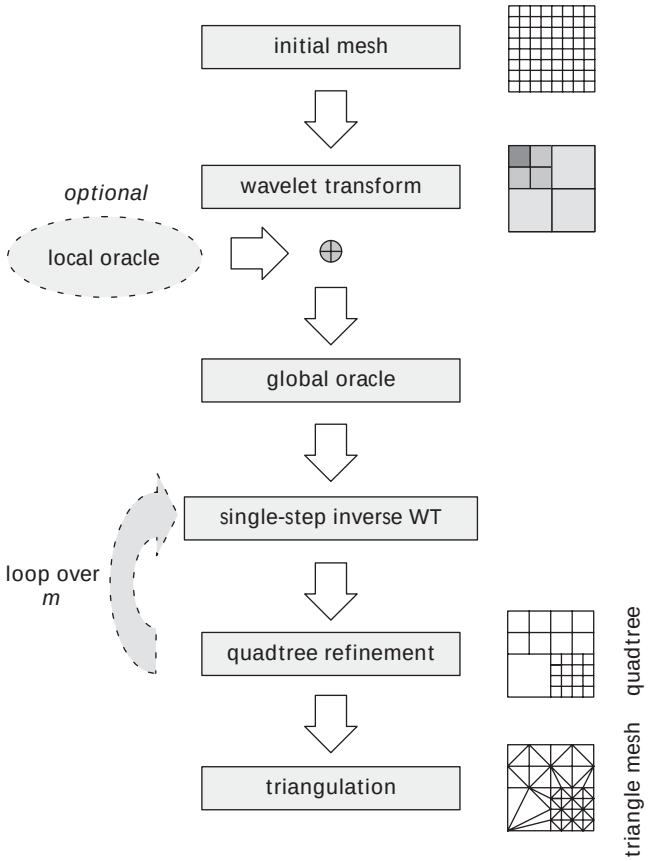


Figure 5.7 The process pipeline for quadtree-based vertex removal employing both the single-step inverse wavelet transform and iterative quadtree coarsening.

recovers the full size detail signals Δf^m represented by all wavelets at $m = 1, \ldots, M$ and by the scaling functions f^M. This can be accomplished by reversing the trace of each detail signal from the original down the decomposition pyramid. In other words, any detail signal Δf^m at iteration depth m can be obtained from the respective wavelet coefficients by subsequent filtering and upsampling. The final output results from superimposing all detail signals:

$$f(x) = \sum_{m=1}^{M} \Delta f^m + f^M. \tag{5.4}$$

Note that this procedure requires additional computation, but although the wavelet coefficients are arranged on a dyadic grid, the Heisenberg principle prevents one from using them as a direct criterion for vertex removal [98].

5.3.2 Vertex Removal in Regular Meshes

So far, some mathematical criteria have been elaborated for approximating a surface data set, sampled on a regular grid, using a multi-resolution hierarchy. In order to build an adaptive surface triangulation, however, it is necessary to remove unimportant mesh vertices and to find a triangulation of the remaining ones. The basic criterion by which a mesh

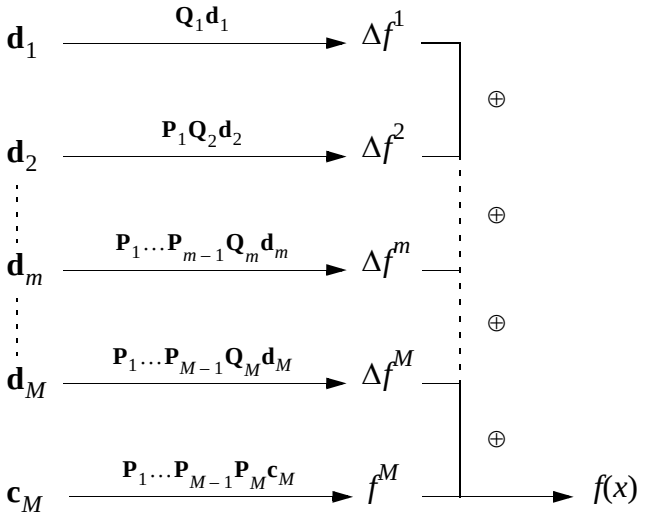


Figure 5.8 Single-step inverse wavelet transform which is used to recover detail signals of individual subbands.

vertex is labeled as unimportant is given by the mathematical framework of the wavelet transform. In contrast to existing methods [69] which base on linear spline wavelets, this scheme can be generalized to any type of wavelet. Therefore, the criteria require more elaboration. Keeping in mind that any triangulation of the surface provides a planar approximation, the error between the original surface function $f(x, y)$ and the bilinear interpolant provided by a triangle has to be bound. Supposing furthermore that the initial data is expanded by wavelet bases, the detail signal in iteration m is used to decide on whether or not each $2m + 1$ th mesh vertex is necessary for the triangle approximation. First, each second vertex is visited and the value of the detail signal of iteration $m = 1$ is analyzed. If the detail signal Δf^1 in some neighborhood of vertex n is sufficiently small, then the vertex is considered not to be important and the approximation can be accomplished by a linear interpolation between vertex $n - 1$ and $n + 1$. This scheme can now be applied recursively by subsequent computation of the detail signals Δf^m; $m = 1, \ldots, M$ and by visiting all dyadic vertices at positions $n = 2mk + 1$. Once the detail signal is sufficiently small and the adjacent vertices in step $m - 1$ are already removed, the current vertex is labeled as well.

As a consequence, this procedure results in recursively building a quadtree representation of the initial mesh by removing dyadic vertices. Figure 5.9 illustrates the thinning method which finally figures out the symbolic quadtree representation of the mesh vertices depicted as an example in Figure 5.10. The nodes of the quadtree contain either pointers to some child-nodes, or in case of leaves, point to the entries of a vertex list.

Note again that the growth of the quadtree is entirely controlled by a single energy threshold τ, the global oracle, embedded in the function space of the wavelets. The final maximum depth of the tree depends on the upper decomposition bound M of the wavelet transform.

At first glance, it seems to be natural to take a particular wavelet coefficient to decide on whether a vertex can be removed. Unfortunatly, this only goes well in the trivial case of linear splines. This is ultimately founded in Heisenberg's uncertainty principle, which determines the lower bound of the spatial frequency localization (see Section 2.1.2 on page 9). Extending the methods to arbitrary types of wavelets, however, requires further

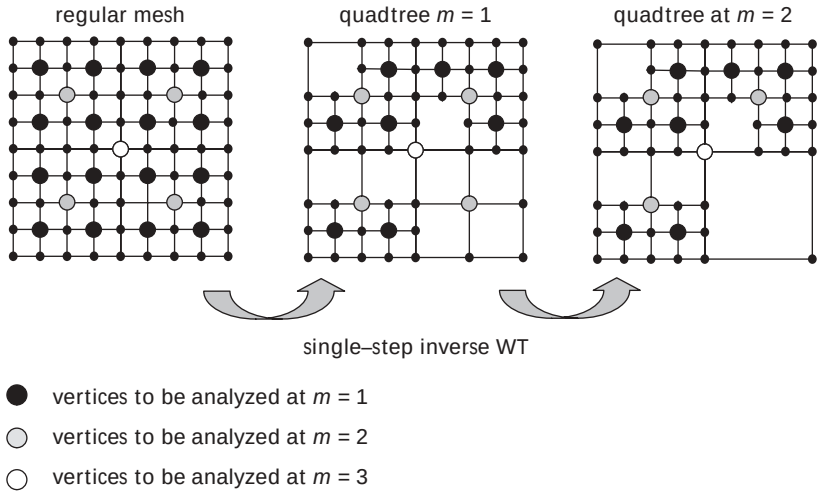


Figure 5.9 Bottom-up construction of a quadtree from a regular mesh by analyzing the detail signals of the WT at each dyadic vertex.

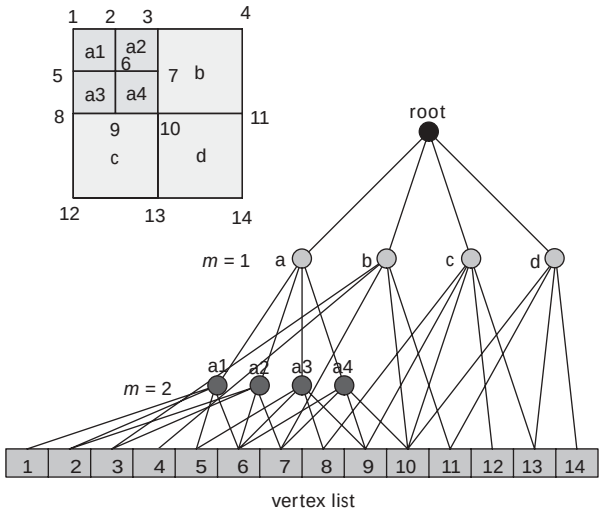


Figure 5.10 Symbolic representation of the mesh using a quadtree data structure.

elaboration on the criteria. If a wavelet has a particular spatial support, the inner product with the data exactly represents its contribution in that region. In the general case, however, it is not possible to recover the contribution of the wavelet at a specific vertex position in that region from the coefficients. Moreover, this would presume a wavelet, whose spatial localization drops to zero, such as with the lazy wavelet [99]. Obviously, more appropriate criteria than wavelet coefficients have to be found.

Therefore, to finally decide on whether a mesh vertex can be removed or not, we propose the following three criteria, which additionally help to preserve the topology of the tree. A vertex will only be removed from the mesh if all criteria are met:

1. *Wavelet-criterion:* A vertex at iteration level m can be removed if the sum of the squares of its difference signal and those within a 4-neighborhood at resolution m are less than an upper bound ε (see Figure 5.11a):

$$\Delta_{\mathbf{A}}^2 + \Delta_{\mathbf{B}}^2 + \Delta_{\mathbf{C}}^2 + \Delta_{\mathbf{D}}^2 + \Delta_{\mathbf{P}}^2 \leq \varepsilon \tag{5.5}$$

Even though the difference signals are assumed to be set to zero upon removal, numerical reasons require to set ε to a small positive number.

2. *Resolution-criterion:* A vertex at iteration level m can be removed iff the four surrounding vertices at level $m-1$ have been removed in a previous step (Figure 5.11b).

3. *m to $m-2$-criterion:* A vertex can be removed if the resulting cell is not adjacent to any cell of a level heigher than $m-2$. Thus, growth is restricted to cell transitions from m to $m-2$, which simplifies the triangulation algorithm (Figure 5.11c).

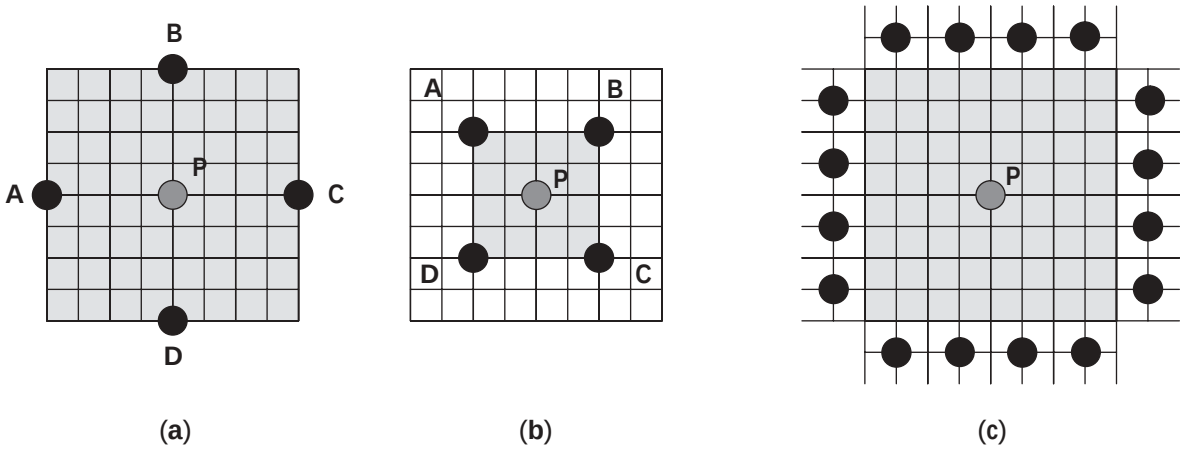


Figure 5.11 Illustration of the different criteria to decide on the vertex removal. **a:** wavelet criterion. **b:** resolution criterion. **c:** m to $m-2$-criterion.

Another aspect of the method is illustrated in Figure 5.12a, where vertex $\mathbf{P}$ is analyzed. Suppose that $\mathbf{P}$ meets all of the above criteria. If $\mathbf{P}$ is removed, however, and if $\mathbf{P}_U$ and $\mathbf{P}_L$ have already been removed, that is, if two adjacent cells have the same resolution, then the vertices $\mathbf{A}$ and $\mathbf{B}$ on the cell boundaries have to be rejected, too. Hence, when traversing the vertex array from the upper left to lower lower right corner, one has to check as well the upper and the left vertices at the same level m.

This additional criterion eventually generates a partitioning of the initial array into different regions (see Figure 5.12b). Within these regions, only the left, the upper or both adjacent vertices have to be checked.

5.3.3 Look-up Tables for Local Triangulations

Once the tree is built from the above procedure, the quadtree cells have to be triangulated. A generic problem arising from meshing hierarchies of rectangular surface patches is the occurrence of cracks [5]. A crack occurs if no thought has been given to adjacency of quadtree cells at different depth and, hence, different resolution. The surface may break up, holes may appear and any consistency required for normal interpolation gets lost. Non-manifold surfaces will be the result. Figure 5.13 shows a crack as well as a way of modifying the triangulation to avoid it.

The scheme introduced here for fast and consistent cell triangulation is based on the following observation: Consider Figure 5.14 where two adjacent cells are depicted along with topological arrangements that may occur for transitions from level m to $m-1$ and to $m-2$. There are only five different cases at the respective cell boundary. Presuming the growth of the quadtree as being restricted so that only transitions up to $m-2$ are possible

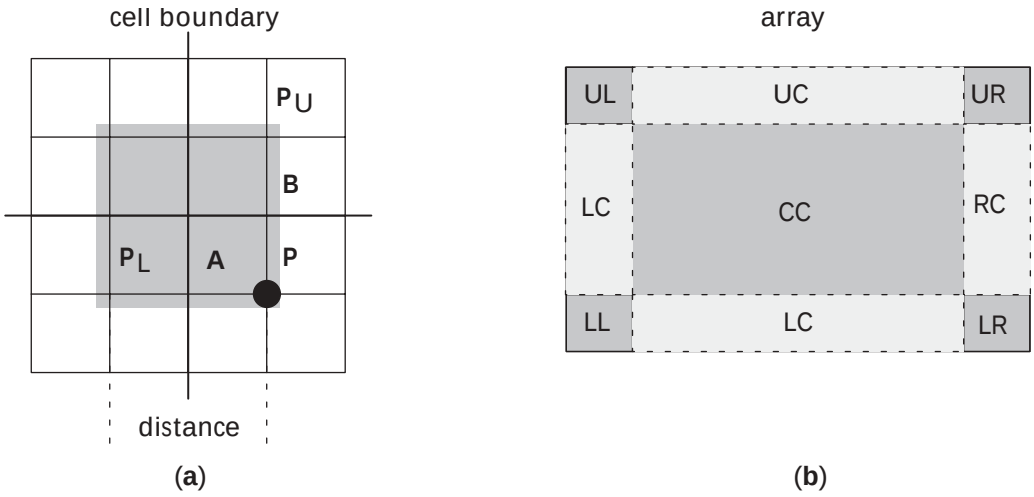


Figure 5.12 **a:** Criteria to remove vertices on the edges of adjacent cells of the same resolution. **b:** Resulting partitioning of the initial array.

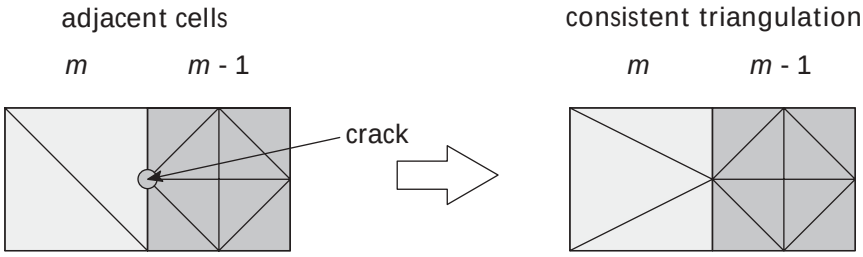


Figure 5.13 The occurrence of cracks at the boundaries of adjacent quadtree cells of different resolution.

(*resolution criterion*), the set of possible arrangements of vertices at the four cell boundaries can now be derived from Figure 5.14. Moreover, some look-up tables may be constructed, containing the triangulations as explained below.

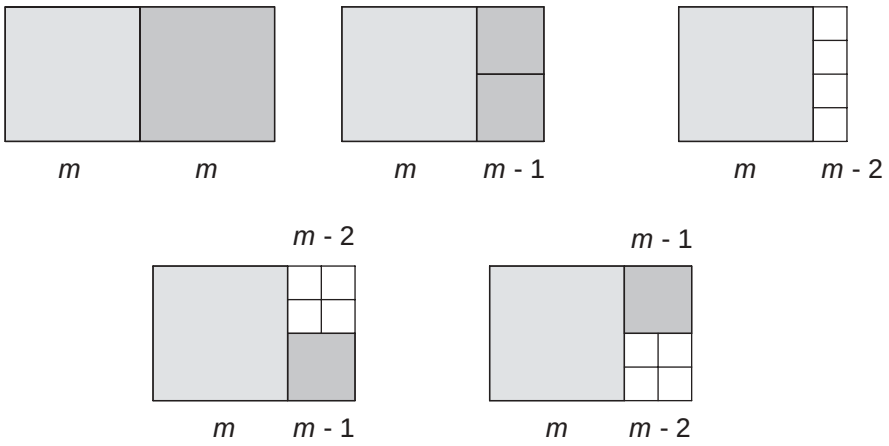


Figure 5.14 Topology of mesh nodes of adjacent cells for different resolutions m, $m-1$, and $m-2$.

For cell transitions from m to $m-1$, a look-up table with 16 entries is built as depicted in Figure 5.15. The central idea of the algorithm is to first solve the triangulation within

each cell for m to $m-1$. This is accomplished by analyzing the mesh vertices along each cell edge.

Fast computation of the look up table entry can be accomplished by a binary outcode [68], generated from bitwise addition of the flags of the respective edge vertices, as indicated in Figure 5.15.

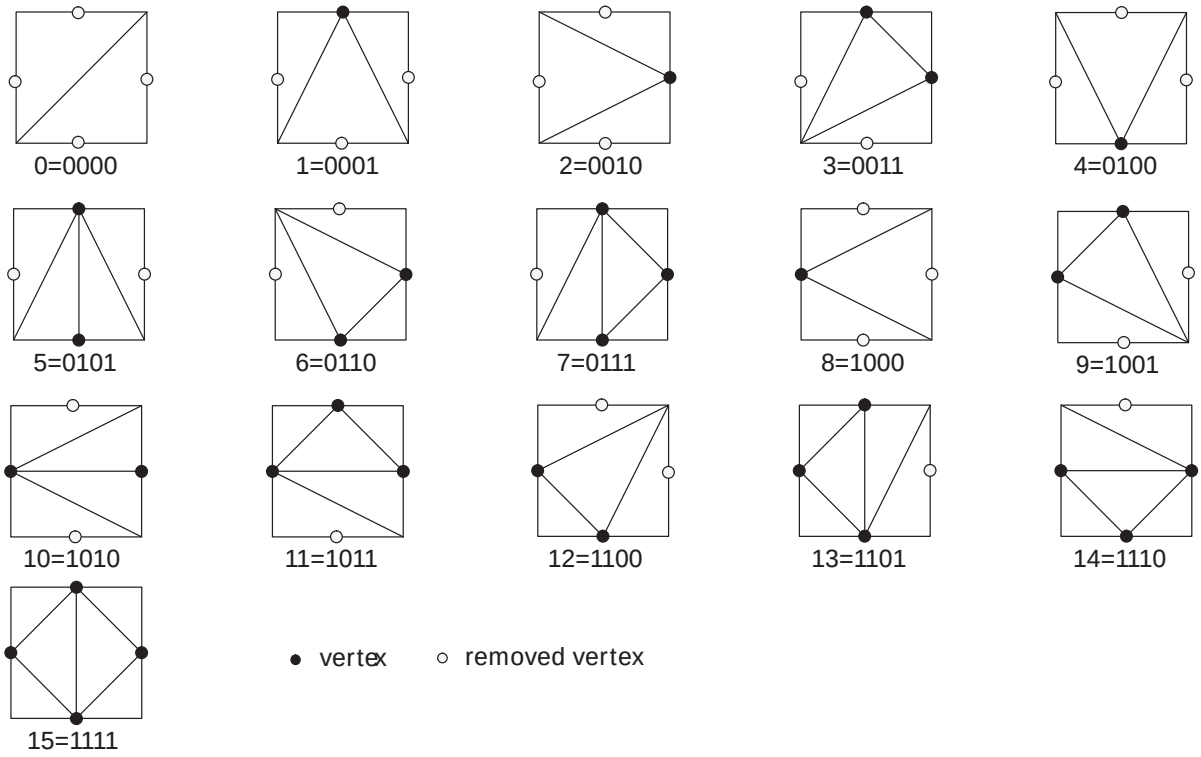


Figure 5.15 Look-up table representing the optimal triangulation of all possible cases from m to $m-1$ with corresponding outcode.

Once the corresponding look-up table entry is identified, mesh vertices which account for the $m-2$ transitions are considered. This may cause some triangles to split up into two pieces, as shown in Figure 5.16. Consequently, the algorithm first handles the triangulation for the transition from m to $m-1$ and then decides on the corresponding subcase by simply analyzing the flags of all intermediate vertices responsible for transitions from m to $m-2$. Although this results in 625 possible cases, the total number of triangles required does not exceed 96. These triangles are stored in a look-up table depicted in Figure 5.17.

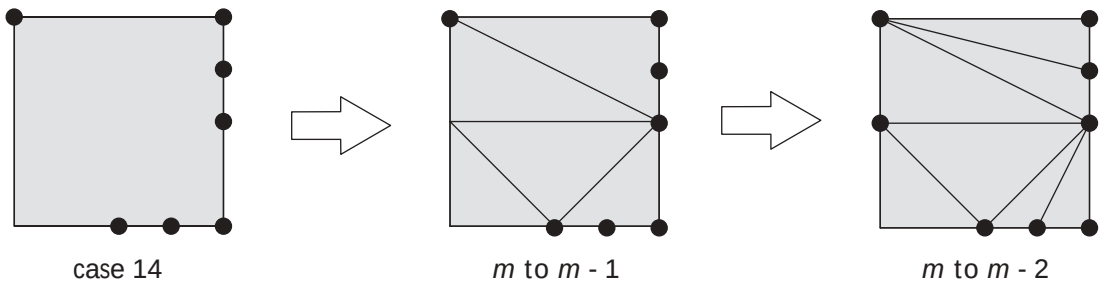


Figure 5.16 Cell triangulation for cases from m to $m-2$ as derived from a look-up table entry of m to $m-1$.

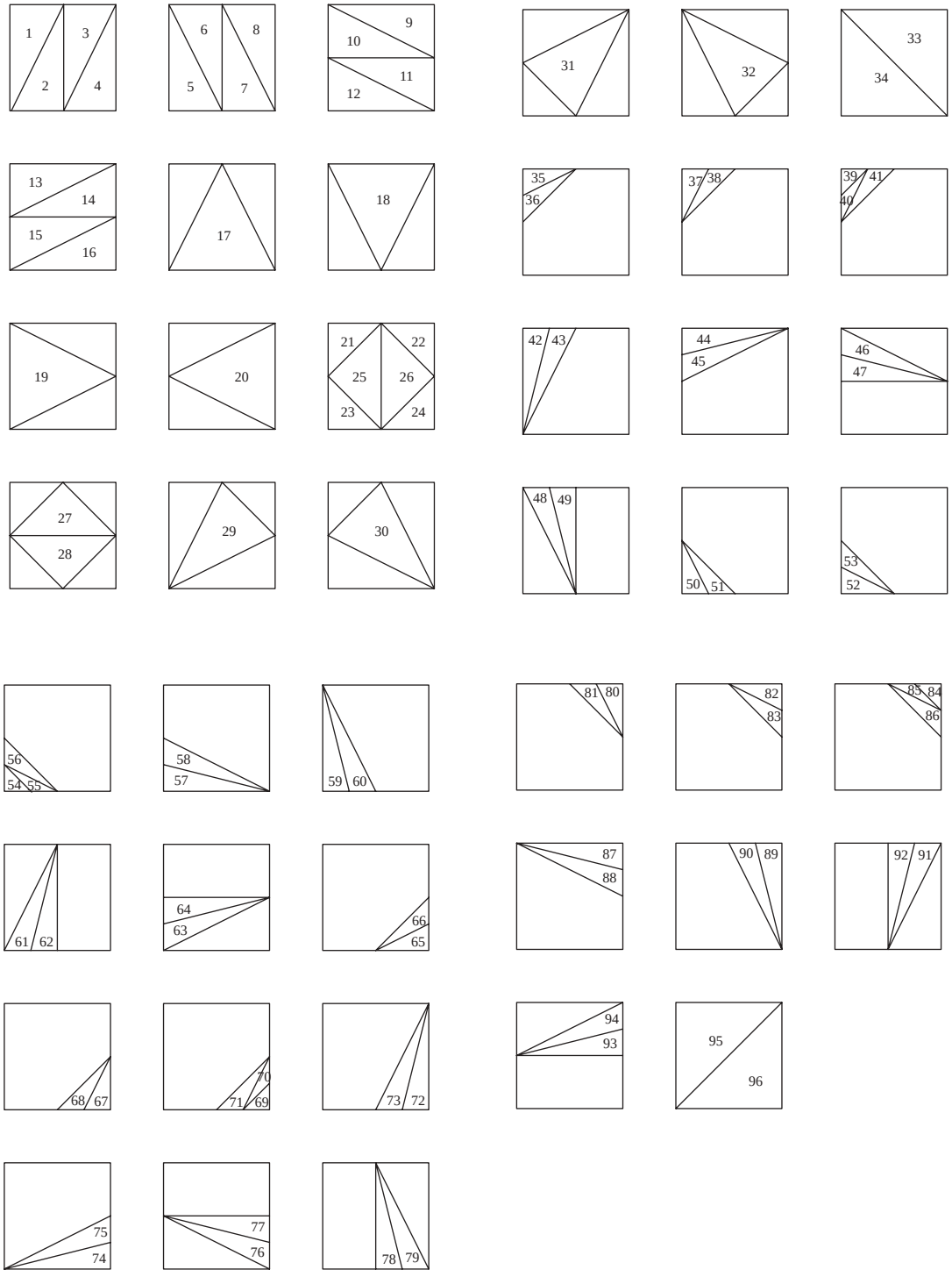


Figure 5.17 Look-up table with all 96 possible triangles which are necessary for transitions from m to $m-2$.

All subcases are hardcoded and contain references to these look-up table entries. Note that although there are 625 cases, only one computation of the outcode and at most eight additional tests are necessary to compute the triangulation. It is clear that one ends up with a very efficient algorithm by doing the meshing without any geometric computation but only by checking vertices along the cell edges.

The corresponding pseudocode for the recursive quadtree traversal and meshing is given with:

```
// The initial array has a size of (N+1)(N+1).
// Let N be a power of 2, N = 2I.
// Each cell is addressed by its upper left corner vertex.
// root cell
x = y = 0;
i = I;
traverse_quadtree(x,y,i);
procedure traverse_quadtree(x,y,i)
{
  // compute center vertex of son cells
  mh = 2i-1;
  xmh = x+mh;
  ymh = y+mh;
  if (i>0) and flag(xmh,ymh)
  {
    i = i-1;
    //analyze the son cells
    traverse_quadtree(x,y,i);
    traverse_quadtree(xmh,y,i);
    traverse_quadtree(x,ymh,i);
    traverse_quadtree(xmh,ymh,i);
  } else
  triangulate(x,y,i);
}
```

5.4 DELAUNAY-BASED VERTEX INSERTION

The quadtree-based vertex removal method introduced in Figure 5.3 has some limitations regarding progressivity, adaptivity, and easy extension into higher dimensions. The bottom-up approach requires successful transmission of all first-level detail coefficients before any vertices can be analyzed and eventually be removed. For a 2-D data set, the amount of data to be transmitted is equal to three quarters of the whole data. Furthermore, since only transitions between two consecutive levels are permitted, the adaptivity of the approximated mesh is limited. Extension into three dimensions is possible, but topological complexity and size of the look-up-tables would be increased substantially [93]. In addition, the simplicity and elegance of the algorithm for two-dimensional data would not be maintained.

The following section addresses a novel top-down vertex insertion method which enables a client to compute geometric reconstructions adaptively and progressively from the incoming bitstream of data. When seeking an appropriate algorithm, computational performance and invariance to the dimensionality are important considerations. The well-known algorithm of Douglas and Peucker [29] is a good starting point. First, its initial form in a non-parametric 1-D setting is introduced and its application in multi-resolution representations is illustrated. Here, special emphasis is given to the extension of the method for progressive reconstruction. Next, the method is generalized to multi-dimensional and parametric settings and some examples of its use are given. The versatility of the introduced method does not impose any restriction on subsequent triangulation methods, which can range from constraint Delaunay triangulation [81] to fast look-up tables [43].

5.4.1 Douglas-Peucker Algorithm

The Douglas-Peucker algorithm is a very popular and high-quality method for representing lines and curves extracted from digitized terrain models [29]. A profound theoretical analysis of this method is given in [3].

The algorithm starts with the two endpoints of the curve: The starting point is called *anchor* and the endpoint is called *floating point.* These two points define a linear approximation of the curve. All intermediate points are inspected to find the point with the largest distance to the initial approximation. If this distance is larger than a particular threshold ε_0, this point becomes the new floating point and the approximation is refined. The floating point is steadily moving towards the anchor until the measured distance is below the threshold. The last floating point then becomes the new anchor and the algorithms is repeated. Figure 5.18 illustrates the Douglas-Peucker algorithm on a simple example.

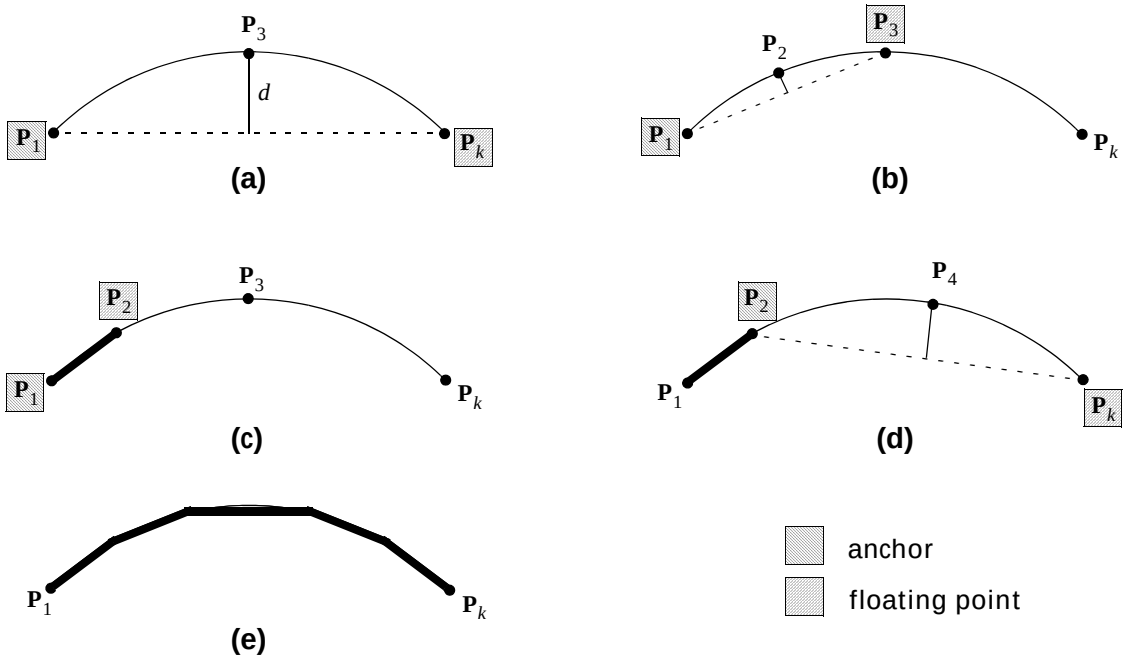


Figure 5.18 The Douglas-Peucker algorithm: **a:** Initialisation: $\mathbf{P}_1$ is the anchor, $\mathbf{P}_k$ is the floating point, $\mathbf{P}_3$ has maximal distance to $\overrightarrow{\mathbf{P}_1\mathbf{P}_k}$. **b:** $\mathbf{P}_3$ becomes the new floating point, $\mathbf{P}_2$ has maximal distance to $\overrightarrow{\mathbf{P}_1\mathbf{P}_3}$. **c:** $\mathbf{P}_2$ becomes the new floating point, the remaining intermediate points are closer to the approximation than the threshold ε_0. **d:** $\overrightarrow{\mathbf{P}_1\mathbf{P}_2}$ is a sufficient approximation for the original curve segment, $\mathbf{P}_2$ becomes the new anchor, $\mathbf{P}_k$ the new floating point. **e:** The final curve approximation.

Figure 5.19 shows an example of the quality of the curve approximation. Apparently, this algorithm can be optimized to yield a lower algorithmic complexity. Several variants are proposed in [29].

5.4.2 One-Dimensional Setting

This section introduces several improved schemes suitable for both non-progressive and progressive settings.

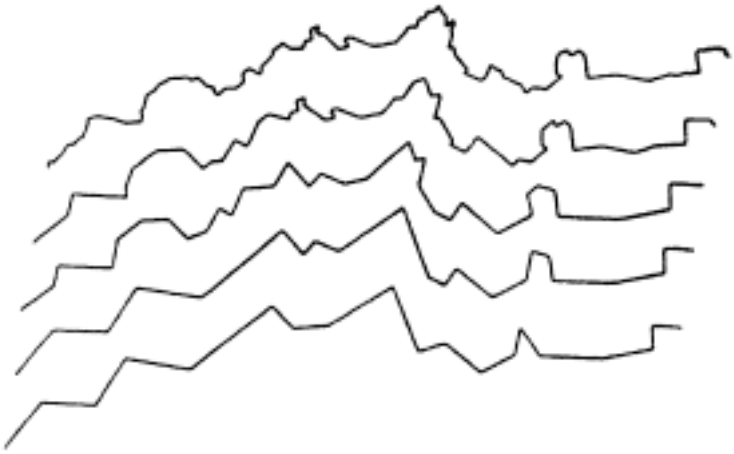


Figure 5.19 Example of a curve approximated with the Douglas-Peucker algorithm for different error thresholds (from [29]).

Non-progressive insertion. In order to devise a top-down vertex insertion strategy, let us first consider the one-dimensional setting. Here, the problem reduces to finding a strategy for the reduction of line segments in piecewise linear approximations. Inspired by the Douglas-Peucker algorithm [29] this method is extended to a recursive version, which is illustrated in Figure 5.20. It starts by connecting the first point of a curve, $\mathbf{P}_0$, with the last point $\mathbf{P}_k$. All intermediate points representing the curve are compared against the line segment $\overrightarrow{\mathbf{P}_0\mathbf{P}_k}$ and the point with the largest distance, for instance $\mathbf{P}_2$, is identified. If its distance exceeds a predefined threshold ε_0, the vertex is considered *important* and is labeled. The initial line segment is split into two halves, on each of which the algorithm is applied recursively. Obviously, the quality of the approximation can be controlled by a distance threshold. The advantage of this extension to the original method lies in the tree type refinement of the vertex analysis coming along with the recurrence relations.

