New Techniques for the Modeling, Processing and Visualization of Surfaces and Volumes

Christian Rössl

Max-Planck-Institut für Informatik
Saarbrücken, Germany

Dissertation zur Erlangung des Grades
Doktor der Ingenieurwissenschaften (Dr.-Ing)
der Naturwissenschaftlich-Technischen Fakultät I
der Universität des Saarlandes

Eingereicht am 11. März 2005 in Saarbrücken.

Betreuender Hochschullehrer — Supervisor
Prof. Dr. Hans-Peter Seidel,
Max-Planck-Institut für Informatik, Saarbrücken, Germany

Gutachter — Reviewers
Prof. Dr. Hans-Peter Seidel,
Max-Planck-Institut für Informatik, Saarbrücken, Germany

Prof. Dr. Marc Alexa,
Technische Universität Darmstadt, Germany

Prof. Dr. Pierre Alliez,
Institut National de Recherche en Informatique et en Automatique,
Sophia-Antipolis, France

Dekan — Dean
Prof. Dr. Jörg Eschmeier,
Universität des Saarlandes, Saarbrücken, Germany

Datum des Kolloquiums — Date of Defense
20. Juli 2005

Christian Rössl
Max-Planck-Institut für Informatik
Stuhlsatzenhausweg 85
66123 Saarbrücken, Germany
roessl@mpi-sb.mpg.de

Abstract

With the advent of powerful 3D acquisition technology, there is a growing demand for the modeling, processing, and visualization of surfaces and volumes. The proposed methods must be efficient and robust, and they must be able to extract the essential structure of the data and to easily and quickly convey the most significant information to a human observer. Independent of the specific nature of the data, the following fundamental problems can be identified: shape reconstruction from discrete samples, data analysis, and data compression.

This thesis presents several novel solutions to these problems for surfaces (Part I) and volumes (Part II). For surfaces, we adopt the well-known triangle mesh representation and develop new algorithms for discrete curvature estimation, detection of feature lines, and line-art rendering (Chapter 3), for connectivity encoding (Chapter 4), and for topology preserving compression of 2D vector fields (Chapter 5). For volumes, that are often given as discrete samples, we base our approach for reconstruction and visualization on the use of new trivariate spline spaces on a certain tetrahedral partition. We study the properties of the new spline spaces (Chapter 7) and present efficient algorithms for reconstruction and visualization by iso-surface rendering for both, regularly (Chapter 8) and irregularly (Chapter 9) distributed data samples.

Kurzfassung

Mit der Einführung leistungsfähiger Verfahren zur Erfassung von dreidimensionalen Daten stellt sich verstärkt die Frage nach Techniken zur Modellierung, Verarbeitung und Visualisierung von Flächen und Volumen. Solche Methoden müssen effizient und robust sein, gleichzeitig müssen sie es ermöglichen, die wesentliche Struktur der Daten zu extrahieren und somit einem Anwender einfach und schnell die wichtigsten Informationen zu vermitteln. Unabhängig von der speziellen Beschaffenheit der Daten stellen sich dabei die folgenden grundlegenden Probleme: Rekonstruktion von diskreten Meßwerten, Datenanalyse und Datenkompression. Diese Dissertation beschreibt einige neue Lösungsansätze zu diesen Problemen für Flächen (Teil I) und Volumen (Teil II).

Für Flächen verwenden wir die wohletablierte Darstellung als Dreiecksnetze. Wir entwickeln neue Algorithmen zur diskreten Krümmungsanalyse, der Bestimmung von Flächencharakteristika und der Erzeugung von Strichzeichnungen (