Progressive insertion. In order to provide progressive insertion of vertices in combination with progressive reconstruction of the wavelet subbands, the non-progressive insertion as introduced above has to be extended. First, all scaling function coefficients are decoded and the corresponding vertices are candidates for insertion. As more and more of the wavelet subbands are decoded, additional vertices can be inserted into the current approximation. The main difference to the non-progressive insertion scheme is the fact that not all possible candidates are available for inspection when the algorithm starts.

Hence, the progressive insertion requires an extension of the recursive algorithm. We need to maintain a list of curve segment approximations form precedent levels. Upon decoding of a new wavelet subband, all segments are inspected for insertion of additional vertices from the newly decoded vertices. If one new vertex is inserted with respect to the threshold ε_0, the corresponding curve segment is split into two new segments, which are in turn processed recursively.

Figure 5.21 depicts a simple example for progressive approximation of a sine wave decomposed to level $M = 5$. The reconstruction of the scaling function in Figure 5.21a only provides sufficient information for every 2^{-M}th vertex. In every subsequent step,

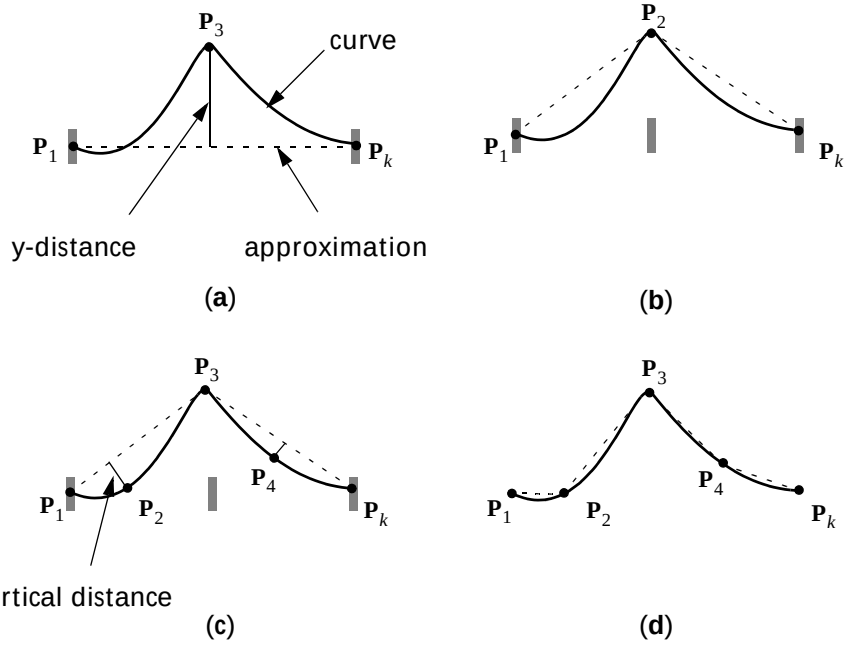


Figure 5.20 **a:** Recursive algorithm assuming a smooth representation of the underlying curve: **a:** P_2 has largest vertical distance. **b:** new approximation after insertion of P_2. **c:** example for vertical distance metric. **d:** final result.

more possible candidates for insertion are decoded and might be inserted into the approximation. With each additional level, the quality of the approximation is improved.

The progressive scheme, however, has some possible disadvantages over the non-progressive version: The first approximation level with information from only the scaling functions might still be too imprecise. It is probable that some vertices inserted at that level would not have been inserted if more information from later levels, which contain the detail information, had already been present. Artifacts arising from this effect are very well visible in the sawtooth example depicted in Figure 5.22. The cubic B-spline scaling functions cannot accurately represent the discontinuities at the maxima of the sawtooth function. The extrema of the approximation are shifted to the left with respect to the original course of the function. With each new wavelet subband arriving, the maximum drift to the right towards the correct location and additional vertices are inserted. Hence, too many superfluous vertices have been inserted in comparison with the approximation resulting from the non-progressive algorithm.

It would be possible to remove these additional vertices from the approximation in the one-dimensional case. This would basically mean, however, that the non-progressive insertion would have to be repeated after the progressive algorithm has finished. Of course, this is not tolerable, since the desired effect of the progressive insertion is to distribute the total cost of refinement over time. In higher dimensions, subsequent removal of vertices would require to design unnecessarily complex data structures.

A good alternative to subsequent vertex removal is to use an adaptive threshold depending on the approximation level. The scaling functions, for example, provide only enough information to coarsely approximate the data. Consequently, it is not practical to use the same small error tolerance values that are used for the final approximation, when much detail information is available. It is cumbersome to specify multiple threshold values for

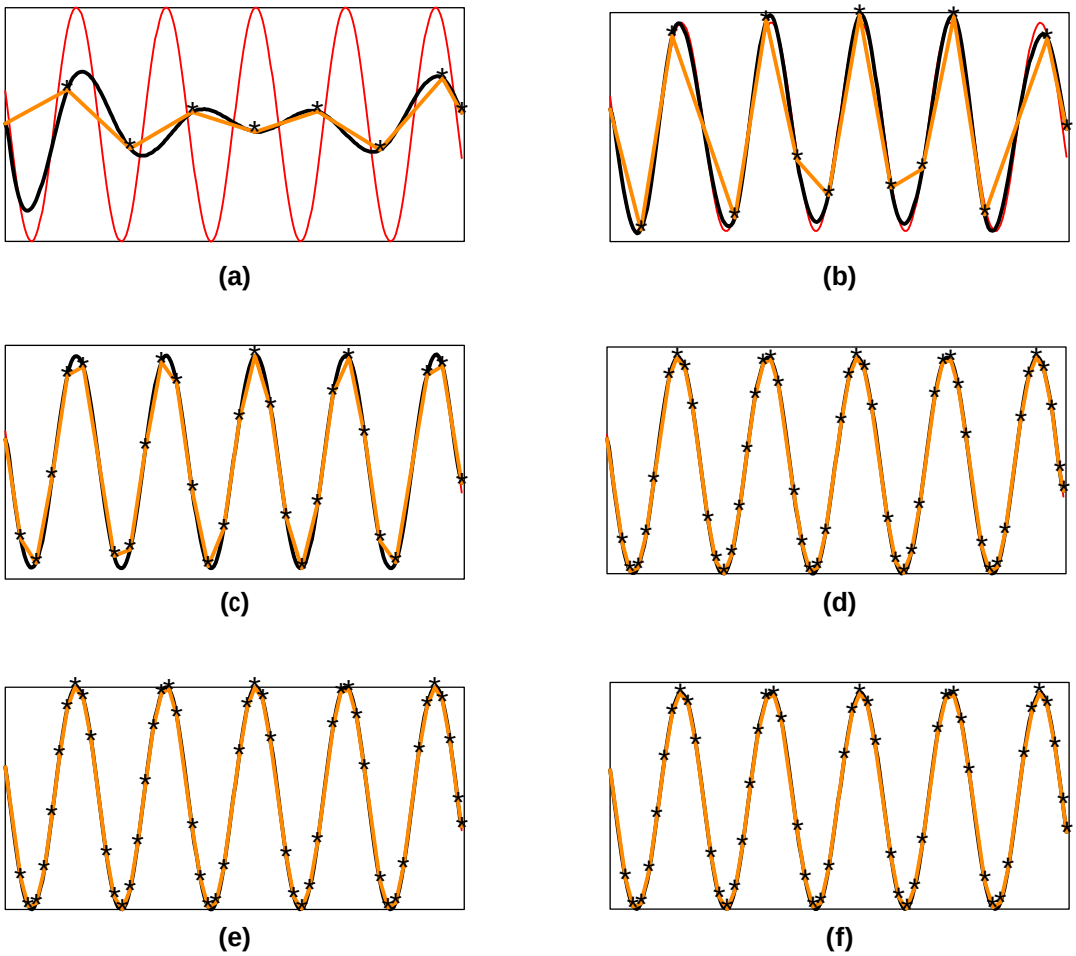


Figure 5.21 Progressive vertex insertion: Approximation of a sine wave which has been decomposed to level $M = 5$. **a:** 3.8% of the vertices at level 1 (scaling functions). **b:** 6.0% at level 2. **c:** 11.5% at level 3. **d:** 21.7% at level 4. **e:** 22.1% at level 5. **f:** 22.1% at level 6.

each level. Therefore, a single threshold ε_0 is weighted exponentially. For an M-level decomposition of the data, the threshold ε_m for level m is determined as follows:

$$\varepsilon_m = \varepsilon_0 a^{M-m+1} \tag{5.6}$$

with $a \geq 1$ and $m \in [1, M+1]$. The original threshold ε_0 is thus employed for the final level only. When a large basis factor a is used, the resulting approximation is equal to the non-progressive approximation. In this case, vertices are only inserted during the final level and all previous levels do not have any effect. See Table 5.5 for an example with varying a and fixed ε_0.

Distance metrics. The distance between a candidate vertex for insertion and the piecewise-linear approximation can be computed in different ways. Figure 5.23 illustrates the setting for three possible criteria in one dimension.

- *Criterion one* determines the closest distance d_1 between a vertex and current approximation:

$$d_1 = d_2 \cos\varphi, \tag{5.7}$$

where d_2 represents the vertical distance parallel to the y-axis in the plane.

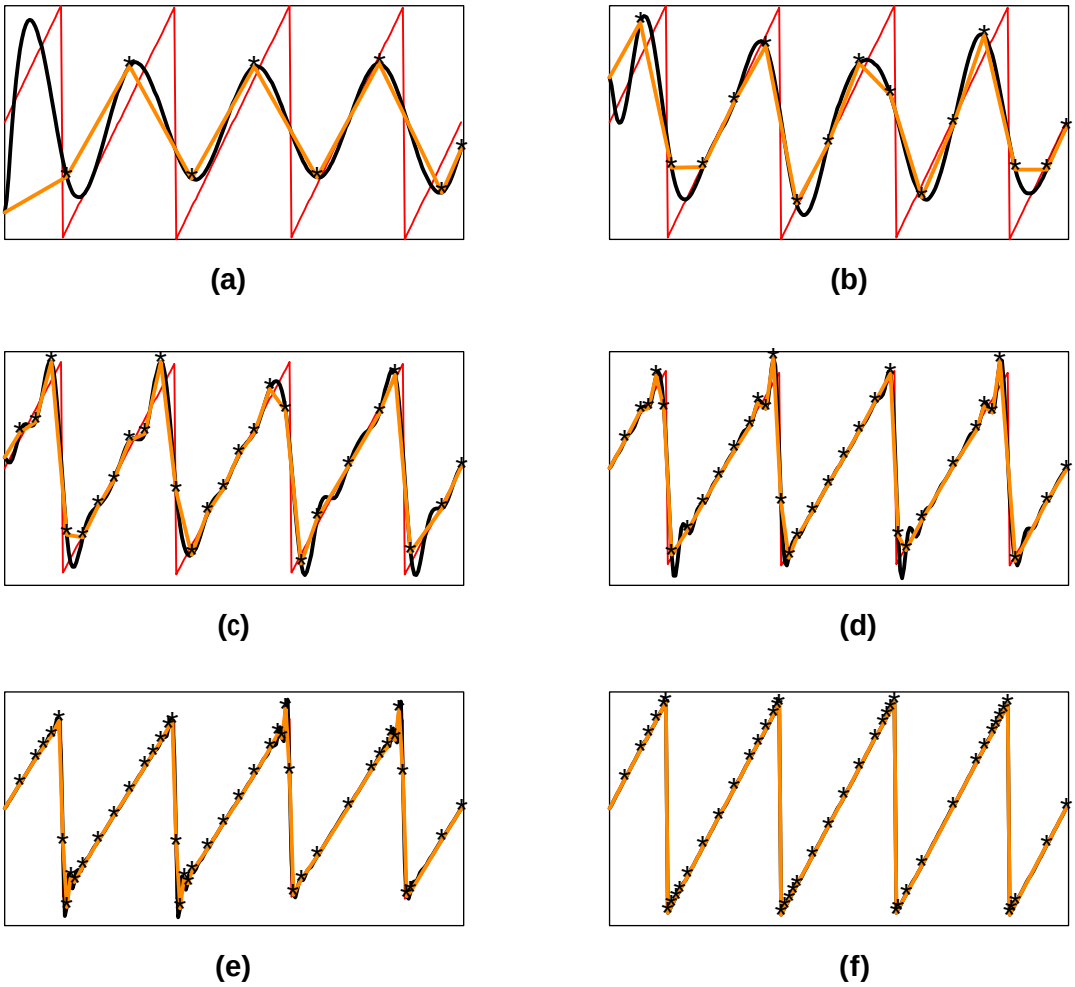


Figure 5.22 Progressive vertex insertion: Approximation of a sawtooth which has been decomposed to level $M = 5$. **a:** 3.8% of the vertices at level 1 (scaling functions). **b:** 6.8% at level 2. **c:** 11.5% at level 3. **d:** 14.9% at level 4. **e:** 20.4% at level 5. **f:** 22.1% at level 6.

Table 5.3 Percentage of vertices inserted into the mesh with varying a and fixed ε_0.

a	m = 1	m = 4	m = M =7
1	0.9%	6.3%	36.2%
1.4	0.9%	6.3%	36.1%
1.8	0.8%	5.9%	35.5%
2	0.8%	5.8%	35.3%
3	0.2%	3.8%	35.2%
4	0.2%	1.2%	29.5%
6	0.2%	0.2%	28.8%
10	0.2%	0.2%	28.8%
non-progressive			28.7%

- *Criterion two* uses d_2 only, which is usually larger that d_1.

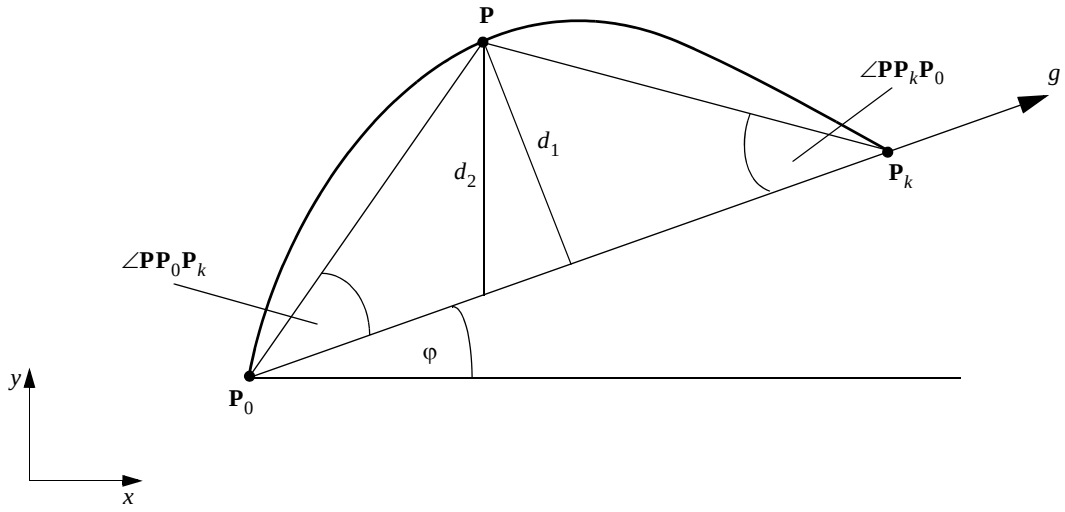


Figure 5.23 Illustration of three different distance criteria.

- *Criterion three* is based on the two angles $\angle \mathbf{PP}_0\mathbf{P}_k$ and $\angle \mathbf{PP}_k\mathbf{P}_0$, which ensures that the criterion is independent of the actual length of the line segment where the new vertex is to be inserted.

The two examples depicted in Figure 5.24 point out the limitations of the three criteria.

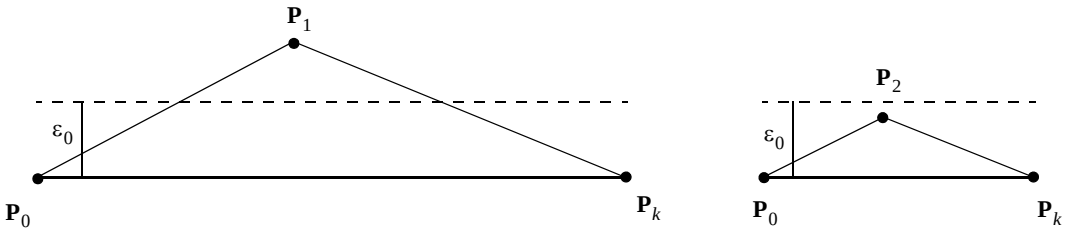


Figure 5.24 Limitations of the three distance criteria.

If $\mathbf{P}_0$ and $\mathbf{P}_k$ are further away from each other, the threshold for the third criterion needs to be larger, which is equivalent to the two angles being larger, than for the other criteria in order to allow insertion of $\mathbf{P}_1$. This is intuitively correct since for long segments, only distant vertices are serious candidates for insertion. If, however, $\mathbf{P}_0$ and $\mathbf{P}_k$ are closer together, criteria one and two neglect vertex $\mathbf{P}_2$ due to its smaller distance to the approximation. In other words, criteria one and two are not scale invariant. Figure 5.25 depicts a slice through a digital terrain model along with a comparison of all three criteria.

The two distance-based criteria clearly outperform the angle-based criterion. The last criterion chooses to many vertices close to the local extrema of the terrain. The other two criteria hardly differ at all in practice. Only for $\varphi > 45°$, the first criterion is noticeably better. On the other hand, evaluation of the first criterion is too expensive in higher-dimensional settings. For that reason, criterion two is a good choice. The problem of scale-dependence can be solved by scaling ε_0 according the extent of the data interval:

$$\bar{\varepsilon}_0 = \varepsilon_0(b-a), \tag{5.8}$$

when $[a, b]$ is the data interval.

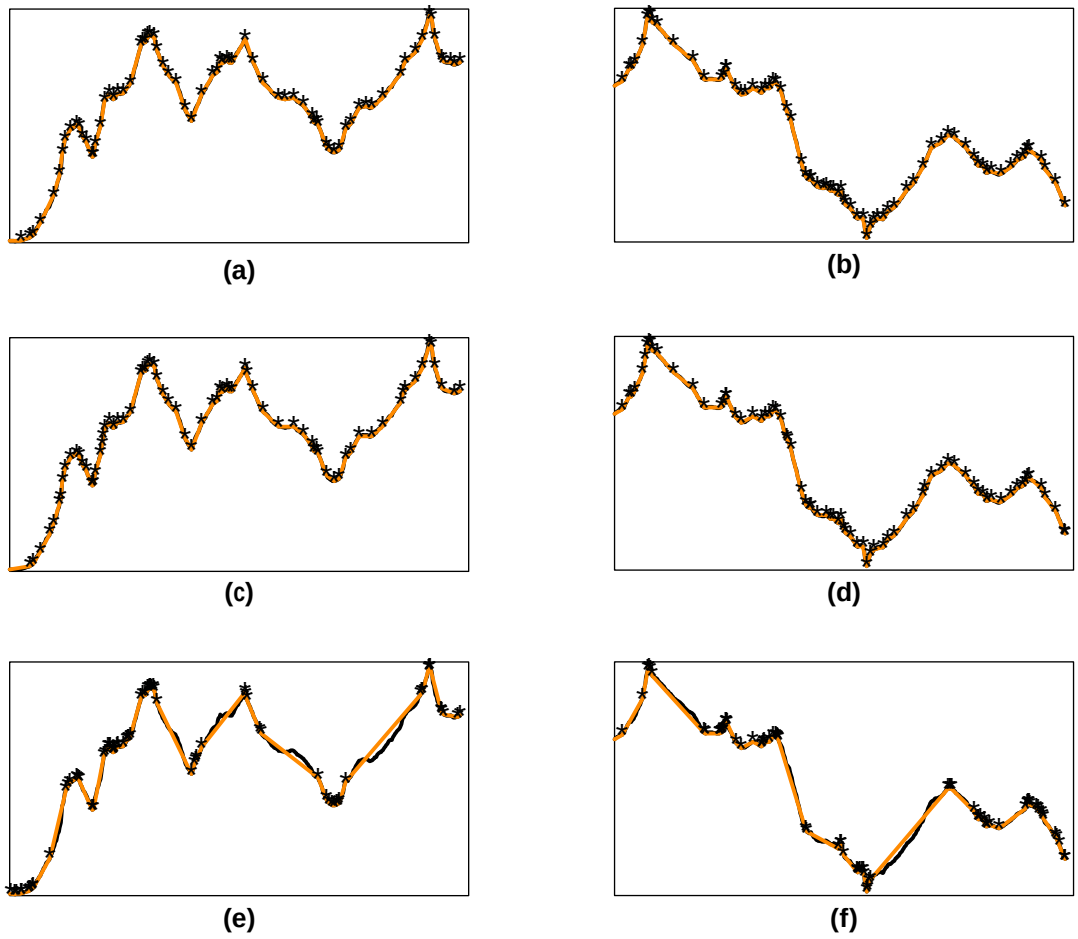


Figure 5.25 Comparison of the three distance criteria on two vertical slices through a digital terrain approximation comprising 153 out of 1024 possible vertices. **a:** Criterion one, RMS error 0.11. **b:** Criterion one, RMS error 0.08. **c:** Criterion two, RMS error 0.11. **d:** Criterion two, RMS error 0.08. **e:** Criterion three, RMS error 3.9. **f:** Criterion three, RMS error 2.3.

The approximation of parametric data sets will be discussed in Section 5.4.4.

5.4.3 Generalizations to Multiple Dimensions

Generalizations of the method towards multi-dimensional non-parametric data is straightforward. Starting from an initial grid, as depicted in Figure 5.26, the algorithm seeks the vertex P with the maximum distance and subdivides the field into four (in 2-D) or eight (in 3-D) subcells on which the method is applied recursively. In these cases, the distances to the bilinear and trilinear interpolants of the cell vertices are computed, respectively.

Recalling the multi-resolution B-spline approximation of the data motivates the extension of the algorithm towards channelwise progressive point insertion. Therefore, the algorithm analyzes mesh vertices progressively and labels unimportant points as new data comes in. In 2-D, for instance, the basic idea is to start from an initial vertex field of resolution 2^{m-M} in each direction, where M represents the maximum iteration. The vertices are provided by the scaling function approximation $f^M(x, y)$ and are processed further by our algorithm. To define a distance metric, we assume a bilinear interpolant between the vertices which approximates the B–spline scaling function representation. If the differ-

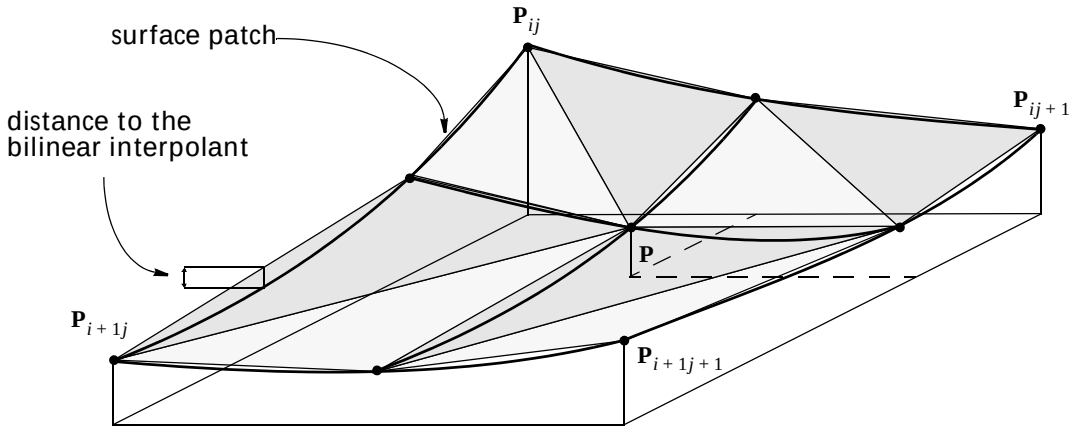


Figure 5.26 Extension towards multiple dimensions exemplified for non-parametric data: 2D version. The underlying B-spline patch is outlined in bold. A new vertex is inserted at position $\mathbf{P}$ and the distance is computed with respect to the bilinear interpolant of $\mathbf{P}_{ij}, \mathbf{P}_{ij+1}, \mathbf{P}_{i+1j}, \mathbf{P}_{i+1j+1}$.

ence signal $\Delta f^m(x, y)$ is received, the resolution is refined by two and all newly inserted vertices are checked in conformance with the distance metric. If required, they will be inserted.

In order to compute the intermediate vertices for each iteration, an inverse wavelet transform has to be applied to all coefficients of a given iteration level m as soon as they are received and decompressed.

Note that the N-tree type cell structure would enable computing very fast meshings using look-up tables, such as the ones presented in Section 5.3 [43]. An example of progressive point insertion is depicted in Figure 7.6, where the mesh is refined gradually with each wavelet subband arriving at the client side.

5.4.4 Parametric Data Sets

A parametric version of the introduced algorithm can be constructed as elucidated below. For reasons of simplicity, we restrict our description to 2-D parameter spaces, however higher dimensional spaces can be easily generalized from that. The conceptual components of our pipeline are illustrated in the diagram of Figure 5.27. We assume an initial parametric B–spline surface $\mathbf{s}(u, v)$ to be defined by its vector valued control points $\mathbf{c}_{ij}^0 = (c_{ij,x}^0, c_{ij,y}^0, c_{ij,z}^0)^{\mathrm{T}}$ at iteration $m = 0$.

$$\boldsymbol{s}(u, v) = \sum_{i=1}^{I} \sum_{j=1}^{J} \mathbf{c}_{ij}^0 \phi_j^0(v) \phi_i^0(u) \tag{5.9}$$

$J \cdot I$: number of tensor product B–spline scaling functions.

Thus, compression has to proceed separately on the x, y and z components of the control vertices. Specifically, the wavelet transform and the oracles operate for now independently on the individual coordinates.

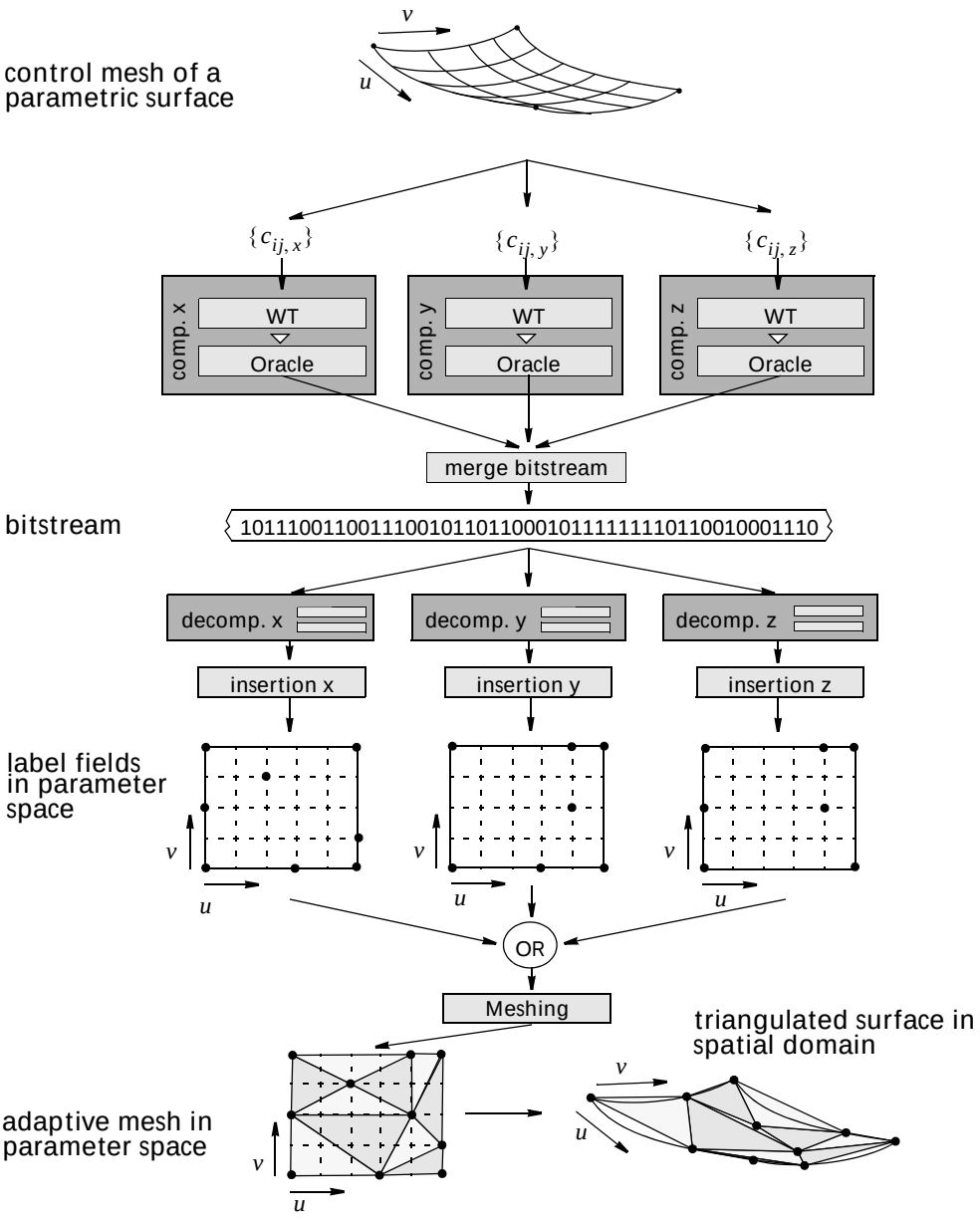


Figure 5.27 Illustration of the conceptual components for parametric compression and reconstruction.

However, things become more complicated upon reconstruction, which operates again in parameter space (u, v) independently for the spatial coordinates $s_x(u, v)$, $s_y(u, v)$, and $s_z(u, v)$. As a result three binary label fields are generated indicating the importance of individual vertices in parameter space for subsequent triangulation. Unfortunately, different results are obtained for $s_x(u, v)$, $s_y(u, v)$, and $s_z(u, v)$, and we have to decide on the final removal. This decision is accomplished by applying a Boolean OR operator over the individual vertex fields, a motivation of which is given as follows: As explained earlier, the non-parametric version of our removal strategy holds for linear approximations in terms of triangulations and thus refines the mesh in spatial regions, where the underlying function features nonlinear behavior. In the parametric setting similar criteria are valid for a linear approximation of a surface. The mesh has to be refined in those regions where the surface shows nonlinear behavior, that is where the local curvature is not equal to zero.

This, however, happens if either $s_x(u, v)$, $s_y(u, v)$ or $s_z(u, v)$ indicate nonlinearity. Obviously, the Boolean OR of the label fields considers a vertex important if one or more of the three coordinates behave locally nonlinear.

The usefulness of this approach can be seen in Figure 5.28, where a parametric surface is compressed and reconstructed with different parameter settings. Here, we end up with a dense mesh in spatial regions of high curvature and simplification occurs in regions of local planarity. Usually, thin triangles, which are also called slivers, are not desired in geometric applications. In the case of this chess figure, however, thin triangles provide for the best and efficient approximation of the middle part of the model.

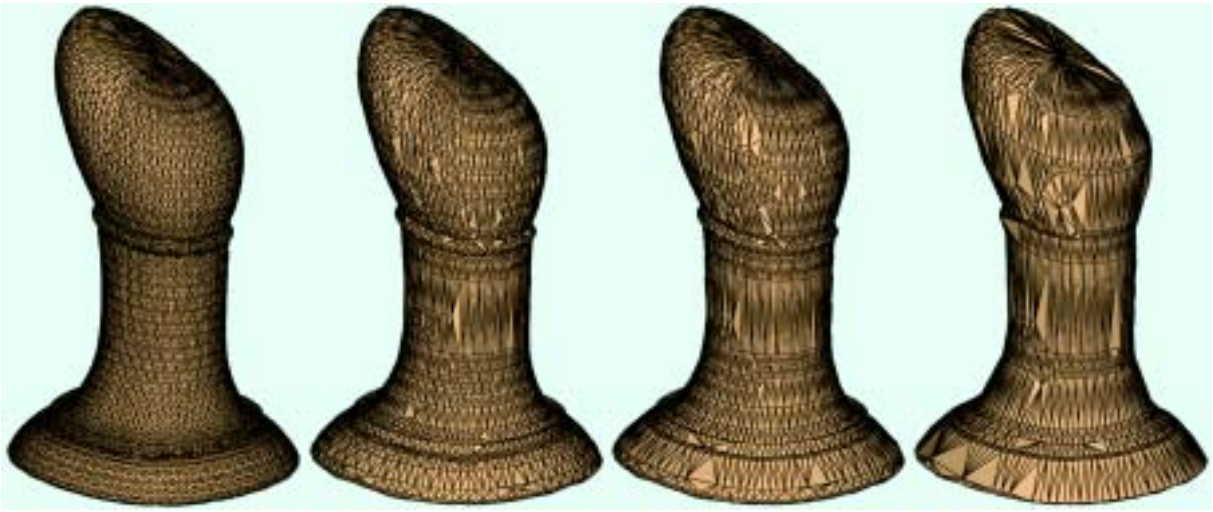


Figure 5.28 Compression and reconstruction of a parametric B–spline surface for different levels of linear approximation. **a:** $\varepsilon_0 = 0$, 19602 triangles (100%). **b:** $\varepsilon_0 = 0.005$, 43% triangles. **c:** $\varepsilon_0 = 0.01$, 34% triangles. **d:** $\varepsilon_0 = 0.02$, 22% triangles. [AVS©]

5.4.5 Texture Compression

In many applications in computer graphics, surfaces are attributed with additional color or texture information (see Section 3.2.2). It is advantageous to compress and transmit texture information along with geometric data. It is possible to extend the geometry compression pipeline to handle texture data. A texture is essentially a 2-D image with one or multiple channels. In most applications, color textures with red, green, and blue color channels are used. Sometimes, additional opacity information is stored in a separate alpha channel.

Color space. To yield better compression results, the image texture should be transformed into a color space other than RGB. In the RGB color space, information is spread over all channel. Other color spaces, such as YIQ and YUV—which are used in the analog PAL and NTSC television standards, respectively—separate luminancy and chrominancy of a pixel. Y is linked to the component of luminacy and U, V and I, Q are linked to the components of chrominancy.

Is is interesting that the entropy of the Y channel is much higher than that of the chrominancy channels. Thus, it is possible to yield better compression rates for the chrominancy.

The range of Y is limited to $[0, 1]$, whereas U, V, I, and Q can have negative as well as positive values and their range cannot be determined in general. The YC_BC_R color space defined in the CCIR Rec. 601-2 standard [83], which specifies the format for digital video images, does not share these disadvantages. Y remains the component of luminancy, but C_B and C_R become the respective components of blue and red. The green component can be determined relative to the other components.

For RGB value in the range of $[0, 255]$, the corresponding Y channel has a range of $[16, 235]$ and the C_B and C_R channels have both a range of $[16, 240]$.

Encoding from RGB into YC_BC_R involves the following transformation:

$$\begin{bmatrix} Y_{602} \\ C_B \\ C_R \end{bmatrix} = \begin{bmatrix} 16 \\ 128 \\ 128 \end{bmatrix} + \frac{1}{256} \begin{bmatrix} 65.738 & 129.057 & 25.064 \\ -37.945 & -74.494 & 112.439 \\ 112.439 & -94.154 & -18.285 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}. \tag{5.10}$$

To decode YC_BC_R back to RGB components in the range $[0, 255]$, the following transformation can be applied:

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \frac{1}{256} \begin{bmatrix} 298.082 & 0.0 & 408.583 \\ 298.082 & -100.291 & -208.120 \\ 298.082 & 516.411 & 0.0 \end{bmatrix} \left(\begin{bmatrix} Y_{601} \\ C_B \\ C_R \end{bmatrix} - \begin{bmatrix} 16 \\ 128 \\ 128 \end{bmatrix} \right) \tag{5.11}$$

Compression pipeline. The pipeline for image texture compression is depicted in Figure 5.29. After transforming the RGB image into YC_BC_R color space, the channels are compressed individually and merged into a single bitstream. After decompression, the approximated data is transformed back into RGB space.

Note that the codec used for image compression is essentially the same as it the one used for geometry compression. Since the codec is optimized for geometry compression, its performance for image compression cannot compete with dedicated image compression codecs. This is due to the fact that it operates on 32 bit floating point values instead of byte values. Hence, eight bit color channels of image textures are converted to 32 bit floats during the process of compression. Taking this into account, the wavelet-based codec performs remarkably well and demonstrates the versatility of the approach. Methods specifically designed for image compression, however, will provide for better performace [91].

Experimental results of compression and progressive reconstruction of a textured digital terrain model are given in Chapter 7.

5.5 ISO-CONTOURING

In the following section the problem of implicit reconstruction is addressed. In practice, implicit structures are mostly iso-lines or iso-surfaces. An advantageous feature coming along with the multi-resolution B–spline representation is the higher order continuous approximation of the underlying data. Although any computation bases on adaptive triangulations obtained from previous procedures, this property can be exploited to reconstruct implicit structures more precisely. For instance, in 2-D data sets, piecewise linear representations of iso-lines can be recovered by immediate computation of the intersec-

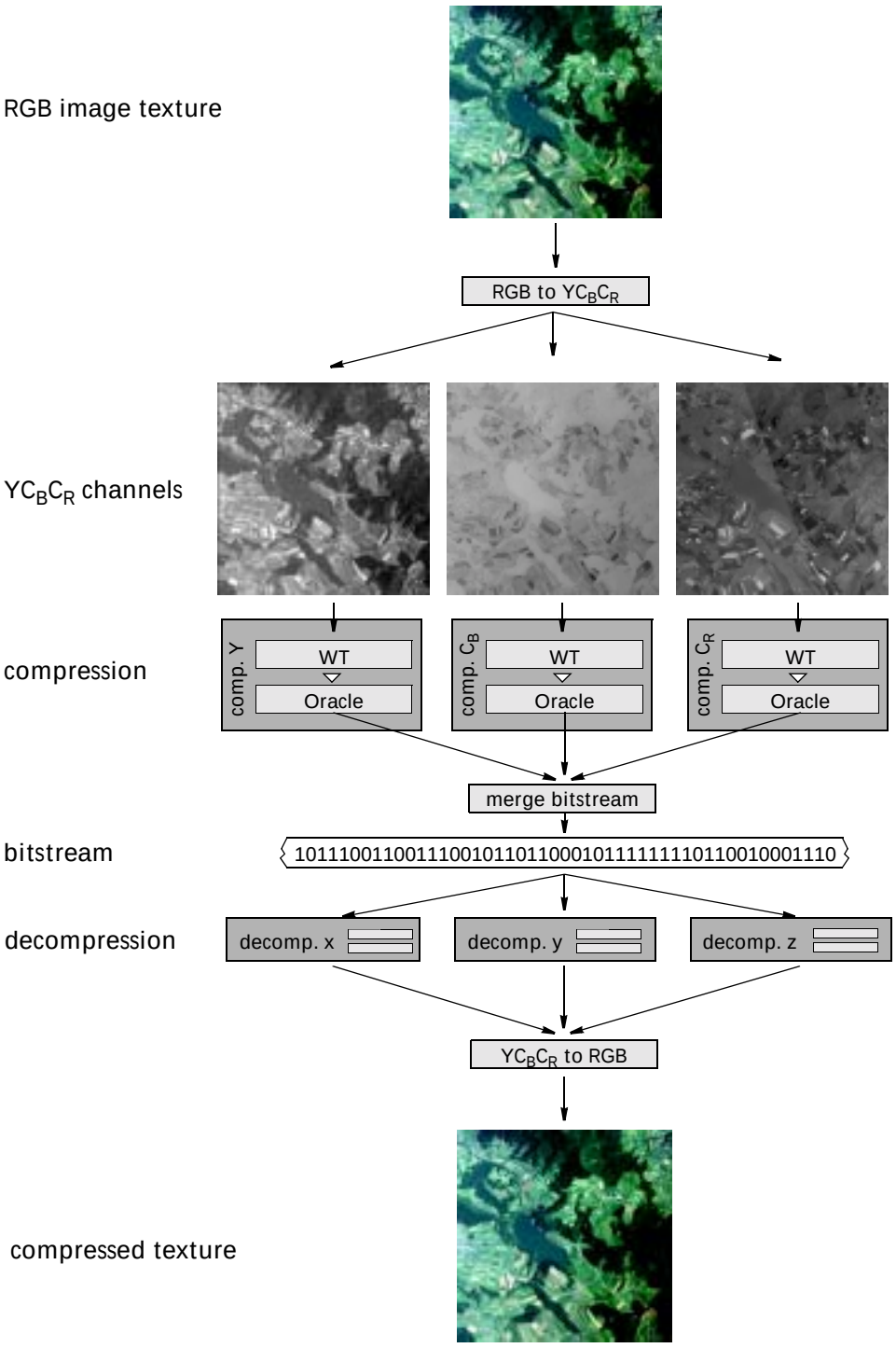


Figure 5.29 Illustration of the conceptual components for the color image texture compression pipeline. [Swissphoto©]

tions along the triangle edges from the B–spline approximation. Similar procedures hold for iso-surface reconstructions from tetrahedralizations. The cubic polynomials perform data smoothing and cancel out most of the jags and discontinuities commonplace in standard methods.

5.5.1 Iso-lines

In order to handle iso-lines we start from the initial B–spline description of the underlying 2D height function $f(x, y)$ and obtain an implicit formulation by

$$f(x, y) = \sum_{i=1}^{I} \sum_{j=1}^{J} c_{ij}^{0} \phi_j^0(y) \phi_i^0(x) = \tau \tag{5.12}$$

τ : iso-value.

Recalling the approximation properties of B–splines we recommend to precompute an interpolation problem to get the appropriate coefficients c_{ij}^{0} related to the data samples to be interpolated. These types of interpolations are extensively investigated and relate tightly to inverse B–spline filtering problems which perform in linear time [106].

For Equation 5.12 we provide a polyline approximation using a marching triangle-like look-up table which operates on a triangle mesh representing $f(x, y)$. A slight extension of the look-up table enables one to extract those parts of the surface which are interior to the iso-line, that is whose function values $f(x, y) > \tau$. The vertices of the describing polygonal hull are given by the intersections of the iso-line with triangle edges, such as shown in Figure 5.30a.

Note that in cases 011, 101, and 110 the initial triangle representing the surface is split into two primitives. The intersection of the iso-line, implicitly defined by Equation 5.12, with the triangle edge is calculated using a binary search along the edge. Here we exploit the regional separation with respect to τ provided by the iso-line.

Figure 5.31 illustrates the performance of the method on digital terrain data. We employed progressive triangular approximations of different quality to compute both iso-lines and to extract interior regions. Comparing the iso-lines depicted in Figure 5.31a that are computed by our method with those of a linear interpolation shown in Figure 5.31b reveals the superiority of the approach. Figure 5.31c depicts the same data set at a higher approximation level.

5.5.2 Iso-surfaces

Similar relations hold for the generation of iso-surfaces and interior or exterior volumes enclosed by iso-surface. Here we start from a B-Spline volume approximation of $f(x, y, z)$:

$$f(x, y, z) = \sum_{i=1}^{I} \sum_{j=1}^{J} \sum_{h=1}^{H} c_{ijh}^{0} \phi_h^0(z) \phi_j^0(y) \phi_i^0(x) = \tau \tag{5.13}$$

$H \cdot J \cdot I$: number of tensor product volume B–splines.

After solving the initial B–spline interpolation problem the iso-surface is obtained by a marching tetrahedron [7] algorithm, where the intersections of the surface with the tetrahedral edges are computed using the binary search on the B–spline volume. Again, a little work on the look-up table enables one to extract interior or exterior volume segments which are important for many applications, such as finite element simulations [60]. Figure 5.30b exploits symmetry and illustrates the 5 out of 16 cases arising upon triangulation giving the connectivity for the extraction of interior volumes.

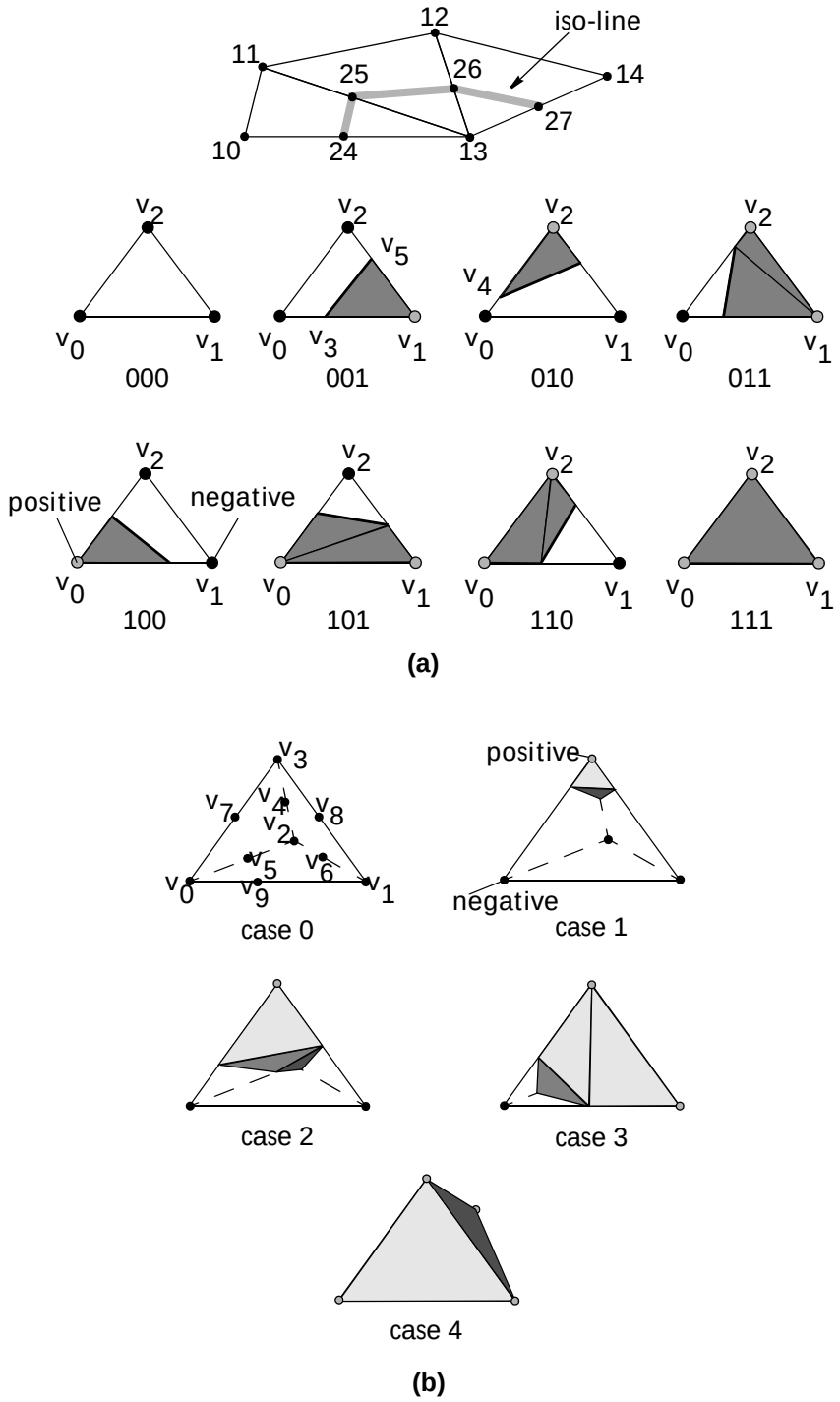


Figure 5.30 Polyline approximation of iso-lines: **a:** Iso-line as computed by intersections with the triangle edges and look-up table to extract interior or exterior parts of the surface. **b:** Generation of iso-surfaces and interior volumes using a marching tetrahedron algorithm: 5 basis cases arising upon triangulation. The connectivity table for generation of interior volumes is presented in Table 5.4.

Note especially that individual tetrahedra may split up into three primitives for representing the bounding surface of the interior volume.

The results given in Figure 7.11 illustrate the approximation behavior of the method, where standard marching cubes and marching tetrahedron with linear interpolation are

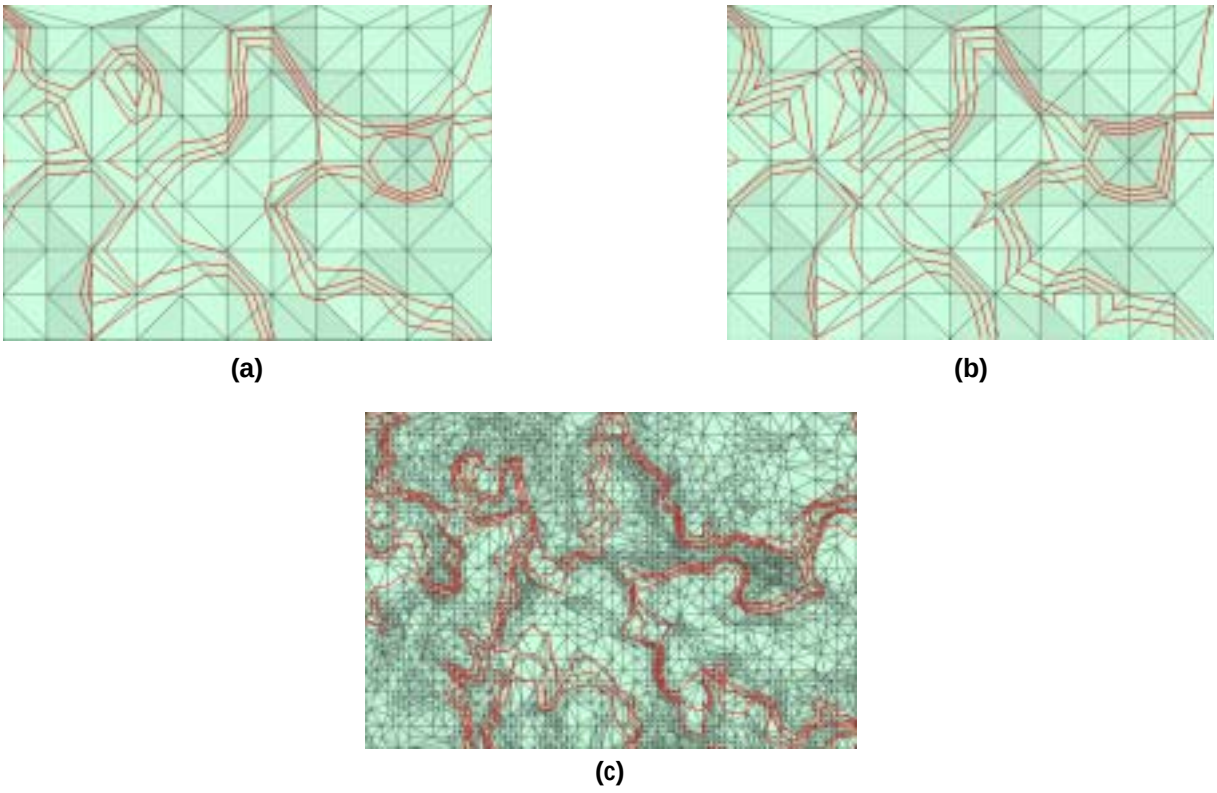


Figure 5.31 Extraction of iso-lines. **a:** Cubic method. **b:** Linear method. **c:** Cubic method at higher reconstruction level. [DHM©]

Table 5.4 Connectivity table for generation of interior volumes (see Figure 5.30b).

No.	v_0	v_1	v_2	v_3	Tetrahedra (vertex list)
0	-	-	-	-	{}
1	-	-	-	+	{ {v_4, v_7, v_8, v_3} }
2	-	-	+	+	{ {v_8, v_5, v_6, v_2}, {v_5, v_8, v_7, v_3}, {v_8, v_5, v_2, v_3} }
3	-	+	+	+	{ {v_7, v_5, v_9, v_2}, {v_3, v_1, v_2, v_7}, {v_7, v_2, v_9, v_1} }
4	+	+	+	+	{ {v_0, v_1, v_2, v_3} }

contrasted to the B-spline binary search algorithm. Although intersection computation is more expensive we observe that the resulting iso-surface is figured out more precisely and smoothly using the new approach.

5.6 IMPLEMENTATION

The core components of the wavelet-based surface and volume compression pipeline described in this chapter, require careful and efficient implementation to achieve adequate performance. This section discusses some important implementation issues as well as algorithmic complexity of the compression and both meshing strategies which have been introduced above. For implementation details of the fast wavelet transform, the reader is referred to Section 2.5.

5.6.1 Compression

One of the main stages of the compression pipeline—the fast wavelet transform—has already been discussed in detail in the previous section. The remaining stages are all in $O(N)$ with one exception: The complexity of the global oracle for bi-orthogonal wavelets is $O(N^2)$. The complexity of the reconstruction is $O(N)$ as well.

Therefore, the wavelet-based compression pipeline introduced in this chapter outperforms other compression methods, including those based on fast Fourier transforms or fast cosine transforms. Thus, it is very well fit for compression of large data sets. The implementation of the compression pipeline is described in detail in [113]

5.6.2 Meshing

The two different meshing schemes which have been described in Section 5.3 and Section 5.4 result in different implementation strategies and data structures.

Quadtree-based vertex removal. One of the very advantages of this method is the low algorithmic complexity for the quadtree meshing. Although the initial inverse wavelet transform to compute the detail signal (see Section 5.3.1) has to be modified, the complexity still remains $O(N)$.

Some investigations can be carried out for the complexity of building up the quadtree: If, in the worst case, the traversal is done up to the maximum depth of the wavelet transform, M, at worst four *energy tests* for the wavelet criterion, four *resolution tests,* and 16 tests for the *m to m-2 criterion* have to be performed. Due to the dyadic structure of the vertices to be analyzed, we end up in the two-dimensional setting with

$$\begin{aligned} C^A &= 24 \sum_{m=1}^{M} 2^{2I-2m} \\ &= 2^{2I}\left(8 - \frac{8}{4^M}\right) \\ &= O(N^2) \end{aligned} \tag{5.14}$$

to test the importance of vertices, which is still linear with respect to the overall number of mesh vertices $N^2 = 2^{2I}$.

The traversal of the array is carried out by the recursive procedure of depth M given in the pseudocode in Section 5.3.2. In the worst case of none of the vertices being labeled as unimportant, its complexity is quantified by

$$\begin{aligned} C^B &= S^{I-M} \sum_{m=0}^{M-1} 4^m \\ &= \frac{2^{I+M} - 2^{I-M}}{3} \\ &= O(2^{2I}) \\ &= O(N^2) \end{aligned} \tag{5.15}$$

It has to be noted, however, that the worst case analysis provides only a theoretical upper bound. In general, vertices labeled as unimportant reduce the computational costs dramatically. A single vertex removal in recursion depth m cuts a whole subtree and saves

$$B = \sum_{k=m}^{M} 4^{k-m} = \frac{4^{M-k+1} - 4}{3} \tag{5.16}$$

tests.

Additional details on the implementation of this scheme is given in [36].

Delaunay-based vertex insertion. In order to support progressive vertex insertion in a top-down fashion, we need a data structure which records results (i.e., previously inserted vertices) from precedent iteration steps.

The implementation employs a tree type data structure to maintain the individual cells representing the mesh. The tree grows iteratively as progression proceeds. This representation, however, contains redundant information since only active segments, which represent all vertices of the current level, are required. Thereupon, it would be practical to only store the active segments in a linked list, evading the need to store redundant vertices. After each iteration the entries of the list represent the remaining cells, which can be triangulated with appropriate methods. Figure 5.32 further elucidates the data representation.

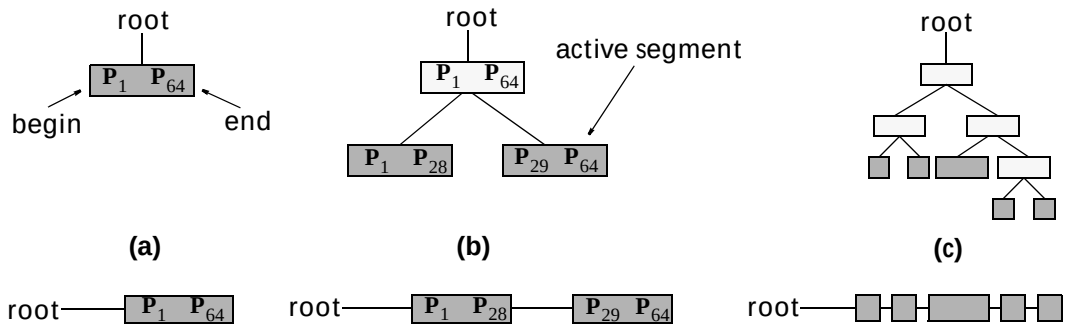


Figure 5.32 Construction of a 1-D tree data structure with 64 vertices and its growth during progression. The equivalent list structure is given below. **a:** First segment at the beginning. **b:** Insertion of $\mathbf{P}_{29}$ causes a split into two segments. **c:** Final tree after inserting all vertices.

Unfortunately, a simple linear list is not sufficient to represent the data. This applies particularly to the two- and three-dimensional case, where it is not guaranteed that the corners of a cell have already been inserted. Furthermore, it is difficult to access the set of vertices which results from the single iteration steps. It is usually the case that those vertices inserted in precedent levels remain in the set of vertices of later levels. In other words, the number of vertices is strictly increasing. A possible solution for both problems is to keep a list of vertices, ordered according to the iteration level at which they have been inserted. After each iteration step, the current number of vertices is recorded, which allows one to determine the exact number of vertices inserted at any given level. Note that the total number of vertices in the data set is known in advance. Thus, it is advantageous to use an array instead of a linked list. It is very handy to utilize an additional array of Booleans recording whether a vertex has already been inserted. An appropriate data structure is as follows:

```
class Node {
  int start;
  int len[3];
  Node* next;
};

Node root;        // first element
float inValues;   // array of vertices of the uniform input data
int nn;           // total number of inserted vertices
int vertices[];   // ordered list of inserted vertices
bool inList[];    // boolean flag for inserted vertices
int nnodes[];     // # intermediate vertices at each level
// ...
```

The size of the arrays is determined by the number of input values.

The complexity of this method is in the worst case $O(N\log N)$: We start with a single segment containing N vertices where all vertices have to be inspected. This segment is then split into two times the dimension segments, where each vertex is inspected a second time. This process is repeated recursively until all segments do only contain two vertices. In other words, to insert all N vertices, we need $\log N$ recursion steps where all vertices are inspected. If only k vertices are inserted, the complexity reduces to $O(N\log k)$, since we only need k segments in the list.

In the case of progressive vertex insertion, not all vertices have already been transmitted and decoded. With a maximum decomposition level M, we end up with $M+1$ iterations. The number of vertices of each step is given for different dimensions in Table 5.5.

Table 5.5 Number of vertices a each level of progression.

Level	1-D	2-D	3-D
1	$2^{-M}N$	$4^{-M}N$	$8^{-M}N$
2	$2^{-M+1}N$	$4^{-M+1}N$	$8^{-M+1}N$
m	$2^{-M+m-1}N$	4^{-M+m-1}	8^{-M+m-1}
$M+1$	N	N	N

At each progression level $m>1$, the number of vertices can only double (one-dimensional setting). Hence, only one step with N inspections is required and for $m>1$, the complexity is $O(N)$. Since the complexity for $m=1$ is $O(N\log N)$, the total complexity if the progressive vertex insertion scheme is $O(N\log N)$. See [113] for more details on the implementation.

The vertices are subsequently triangulated using Delaunay triangulation (see Section 3.3.1). The lower bound of the complexity of the Delaunay triangulation is $O(N\log N)$ [81]. For the two-dimensional setting, Shewchuk's *Triangle* library is employed [92]. For higher-dimensional Delaunay triangulation, the *qhull* library by Barber et al. [4], which does not perform as well as *Triangle* in 2-D, can be used. For an in-depth analysis of these algorithms, the reader is referred to the corresponding references.

6

PROGRESSIVE TETRAHEDRALIZATIONS

The wavelet-based vertex removal and insertion schemes introduced in the previous chapter are very well suited for efficient representations of various kinds of uniform data in multiple dimensions. An increasing demand for purely irregular and unstructured mesh representations, however, shows the requirement for the development of new methods providing even greater flexibility.

Progressive meshes [52] and its generalizations to higher dimensions [80] proofed to be an extremely powerful notion for the efficient representation of triangulated geometric objects at different levels-of-detail. Although a general formulation for arbitrary triangulations has already been given in [80], the special case of *progressive tetrahedralizations* (PT) is of enormous practical importance, since it can be used as a sophisticated representation for a large variety of computations. Finite element discretizations, from where our contribution was motivated, are one example. Here, sophisticated computational methods try to find an optimal balance between refinements of the mesh and of the polynomial degree of the basis functions. Other important applications of progressive tetrahedralizations comprise interpolation and rendering of scattered volume data, where successively refinable methods would definitely improve the performance of existing approaches. A general overview of various mesh simplification methods, including those based on edge collapses, can be found in [49].

However, regardless of the brilliance and simplicity of the idea of edge collapsing, a brute force implementation of the method may rapidly destroy the consistency of the mesh. Various artifacts can be introduced, such as flipped, intersected and degenerated tetrahedra, which in turn may impede any visualization method. An example is given in Figure 6.1. Here two boundary tetrahedra intersect due to an edge collapse in a locally concave mesh region.

This chapter presents a novel method for the hierarchical and progressive representation of tetrahedral meshes with arbitrary connectivity and we elaborate on some pitfalls

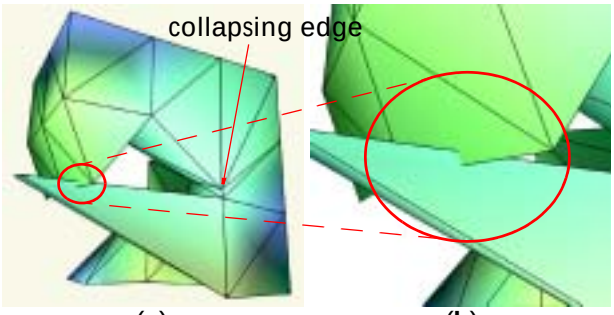


Figure 6.1 Intersection of two tetrahedra: **a:** The edges of two non-adjacent tetrahedra bounding the volume intersect while collapsing an edge in a locally concave mesh region. **b:** Close-up of the intersection.

and fallacies people might get caught in when trying to implement the method of edge collapsing for tetrahedral meshes. Specifically, we address the issue of defining appropriate cost functions. Unlike the elegant geometric approach presented in [52], we must account for volume and application specific properties, such as volume preservation, gradient estimation of the underlying data or aspect ratio of the simplex. In addition, we devised a sequence of tests to ensure a robust and consistent progressive tetrahedralization, which have first been introduced in [94]. Dey and colleagues [28] have independently presented an in-depth analysis of topology preserving edge collapses.

6.1 PRINCIPLES

In a progressive mesh representation, for both triangular and tetrahedral meshes, a mesh with scalar attributes s_i assigned to each vertex $\mathbf{v}_i$, is defined as

$$(M^0, \text{vsplit}_0, \text{vsplit}_1, \ldots, \text{vsplit}_{n-2}, \text{vsplit}_{n-1}) \tag{6.1}$$

where M^0 is some coarse base mesh and vsplit_i are vertex split operations which reconstruct the original mesh $M = M^n$ from M^0:

$$M^0 \xrightarrow{\text{vsplit}_0} M^1 \xrightarrow{\text{vsplit}_1} \ldots \xrightarrow{\text{vsplit}_{n-1}} M^n . \tag{6.2}$$

Conversely, M^0 is derived from M through a series of edge collapse operations ecol_i which are inverse to vsplit_i:

$$M^n \xrightarrow{\text{ecol}_{n-1}} M^{n-1} \xrightarrow{\text{ecol}_{n-2}} \ldots \xrightarrow{\text{ecol}_0} M^0 . \tag{6.3}$$

Each ecol_i replaces an edge e_i with endpoints $\mathbf{v}_a$ and $\mathbf{v}_b$ with a new vertex $\bar{\mathbf{v}}_a$. As opposed to some other methods [89], the topology of the mesh is preserved, that is, all instances of M are homeomorphic. In the tetrahedral setting, degenerations of tetrahedra into lower dimensional simplices is prohibited. The set of cells sharing e_i will be called $\{icells_i\}$. Thus, an edge split adds the tetrahedra in $\{icells_i\}$ to the list of active elements. Conversely, the set of non-vanishing cells affected by the associated edge collapse is called $\{ncells_i\}$.

For the triangular setting, edge collapse and vertex split operations are illustrated in Figure 6.2. Note that, except for the mesh boundary, a single edge collapse operation

ecol_i removes always two triangles from the mesh, in other words, $\{icells_i\}$ comprises two cells.

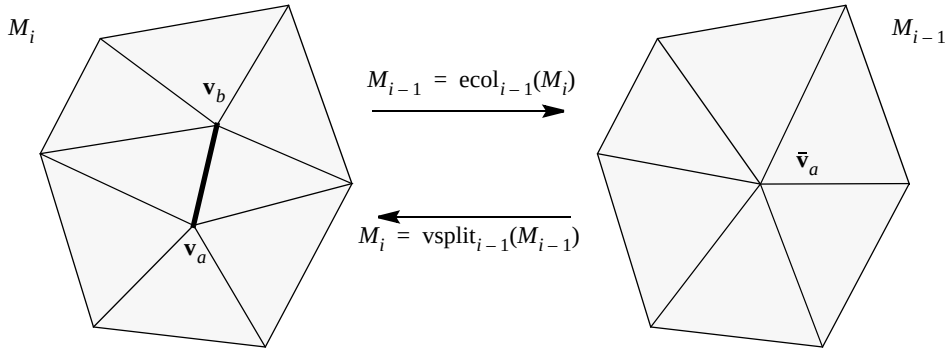


Figure 6.2 Edge collapse and vertex split in a triangular mesh. Edge $(\mathbf{v}_a, \mathbf{v}_b)$ is collapsed into vertex $\mathbf{v}_a$.

Figure 6.3 depicts an edge collapse operation in a tetrahedral mesh. All tetrahedra sharing e_i vanish, whereas all tetrahedra sharing only one of the vertices of the edge change in shape. Unlike as for triangular edge collapses, the number of cells in $\{icells_i\}$ can be arbitrarily large. This indicates that a single edge collapse in a tetrahedral mesh can influence a larger part of the mesh as it is the case for triangular meshes.

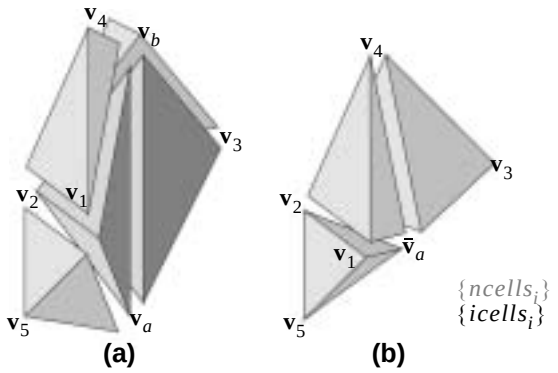


Figure 6.3 Edge collapse in a tetrahedral mesh: **a:** Mesh before collapsing edge $(\mathbf{v}_a, \mathbf{v}_b)$. **b:** Configuration after collapse with resulting vertex $\bar{\mathbf{v}}_a$. (Note: the tetrahedra are shrunk to emphasize the underlying three-dimensional structure).

The fundamentals of these two operations on meshes is based on the concept of collapsing and shelling, which has been introduced in Section 3.1.4.

We can make the important observation that the coarse base mesh M^0 and the sequence of vertex split operations $\{\text{vsplit}_i\}$ provides all necessary information to reconstruct the original mesh M^n losslessly, as well as any intermediate approximation M^i.

Given these basic operations, we can devise an efficient and elegant algorithm for representing meshes with arbitrary connectivity.

6.2 BASIC ALGORITHM

In order to derive a progressive mesh representation $(M^0, \{\text{vsplit}_i\})$ for an arbitrary input mesh M^n, we need to determine a good sequence of edge collapse operations $\{\text{ecol}_i\}$ which transform M^n to M^0. A greedy, yet powerful strategy is to assign a specific value to each edge which reflects the "cost" when this edge is collapsed. Hence, important edges are assigned high costs. The cost value for each edge is determined by special cost functions, which are explained in detail in Section 6.3.

In a preprocessing step, all edges in the mesh are sorted into a priority heap data structure *(edgeheap)* according to their associated cost value, where the least "expensive" edge is placed on top. Figure 6.4 sketches the basic algorithm for performing a sequence of edge collapse operations.

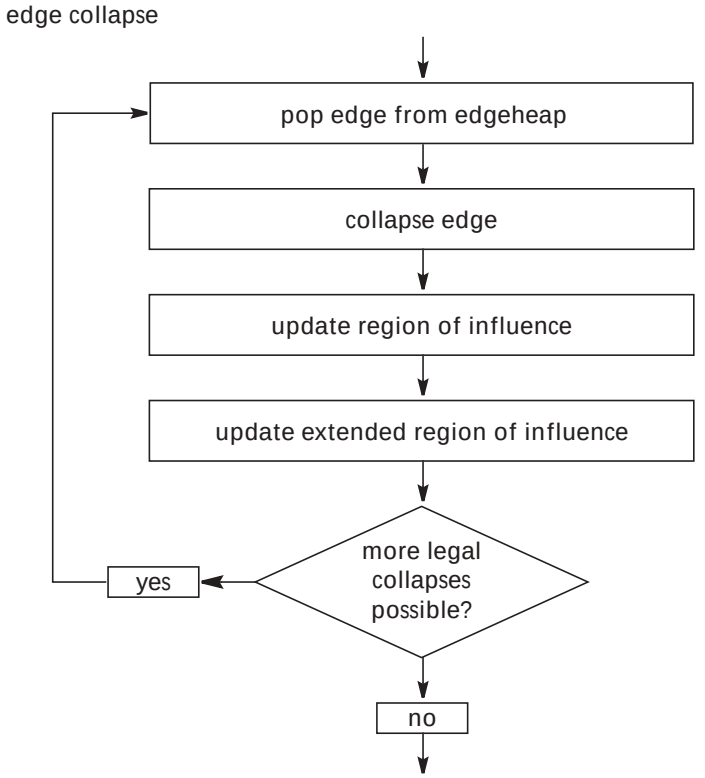


Figure 6.4 Sequence of basic edge collapse operations.

The topmost edge (i.e., the edge with the smallest cost) is removed from the edgeheap. Before we can perform the edge collapse, however, we have to ensure that this operation does not destroy the consistency of mesh by introducing non-manifold vertices or faces due to degenerate cells or cell intersections. Special functions which determine whether an edge collapse operation is legal are introduced in Section 6.4.

After a successful edge collapse ecol_i, we have to make some adjustments in $\{ncells_i\}$. Figure 6.5 illustrates all cells that are directly influenced by ecol_i.

Any edges which constitute $\text{star}(\mathbf{v}_a) \cup \text{star}(\mathbf{v}_b)$ may change their length, which might directly influence on their associated cost values. Therefore, the cost of these edges has to be recomputed and the edgeheap has to be updated. Depending on the specific definition of the cost functions, the edge cost might not only be influenced by vertex and edge prop-

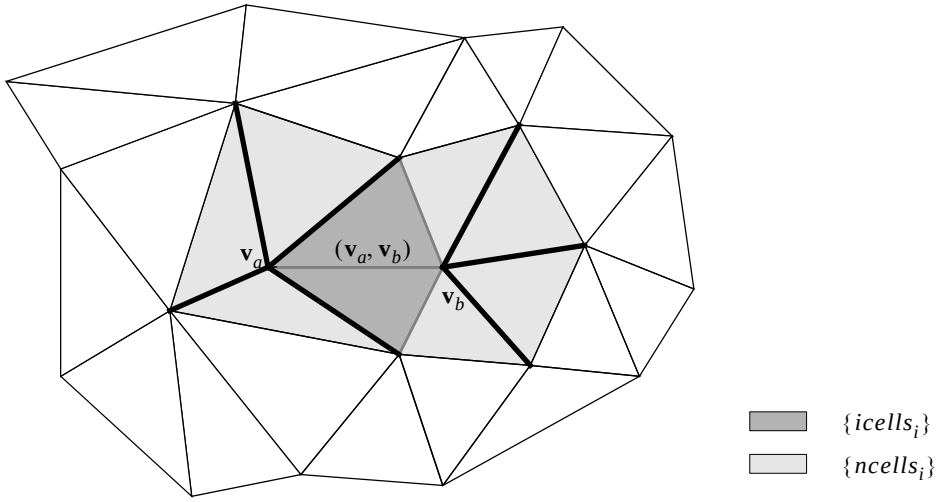


Figure 6.5 Region of influence for ecol_i collapsing $(\mathbf{v}_a, \mathbf{v}_b)$.

erties, but also from cell properties of all cells sharing this edge. This defines an extended region of influence of an edge as depicted in Figure 6.6.

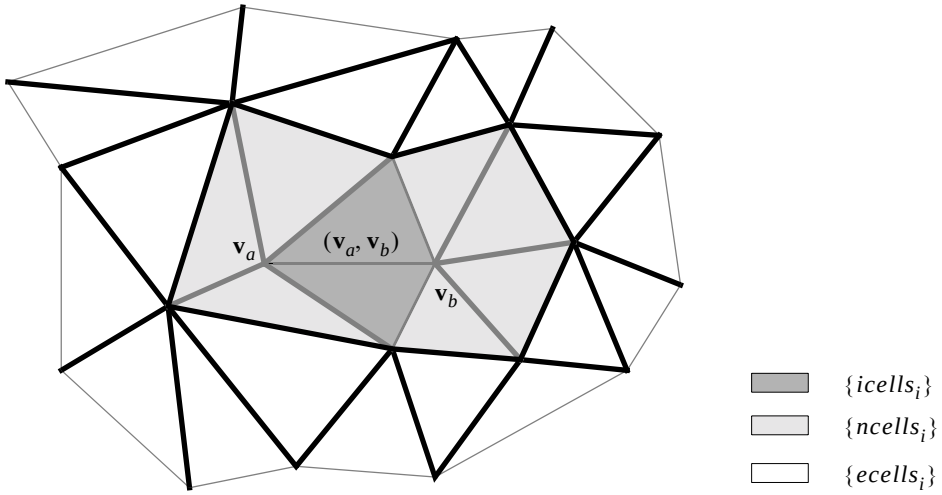


Figure 6.6 Extended region of influence of edge $(\mathbf{v}_a, \mathbf{v}_b)$.

The extended region $\{ecells_i\}$ comprises the union of the star()s of all vertices on the boundary of $\{ncells_i\}$. In summary, the following adjustments have to be made after each edge collapse:

1. $\text{star}(\mathbf{v}_a) \cup \text{star}(\mathbf{v}_b)$: Update of edge lengths and cost values.
2. $\{ncells_i\}$: Update of cell properties.
3. $\{ecells_i\}$: Update of cost values.

In order to compute a sequence of robust, non-degenerate and consistent meshes, the following aspects have to be considered:

- *Cost functions* which determine the order of ecol operations depending on desired mesh optimization criteria.

- *Boundary and feature edges,* which should be preserved, can be checked during a preprocessing step.
- *Intersections and inversions* of tetrahedra inside and outside of

$$\{\{icells_i\} \cup \{ncells_i\}\},$$

such as the one in Figure 6.1, have to be processed at run time. The remainder of this chapter elaborates on the details of these issues.

6.3 COST FUNCTIONS

Various elegant algorithms [35, 52] based on the ecol/vsplit paradigm use cost functions optimized for triangular surfaces, often accounting for distance measures, triangle shape, and others. In tetrahedral meshes, however, we have to redefine the terms of the cost function considering other features, like volume preservation or gradients. Although many different measures are conceivable to control the simplification process, the following ones yield a good balance between required degrees of freedom and the difficulty of parameter optimization.

Thus, in our setting, for each edge $e_i = (\mathbf{v}_a, \mathbf{v}_b)$, the associated edge collapse operation $\text{ecol}_i(a, b)$: $M^i \leftarrow M^{i+1}$ is assigned the following cost:

$$\Delta E(e_i) = \Delta E_{\text{grad}}(e_i) + \Delta E_{\text{vol}}(e_i) + \Delta E_{\text{equi}}(e_i) + \Delta E_{\text{edgelen}}(e_i) + \Delta E_{\text{normal}}(e_i). \quad (6.4)$$

6.3.1 Gradient energy

The first term ΔE_{grad} is defined as

$$\Delta E_{\text{grad}}(e_i) = w_{\text{grad}} \cdot |s_a - s_b| \quad (6.5)$$

and forms a simplified measure for the difference of underlying scalar volume function along the edge e_i. Hence, edges with considerably differing scalar attributes are assigned high costs. This term of the cost function applies only to volumetric data sets with scalar attributes.

6.3.2 Volume energy

When collapsing edges and removing tetrahedra from the mesh, the overall volume tends to decrease, that is the mesh is shrinking. Therefore, a second term ΔE_{vol} penalizing volume changes is introduced:

$$\Delta E_{\text{vol}}(e_i) = w_{\text{vol}} \cdot \left(\sum_{T_j \in \{ncells_i\}} (\text{vol}(T_j) - \text{vol}(\bar{T}_j)) + \sum_{T_j \in \{icells_j\}} \text{vol}(T_j) \right). \quad (6.6)$$

T_j denote all tetrahedra in the set of neighborhood cells $\{ncells_i\}$ of e_i and introduced cells $\{icells_i\}$, respectively. $\bar{T}_j$ is the tetrahedron after the collapse and $\text{vol}(T_j)$ its volume. Note that only simplices in $\{icells_i\} \cup \{ncells_i\}$ can contribute to volume changes.

6.3.3 Equilateral energy

Especially in FEM applications it is often required that tetrahedra sustain equilateral shape. ΔE_{equi} can be employed to balance the edge length of individual tetrahedra:

$$\Delta E_{\text{equi}}(e_i) = w_{\text{equi}} \cdot \sum_{T_j \in ncells_i} \left(\sum_{\{a,b\} \in T_j} (l_{a,b} - m_j)^2 - \sum (\dot{l}_{a,b} - \overline{m}_j)^2 \right) \tag{6.7}$$

with edge length

$$l_{a,b} = |\mathbf{v}_a - \mathbf{v}_b|$$

and average edge length

$$m_j = 1 / |T_j| \cdot \sum_{\{a,b\} \in T_i} l_{a,b} .$$

6.3.4 Edge length energy

Note that the initial mesh M^n will be generated usually from some triangulation scheme. Depending on the application context and the desired mesh features it can be advantageous to include

$$\Delta E_{\text{edgelen}}(e_i) = w_{\text{edgelen}} \cdot |\mathbf{v}_a - \mathbf{v}_b| \tag{6.8}$$

into the cost function thereby enforcing short edges to be collapsed earlier.

6.3.5 Surface normal energy

For triangular surface meshes, the difference between the vertex normals of $\mathbf{v}_a$ and $\mathbf{v}_b$ are a good estimate of the local curvature of the mesh. If e_i lies within a region of high curvature, an edge collapse would flatten that region. Hence, this edge should be associated with an appropriate cost value defined as

$$\Delta E_{\text{normal}}(e_i) = w_{\text{normal}} \cdot (\mathbf{n}_a \cdot \mathbf{n}_b) \cdot |l_{a,b}|^2 , \tag{6.9}$$

where $\mathbf{n}_i$ denotes the normal vector in vertex $\mathbf{v}_i$ and $(\mathbf{n}_i \cdot \mathbf{n}_j)$ is the usual vector product.

Note that surface mehes, ΔE_{normal} is roughly equivalent to ΔE_{grad} in the volumetric setting.

Each of these terms can be weighted individually by coefficients w_{grad}, w_{vol}, w_{equi}, w_{edgelen}, and w_{normal}, respectively, to allow adoption to specific data sets and applications.

6.4 TOPOLOGICAL CONSIDERATIONS

Unfortunately, brute force selection of edges according to the cost function from above can introduce mesh inconsistencies, like degeneration, folding, intersection, or loss of individual features.

6.4.1 Static Tests

In order to avoid these types of artifacts, some *static* tests can be carried out prior to building the edgeheap. Before the test criteria are going to be introduced, some properties of edges and vertices have to be defined:

- *boundary edge:* an edge e_i is called a boundary edge if it lies on the boundary ∂ of the mesh. This can easily be determined for triangulated two-manifolds, where $\{icells_i\}$ contains less than two cells if and only if e_i is a boundary edge. For tetrahedral meshes, we cannot use this criterion, because we do not have an upper bound for the number of cells in $\{icells_i\}$. Instead, we can determine whether e_i is not a boundary edge by analyzing the union set $\{U_i\}$ of all vertices in $\{icells_i\}\backslash\{\mathbf{v}_a, \mathbf{v}_b\}$. If each vertex appears only once in this union, e_i is not on the boundary. Algorithmically, this can efficiently be determined by inserting every pair of vertices constituting an edge in $\{icells_i\}$ by using an exclusive insert operation. In other words, a vertex $\mathbf{v}_i$ is only inserted into $\{U_i\}$ if it is not already contained in the set. Otherwise, it is removed from $\{U_i\}$. After all endpoints of all edges in $\{icells_i\}$ have been inserted in this fashion, e_i can be marked as a boundary edge if $\{U_i\}$ is non-empty.
- *boundary vertex:* a vertex v_i is called a boundary vertex if at least one edge incident to v_i is a boundary edge.
- *boundary face:* a triangular cell face f_i is called a boundary face if all its edges are boundary edges.
- *boundary cell:* a tetrahedral cell T_i is called a boundary cell if at least one of its faces is a boundary face.

In a preprocessing step, all boundary vertices and edges are marked. Table 6.1 lists the five different combinations of boundary edges and vertices. Only cases one and five pass the consistency test for legal edge collapses.

Table 6.1 The five possible cases for combinations of boundary edges and boundary vertices. The examples refer to Figure 6.7.

case	e	$\mathbf{v}_a$	$\mathbf{v}_b$	legal	example
1				yes	$(\mathbf{v}_3, \mathbf{v}_4)$
2		*boundary*		optional[a]	$(\mathbf{v}_1, \mathbf{v}_{14})$
3			*boundary*	optional[a]	$(\mathbf{v}_3, \mathbf{v}_2)$
4		*boundary*	*boundary*	no[b]	$(\mathbf{v}_8, \mathbf{v}_{10})$
5	*boundary*	*boundary*	*boundary*	yes[c]	$(\mathbf{v}_0, \mathbf{v}_1)$

a. cases two and three induce “dents” on the boundary surface which may not be desired.
b. case four introduces degenerated cells, since $\mathbf{v}_9$ is not deleted.
c. a boundary edge always implies that its vertices are boundary vertices. Simplifications on the boundary are allowed.

Evaluation of case four can be used to ensure the preservation of the mesh boundary, both in the triangular and the tetrahedral setting. By collapsing an edge which did not pass this test, a boundary vertex could drift away from the boundary, destroying its shape. This test is an alternative to the use of the volume energy cost function term introduced above. Using a static check is more rigorous, because it maintains the boundary under any circumstance, whereas the use of the cost function might eventually allow to collapse that edge. Deciding for one of the other alternative is depending on the application.

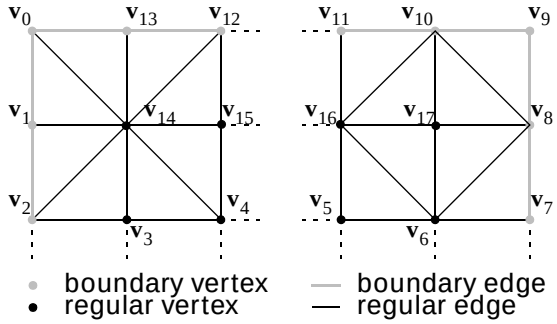


Figure 6.7 Naming conventions used for static consistency tests. For simplicity, the examples are depicted in 2D.

Interior edges (case one) have to be further checked dynamically for possible intersections and cell orientation changes.

A normal flipping heuristic [84] can be generalized to circumvent folding or self-intersection of cells. This can easily be implemented by testing for sign changes of surface normal vectors and by analyzing the volume of all tetrahedra $T_i \in \{ncells_i\}$ before and after the collapse. Recall that the volume of a tetrahedron is defined by the parallelepipedial product of its three edges e_i, e_j and e_k:

$$\frac{1}{6}[e_i, e_j, e_k] \;=\; \frac{1}{6}\langle e_i \times e_j, e_k \rangle \,. \tag{6.10}$$

If the volume of one of the neighboring tetrahedra becomes negative, tetrahedral folding occurs. In this case the edge fails the consistency test. In addition, this test avoids degenerate cells by setting a lower volume threshold to be retained after the collapse.

6.4.2 Dynamic Tests

Unfortunately, not all inconsistencies can be fixed with the static tests introduced in Section 6.4.1. Some severe problems arise *dynamically* while performing individual ecol operations and further testing is required on the fly.

Let us start from the following observation: Edge collapses can cause global intersections of tetrahedra, a simple example of which is shown in Figure 6.8.

Since not only neighboring cells can be affected with intersections, elaborate tests are required to detect and to prevent those potential inconsistencies. The simplest algorithm would involve a brute force edge–cell test with $O(N^2)$ time–complexity for each collapse. Under certain assumptions, a closer look at the problem reveals a more efficient solution of this problem.

We assume for the following investigations that the input mesh is intersection-free and connected. We make no assumption about the genus of the mesh, though. The problem can further be divided into checking interior edges and edges on the mesh boundary.

Interior edges. After passing the static tests described in Section 6.4.1, we can ensure that any non-boundary edge e_i (i.e., any interior edge) cannot have a boundary vertex as endpoint. If the set $\{\{icells_i\} \cup \{ncells_i\}\}$ contains no boundary edge, its boundary forms a polytope entirely wrapping the edge. A collapse of the edge, however, does not affect the boundary of the polytope, whose disjoint triangulation is given by the tetrahedra $\bar{T}_j \in \{ncells_i\}$. Thus, intersections can only occur with boundary cells and intersection

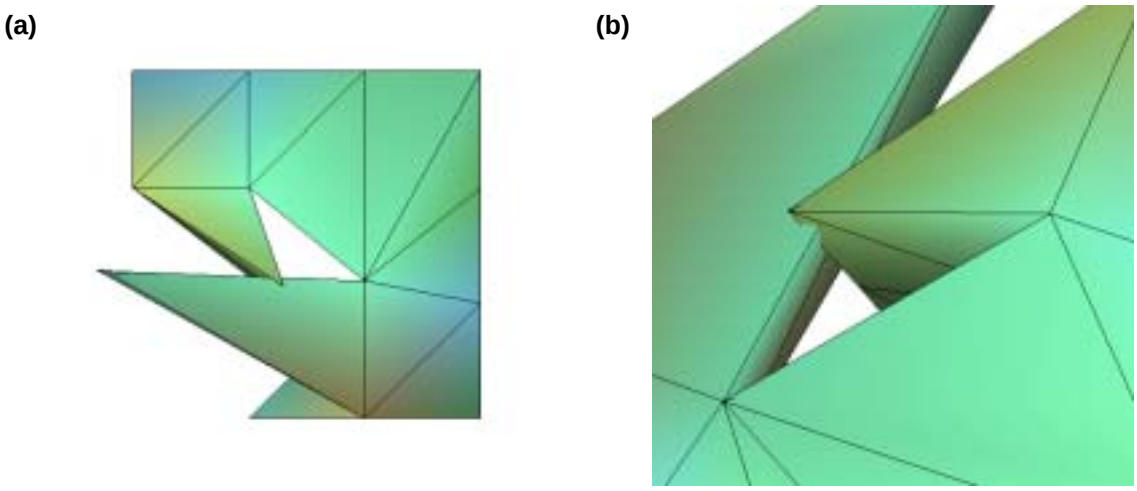


Figure 6.8 Tetrahedral intersection: **a:** Two tetrahedra intersect after an edge collapse. **b:** Close-up view of the intersection.

tests can be restricted to the mesh boundary. A collapse of an interior edge is depicted in Figure 6.9.

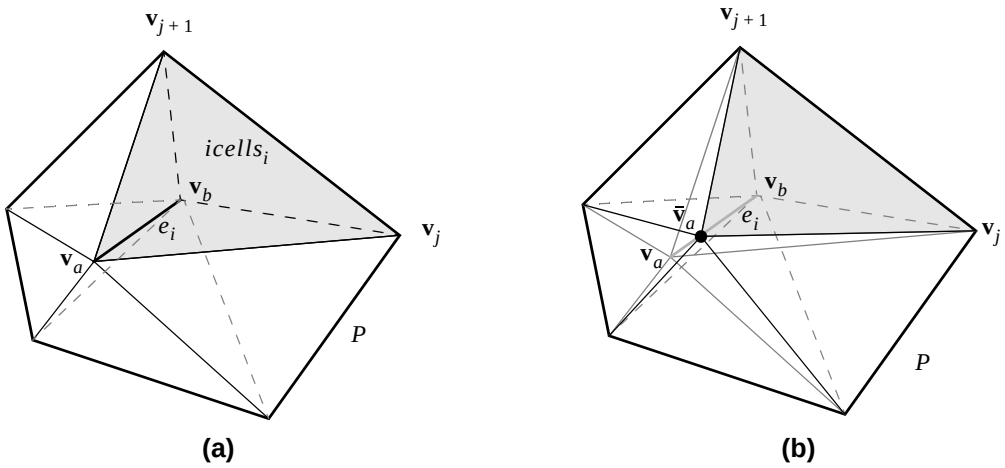


Figure 6.9 Tetrahedron $icells_i$ in polygon P vanishes after collapsing edge $e_i = (\mathbf{v}_a, \mathbf{v}_b)$ into $\bar{\mathbf{v}}_a$. It is replaced by faces(star($\bar{\mathbf{v}}_a$)). **a:** Before the collapse. **b:** During the collapse.

Exterior edges. Figure 6.10a depicts the top view of a tetrahedral mesh where T_j is a boundary cell that is close to the boundary of the mesh without intersecting it. Let e_i be the edge to be collapsed next and let vertex $\mathbf{v}_b$ be closer to the viewpoint than $\mathbf{v}_a$. The situation after the collapse where T_j is intersecting two faces of $\{ncells_i\}$ is depicted in Figure 6.10b.

In essence, we have to perform triangle–triangle intersection tests [73, 50] in case of boundary edges or vertices which can be carried out as follows: First, we define the set of triangles $\{f_k\}$ containing all boundary faces of tetrahedra $T_j \in \{ncells_i\}$. These are the faces which can change after ecol_i since they all share the new vertex $\bar{\mathbf{v}}_a$. Thus they are our prime candidates for intersection with other boundary faces.

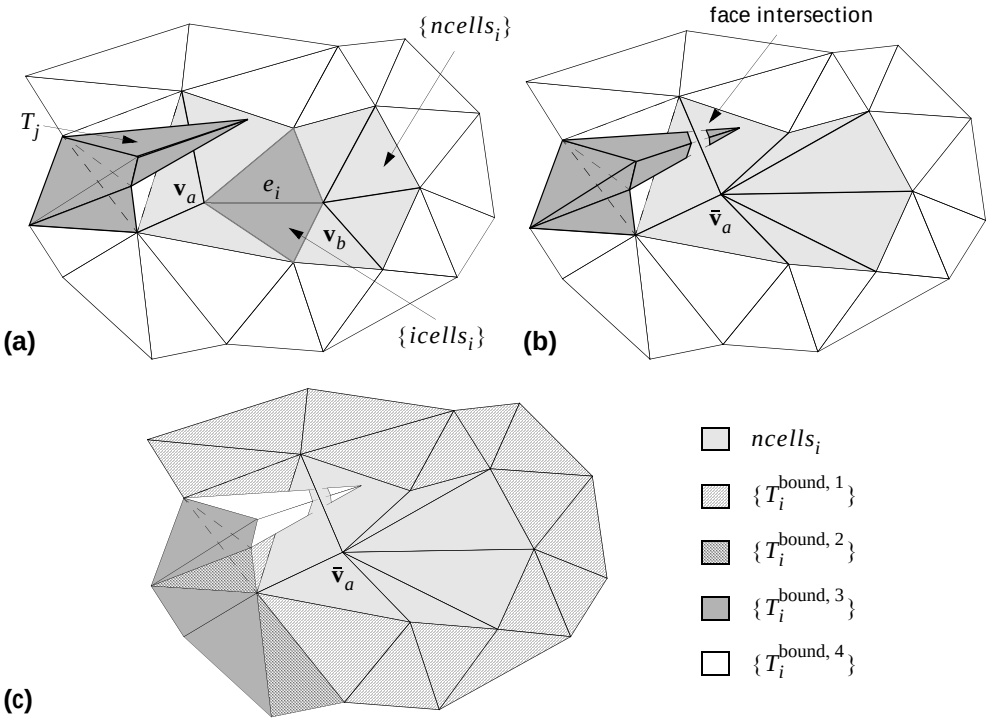


Figure 6.10 Tetrahedron T_j intersects two faces of $\{f_k\}$ after collapsing edge e_i into vertex $\bar{\mathbf{v}}_a$. **a:** Before collapse. **b:** After collapse. **c:** Traversal of the mesh $\{T_i^{\text{bound},\,s}\}$ for iteration steps one to four.

In order to avoid testing these faces against all other boundary tetrahedra, we propose the following iterative method: The algorithm starts from an initial set $\{T_i^{\text{bound}}\}$ containing the subset of all boundary tetrahedra in the one-neighborhood of $ncells_i$.

Let us take the first element of $\{T_i^{\text{bound}}\}$ and test for intersection with all faces in $\{f_k\}$. If the test fails and no intersection occurs, the tetrahedron is labeled as visited and we can proceed to the next element of $\{T_i^{\text{bound}}\}$. Otherwise we can abort the test and reverse the current ecol_i operation. The restriction to the boundary faces of each tetrahedron simplifies the intersection test, since many cells have only one boundary face (see Figure 6.10). After probing all cells, all visited tetrahedra in $\{T_i^{\text{bound}}\}$ are replaced with their non-visited neighboring boundary cells and thereby traverse the mesh. The $\{T_i^{\text{bound},\,s}\}$ are shown in Figure 6.10c for different iteration steps.

The following pseudo code summarizes the principle steps of the intersection test:

```
intersection_test {
  // Tbound[i,m]: boundary cells T_i at iteration s
  // f[k]: boundary faces in ncells_i
  // calculate_intersections: triangle-triangle
  // intersection test
  // S: maximum iteration level
  s = 0;
  while (Tbound[i,s] not empty && s < S){
    forall f[i] in Tbound[i,s]
      forall f[k]
        calculate_intersections(f[i], f[k]);
    update Tbound[i,s];
    s++;
  }
}
```

Note that the test asymptotically traverses all sharp faces of the mesh and consumes time proportional $O(F \cdot B)$, where F stands for the number of elements in $\{f_k\}$ and B for the overall number of boundary faces in the data set. As it will be demonstrated in Section 7.2, one can restrict the number of iteration steps to an upper bound in practice.

6.5 IMPLEMENTATION

The remaining sections of this chapter summarize important implementation issues regarding data structure design, algorithmic complexity and requirements for performing geomorphs. The implementation for progressive tetrahedralizations has been carried out to handle both triangular and tetrahedral meshes.

6.5.1 Data Structures

The objective of the progressive tetrahedralization method is the transformation of an unstructured input mesh into a coarse base mesh and a sequence of vertex split operations. Hence, the basic data structure is a `vsplit` record containing all necessary information to reconstruct the original mesh by carrying out vertex splits. The structure of a `vsplit` record is shown in Figure 6.11.

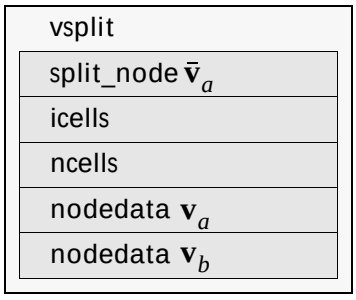


Figure 6.11 The `vsplit` record.

Minimum information for each record comprises the index of $\bar{\mathbf{v}}_a$, the connectivity of all cells in $\{icells_i\}$ and $\{ncells_i\}$, as well as data for $\mathbf{v}_a$ and $\mathbf{v}_b$. Since edges in our representation are not directed, we can assume without loss of generality that the index of $\mathbf{v}_a$ is less than that of $\mathbf{v}_b$. Therefore, it is not necessary to store the index of $\mathbf{v}_b$ as we always exchange it with the last index in the node list during the collapse. This ensures that the index of $\mathbf{v}_b$ is identical to the current size of the node list. Node data for both vertices can be delta coded.

Connectivity of $\{ncells_i\}$ comprises the indices of neighboring cells of $\mathbf{v}_b$, only. Obviously, the connectivity of the neighborhood of $\mathbf{v}_a$ is not affected by an edge collapse or a vertex split. Excluding $\{icells_i\}$ from this examination, the sets of neighborhood cells of $\mathbf{v}_a$ and $\mathbf{v}_b$, respectively, are disjunct. During reconstruction, we can replace $\mathbf{v}_a$ with $\mathbf{v}_b$, as both vertices have earlier been collapsed into $\bar{\mathbf{v}}_a$, whose index is identical to that of $\mathbf{v}_a$.

In the case of $\{icells_i\}$, indices have to be stored for all vertices except for $\mathbf{v}_a$ and $\mathbf{v}_b$, which are stored separately in `vsplit` and can easily be derived on the fly. Let us consider an example in the tetrahedral setting. Given the connectivity $K = \{v_1, v_2, v_3, v_4\}$ and $v_a, v_b \in K$. Note that v_i denotes the index of vertex $\mathbf{v}_i$. Let further be

o_{orig} = orientation(K) the original orientation of the tetrahedron as function of K. The subset $K' = K \backslash \{v_a, v_b\} = \{a, b\}$ contains the remaining indices that have to be stored with `icells` in `vsplit` in original order.

Figure 6.12 for the triangular setting and Figure 6.13 for the tetrahedral setting illustrate the alternating orientation of the cells for different permutations of the vertices. For reconstruction purposes, we are not interested in the exact connectivity of a single cell, but only in a permutation with consistent orientation. For that reason, orientation o_{orig}, which requires only one bit, is stored.

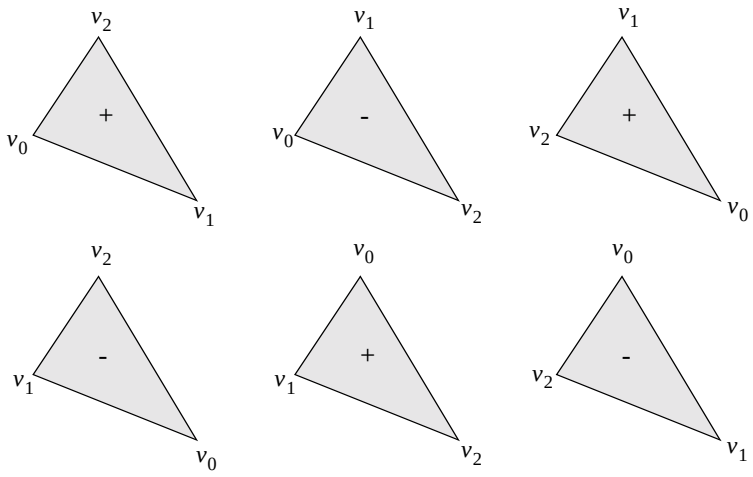


Figure 6.12 The six permutations of vertices result in alternating orientations of the triangle.

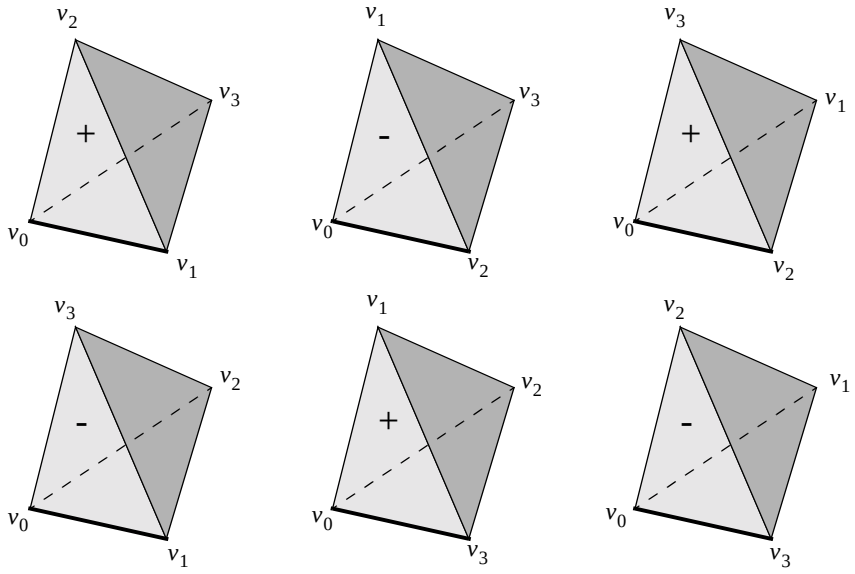


Figure 6.13 Six out of 24 possible permutations of tetrahedral vertices with alternating cell orientation (vertex v_0 is fixed).

Given v_a, v_b and $K' = \{a, b\}$, only two cases have to be considered for the reconstruction of $\overline{K}$:

1. $\overline{K} = \{v_a, v_b, a, b\}$ for $o_{\text{orig}} = \text{orientation}(\{v_a, v_b, a, b\})$
2. $\overline{K} = \{v_b, v_a, a, b\}$ for $o_{\text{orig}} = \text{orientation}(\{v_b, v_a, a, b\})$.

Since $\{a, b\}$ have been stored in original order, 12 different combinations remain for the connectivity as listed in Table 6.2.

Table 6.2 All 12 combinations for tetrahedron connectivity and resulting orientation and outcodes for cells with positive orientation.

K	orientation(K)	outcode	K	orientation(K)	outcode
v_a, v_b, a, b	+	41200	a, v_a, v_b, b	+	40120
v_a, a, v_b, b	-		a, v_a, b, v_b	-	
v_a, a, b, v_b	+	41002	a, v_b, v_a, b	-	
v_b, v_a, a, b	-		a, v_b, b, v_a	+	40201
v_b, a, v_a, b	+	42010	a, b, v_a, v_b	+	40012
v_b, a, b, v_b	-		a, b, v_b, v_a	-	

Table 6.3 All six combinations for triangle connectivity and resulting orientation and outcodes for cells with positive orientation.

K	orientation	outcode
v_a, v_b, a	+	3120
v_b, v_a, a	-	
v_a, a, v_b	-	
v_b, a, v_a	+	3201
a, v_a, v_b	+	3012
a, v_b, v_a	-	

A fast method is devised to determine the correct case from Table 6.2 using an unique outcode generated from the order of the indices in $\overline{K}$. The first digit encodes the number of vertices in a cell (i.e., 3 for triangles, and 4 for tetrahedra), which allows for a consistent implementation of both triangular and tetrahedral settings. The following digits encode the original order of the indices, in other words, 1 for v_a, 2 for v_b and 0 for either of the remaining indices a and b. (Table 6.3 list all combinations and the outcodes for the triangular setting). Based on these outcodes, it is possible to determine whether or not the orientation differs from the original and, accordingly, which of the two cases is applicable.

The single bit indicating and orientation change is stored in the first index value of K' using bitwise negation. Hence, the resulting index is determined by $\tilde{a} = -(a + 1)$. See Figure 6.14 for an example.

6.5.2 Complexity

Complexity analysis can be divided into two parts: Transformation of an unstructured input mesh into a progressive representation using ecol operations, and reconstruction of

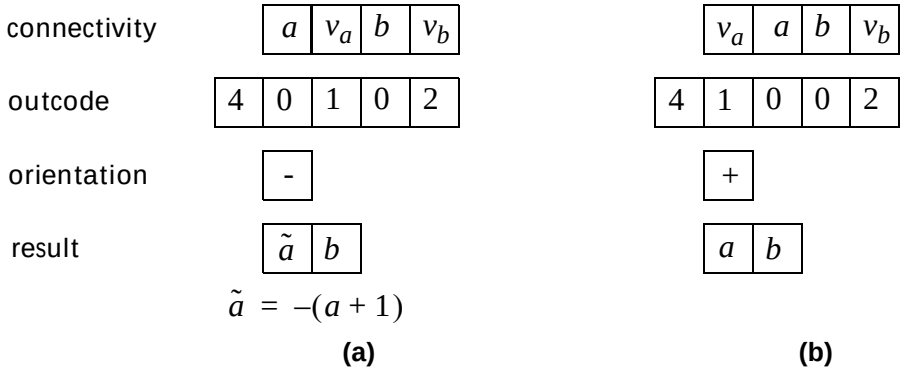


Figure 6.14 Example for encoding connectivity and orientation of two tetrahedral cells. **a:** negative orientation. **b:** positive orientation.

a mesh at arbitrary resolution using vsplits. The beauty of this method is that a single vsplit_i can be performed in time linear to the size of $\{ncells_i\}$, which can be neglected with regard to the overall number of cells in the data set. Hence, the major time involved is concentrated in the carrying out of the vertex splits, which includes evaluation of all cost functions.

In general, time-complexity is proportional to the number of edge collapses (i.e., the number of edges $N_{\text{edges}} = \text{sizeof}(\{e_i\})$ in the input mesh). Average time t_{collapse} for a single edge collapse operation comprises components for cost function evaluation, recomputes for entries in $\{ncells_i\}$ and edgeheap manipulations. For cost functions with constant cost per cell in $\{ncells_i\}$, we yield

$$t_{\Delta E} = \sum_i t_{\Delta E_i}, \tag{6.11}$$

which can be considered as constant per collapse. This is also true for most of the check functions which have been discussed above:

$$t_{\text{check}} = \sum_i t_{\text{check}_i}. \tag{6.12}$$

The dynamic intersection test for tetrahedral cells is an exception, which will be discussed separately.

After each successful edge collapse, certain values computed for each entry during in the edgeheap during preprocessing—such as cell volume and edge length—have to be recomputed:

$$t_{\text{update}} = \sum_i t_{\text{update}_i}. \tag{6.13}$$

The complexity of these updates is also dependent on the size of $\{ncells_i\}$ and assumed to be constant.

Finally, the updated cost values for each edge have to be propagated in the edgeheap. In general, this operation is $O(\log N_{\text{edges}})$, but for small changes we can achieve linear-time complexity as well: Each edge stores its position within the heap and can thus move relative to that position upon cost value changes. For each update due to an edge collapse,

cost value changes are usually small. As a result, updated edges move only within a small distance relative to their original position. This distance can also be expressed as

$$d_{\text{heap}} = c \cdot \log(\text{sizeof(edgeheap)}), \tag{6.14}$$

where c smaller than one (e.g., 0.125 for a maximal distance of 1/8 of the height of the heap). For very small cost value changes, as it is the case during updates of the extended region of influence, d_{heap} is assumed to be constant.

Combining all of the above components we yield

$$t_{\text{collapse}} = t_{\Delta E} + t_{\text{check}} + t_{\text{update}} \in O(N_{\text{edges}}). \tag{6.15}$$

Obviously, the intersection check is the algorithmic bottle-neck of our implementation. Although we have optimized the algorithm to inspect only the boundary for intersections, the ratio between boundary cells and interior cells is predominantly dependent on the specific data set. Therefore, we end up with a worst case of $O(F \cdot B)$.

Figure 6.15 depicts an example of a medical volume data set comprising approximately 90,000 edges. The number of edges on the edgeheap is steadily decreasing, while the number of updates per edge collapse remains constant over time.

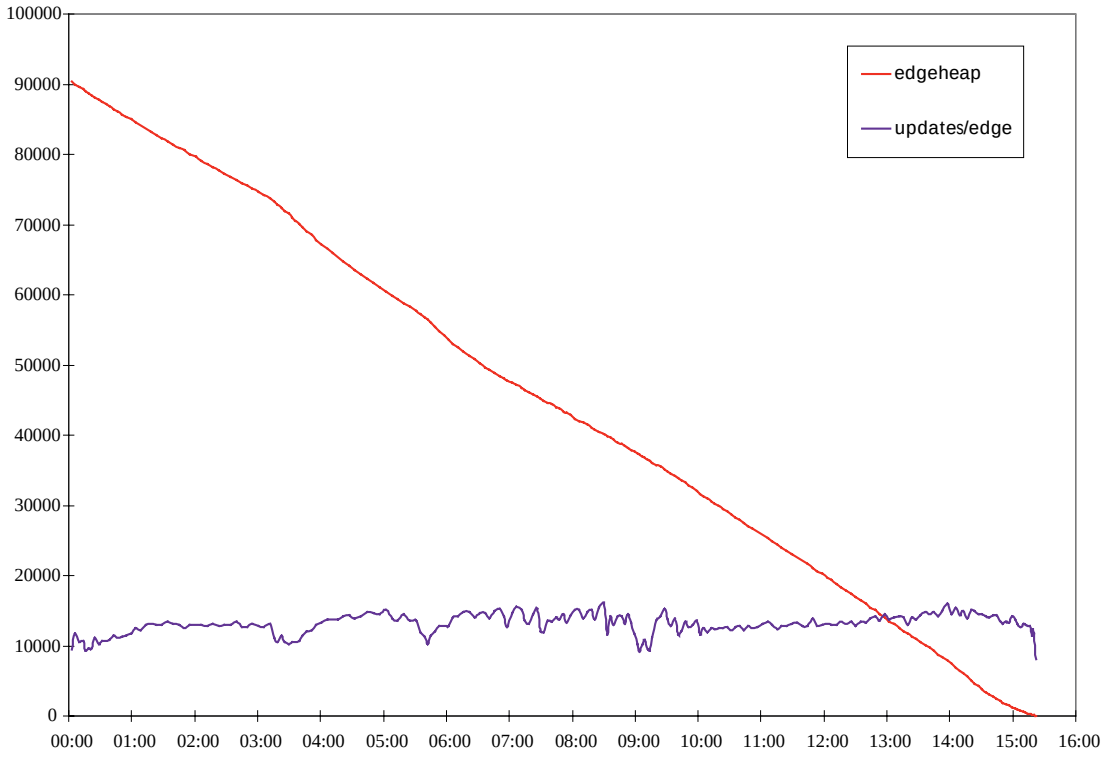


Figure 6.15 Example for the complexity of a transformation into a PT representation. The number of edges on the edgeheap is steadily decreasing while the number of updates per edge collapse remains constant.

6.5.3 Geomorphing

Geomorphing is an extension of the reconstruction algorithm that avoids popping artefacts arising from regular vertex splits. When a mesh is reconstructed interactively or progressively, splitting a vertex and thereby inserting (possibly large) cells causes visual

discontinuities, usually referred to as popping. This can be avoided in an elegant fashion by morphing between two mesh instances M^i and M^{i+1} [52]. This can be carried out by interpolating between the positions of all vertices that change due to a set of subsequent vertex splits. Figure 6.16 depicts a geomorph sequence for 20 simultaneous vertex splits.

Geomorphing does not require any modification to the edge collapse operations or the internal data structure which represent a PT, because no additional information is necessary. Note that a geomorph $G(M^i, M^j, \alpha)$ between M^i and M^j is equivalent to $G(M^j, M^i, 1-\alpha)$ in the opposite direction. For that reason, only geomorphing in one direction (e.g., $j > i$) has to be considered.

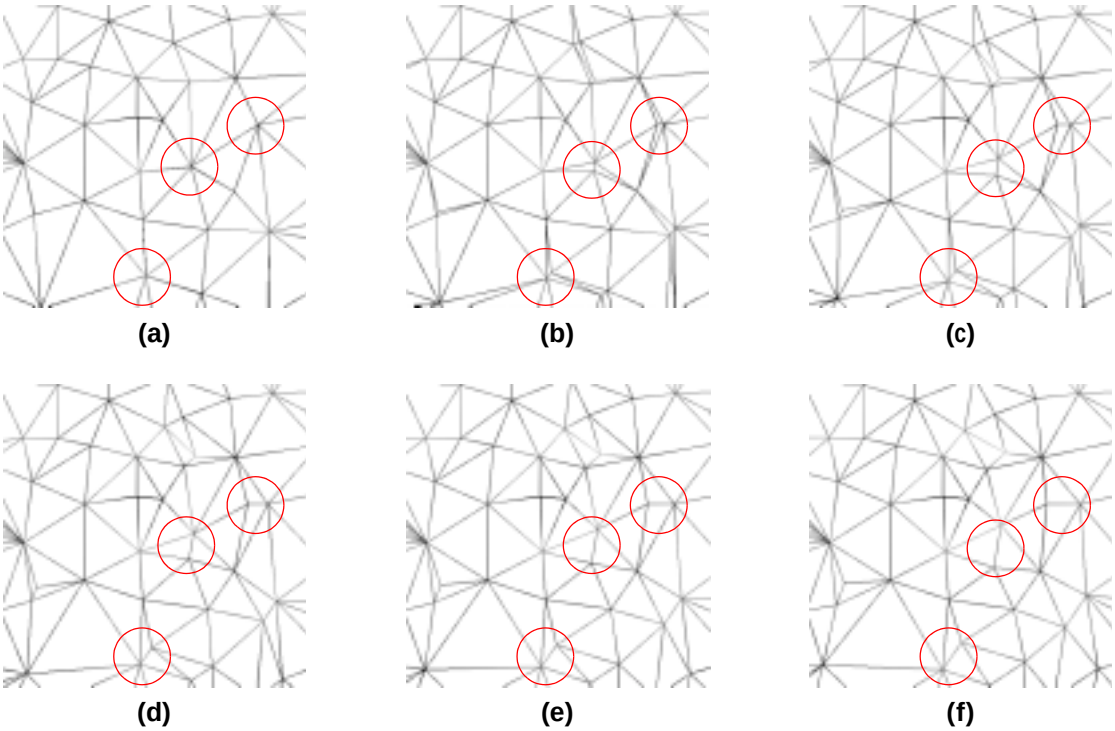


Figure 6.16 Illustration of a geomorph for a triangular mesh. 20 vertex splits are carried out simultaneously while the interpolation parameter is increased from 0.0 to 1.0: **a:** $\alpha = 0.0$. **b:** $\alpha = 0.2$. **c:** $\alpha = 0.4$. **d:** $\alpha = 0.6$. **e:** $\alpha = 0.8$. **f:** $\alpha = 1.0$.

Geomorphs between M^i and M^j can easily be implemented using a Boolean array `morph_array` of size j. An array entry can either be `true`, if the corresponding vertex position is being changed during the geomorph, or `false`, if the position is fixed. All entries of `morph_array[k<i]` are initialized to `false`. The remaining entries `morph_array[i<=k<j]` are marked `true`. In other words, any vertex in the start mesh M^i is marked as remaining fixed and any additional vertex in M^j is marked as being morphed. For each vertex split operation, the index of the next `split_node` $\bar{\mathbf{v}}_a$ is obtained from the `vsplit` record and the corresponding entry in the array is set to `true`. This ensures that every vertex which is going to be translated during the geomorph is marked.

After initialization of the array, position and attributes of the vertices $\mathbf{v}_a$ and $\mathbf{v}_b$ are interpolated from the starting position of $\bar{\mathbf{v}}_a$ to their original position. Note that it is possible to specify an arbitrary number of independent vertex splits which can be morphed simultaneously. For more details on the implementation see [109].

7

EXPERIMENTS AND COMPARISON

This chapter features some experimental results of the methods introduced in this thesis. To evaluate the performance of the different methods introduced in the previous chapters, separate stages of the compression pipeline are discussed individually. First, results from the wavelet-based representations of Chapter 5 are presented. This is followed by results from progressive tetrahedralizations introduced in Chapter 6. We conclude with a comparison of the different methods discussing advantages and disadvantages.

7.1 WAVELET-BASED REPRESENTATIONS

This section reflects the versatility of the wavelet-based surface and volume representation schemes.

7.1.1 Surface Compression

Two different digital terrain models are used to demonstrate the codec performance of the proposed scheme. Both experiments are carried out in a non-adaptive setting to focus on the compression behavior.

***Matterhorn* digital terrain model.** The first experiment employs a digital terrain data set of the Matterhorn. The original resolution of the data is 701 by 481 vertices with 25 meter spacing. Due to the fact that our implementation of the fast wavelet transform operates on resolutions which are multiples of two in each direction (plus some additional vertices equal to the degree of the basis function), the data are cropped to 515 in x-direction and expanded to 515 in y-direction as described in Section 2.5. In this example, the the upper decomposition level for the wavelet transform is set to $M = 6$. The resulting images are depicted in Figure 7.2 and quantitative measurements are listed in Table 7.1.

Note that there are almost no visual differences to the original approximation up to a compression ratio of 30:1. Even a ratio of 172:1 delivers convincing results with high visual details. This is also reflected in the PSNR values in Table 7.1 and the rate-distortion graph in Figure 7.1.

Table 7.1 Compression of the Matterhorn digital terrain model for different compression settings (see Figure 7.2). [DHM©]

fig.	coefficients [%]	quantization [bits]	compression ratio	compression rate [bits/vertex]	RMSE [%]	PSNR [dB]
(a)	100	20	2.17:1	14.73	0.000	120.34
(b)	80	20	2.50:1	12.80	0.006	89.51
(c)	60	15	4.58:1	6.99	0.017	77.88
(d)	30	10	13.04:1	2.45	0.079	65.13
(e)	10	10	29.67:1	1.08	0.136	60.36
(f)	1	10	172.41:1	0.19	0.729	45.95
(g)	0.1	10	1111.11:1	0.03	3.124	33.25
(h)	0.05	10	1666.67:1	0.02	5.983	27.63

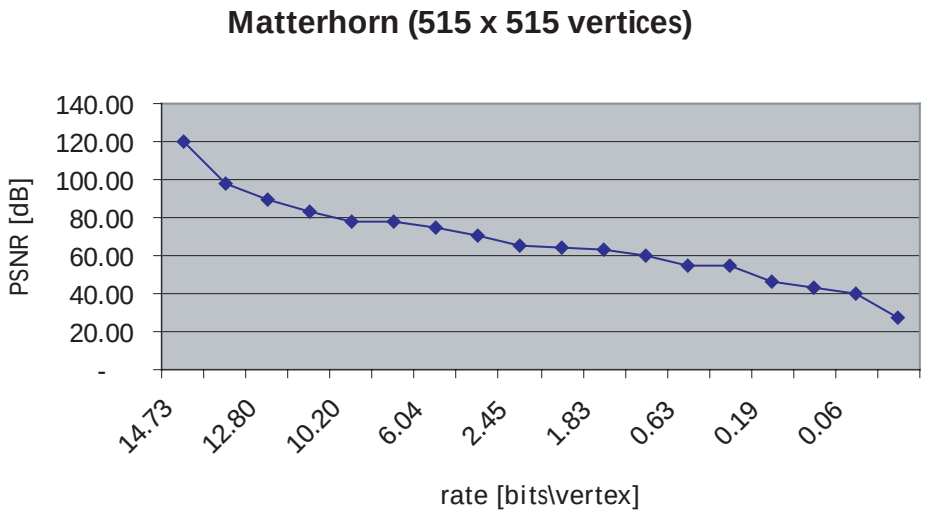


Figure 7.1 Rate distortion graph for the Matterhorn model at different compression levels (see Figure 7.2).

Small ripples which can be seen in Figure 7.2g are caused by single B-wavelets with large support. The coarsest approximation level in Figure 7.2h comprises scaling functions only and the smooth approximation with cubic B-splines is visible.

***Albis* digital terrain model.** The second example features the Albis digital terrain model with $M = 6$. The model has also been expanded to 515 by 515 vertices from the original resolution of 401 by 401. As opposed to the previous example, the distance between two vertices is only 10 meters, providing a more detailed representation of the

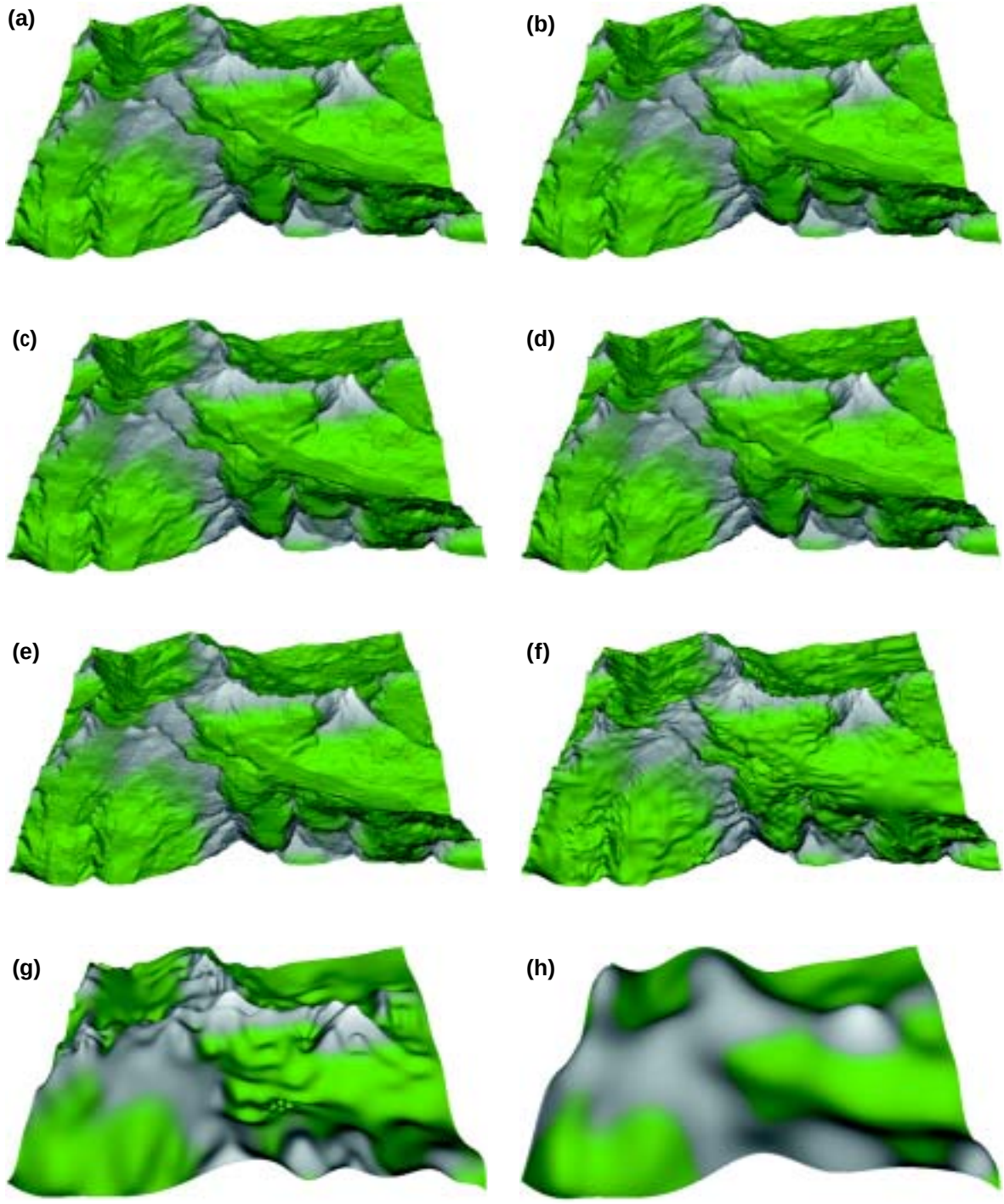


Figure 7.2 Results of compressing the Matterhorn data set. Corresponding quantitative measurements are listed in Table 7.1 and a rate distortion graph is depicted in Figure 7.1.

terrain. The resulting images are shown in Figure 7.4 and quantitative measurements are given in Table 7.2.

The performance of the codec for this model is slightly worse than for the Matterhorn data set. This is caused by the existence of many small-scale details of the model at 10m-resolution, which have high entropy and are harder to compress. At higher compression ratios, these details are likely to vanish resulting in a lower fidelity of the approximation. Nevertheless, we will show in Section 7.1.4 that in combination with surface textures, the

loss of small-scale geometric details can be compensated for. See Figure 7.3 for the rate-distortion graph of this experiment.

Table 7.2 Compression of the Albis digital terrain model for different compression settings (see Figure 7.4). [Swissphoto©]

fig.	coefficients [%]	quantization [bits]	compression ratio	compression rate [bits/vertex]	RMSE [%]	PSNR [dB]
(a)	100	20	1.71:1	18.67	0.000	98.08
(b)	70	20	1.93:1	16.61	0.033	78.21
(c)	50	15	3.69:1	8.67	0.304	59.25
(d)	30	15	5.43:1	5.89	0.738	51.44
(e)	20	10	12.47:1	2.57	2.604	40.54
(f)	4	10	40.65:1	0.79	2.712	40.16
(g)	1	10	119.05:1	0.27	12.173	27.13
(h)	0.05	10	909.09:1	0.04	134.27	6.28

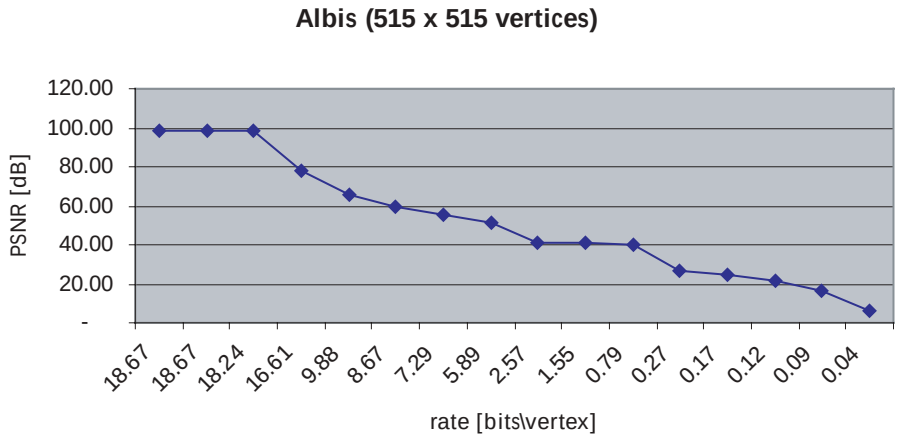


Figure 7.3 Rate distortion graph for the Albis model at different compression levels (see Figure 7.4).

7.1.2 Adaptive Multi-Resolution Surfaces

So far, we have only investigated the compression behavior of the our scheme. The next example depicted in Figure 7.6 combines surface compression and adaptive triangulation at various reconstruction levels. The same terrain model of the Matterhorn as above is used in this experiment. The number of vertices is progressively increased from 0.03% at $m = 6$ to 25.19% at $m = 0$. The settings for the codec are kept constant for this example with 30.00% of the coefficients remaining at 10 bit quantization. We achieve a compression ratio of 13.04:1 for $m = 0$ and a rate of 2.45 bits/vertex. The geometric adaptivity threshold was set to $\varepsilon_0 = 0.0025$.

Note the transition from a (semi-)regular mesh to a highly adaptive approximation as more of the vertices are decoded and progressively inserted into the mesh. The initial semi-

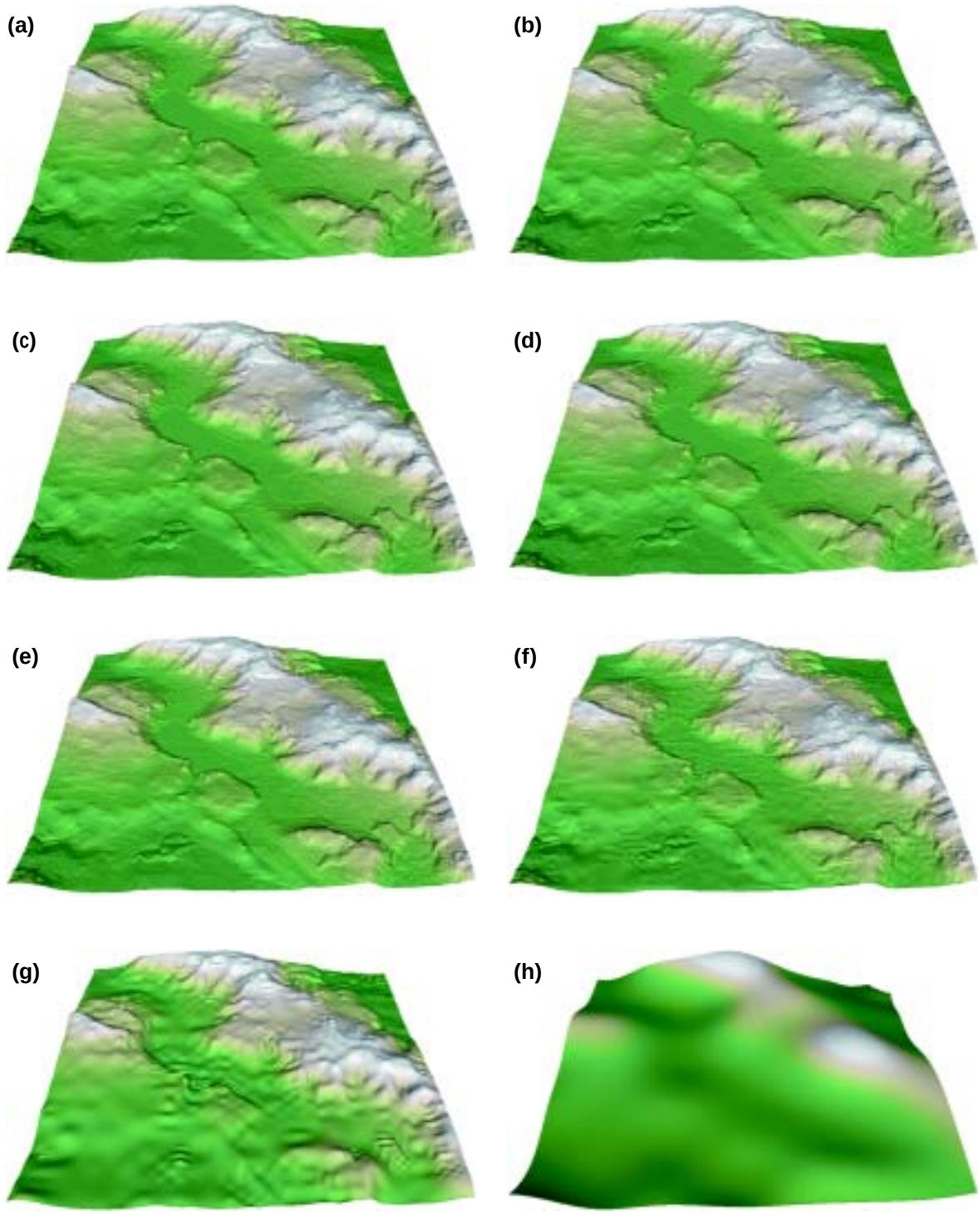


Figure 7.4 Results of compressing the Albis data set. Corresponding quantitative measurements are listed in Table 7.2 and a rate distortion graph is depicted in Figure 7.3.

regularity is caused vertices corresponding to scaling function coefficients, which are inserted into the mesh by default. Comparing the approximation in Figure 7.6d with the (non-adaptive) approximation of the same data in Figure 7.2d, the quality of the adaptive multi-resolution scheme is demonstrated.

Table 7.3 Progressive adaptive reconstruction of the Matterhorn model for fixed compression settings. (see Figure 7.6).

fig.	reconstruction level *m*	# vertices [%]	RMSE	PSNR [dB]
(a)	6	0.03	10.59	27.63
–	5	0.11	10.59	27.63
(b)	4	0.39	4.79	34.52
–	3	1.35	2.69	39.52
(c)	2	3.89	0.59	52.70
–	1	8.81	0.44	55.24
(d)	0	25.19	0.17	63.37

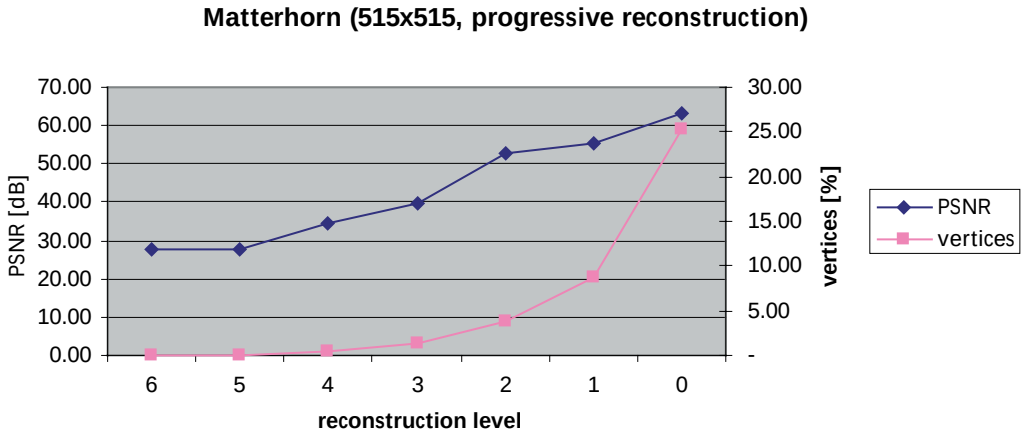


Figure 7.5 Progressive adaptive reconstruction of the Matterhorn model (see Figure 7.6).

7.1.3 Quadtree-based mesh simplification

Results from the quadtree-based mesh simplification scheme are depicted in Figure 7.8. The Matterhorn model has been cropped to 256 by 256 vertices due to constraints of the prototype implementation. As compared with the results of the adaptive Delaunay-based vertex insertion method in Section 7.1.2, the quadtree mesh features a less adaptive, but more structured approximation. This is of course due to the two-level look-up-table, which constraints the growth of the quadtree.

7.1.4 Texture Compression

For this experiment, the Albis data set has been texture-mapped with a high-quality orthophoto that is registered with the terrain model. The original resolution of the orthophoto is 5334 by 5334 pixels with 24 bit color depth. Beforehand, the texture was down-scaled to 1027 by 1027 pixels due to limitations of the available texture memory. The wavelet decomposition comprises seven levels quantized to 10 bits and the number of remaining vertices in the terrain approximation is equal to 30 percent. This results in a rate of 3.87 bits per vertex. The geometric accuracy threshold ε in this example is equal to 0.005.

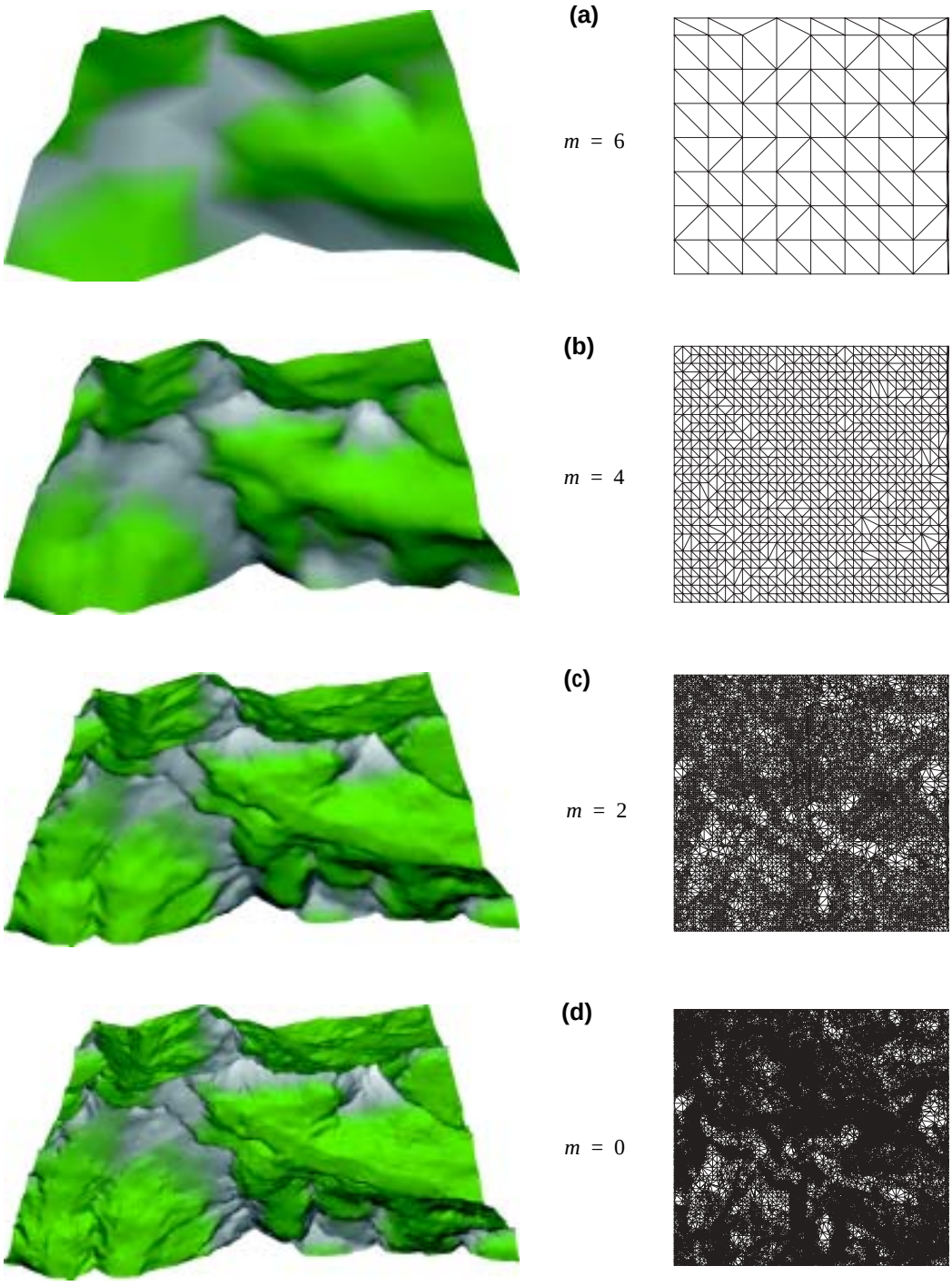


Figure 7.6 Progressive adaptive reconstruction of the Matterhorn model a different levels from $m = 6$ to $m = 0$. See Table 7.3 for quantitative results.

Different settings have been used for the individual color channels of the texture: The Y channel, which has high entropy, is quantized to 12 bits with 70 percent of the wavelets remaining in the approximation. The resulting rate is equal to 8.31. The two color channels, C_B and C_R, The coefficients have been quantized to 10 bits and only 15 percent of the wavelets are employed for reconstruction. In comparison to the Y channel, the rates are significantly lower at 1.86 and 1.84 bits per pixels, respectively. Distortion figures for this experiment are listed in Table 7.5.

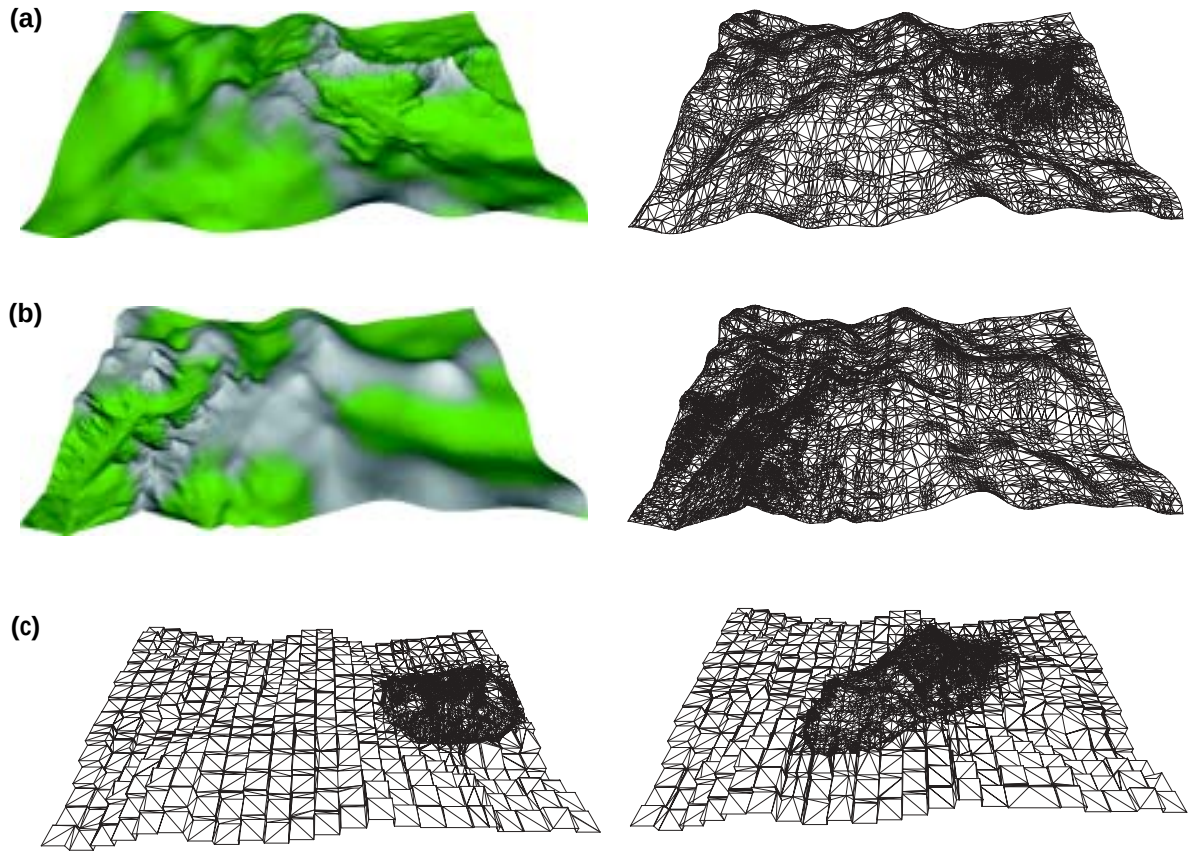


Figure 7.7 Adaptive reconstruction of the Matterhorn model with local level-of-detail control. **a, b:** The local-level-of detail focus is centered at different positions (left: shaded, right: wireframe). **c:** Instead of cubic B-splines, Haar wavelets are used for the approximation. Due to the minimal support of the Haar wavelets, the localization property is nicely illustrated.

Table 7.4 Quadtree-based mesh simplification of the Matterhorn model (see Figure 7.8).

fig.	error threshold	triangles [%]
–	0	99.99
(a)	0.25	68.27
(b)	0.50	53.97
(c)	1.00	42.30
(d), (e)	5.00	19.74
(f)	15.00	11.85

The seven approximation levels have been reconstructed progressively (see Figure 7.10). The rate-distortion graph depicted in Figure 7.9 shows the distortion of the terrain and the texture approximation in comparison with the number of triangles for the different reconstruction levels. Obviously, the distortion of the texture is higher than that of the terrain. As already mentioned in Section 5.4.5, the compression pipeline has not been optimized for image compression and, thus, this behavior could be expected.

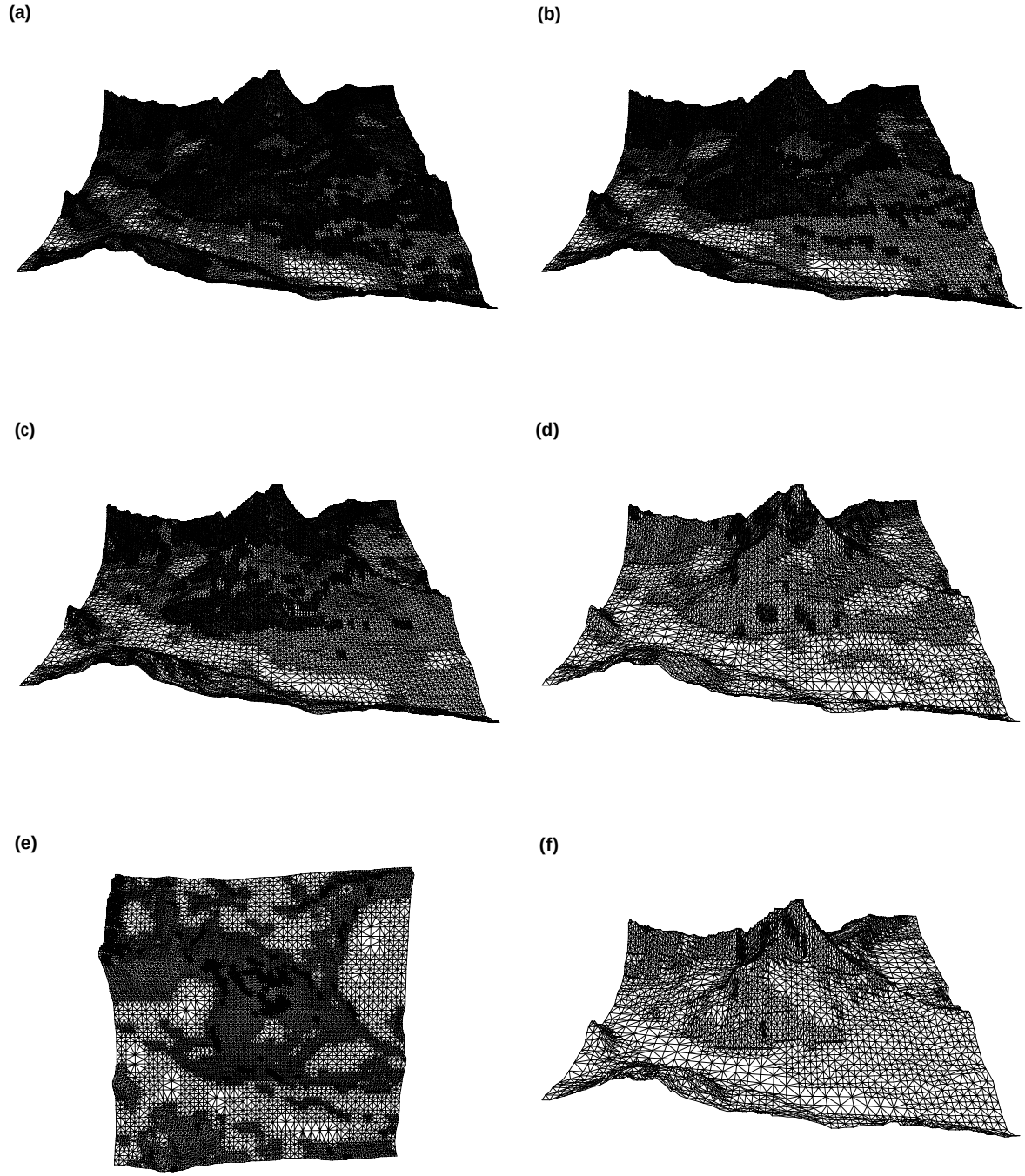


Figure 7.8 Quadtree-based mesh simplification of the Matterhorn (see Table 7.4).

Table 7.5 Progressive adaptive reconstruction of the textured Albis model for fixed compression settings (see Figure 7.10).

fig.	level	# vertices [%]	RMSE terrain	PSNR terrain [dB]	pixels [%]	RMSE texture	PSNR texture [dB]
(a)	1	0.04	2.02	42.03	0.02	64.06	12.00
–	2	0.11	1.22	46.38	0.10	49.23	14.29
(b)	3	0.37	0.63	52.13	0.39	26.13	19.19

Table 7.5 Progressive adaptive reconstruction of the textured Albis model for fixed compression settings (see Figure 7.10). (Continued)

fig.	level	# vertices [%]	RMSE terrain	PSNR terrain [dB]	pixels [%]	RMSE texture	PSNR texture [dB]
–	4	1.16	0.32	58.06	1.56	23.16	20.83
(c)	5	3.13	0.16	63.83	6.25	15.43	24.36
–	6	6.92	0.11	67.01	25	15.01	24.60
(d)	7	23.10	0.08	70.03	100	5.28	33.68

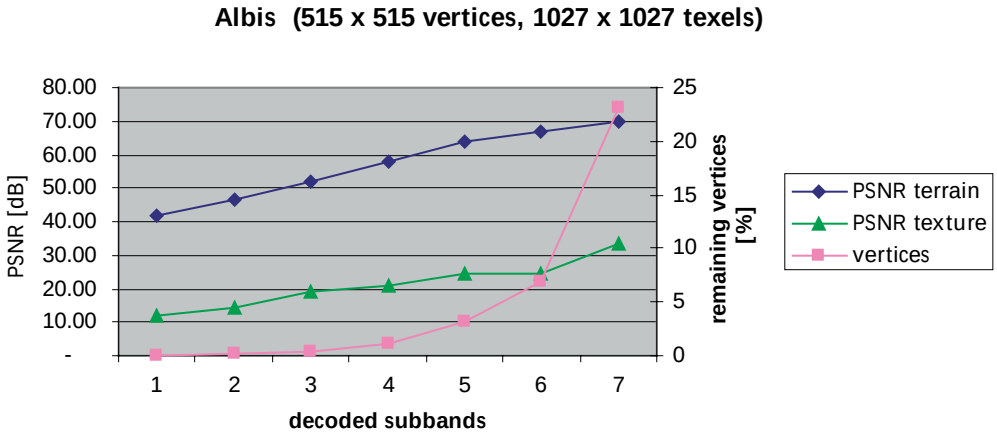


Figure 7.9 Rate-distortion graph for the textured Albis model. The distortion for both the terrain and the texture approximation in comparison to the percentage of remaining triangles is shown.

7.1.5 Local Level-Of-Detail Control

An adaptive approximation of Matterhorn with local level-of-detail control is depicted in Figure 7.7. The focus of the local oracle is centered at different positions. Figure 7.7c nicely illustrates the wavelet localization property: Instead of the cubic B-spline representation, Haar wavelets, which feature a compact support equal to one, are employed.

7.1.6 Volume Compression

For the following investigation, we have selected a medical CT volume data set of a child's skull. The resolution of the data set is 128 by 128 for each of the 62 slices. The compression parameters have been set to 10 bit quantization with four wavelet decomposition levels. 75 percent of the wavelet coefficients are discarded which results in a compression ratio of 10.4:1 and a rate of 3.08 bits per voxel, respectively. Iso-surfaces extracted at level $\tau = 50$ are depicted in Figure 7.11 and corresponding quantitative number are listed in Table 7.6.

Note the high quality of the iso-surface extraction method introduced in Section 5.5.2. The iso-surface shown in Figure 7.11d and Figure 7.11e are both extracted from the identical volume approximation. The first mesh was extracted with our new method and the latter mesh was extracted with the standard marching tetrahedron method. The bumps on

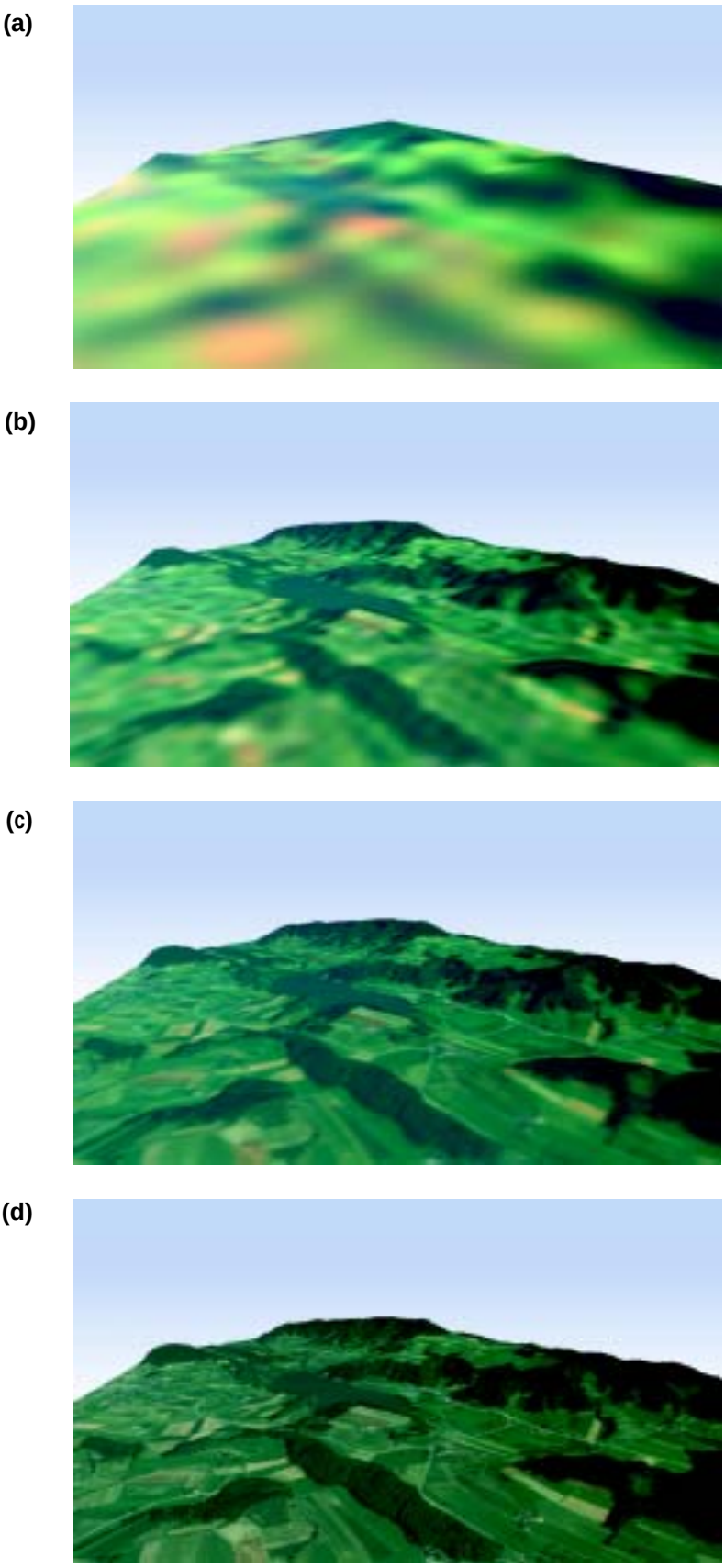


Figure 7.10 Progressive reconstruction of the Albis terrain model, mapped with orthophotos. **a:** level one. **b:** Level three. **c:** Level five. **d:** Level seven. Performance statistics are listed in Table 7.5 and Figure 7.9. [Swissphoto©] (see Color Plate 2 on page 152).

the surface in Figure 7.11e are caused by the linear interpolation of the intersection between the tetrahedral cells and the iso-surface. Apparently, this method fails in highly adaptive settings, whereas our method provides for better results while employing higher-order B-spline interpolation to find the intersection.

Table 7.6 Adaptive volume approximation and iso-surface extraction for fixed compression settings. (see Figure 7.11)

error threshold	vertices [%]	tetrahedra [%]	triangles	RMSE [%] (volume)	PSNR [dB] (volume)
0.075	5.79	6.01	128,923	3.87	28.24
0.05	9.30	9.58	167,933	2.91	30.71
0.025	18.30	18.74	227,676	2.27	32.85
0.01	43.90	45.09	253,428	1.83	34.76

7.2 PROGRESSIVE GEOMETRY REPRESENTATIONS

The following sections feature several experiments with progressive mesh and progressive tetrahedralization representations.

7.2.1 Progressive Surface Meshes

Two examples for progressive surface mesh will be given in this section. For the first example, a topologically simple model of an almond is employed. The second example features the complex and highly detail model of David's head from the Stanford Digital Michelangelo Project [62].

Almond. The full Almond data set consists of 6400 triangles. The following three cost functions have been used to transform the data set into a progressive mesh representation: Edge length energy, surface normal energy and equilateral energy. Because of its simple topology, the original data could be collapsed to a single triangle. The complete transformation took only 8.3 seconds on an SGI Octane R10000@195MHz. Six reconstruction levels up to the original resolution are depicted in Figure 7.12 along with the corresponding error visualizations. We use textures to visualize the error, which provides for higher quality than straight-forward per-vertex shading. The performance statistics for this experiment are listed in Table 7.7. Note that the reconstruction time for each of the levels was below 0.01 seconds and has therefore been omitted from the table.

This simple experiment has been carried out to illustrate the influence of the different cost function terms: The equilateral energy term is responsible for the well shapedness of the mesh. The surface normal energy ensures that the mesh refinement at the sharp boundary of the almond is sufficiently high to accurately represent its original shape.

The error visualization shows that the mesh depicted in Figure 7.12c is virtually an error-free approximation of the data. The relatively large errors in Figure 7.12a are caused by increasingly large triangles, which illustrates the limitation of piecewise-linear approximations of smooth curved surfaces.

David. The original size of the head of Michelangelo's David at one millimeter resolution is 4,000,885 triangles. Due to memory limitations, this mesh has been pre-simplified to

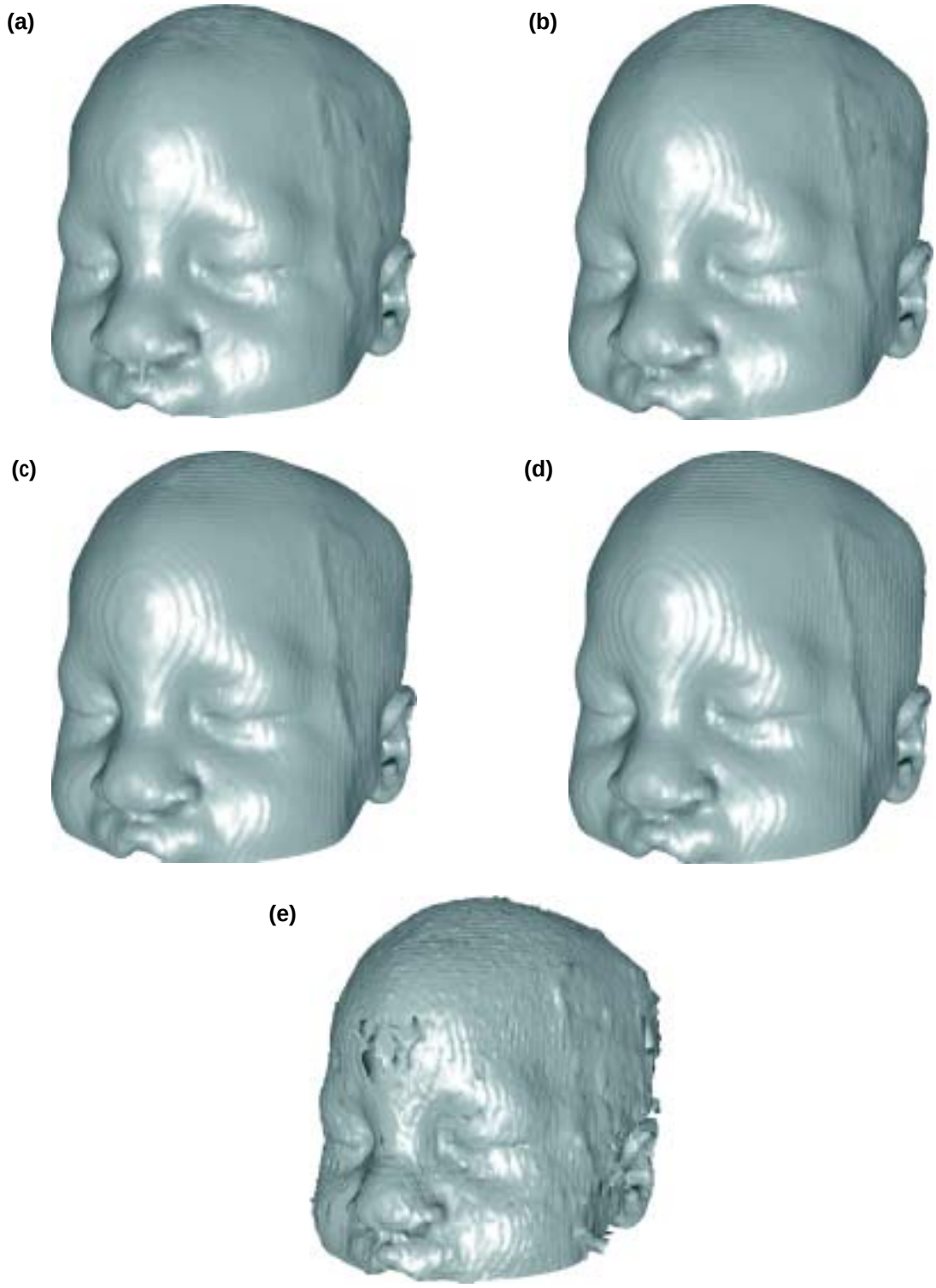


Figure 7.11 Iso-surface extraction of an adaptive volume compression of a child's head CT. **a:** Reconstruction with 6.2 percent of the tetrahedra. **b:** 9.9 percent. **c:** 19.3 percent. **d:** 46.5 percent. **e:** Iso-surface extraction using the standard (linear) marching tetrahedron method from the same approximation that has been used in **d. [AVS©]**

500,000 triangles using a static mesh simplification method provided by AVS/Express 5.0. This simplified mesh has been used as input mesh for the progressive mesh scheme.

Table 7.7 Statistical results for the Almond data set. (see Figure 7.12)

fig	triangles	triangles [%]	geometric MSE [%]
(a)	502	7.84	0.185
(b)	1,002	15.66	0.082
(c)	2,002	31.28	0.034
(d)	3,002	46.91	0.019
(e)	4,002	62.53	0.010
(f)	6,400	100.00	0.000

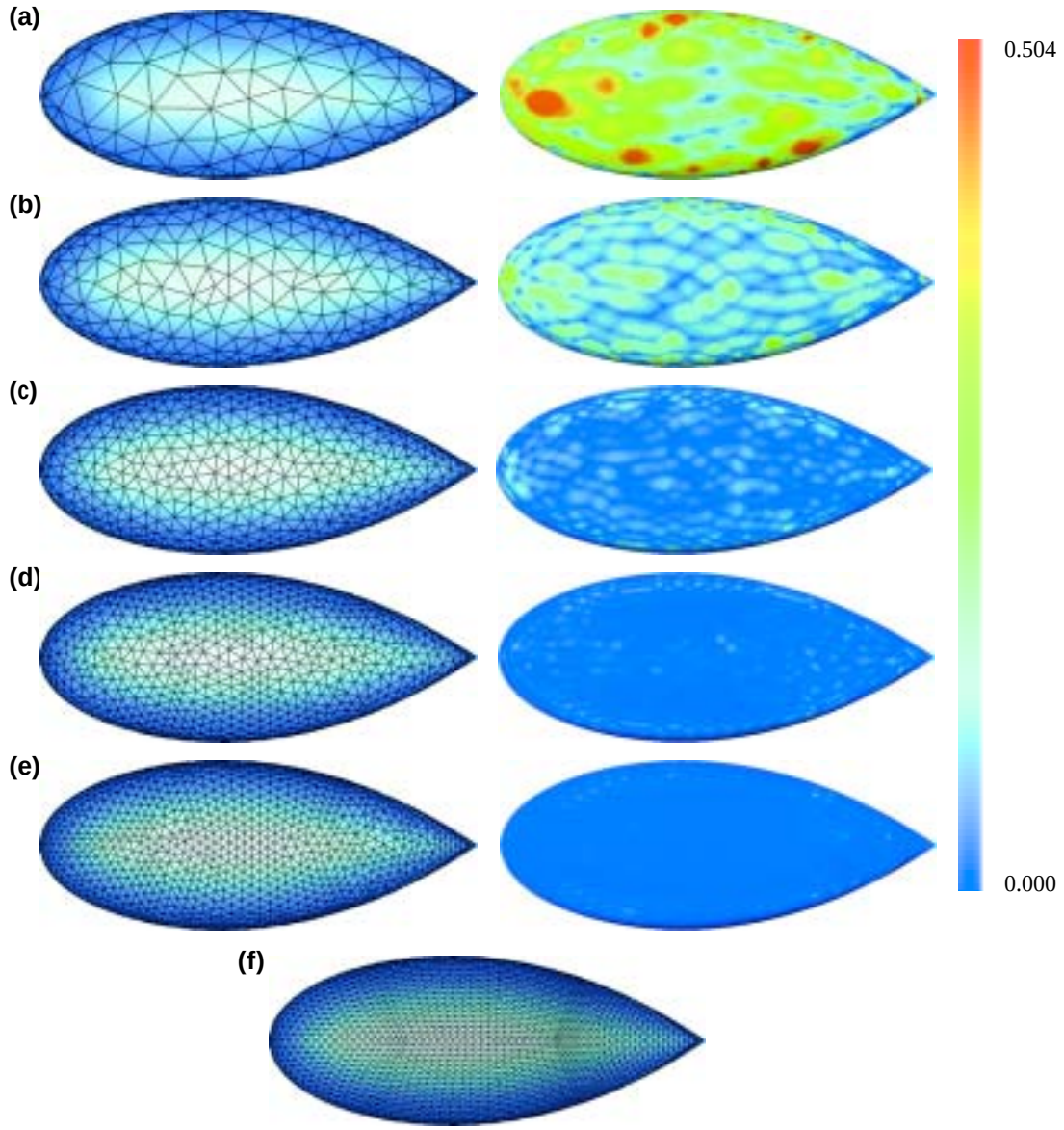


Figure 7.12 Progressive mesh representation of the Almond data set. **a:** 502 triangles. **b:** 1002 triangles. **c:** 2002 triangles. **d:** 3002 triangles. **e:** 4002 triangles. **f:** 6400 triangles. The corresponding errors are mapped onto the meshes on the right side. (see Color Plate 1 on page 151). [AVS©]

As with the previous example, we have employed the same three cost functions to transform the model into a progressive mesh representation, which took 786 seconds. Statistical results including geometric error and reconstruction time are listed in Table 7.8. Again, timings are measured on an SGI Octane R10000@195MHz.

The error visualization shows the high quality of the approximation. Note that most of the error is first introduced in the hair region and around the eyes. The curly hair is highly detailed and topologically complex.

This data set, however, points out a fundamental limitation of geometry-based surface approximations: This tremendously complex model is very difficult to approximate with mesh representations efficiently. Other models which have been acquired in the Digital Michelangelo Project even feature hundreds of billions of triangles, most of which are in sub-pixel resolution. If those modes are used for rendering purposes only without the requirement for connectivity information, pixel-based rendering methods might prove successful [79, 62].

Table 7.8 Statistical results for the progressive mesh representation of David's head form the Digital Michelangelo Project. (see Figure 7.13)

fig	triangles	triangles [%]	reconstruction time [s]	geometric MSE [%]
(a)	500,000	100.00	1.38	0.000
(b)	204,851	40.97	0.44	0.020
(c)	104,859	20.97	0.18	0.039
(d)	54,872	10.97	0.08	0.065

7.2.2 Progressive Tetrahedralizations

For the following investigations, an irregular mesh of a turbine blade was selected. The original data set consists of 576,576 tetrahedra with scalar node data representing pressure between the blades. Figure 7.15 shows results with various levels of reconstruction, different settings of the cost functions, and extracted iso-surfaces, both for the original mesh and for a selected subset with only one blade. Table 7.9 lists the performance statistics and parameter settings used to generate the example meshes. M^n denotes the original number of tetrahedra and M^i is the number of tetrahedra of the reconstructed mesh. ΔE_{grad}, ΔE_{vol}, and ΔE_{equi} indicate the individual cost function terms used for the example. *Time* shows the computation time for a full mesh collapse, *i* the number of vsplits, and *Hits* the number of dynamically detected intersections ($s = 2$). The performance of our algorithms was measured on an SGI Octane R10000@195MHz. The reconstruction time for the examples was between 0.1s and 0.8s, depending on the number of splits.

Figure 7.14 shows iso-surfaces of the blades, extracted from M^{15000} at iso-value 5.0. A slice through the mesh is displayed in order to depict the irregularity of the mesh reconstruction. Note especially the high quality of the iso-surfaces which confirms the effectiveness of ΔE_{grad}. A subset of the mesh with one blade is shown in Figure 7.15a-b for two different resolutions. In order to balance the individual terms of ΔE, the weights were set to $w_{\text{grad}} = 1$, $w_{\text{vol}} = 500$, and $w_{\text{equi}} = 200$, respectively.

In Figure 7.15c, the mesh is cut to render its internal structure. In order to emphasize the influence of individual cost function terms we computed Figure 7.15d-f. We observe

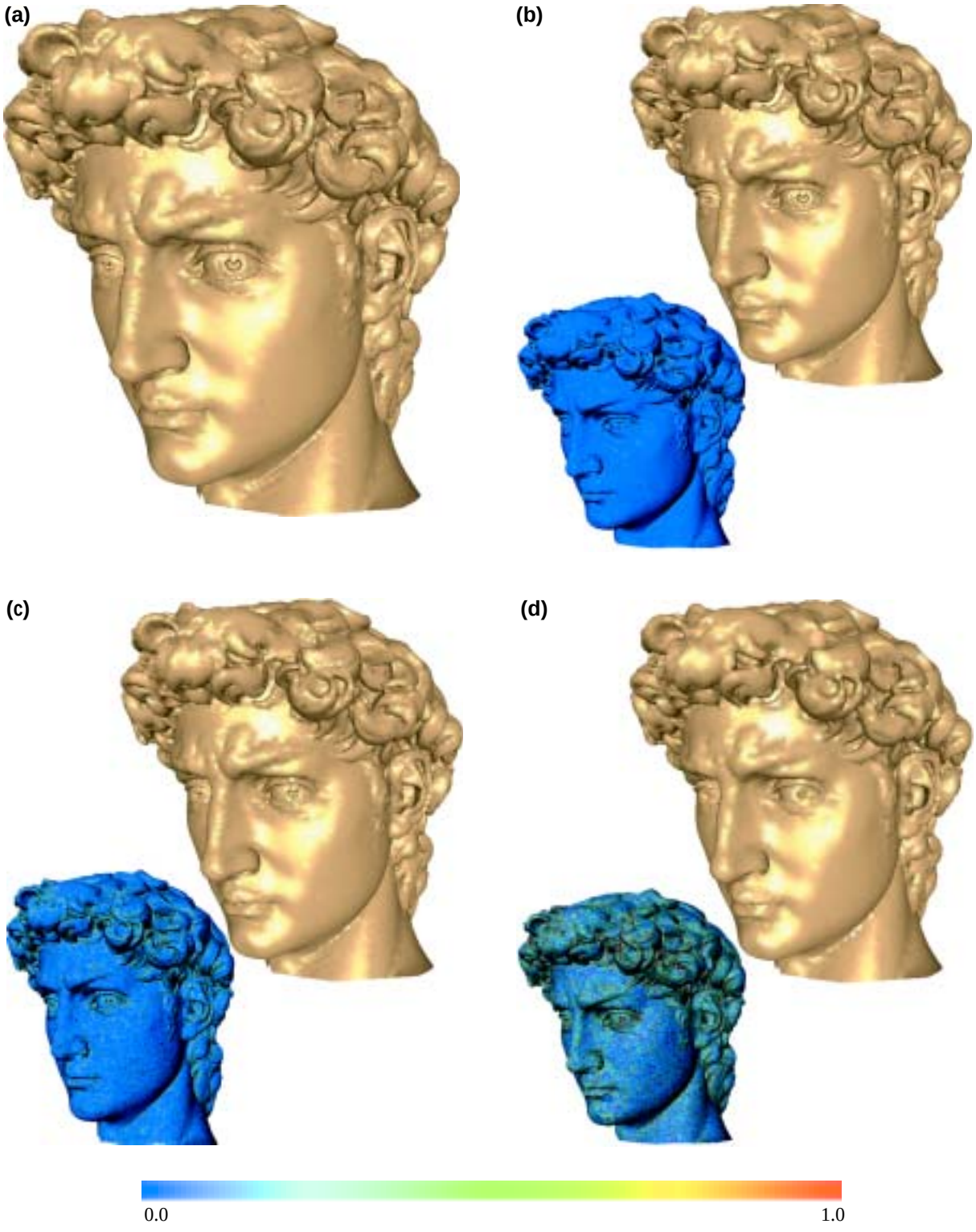


Figure 7.13 Progressive mesh approximation of the head of Michelangelo's David. **a:** Input mesh. **b:** Approximation comprising 40.97 percent of the triangles. **c:** 20.97 percent. **d:** 10.97 percent. [Michelangelo©] (see Color Plate 3 on page 153).

that each of the energy terms stands for a specific feature. ΔE_{grad}, for instance in Figure 7.15d reconstructs the blade region very well, but violates volume preservation and

Table 7.9 Parameter settings and performance statistics for the results shown in Figure 7.15.

fig.	M^n	M^i	E_{grad}	E_{vol}	E_{equi}	time [s]	i	hits
(a)	576,576	117,139	✓	✓	✓	4h57'	15,000	40,415
(b)	78,624	8,317	✓	✓	✓	34'23"	1,000	6,040
(c)	78,624	13,327	✓	✓	✓	34'23''	2,000	6,040
(d)	78,624	13,327	✓	✓	✓	34'23''	2.000	6,040
(e)	78,624	10,554	✓	×	×	19'26''	1,000	2,386
(f)	78,624	7,440	×	✓	×	41'58''	500	7,919
(g)	78,624	8,169	×	×	✓	39'35''	1,000	4,798

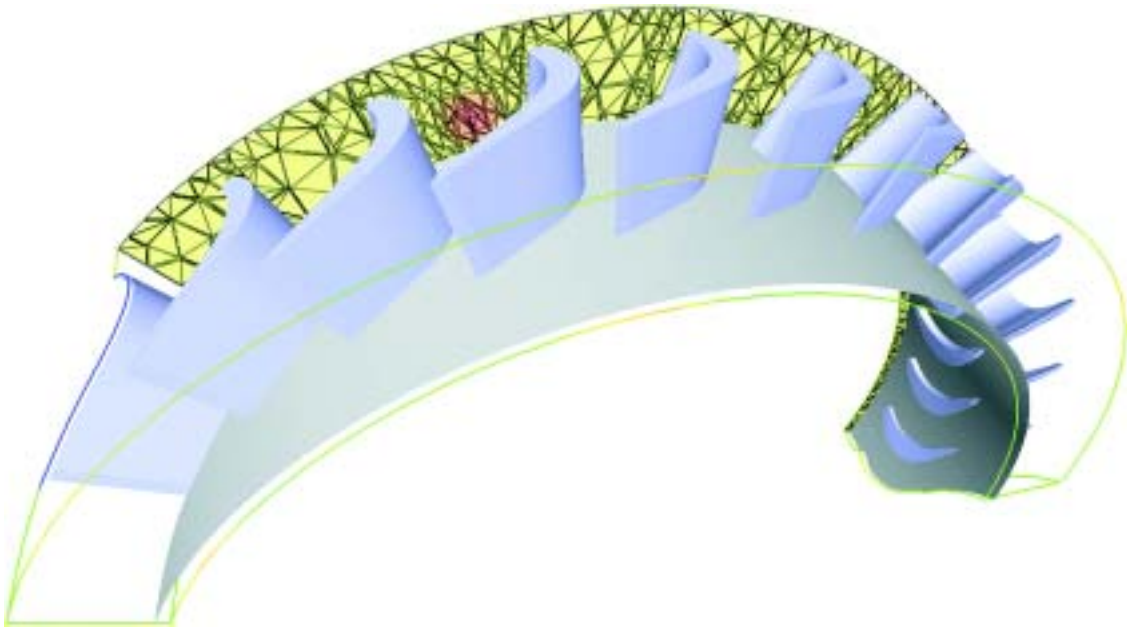


Figure 7.14 Extraction of iso-surfaces of the turbine blades using our higher-order marching tetrahedra method. [AVS©]

produces poorly shaped simplices. As expected, ΔE_{vol} sustains the volume, whereas ΔE_{equi} preserves the equilateral shape of the tetrahedra.

Usually, it can not be guaranteed that all intersections are detected with $s = 2$. For the above example the total number of intersection hits with $s \to \infty$ is 6,116, which means that 98.8% of the intersections have already been detected at iteration level two.

7.3 COMPARISON

The concluding section of this chapter includes two comparative experiments for the new surface and volume approximation methods developed in this thesis. We will analyze potential strengths and weaknesses of each of the methods. This comparison will focus on the geometric approximation performance of the three methods. Although the PT method can process a wide range of unstructured surface and volume data sets, we will restrict these

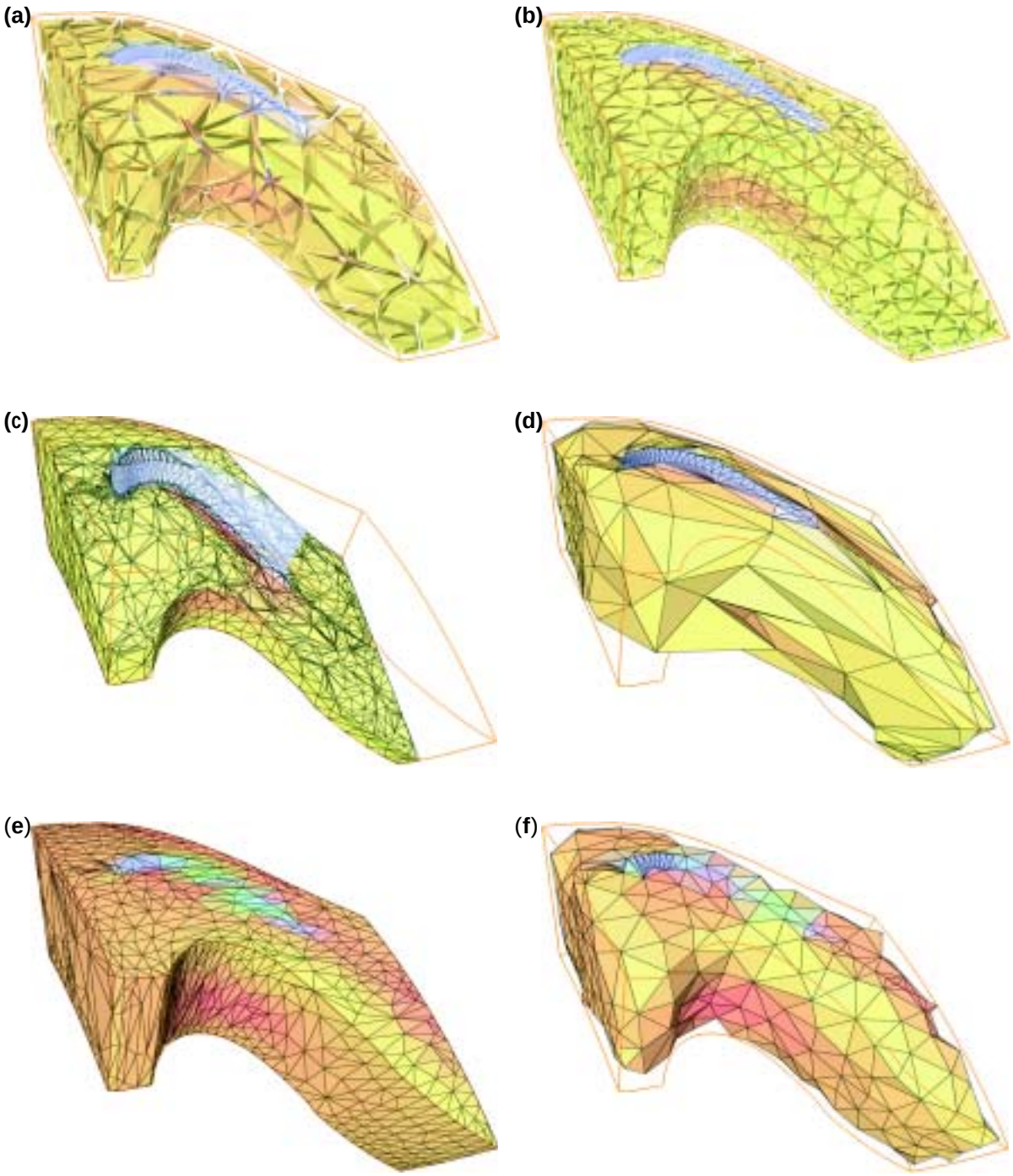


Figure 7.15 PT representations of an irregular turbine blade mesh. **a, b:** Part of the data set at different reconstruction levels with shrunk tetrahedra. **c:** Part of the mesh is cut to render its internal features. **d:** PT with ΔE_{grad}. **e:** ΔE_{vol}. **f:** ΔE_{equi} only. Parameter settings and performance statistics are listed in Table 7.9. [AVS©]

experiments to uniform input data, due to the constraints of the tensor product wavelet transform used by the other two methods.

7.3.1 Surface Approximation

In this section, we will investigate the approximation behavior of the quadtree-based vertex removal, the Delaunay-based vertex insertion and progressive meshes. We have selected the Matterhorn data set, which has been cropped to 256 by 256 vertices. For the transformation into the progressive mesh representation, the following cost function terms have been used: Normal energy, edge length energy and equilateral energy, weighted with 50, 1 and 5, respectively. The total transformation took 2'58" on an SGI Octane R10000. The reconstruction parameters for all three methods have been adjusted in order to yield the same number of triangles for each of the five reconstruction levels. The geometric approximation errors have been calculated with *Metro* [17] and the error visualization is depicted in Figure 7.16. Numerical results for this experiment are listed in Table 7.10. In addition to the absolute mean-square error, we have included the relative error with respect to bounding box of the model.

Table 7.10 Error statistics for the Matterhorn comparison experiment.

triangles	progressive mesh		vertex insertion		vertex removal	
	MSE	MSE [%]	MSE	MSE [%]	MSE	MSE [%]
89,480	0.01455	0.003899	0.0245	0.006562	0.1183	0.03159
70,740	0.02507	0.006717	0.03019	0.008087	0.1591	0.04249
55,449	0.03517	0.009422	0.03846	0.0103	0.2079	0.0555
25,878	0.07775	0.02082	0.07688	0.02059	0.3804	0.1015
15,529	0.118	0.03163	0.1188	0.03184	0.5516	0.1473

We can see from the statistics and the error visualization that both, the progressive mesh and the Delaunay-based vertex insertion methods outperform the quadtree-based vertex removal. Due to the greater adaptivity of the first two methods, they achieve better approximation results than the latter method, which is constraint by the two-level look-up-table triangulation. A closer look at the quadtree-based method, however, reveals that its performance is still satisfactory. The relative error at the coarsest approximation level is below 0.15 percent.

7.3.2 Volume Approximation

For the final experiment, the child's head data set has been employed. We have cropped the original data to a representative fragment with 32 by 32 by 32 voxels or 178,746 tetrahedra. Figure 7.17 depicts a semi-transparent view of this model.

This data set has been processed with both, the Delaunay-based vertex insertion and the progressive tetrahedralization methods. Performance statistics for this experiment are listed in Table 7.11 and corresponding graphs are shown in Figure 7.18. The timing figures for the PT do not include the initial transformation into the PT representation, which took 10'40" without and 41'3" with full intersection testing. Three tetrahedral intersections were detected during the process and the corresponding collapse operations have been evaded. The original mesh and its approximation have both been sampled at 1,000,000 randomly distributed positions to determine the approximation error.

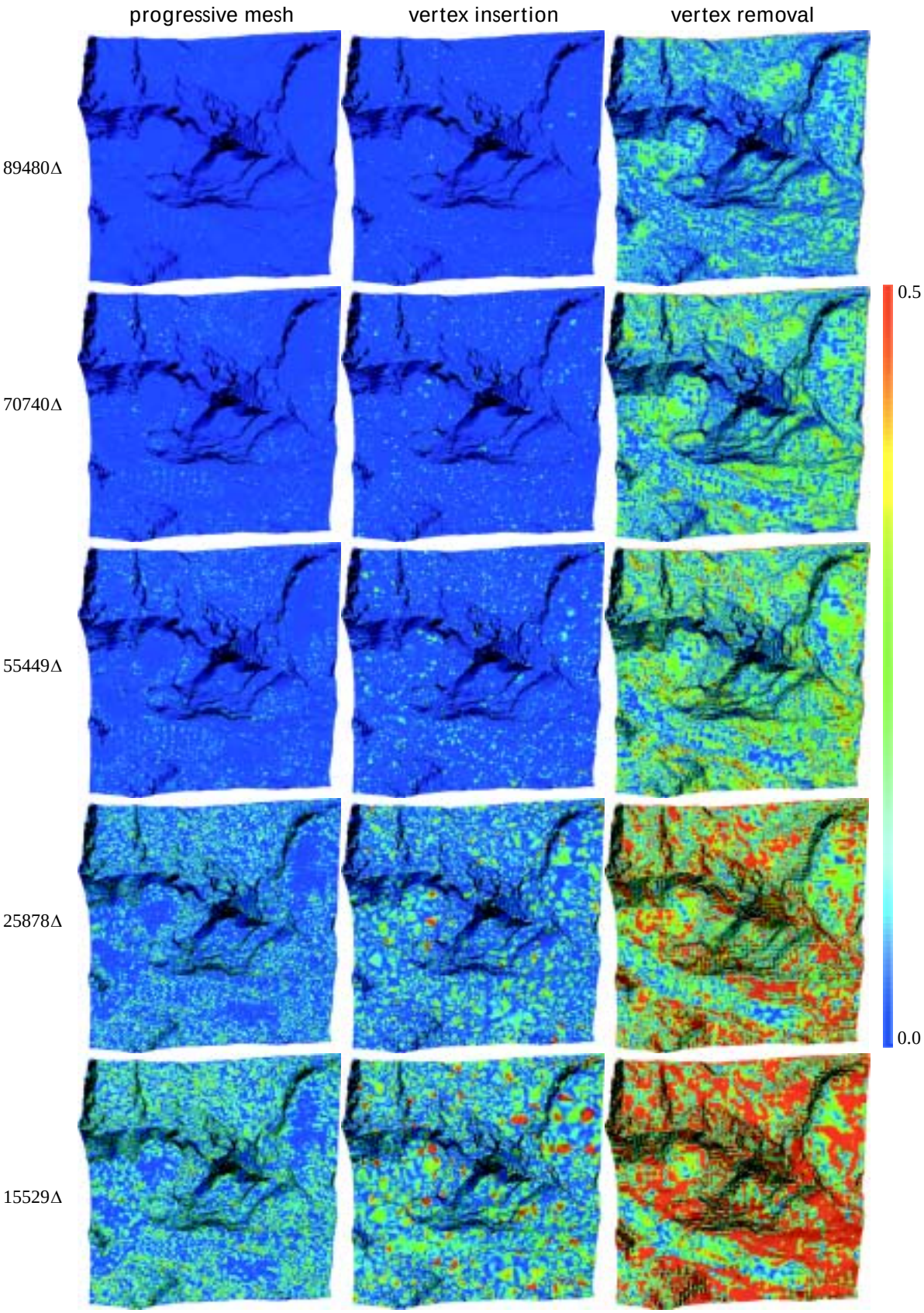


Figure 7.16 Error visualization of the Matterhorn model for the three different mesh approximation methods at various reconstruction levels. The magnitude of the absolute mean-square geometric error corresponds to the colorbar on the right. (see Color Plate 4 on page 154).

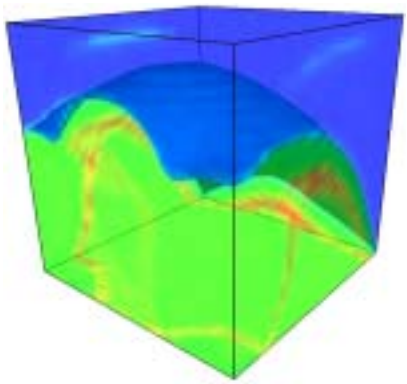


Figure 7.17 Fragment of the child's skull data set which is employed for the comparison of the Delaunay-based vertex insertion and the progressive tetrahedralization methods. [AVS©]

It is interesting to see that the lines of the distortion graph for both methods take an almost identical course. The performance of the PT methods, however, drops off for the last sample at 10,000 tetrahedra. This behavior can be explained by the fact that the order of priority of the edge collapses is determined by a greedy algorithm. After large number of successful edge collapse operations, it becomes more an more difficult to select edges with low cost without compromising the correct topology of the mesh. This fact can usually be neglected for two-manifold meshes where the topology is much simpler, but it has a considerable influence on the volumetric setting. A global optimization scheme would certainly improve the mesh quality at low approximation levels, but this is probably not feasible for large applications.

Without taking preprocessing into account, the PT method clearly outperforms the Delaunay-based vertex insertion scheme in terms of reconstruction time by almost two orders of magnitude. The bottleneck of the latter scheme, however, is the Delaunay tetrahedralization which only performs in $O(N \log N)$, where N is the number of voxels. A faster tetrahedralization scheme could significantly improve the overall performance.

Table 7.11 Performance statistics for the volume comparison experiment.

tetrahedra	progressive tetrahedralizations		vertex insertion	
	PSNR [dB]	time [s]	PSNR [dB]	time [s]
178,746	81.33	0.50	75.37	20.0
120,000	58.06	0.31	54.16	14.5
100,000	55.75	0.24	53.58	12.0
80,000	53.35	0.18	53.06	11.2
60,000	51.76	0.11	51.09	10.0
40,000	43.64	0.08	48.59	4.6
20,000	42.77	0.05	45.04	2.5
10,000	22.92	0.01	40.52	1.6

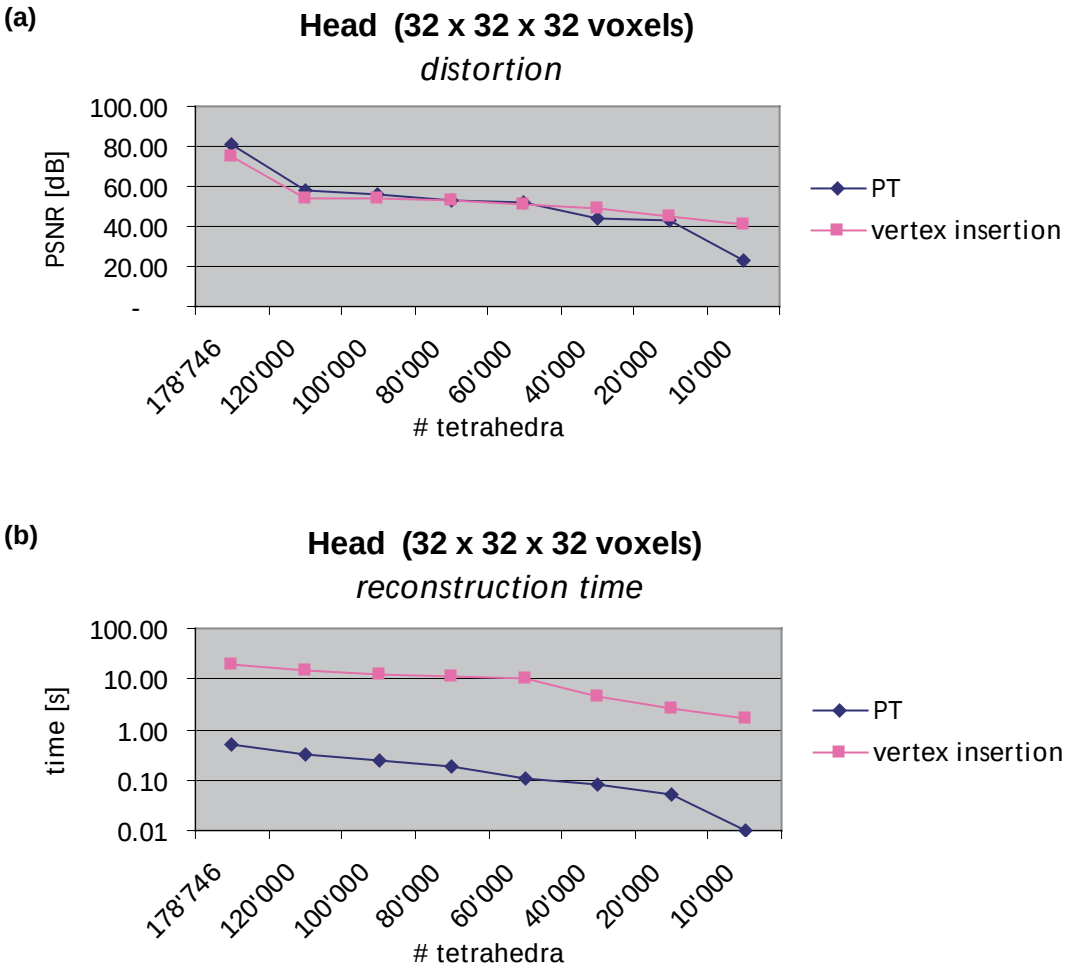


Figure 7.18 Performance statistics for the volume comparison experiment. **a:** PSNR of both methods for decreasing number of tetrahedra. **b:** Reconstruction times (note the logarithmic scale of the time axis).

8

CONCLUSIONS AND OUTLOOK

This chapter summarizes the results achieved in this thesis and gives outlook on further research directions and possible extensions.

8.1 CONCLUSIONS

In this dissertation, we have devised a new geometry compression and reconstruction pipeline with support for uniform data in two- and three-dimensional settings. Our work is based on the powerful theory of multi-resolution B-spline wavelets. Two different hierarchical mesh simplification schemes, governed by the underlying wavelet approximation, have tightly been integrated. Results from these methods have been compared to the popular progressive mesh algorithm and our novel extension to unstructured tetrahedral meshes.

Wavelet-based geometry compression. In Chapter 5, we have presented a new geometry compression pipeline. The combination of wavelet theory (Chapter 2) and different compression methods (Chapter 4), which have successfully been employed in other application fields, lead to a new approach to geometry compression for uniform data in multiple dimensions. The underlying B-spline bases ensure smooth and continuous approximations while providing good compression performance.

We have presented an efficient implementation of the fast wavelet transform, taking boundary problems into account: Whenever dealing with bounded manifolds, basis functions overlapping the boundary are likely to arise. When using truncated approximations, most standard solutions will introduce more or less artifacts in the reconstructed model. We have chosen a solution incorporating endpoint-interpolating B-spline bases, which are constructed by multiple knot insertion at the boundaries and that completely prevent us from having boundary distortions. Careful implementation allows for a wavelet transform without performance loss when compared to the standard fast wavelet transform.

Several oracles, which select a relevant subset of basis functions for reconstruction, have been introduced. Global oracles analyze the data in wavelet domain and reject an applica-

tion-specific percentage of wavelet coefficients in the least-squares sense. They directly influence the compression performance of the whole pipeline by replacing adjacent small coefficients with sequences of zeros, which are subject to further encoding. A complement to global oracles are local oracles: Instead of globally specifying a certain percentage of basis functions to be rejected, spatial regions, where all significant information is to be retained, can be defined by the user. Loss of information only occurs outside of those regions. Hence, local oracles provide for selective refinement of the reconstruction and can be looked upon as electronic magnifying glasses.

Although not specifically designed and optimized for image compression, we have shown in Section 5.4.5 and Section 7.1.4 that our pipeline performs remarkably well for progressive compression of texture attributes on digital terrain data.

Subsequent to global and local oracle operations, the resulting sequences of zero-valued coefficients are encoded using run-length coding an entropy coding.

We have demonstrated the performance of the geometry compression pipeline in Section 7.1 for data sets from different application domains in two and three dimensions.

Bottom-up vertex removal. After decompression of the data by reversing the steps described above, we have employed adaptive mesh reconstruction using a fast an elegant method introduced in Section 5.3. The analysis of vertices is carried out by stepwise progressive reconstruction of the wavelet coefficient channels from fine to coarse level. This enables us to reject vertices at dyadic positions, resulting in the construction of a quadtree data structure.

Consistent and efficient triangulation of the quadtree could be guaranteed by the use of a novel two-level look-up-table, which ensures crack-free meshing.

The main advantage of this approach is its fast and elegant implementation, which does not require any geometric calculations upon triangulation. Due to the constraints of the look-up-table, however, the semi-structured approximation limits the adaptivity of the resulting mesh. This is reflected in the numerical results presented in Section 7.3, which show that we cannot achieve the same performance as with the other two methods presented in this thesis.

Furthermore, the bottom-up nature of the algorithm prevents us from allowing progressive transmission and reconstruction. Before we can start with the construction of the quadtree, we have to receive all detail information at the finest level, which comprises two thirds of the original data (uncompressed).

Those drawbacks have motivated us to investigate alternative reconstruction strategies which overcome these problems while still allowing tight integration into our compression pipeline.

Top-down vertex insertion. The Delaunay-based top-down vertex insertion method described in Section 5.4 provides a flexible and adaptive geometry reconstruction scheme. Inspired by the popular Douglas-Peuker algorithm for one-dimensional curve simplification, we have devised a new scheme for higher-dimensional settings that has been carefully adapted to our needs.

Using a bottom-up instead of a top-down approach opens the possibility for channel-wise progressive reconstruction. By transmitting coarse wavelet channels first, which only comprise a small subset of the total number of coefficients, we can refine the geometric representation adaptively. Since we are not constrained by a look-up-table, but employ Delaunay triangulation to determine the mesh connectivity, we can generate approxima-

tions which adapt better to the original data. In terms of the geometric approximation error, this method clearly outperforms the quadtree-based scheme (see Section 7.3).

On the other hand, the algorithmic complexity of the vertex insertion method, which is largely determined by the cost of the Delaunay triangulation, is considerably higher (see Section 5.6.2). Hence, for very large data sets and applications where fast triangulation on the fly is more important than very low approximation error, the quadtree-based method is still attractive.

Progressive tetrahedralizations. Progressive mesh representations have proven to be an excellent alternative to wavelet-based approximations in a variety of application fields. Inherently, they provide support for unstructured input meshes, which is only difficult to achieve with tensor product constructions of higher-dimensional multi-resolution methods.

In order to compare our wavelet-based geometry reconstruction scheme with the progressive mesh method, we have devised an extension of progressive meshes to progressive tetrahedralizations (Chapter 6). Mesh consistency problems such as flipping of tetrahedra or intersection, which violate the manifold property of a mesh, frequently arise in the tetrahedral setting and have consequently been addressed in Section 6.4.

The accuracy of the progressive mesh and the progressive tetrahedralization methods is comparable to that of our Delaunay-based vertex insertion scheme (see Section 7.3). Direct comparison for surface triangulations has shown that we could even achieve better results with progressive meshes. This does not come as surprise, because we were able to employ data-specific cost functions which govern the transformation from the input mesh into the progressive mesh representation.

Similar results could be obtained for tetrahedral meshes. For very coarse approximations, however, we have realized that the greedy method that determines the ordering of the edge collapse operations, is additionally constrained by the topological considerations described earlier. For that reason, we could not obtain the same accuracy as with our vertex insertion scheme under those circumstances.

Another advantage of progressive tetrahedralizations, besides its fine grain progressivity, is its very short reconstruction time—even for larger models. This clearly compensates for the time consuming transformation comprising cost function evaluations, consistency checks and edge collapses, which can be carried out offline as a preprocessing step, though.

8.2 OUTLOOK

In this dissertation we have presented an entire geometry compression and reconstruction pipeline for uniform input data. We have demonstrated the versatility of our approach with multiple data sets from different application fields.

Notwithstanding the positive results derived from this work, we believe that there is still room for improvement.

The use of alternate basis functions should be investigated. Despite the fact that we were able to successfully apply cubic B-spline basis in our approximation scheme, other bases might be designed, which show even better compression behavior while featuring smaller compact support. Moreover, a possible extension to unstructured settings should be taken into considerations. The tensor product approach to the construction of higher-dimensional B-spline bases currently prohibits us from processing data with non-uniform

parametrization. The are two conceivable ways to attack this problem: Appropriate basis function with inherent support for scattered data could be devised. The methods proposed in [99, 16] point towards that direction. A different approach could be to subdivide the input mesh into small patches that are sufficiently flat. Those patches could be further processed in a fashion similar to height fields. The necessary reparametrization to obtain uniformly distributed vertices could be carried out in advance.

Another interesting area of future research is the investigation of out-of-core methods. Very large terabyte-scale models, often the result from complex numerical simulations, cannot be processed as a whole in main memory. Efficient methods could be developed, which allow partial loading and processing of data without suffering severe performance loss. A method for out-of-core mesh simplification has recently been presented in [64].

The oracles and reconstructions schemes that have been devised in this thesis are governed by the L^2-norm. Additional criteria based on the human perceptual system or on screen-space error could further optimize the quality of the mesh approximations when primarily employed for rendering purposes. The design and implementation of our progressive tetrahedralization method provides an flexible interface for additional error criteria.

Finally, hierarchical point-based approximation and reconstruction methods might present a very promising alternative to mesh-based representations. For visualization purposes, screen resolution is an upper bound of the required complexity of geometric models: At resolutions where the average size of a single triangle is at or even below pixel size, connectivity information between vertices becomes more and more irrelevant and could thus be entirely omitted. If the model has been sampled at very high rates, however, a hierarchical representation would provide the possibility to zoom into the data up to sampling resolution without loosing details.

A

SOFTWARE COMPONENTS

The following appendix provides a short overview of the software system which has been developed during the coarse of this thesis. To ensure high flexibility of our prototype, we have realized our methods as software components within the AVS/Express framework.

AVS/Express [1] is a commercially available multi-platform, component-based software environment with strong emphasis on building scientific visualization applications. The base system provides a large number of standard components at different abstraction levels, as well as flexible user-extendable data structures, suitable for a multitude of application fields. The internal data model is based on a “data-reference” architecture, where individual components of hierarchically defined data structures can be referenced and used as input for computational objects. Component libraries with support for data input and output, data analysis and processing, as well as various data display techniques are available.

Furthermore, a C++ application programming interface (API) enables one to seamlessly integrate new components to the system. Components can basically be defined at two levels:

- *Module:* A compound data structure that aggregates any number of basic data types, other modules and methods. Modules are mapped to C++ classes and can thus be used directly in C++ applications. See Figure A.1 for an example of a user-defined module.
- *Macro:* Combinations of modules, macros and connections. Macros are used to build multi-level hierarchies of computational and user interface components. Figure A.2 depicts an example of macro for wavelet-based data compression. High level macros represent the user application. Connection between macros and modules, which define the data flow of the application, can be made interactively using a network editor. Only when the output of one component changes, other components that are connected with the first are triggered and existing update methods are executed.

We have developed a total of 158 modules and 471 macros for the methods presented in this thesis. This includes low-level modules such as the fast wavelet transform or the global

Figure A.1 AVS/Express module for the B-spline wavelet transform component. Different data types are color-coded.

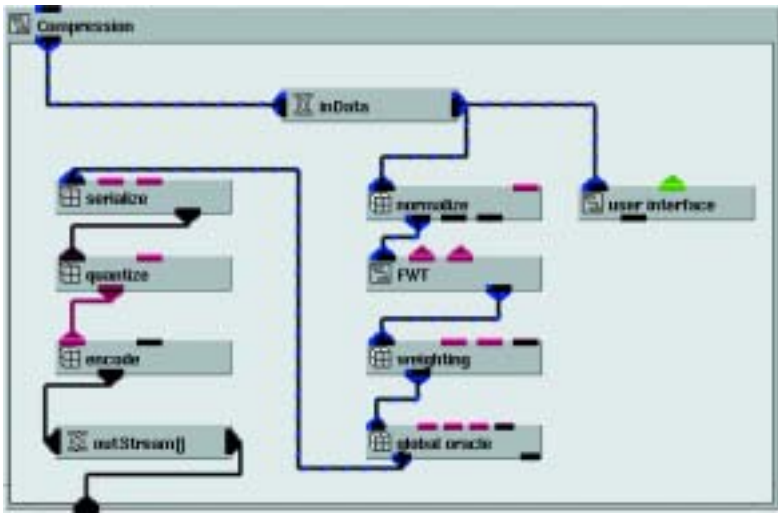


Figure A.2 AVS/Express macro for data compression. The individual components for the compression pipeline are connected forming a data-reference graph.

oracle, as well as high-level macros that encompass the full geometry compression pipeline including user interface.

We have combined these components together with AVS/Express standard modules and macros to build sophisticated yet flexible application networks. Figure A.3 shows the top-level components of an application that transforms a geometric models into a progressive mesh representation. The reconstructed mesh is rendered using the provided OpenGL rendering component. The full application window corresponding to the example network is depicted in Figure A.4.

Figure A.3 Six components have been connected to build an application for transforming a data model into the progressive mesh representation.

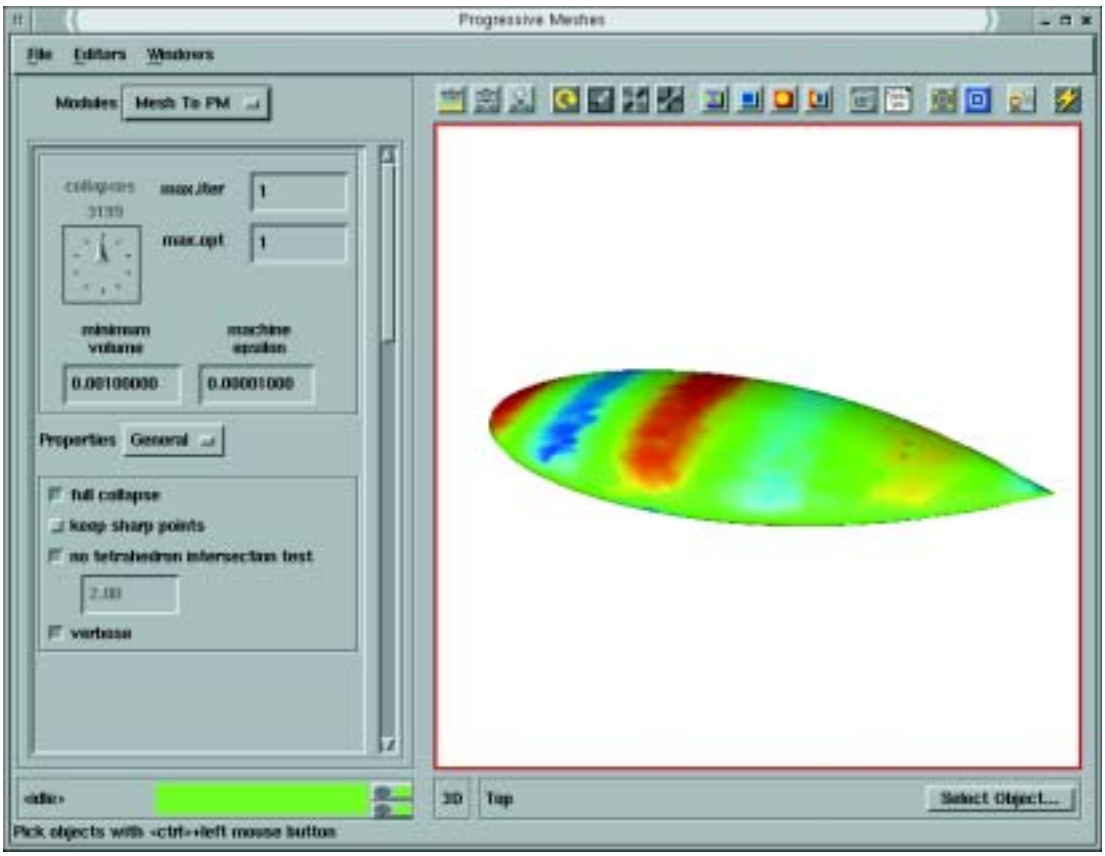


Figure A.4 User interface for the progressive mesh application assembled within AVS/Express.

B

NOMENCLATURE

CHAPTER 2

m_0 . . . . . initial approximation level

M . . . . . maximum approximation level

m . . . . . current approximation level

p, q, r . . . . . basis function translation parameters

j . . . . . order of B-spline basis

$\phi(x)$ . . . . . scaling mother-function

$\tilde{\phi}(x)$ . . . . . dual scaling function

$\phi^{[j]}(x)$ . . . . . B-spline scaling function of order j

$\tilde{\phi}^{[j]}(x)$ . . . . . dual B-spline scaling function of order j

$\phi_{mp}(x)$ . . . . . scaling function (dilation m, shift p)

$\phi^{n}(x)$ . . . . . n-dimensional scaling function

$\tilde{\phi}^{n}(x)$ . . . . . dual n-dimensional scaling function

V_m . . . . . subspace spanned by $\phi_{mp}(x)$

c_{mp} . . . . . scalar coefficient corresponding to $\phi_{mp}(x)$

$V_m^{[j]}$ . . . . . subspace spanned by $\phi_m^{[j]}$

$\psi(x)$ . . . . . wavelet mother-function

$\psi(x)$ . . . . . dual wavelet function

$\psi^{[j]}(x)$ . . . . . B-spline wavelet function of order j

$\tilde{\psi}^{[j]}(x)$ . . . . . dual B-spline wavelet function of order j

$\psi_{mp}(x)$ . . . . . wavelet function (dilation m, shift p)

$\psi^{n}(x)$ . . . . . n-dimensional wavelet function

$\tilde{\psi}^{n}(x)$ . . . . . dual n-dimensional wavelet function

W_m . . . . . subspace spanned by $\psi_{mp}(x)$

d_{mp} . . . . . scalar coefficient corresponding to $\psi_{mp}(x)$

$\mathrm{supp}(f)$ . . . . support of function f

δ_{pq} . . . . . Kronecker-delta function

CHAPTER 3

A, B . . . . . simplices

C . . . . . cell

X, Y . . . . . polehedra

K . . . . . simplicial complex, cell complex

V . . . . . set of vertex positions

$\mathbf{v}_i$ . . . . . vertex

$\mathbf{p}_i$ . . . . . point

$\mathrm{star}(A)$ . . . . . *star* of complex A

$\mathrm{faces}(A)$ . . . . *faces* of complex A

M . . . . . manifold

$\partial\cdot$ . . . . . boundary

$\mathrm{clos}(\cdot)$ . . . . . transitive closure

$\cdot\downarrow\cdot$ . . . . . elementary collapse

$\cdot\Downarrow\cdot$ . . . . . collapse

$\mathrm{ext}(x)$ . . . . . number of vertices of a structured mesh along the x-axis

E^d . . . . . d-dimensional Euclidian space

$d(\mathbf{p}_i, \mathbf{p}_j)$ . . . . Eucidian distance between $\mathbf{p}_i$ and $\mathbf{p}_j$

S . . . . . set of points in E^d

b_{ij} . . . . . perpendicular bisector of $\mathbf{p}_i$ and $\mathbf{p}_j$

H_{ij} . . . . . halplane containing $\mathbf{p}_i$ that is defined by b_{ij}

V_i . . . . . Voronoi polygon assiciated with $\mathbf{p}_i$

Vor(S) . . . . . Voronoi diagram of S

$D(S)$ . . . . . Delaunay triangulation of S

RMS($\mathbf{u}$, $\mathbf{v}$) . . *root mean square* error between two vectors

$e(\mathbf{p}, S)$ . . . . . distance between point $\mathbf{p}$ and surface S

$E(S_1, S_2)$. . . . distance between two surfaces

$E_v(S_1, S_2)$ . . . volume difference between two surfaces

$E_m(S_1, S_2)$ . . mean distance between two surfaces

$e'(\mathbf{p}, S)$ . . . . . signed distance between point and surface

$E^+(S_1, S_2)$. . . positive distance error between two surfaces

$E^-(S_1, S_2)$ . . . negative distance error between two surfaces

CHAPTER 4

$\mathbf{X}$, $\mathbf{Y}$ . . . . . data sequences

$\overline{\mathbf{X}}$ . . . . . compressed data sequence

X_i . . . . . data symbol

$P(X_i)$ . . . . . probability of occurance of symbol X_i

$H(\mathbf{X})$. . . . . entropy of sequence $\mathbf{X}$

$c(X_i)$ . . . . . code word of X_i

α_i . . . . . binary string

Δ. . . . . step size of uniform quantizer

$\{h_n\}$ . . . . . impulse response of a low-pass QMF filter

$\{g_n\}$ . . . . . impulse response of a high-pass QMF filter

$\downarrow 2$ . . . . . downsampling by factor two

$\uparrow 2$ . . . . . upsampling by factor two

CHAPTER 5

m . . . . . intermediate decomposition level

M . . . . . maximum decomposition level

Δf^m . . . . . reconstructed detail signal of level m

f^M . . . . . reconstructed scaling functions signal of level M

τ . . . . . global oracle threshold; iso-value

ε . . . . . geometric threshold

$\mathbf{P}$ . . . . . vertex

$\mathbf{s}(u, v)$ . . . . . parametric B-spline surface with parameters u, v

CHAPTER 6

$\mathbf{v}_i$ . . . . . vertex

$\bar{\mathbf{v}}_i$ . . . . . remaining vertex after collapse (*split vertex*)

e_i . . . . . edge

s_i . . . . . scalar vertex attributes

vsplit_i . . . . . ith vetex split operation

ecol_i . . . . . ith edge collapse operation

M^i . . . . . instance of mesh after i vertex splits

$\{icells_i\}$ . . . set of introduced cells for vsplit_i

$\{ncells_i\}$ . . . set of neighbor cells for vsplit_i

$\Delta E(e_i)$ . . . . . cost value associated with edge e_i

T_i . . . . . tetrahedral cell

N_{edges} . . . . . number of edges in mesh M

α . . . . . interpolation parameter for geomorph operation

$G(M^i, M^j, \alpha)$ geomorph from M^i to M^j

C

REFERENCES

[1] Advanced Visual Systems Inc., Waltham, MA, USA. *Using AVS/Express*, 4.0 edition, July 1998.

[2] N. Ahmed and K. R. Rao. *Orthogonal Transforms for Digital Signal Processing.*. Springer-Verlag, New York, 1975.

[3] L. Balmelli. *Rate-Distortion Optimal Mesh Simplification for Communications*. PhD thesis, École Polytechnique Fédérale de Lausanne, 2000.

[4] C. B. Barber, D. P. Dobkin, and H. Huhdanpaa. "Qhull," 1996. http://www.geom.umn.edu/locate/qhull.

[5] D. R. Baum, S. Mann, K. P. Smith, and J. M. Winget. "Making radiosity usable: Automatic preprocessing and meshing techniques for the generation of accurate radiosity solutions." In T. W. Sederberg, editor, *Computer Graphics (SIGGRAPH '91 Proceedings)*, volume 25, pages 51–60, July 1991.

[6] G. Beylkin, R. Coifman, and V. Rokhlin. "Fast wavelet transforms and numerical algorithms i." *Communications on Pure and Applied Mathematics*, 44(2):141–183, Mar. 1991.

[7] J. Bloomenthal. "An implicit surface polygonizer." In P. Heckbert, editor, *Graphics Gems IV*, pages 324–349. Academic Press, Boston, 1994.

[8] K. Bonnell, M. A. Duchaineau, D. Schikore, B. Hamann, and K. I. Joy. "Constructing material interfaces from data sets containing volume fraction information." In *Proceedings of IEEE Visualization 2000*, 2000.

[9] K. Q. Brown. "Voronoi diagrams from convex hulls." *Info. Proc. Lett.*, 9:223–228, 1979.

[10] J. Capon. "A probabilistic model for run-length coding of pictures." *IRE Transactions on Information Theory*, pages 157–163, 1959.

[11] A. Certain, J. Popovi'c, T. DeRose, T. Duchamp, D. Salesin, and W. Stuetzle. "Interactive multiresolution surface viewing." In H. Rushmeier, editor, *SIGGRAPH 96 Conference*

Proceedings, Annual Conference Series, pages 91–98. ACM SIGGRAPH, Addison Wesley, Aug. 1996. held in New Orleans, Louisiana, 04-09 August 1996.

[12] M. Chow. "Optimized geometry compression for real-time rendering." In *Proceedings of the IEEE Visualization '97*, pages 347–354, 1997.

[13] C. Chui. *An Introduction to Wavelets*. Academic Press, 1992.

[14] C. Chui and E. Quak. "Wavelets on a bounded interval." In D. Braess and L. L. Schumaker, editors, *Numerical Methods of Approximation Theory*, pages 1–24. Birkhäuser Verlag, Basel, 1992.

[15] C. K. Chui and J. Z. Wang. "A cardinal spline approach to wavelets." *Proc. Amer. Math. Soc.*, 113:785–793, 1991.

[16] C. K. Chui, J. D. Ward, K. Jetter, and J. Stoeckler. "Wavelets for analyzing scattered data: An unbounded operator approach." *Appl. Comput. Harmon. Anal.*, 3(3):254–267, 1996.

[17] P. Cignoni, c. Rocchini, and r. Scopigno. "Metro: Measuring error on simplified surfaces." *Computer Graphics Forum*, 17(2):167–174, June 1998.

[18] P. Cignoni, C. Montani, E. Puppo, and R. Scopigno. "Multiresolution representation and visualization of volume data." *IEEE Transactions on Visualization and Computer Graphics*, 3(4):352–369, Dec. 1997.

[19] A. Cohen, I. Daubechies, and J. Feauveau. "Bi-orthogonal bases of compactly supported wavelets." *Comm. Pure and Applied Mathematics 45*, pages 485–560, 1992.

[20] A. Cohen, I. Daubechies, and P. Vial. "Wavelets on the interval and fast wavelet transforms." *Applied and Computational Harmonic Analysis*, 1:54–81, 1993.

[21] J. Cohen, A. Varshney, D. Manocha, G. Turk, H. Weber, P. Agarwal, F. P. B. Jr., and W. Wright. "Simplification Envelopes." In *Computer Graphics Proceedings, Annual Conference Series, 1996 (ACM SIGGRAPH '96 Proceedings)*, pages 119–128, 1996.

[22] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press, Cambridge, Massachusetts, 1994.

[23] A. Croisier, D. Esteban, and C. Galand. "Perfect channel splitting by use of interpolation/decimation techniques." In *Proceedings International Conference on Information Science and Systems.*, Piscataway, NJ, 1976. IEEE Press.

[24] I. Daubechies. "Orthonormal basees of compactly supported wavelets." *Communications of Pure and Applied Mathematics*, 41:909–996, 1988.

[25] I. Daubechies. *Ten Lectures on Wavelets*. CBMS-NSF regional conference series in applied mathematics, no.61, SIAM, 1992.

[26] M. F. Deering. "Geometry compression." In R. Cook, editor, *SIGGRAPH 95 Conference Proceedings*, Annual Conference Series, pages 13–20. ACM SIGGRAPH, Addison Wesley, Aug. 1995. held in Los Angeles, California, 06-11 August 1995.

[27] B. Delaunay. "Sur la sphère vide." *Bull. Acad. Sci. USSR(VII), Classe Sci. Mat. Nat.*, pages 793–800, 1934.

[28] T. Dey, H. Edelsbrunner, S. Guha, and D. Nekhayev. “Topology preserving edge contraction.” Technical report, Raindrop Geomagic Inc., Research Triangle Park, North Carolina, 1998.

[29] D. Douglas and T. Peucker. “Algorithms for the reduction of the number of points required to present a digitized line or its caricature.” *The Canadian Cartographer*, 10(2):112–122, December 1973.

[30] E. Quak and N. Weyrich. “Decomposition and reconstruction algorithms for spline wavelets on a bounded interval.” *Appl. Comput. Harmon. Anal.*, 1(3):217–231, 1994.

[31] M. Eck, T. DeRose, T. Duchamp, H. Hoppe, M. Lounsbery, and W. Stuetzle. “Multiresolution analysis of arbitrary meshes.” In R. Cook, editor, *SIGGRAPH 95 Conference Proceedings*, Annual Conference Series, pages 173–182. ACM SIGGRAPH, Addison Wesley, Aug. 1995. held in Los Angeles, California, 06-11 August 1995.

[32] F. Evans, S. Skiena, and A. Varshney. “Optimizing triangle strips for fast rendering.” In *Proceedings of the IEEE Visualization ’96*, pages 319–326, 1996.

[33] A. Finkelstein and D. H. Salesin. “Multiresolution curves.” In A. Glassner, editor, *Proceedings of SIGGRAPH ’94 (Orlando, Florida, July 24–29, 1994)*, Computer Graphics Proceedings, Annual Conference Series, pages 261–268. ACM SIGGRAPH, ACM Press, July 1994. ISBN 0-89791-667-0.

[34] A. Fournier. “Wavelets and their applications in computer graphics.” Course Notes SIGGRAPH ’94, 1994.

[35] M. Garland and P. S. Heckbert. “Surface simplification using quadric error metrics.” In *Computer Graphics (SIGGRAPH ’97 Proceedings)*, pages 209–216, Aug. 1997.

[36] R. Gatti. “Effiziente Darstellung von dreidimensionalen Geländedaten mittels Wavelettransformationen und Level–of–Detail Steuerung.” Master’s thesis, ETH Zurich, Mar. 1995.

[37] A. Gersho and R. M. Gray. *Vector Qauntization and Signal Compression*. Kluwer Academic Publishers, Norwell, MA, 1991.

[38] H. Gish and J. N. Pierce. “Asymptotically efficient quantizing.” *IEEE Trans. on Information Theory*, IT-14(5):676, Sept. 1968.

[39] M. H. Gross. *Visual Computing*. Springer, 1994.

[40] M. H. Gross. “L^2 optimal oracles and compresion strategies for semi-orthogonal wavelets.” Technical Report 254, Computer Science Department, ETH Zurich, 1996.

[41] M. H. Gross, R. Gatti, and O. Staadt. “Fast multiresolution surface meshing.” In *Proceedings of IEEE Visualization ’95*, pages 135–142. IEEE Computer Society Press, 1995.

[42] M. H. Gross, L. Lippert, and O. Staadt. “Compression methods for visualization.” *Future Generation Computer Systems*, 15(1):11–29, 1999.

[43] M. H. Gross, O. G. Staadt, and R. Gatti. “Efficient triangular surface approximations using wavelets and quadtree data structures.” *IEEE Transactions on Visualization and Computer Graphics*, 2(2), June 1996. ISSN 1077-2626.

[44] R. Grosso, C. Lürig, and T. Ertl. “The multilevel finite element method for adaptive mesh optimization and visualization of volume data.” In *Proceedings of IEEE Visualization ’97*, pages 387–394, 1997.

[45] A. Guéziec. “Surface simplification inside a tolerance volume.” Technical Report RC 20440, IBM T. J. Watson Research Center, 1996.

[46] S. Gumhold, S. Guthe, and W. Straßer. “Tetrahedral mesh compression with the cut-border machine.” *IEEE Visualization ’99*, pages 51–58, October 1999. ISBN 0-7803-5897-X. Held in San Francisco, California.

[47] S. Gumhold and W. Straßer. “Real time compression of triangle mesh connectivity.” In M. Cohen, editor, *SIGGRAPH 98 Conference Proceedings*, Annual Conference Series, pages 133–140. ACM SIGGRAPH, Addison Wesley, July 1998. ISBN 0-89791-999-8.

[48] A. Haar. “Zur Theorie der orthogonalen Funktionen-Systeme.” *Math. Ann.*, 69:331–371, 1910.

[49] P. Heckbert, J. Rossignac, H. Hoppe, W. Schroeder, M. Soucy, and A. Varshney. “Course no. 25: Multiresolution surface modeling.” In *Course Notes for SIGGRAPH ’97*. ACM SIGGRAPH, 1997.

[50] M. Held. “Erit—A collection of efficient and reliable intersection tests.” *Journal of Graphics Tools*, 2(4):25–44, 1997.

[51] J. L. Hennessy and D. A. Patterson. *Computer Organization and Design: The Hardware/Software Interface*. Morgan Kaufmann Publishers, San Francisco, CA, 2 edition, 1998. With a contribution by James R. Larus.

[52] H. Hoppe. “Progressive meshes.” In H. Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 99–108. ACM SIGGRAPH, Addison Wesley, Aug. 1996. held in New Orleans, Louisiana, 04-09 August 1996.

[53] H. Hoppe. “View-dependent refinement of progressive meshes.” In *Proceedings of SIGGRAPH ’97*, pages 189–198, 1997.

[54] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. “Mesh optimization.” In *Proceedings of SIGGRAPH ’93*, pages 19–26, Aug. 1993.

[55] J. Hudson. *Piecewise Linear Topology*. W. A. Benjamin, Inc., 1969.

[56] D. A. Huffman. “A method for the construction of minimum redundancy codes.” *Proceedings of the IRE*, 40:1098–1101, 1951.

[57] B. Jawerth and W. Sweldens. “An overview of wavelet based multiresolution analyses.” *SIREV*, 36(3):377–412, 1994.

[58] K. Keeler and J. Westbrook. “Short encodings of planar graphs and maps.” *Discrete Applied Mathematics*, 58(3):239–252, 1995.

[59] R. Kisseleff. “Wavelet–basierte Isoflächenfindung durch Verwendung adaptiver Tetraedisierungsverfahren.” Master’s thesis, ETH Zurich, Mar. 1996.

[60] R. M. Koch, M. H. Gross, F. R. Carls, D. F. von Büren, G. Fankhauser, and Y. I. H. Parish. “Simulationg facial surgery using finite element models.” In H. Rushmeier, editor, *Computer Graphics (SIGGRAPH ’96 Proceedings)*, pages 421–428, Aug. 1996.

[61] R. M. Koch, S. H. M. Roth, M. H. Gross, A. P. Zimmermann, and H. F. Sailer. “A framework for facial surgery simulation.” Technical Report 326, Computer Science Department, ETH Zurich, 1999.

[62] M. Levoy, K. Pulli, B. Curless, S. Rusinkiewicz, D. Koller, L. Pereira, M. Ginzton, S. Anderson, J. Davis, J. Ginsberg, J. Shade, and D. Fulk. “The digital michelangelo project: 3d scanning of large statues.” *Proceedings of SIGGRAPH 2000*, pages 131–144, July 2000. ISBN 1-58113-208-5.

[63] Y. Linde, A. Buzo, and R. M. Gray. “An algorithm for vector quantization design.” *IEEE Transactions on Communications*, COM-28:84–95, Jan. 1980.

[64] P. Lindstrom. “Out-of-core simplification of large polygonal models.” In *Proceedings of ACM SIGGRAPH 2000*, pages 259–262, July 2000.

[65] P. Lindstrom, D. Koller, W. Ribarsky, L. F. Hughes, N. Faust, and G. Turner. “Real-Time, continuous level of detail rendering of height fields.” In H. Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 109–118. ACM SIGGRAPH, Addison Wesley, Aug. 1996. held in New Orleans, Louisiana, 04-09 August 1996.

[66] L. Lippert. *Wavelet-based Volume Rendering*. PhD thesis, ETH Zurich, 1998.

[67] S. P. Lloyd. “Least squares quantization in PCM.” *IEEE Transactions on Information Theory*, IT-28:127–135, 1982.

[68] W. E. Lorensen and H. E. Cline. “Marching cubes: A high resolution 3d surface construction algorithm.” *Computer Graphics (Proceedings of SIGGRAPH 87)*, 21(4):163–169, July 1987. Held in Anaheim, California.

[69] J. M. Lounsbery. *Multiresolution Analysis for Surfaces of Arbitrary Topological Type*. PhD thesis, University of Washington, Seattle, 1994.

[70] S. G. Mallat. “A theory for multiresolution signal decomposition: The wavelet representation.” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(7):674–693, July 1989.

[71] W. S. Massey. *A Basic Course in Algebraic Topology*. Springer-Verlag, 1997.

[72] R. B. McMaster. “The geometric properties of numerical generalization.” *Geographical Analysis*, 19(4):330–346, Oct. 1987.

[73] T. Möller. “A fast triangle-triangle intersection test.” *Journal of Graphics Tools*, 2(2):15–30, 1997.

[74] J. Neider, T. Davis, and M. Woo. *OpenGL Programming Guide*. Addison-Wesley, 1993.

[75] R. B. Pajarola. *Access to Large Scale Terrain and Image Databases in Geoinformation Systems*. PhD thesis, ETH Zurich, June 1998.

[76] R. B. Pajarola and J. Rossignac. “Compressed progressive meshes.” *IEEE Transactions on Visualization and Computer Graphics*, 6(1):79–93, 2000.

[77] R. B. Pajarola, J. Rossignac, and A. Szymczak. “Implant sprays: Compression of progressive tetrahedral mesh connectivity.” *IEEE Visualization '99*, pages 299–306, October 1999. ISBN 0-7803-5897-X. Held in San Francisco, California.

[78] W. B. Pennebaker and J. L. Mitchell. *JPEG Still Image Data Compression Standard*. Van Nostrand Reinhold, New York, 1993.

[79] H. Pfister, M. Zwicker, J. van Baar, and M. Gross. "Surfels: Surface elements as rendering primitives." *Proceedings of SIGGRAPH 2000*, pages 335–342, July 2000. ISBN 1-58113-208-5.

[80] J. Popovic and H. Hoppe. "Progressive simplicial complexes." In T. Whitted, editor, *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, pages 217–224. ACM SIGGRAPH, Addison Wesley, Aug. 1997. ISBN 0-89791-896-7.

[81] F. P. Preparata and M. I. Shamos. *Computational Geometry*. Springer, New York, 1985.

[82] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Wavelet Transforms*, chapter 13.10, pages 591–606. Cambridge University Press, 2 edition, 1992.

[83] Recommendations of the CCIR. "CCIR Rec. 601-2, Encoding parameters of digital television for studios." International Telecommunication Union, 1990.

[84] R. Ronfard and J. Rossignac. "Full-range approximation of triangulated polyhedra." In *Proceedings of EUROGRAPHICS '96*, pages C67–C76, 1996.

[85] J. Rossignac. "Edgebreaker: Connectivity compression for triangle meshes." *IEEE Transactions on Visualization and Computer Graphics*, 5(1):47–61, January - March 1999. ISSN 1077-2626.

[86] J. Rossignac and P. Borrel. "Multi-resolution 3D approximation for rendering complex scenes." In *Second Conference on Geometric Modelling in Computer Graphics*, pages 453–465, June 1993. Genova, Italy.

[87] C. P. Rourke and B. J. Sanderson. *Introduction to Piecewise-Linear Topology*. Springer-Verlag, 1982.

[88] K. Sayood. *Introduction to Data Compression*. Morgan Kaufmann, San Francisco, 1996.

[89] W. Schroeder. "A topology modifying progressive decimation algorithm." In *Proceedings of IEEE Visualization '97*, pages 205–212, 1997.

[90] W. J. Schroeder, J. A. Zarge, and W. E. Lorensen. "Decimation of triangle meshes." In E. E. Catmull, editor, *Computer Graphics (SIGGRAPH '92 Proceedings)*, volume 26, pages 65–70, July 1992.

[91] J. M. Shapiro. "Embedded image coding using zerotrees of wavelet coefficients." *IEEE Trans. Signal Processing*, 41(12):3445–3462, Dec. 1993.

[92] J. R. Shewchuk. "Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator." In M. C. Lin and D. Manocha, editors, *Applied Computational Geometry: Towards Geometric Engineering*, volume 1148 of *Lecture Notes in Computer Science*, pages 203–222. Springer-Verlag, May 1996. From the First ACM Workshop on Applied Computational Geometry.

[93] O. G. Staadt. "Rekonstruktion von Isoflächen in Volumendaten mittels Wavelet-basierter Octrees." Master's thesis, Darmstadt University of Technology, 1995.

[94] O. G. Staadt and M. H. Gross. "Progressive tetrahedralizations." In *Proceedings of IEEE Visualization '98*, pages 397–402, 1998.

[95] O. G. Staadt, M. H. Gross, and R. Weber. “Multiresolution compression and reconstruction.” In R. Yagel and H. Hagen, editors, *IEEE Visualization ’97*, pages 337–346. IEEE, Nov. 1997.

[96] E. J. Stollnitz, T. D. DeRose, and D. H. Salesin. “Wavelets for computer graphics: A primer, part1.” *IEEE Computer Graphics and Applications*, 15(3):76–84, 1995.

[97] E. J. Stollnitz, T. D. DeRose, and D. H. Salesin. *Wavelets for Computer Graphics: Theory and Applications*. Morgann Kaufmann, San Francisco, CA, 1996.

[98] G. Strang and T. Nguyen. *Wavelets and Filter Banks*. Wellesley-Cambridge Press, 1996.

[99] W. Sweldens. “The lifting scheme: A construction of second generation wavelets.” *SIAM Journal on Mathematical Analysis*, 29(2):511–546, Mar. 1998.

[100] A. Szymczak and J. Rossignac. “Grow & Fold: Compression of tetrahedral meshes.” In *Proceedings of the 5th Symposium on Solid Modeling and Applications*, pages 54–64. ACM, 1999.

[101] G. Taubin and J. Rossignac. “Geometric compression through topological surgery.” *ACM Transactions on Graphics*, 17(2):84–115, Apr. 1998. ISSN 0730-0301.

[102] G. Taubin and J. Rossignac. “3d geometry compression.” ACM SIGGRAPH 99, Course 21, Aug. 1999.

[103] C. Touma and C. Gotsman. “Triangle mesh compression.” *Graphics Interface ’98*, pages 26–34, June 1998. ISBN 0-9695338-6-1.

[104] I. J. Trotts, B. Hamann, K. I. Joy, and D. F. Wiley. “Simplication of tetrahedral meshes.” In *Proceedings of IEEE Visualization ’98*, pages 287–296, 1998.

[105] G. Turk. “Re-tiling polygonal surfaces.” In E. E. Catmull, editor, *Computer Graphics (SIGGRAPH ’92 Proceedings)*, volume 26, pages 55–64, July 1992.

[106] M. Unser, A. Aldroubi, and M. Eden. “Fast b–spline transforms for continous image representation and interpolation.” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(3):277–285, Mar. 1991.

[107] M. Unser, A. Aldroubi, and M. Eden. “On the asymptotic convergence of B-spline wavelets to Gabor functions.” *IEEE Transactions on Information Theory*, 38(2):864–872, 1992.

[108] M. Unser, A. Aldroubi, and M. Eden. “A family of polynomial spline wavelet transforms.” *Signal Processing*, 30:141–162, 1993.

[109] M. Vetter. “Progressive meshes.” Master’s thesis, ETH Zurich, Sept. 1997.

[110] M. Vetterli and J. Kovacevic. *Wavelets and Subband Coding*. Prentice Hall, Englewood Cliffs, NJ, 1995.

[111] G. Voronoi. “Nouvelles applications des parametres continus à la theorie des formes quadratiques. deuxième mémoire: Recherches sur les paralléloèdres primitifs.” *Reine und Angewandte Mathematik*, 134:198–287, 1908.

[112] G. K. Wallace. “The jpeg still picture compression standard.” *Communications of the ACM*, 34(4):30–44, Apr. 1991.

[113] R. Weber. “Wavelet–basierte Kompression und Interpolation von Isoflächen in Volumendaten.” Master’s thesis, ETH Zurich, Aug. 1996.

[114] E. R. White. “Assessment of line-generalization algorithms using characteristic points.” *The American Cartographer*, 12(1):17–27, 1985.

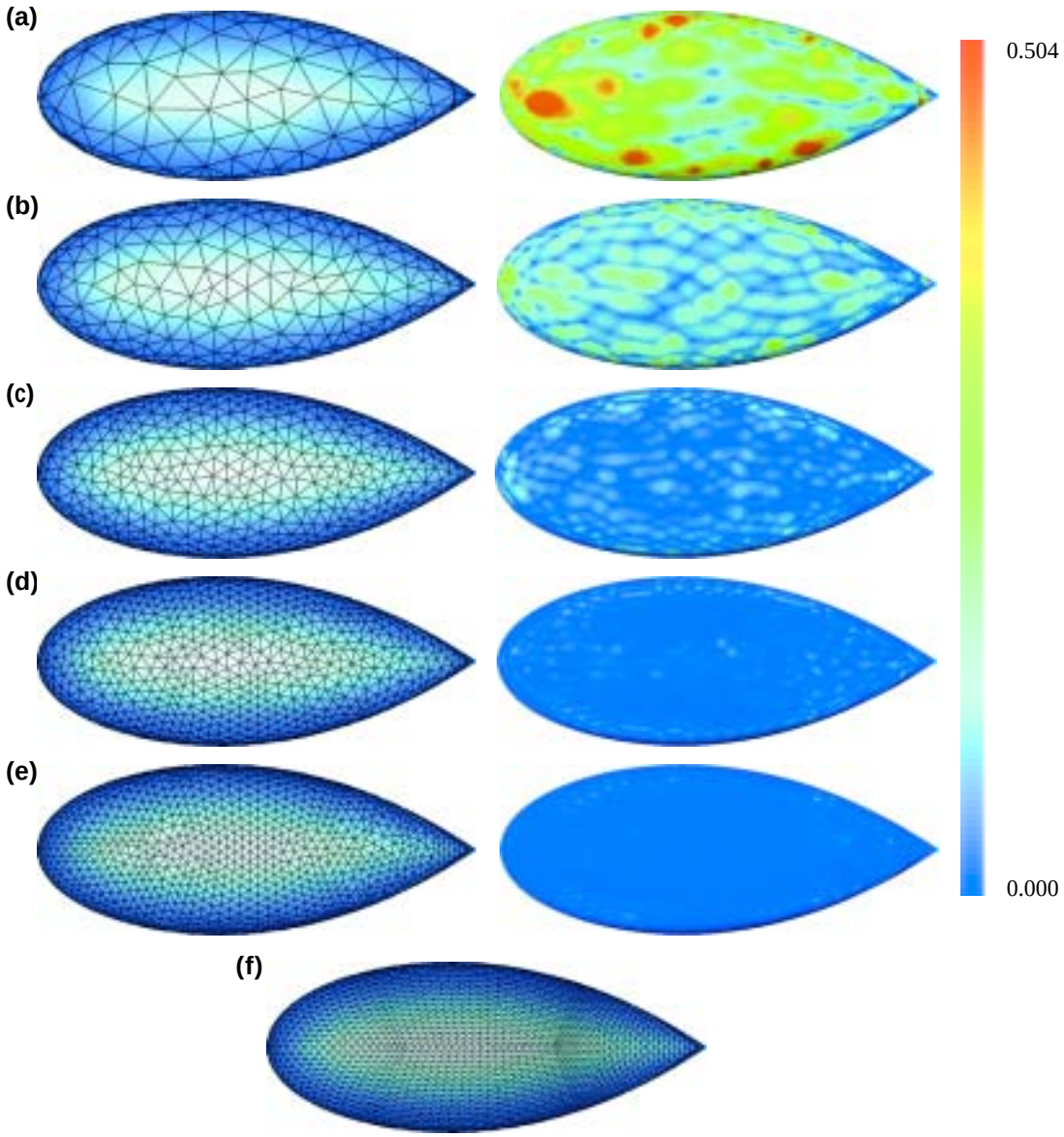
[115] I. H. Witten, R. M. Neal, and J. G. Cleary. “Arithmetic coding for data compression.” *Communications of the ACM*, 30(6):520–540, June 1987.

[116] J. C. Xia, J. El-Sana, and A. Varshney. “Adaptive real-time level-of-detail-based rendering for polygonal models.” *IEEE Transactions on Visualization and Computer Graphics*, 3(2):171–183, 1997.

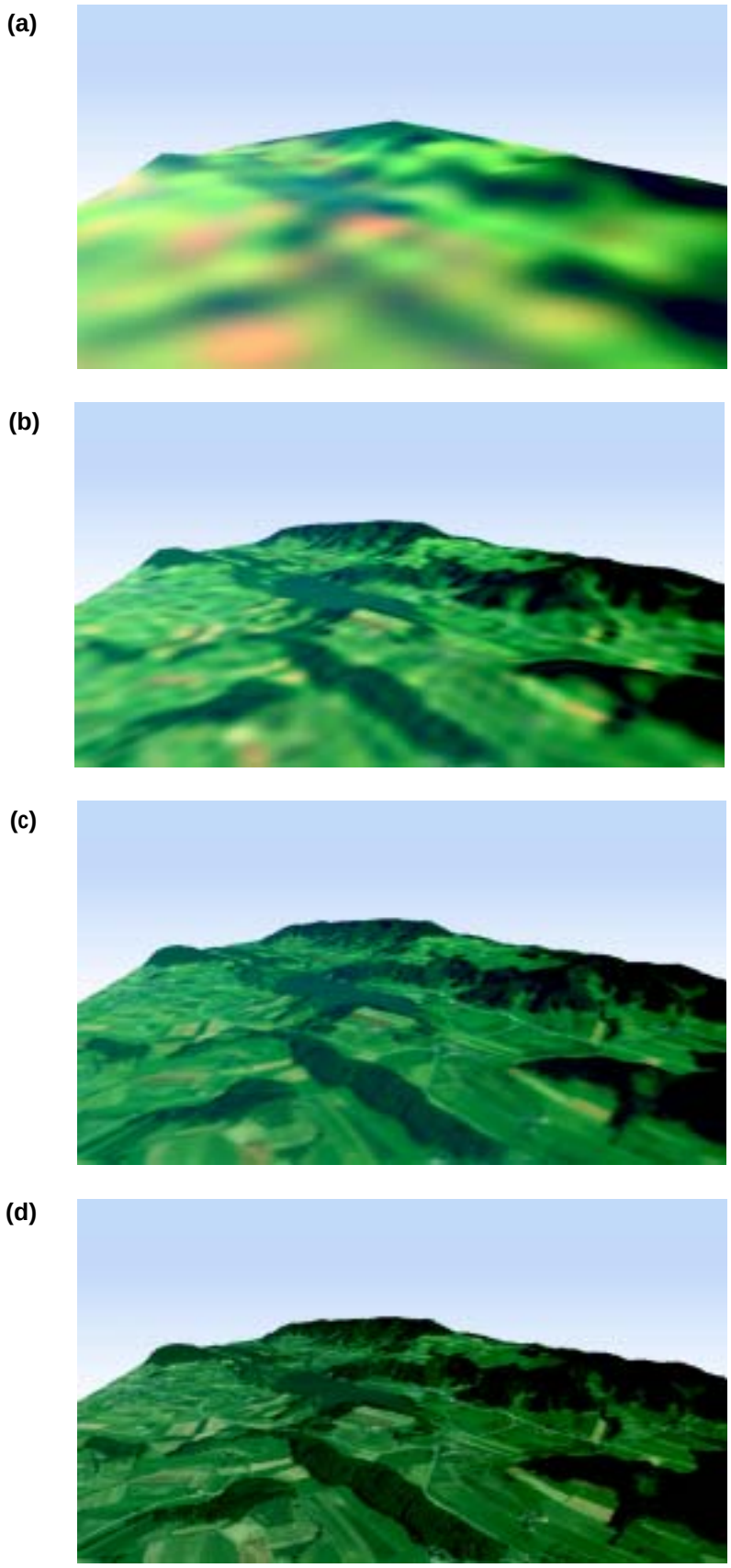
[117] X. Zhang and B. A. Wandell. “Color image fidelity metrics evaluated using image distortion maps.” *Signal Processing*, 70(3):201–214, Nov. 1998.

D

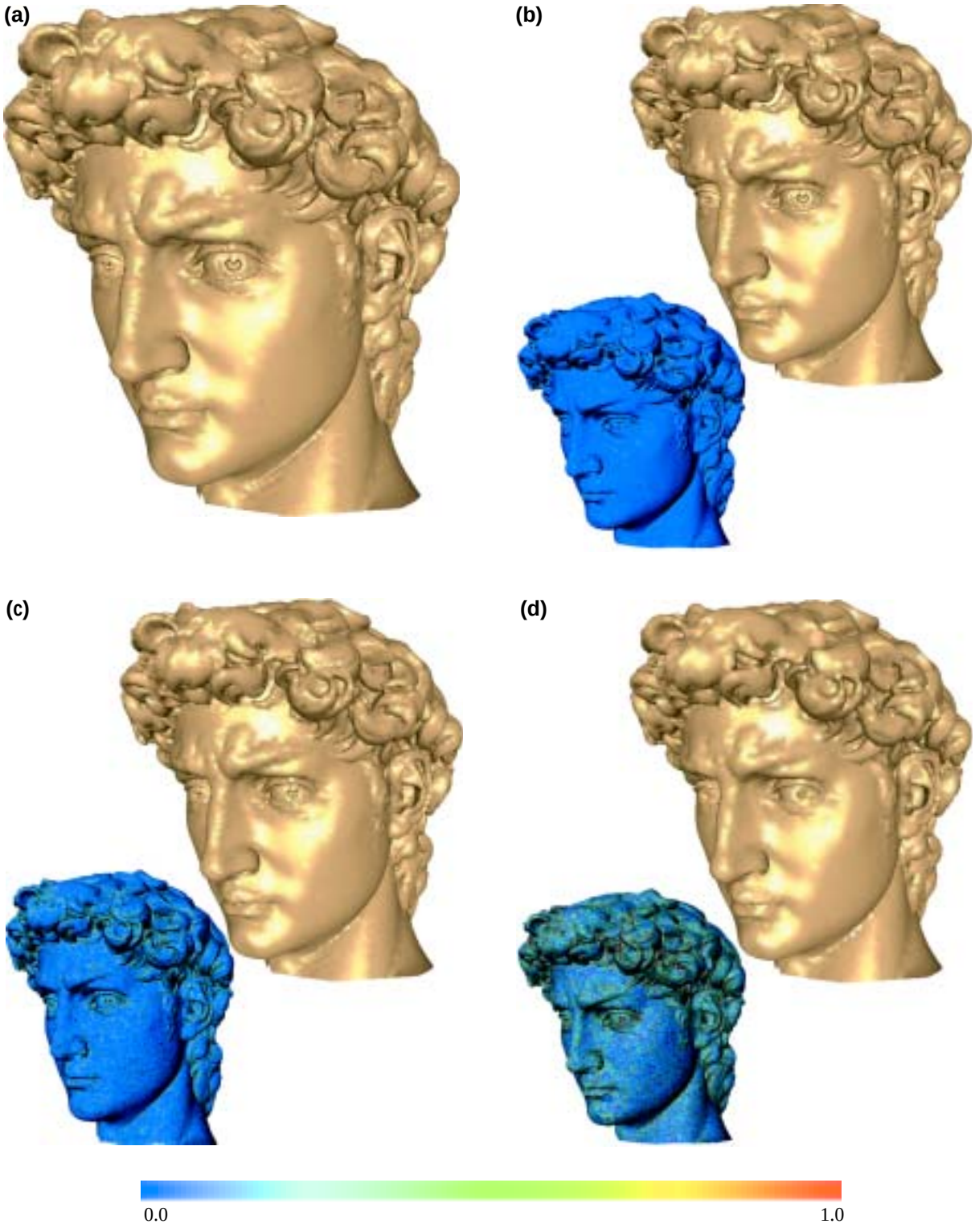
COLOR PLATES



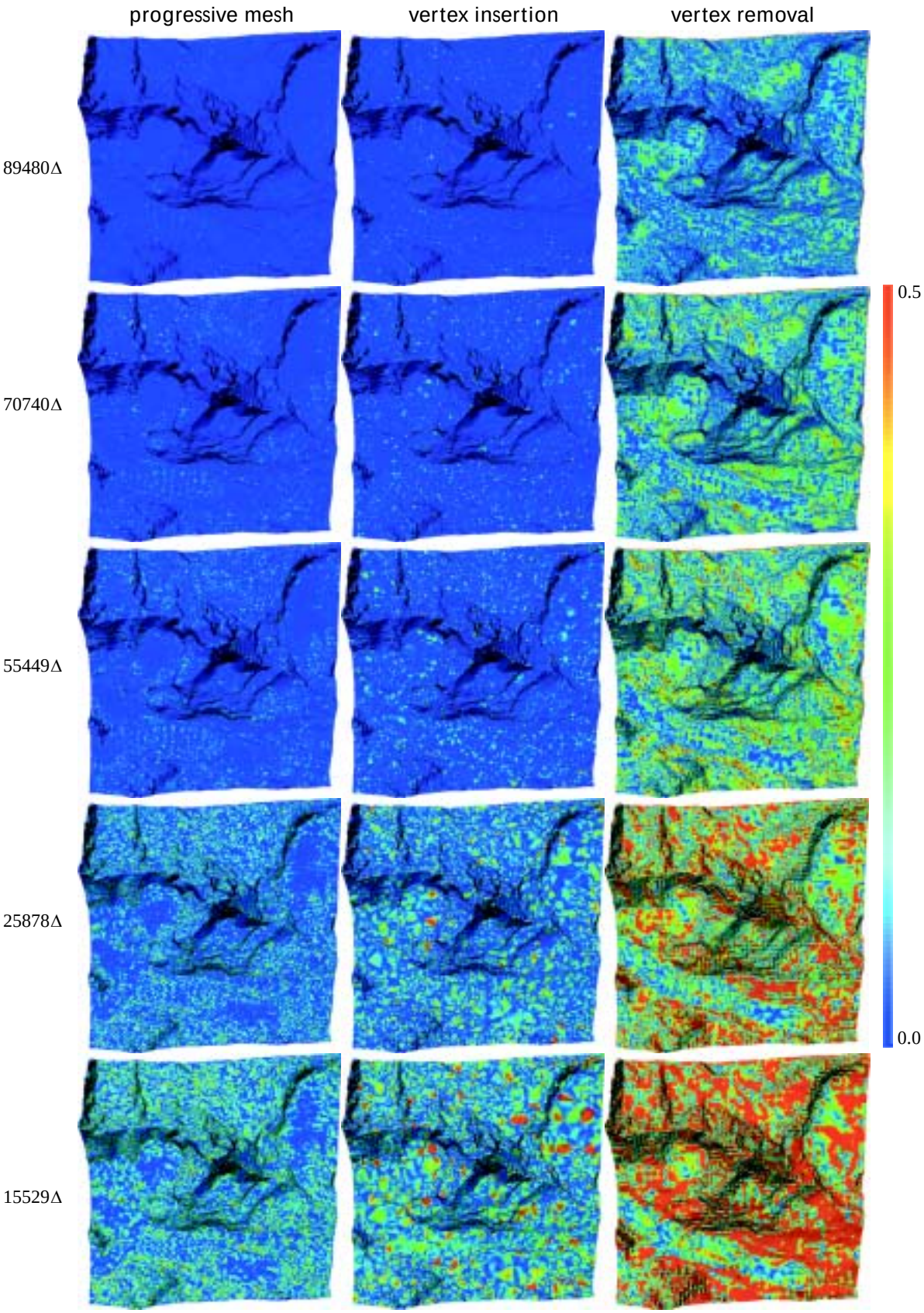
Color Plate 1 Progressive mesh representation of the Almond data set. **a:** 502 triangles. **b:** 1002 triangles. **c:** 2002 triangles. **d:** 3002 triangles. **e:** 4002 triangles. **f:** 6400 triangles. The corresponding errors are mapped onto the meshes on the right side. (see Figure 7.12 on page 122).



Color Plate 2 Progressive reconstruction of the Albis terrain model, mapped with orthophotos at increasing levels. **a:** level one. **b:** Level three. **c:** Level five. **d:** Level seven. (see Figure 7.10 on page 119).



Color Plate 3 Progressive mesh approximation of the head of Michelangelo's David. **a:** Input mesh. **b:** Approximation comprising 40.97 percent of the triangles. **c:** 20.97 percent. **d:** 10.97 percent. (see Figure 7.13 on page 124).



Color Plate 4 Error visualization of the Matterhorn model for the three different mesh approximation methods at various reconstruction levels. The magnitude of the absolute mean-square geometric error corresponds to the colorbar on the right. (see Figure 7.16 on page 128).

E

COPYRIGHTS

[AVS©]	AVS/Express sample data © 1999 Advanced Visual Systems Inc.
[DHM©]	Digital elevation model *DHM 25* © 1994 Swiss Federal Office of Topography.
[Michelangelo©]	The Digital Michelangelo Project © 2000 Stanford University Computer Graphics Laboratory.
[Swissphoto©]	Digital orthophotograph *Albis* © 1997 Swissphoto Vermessung AG.
[Stanford©]	The Stanford 3D Scanning Repository © 2000 Stanford University Computer Graphics Laboratory.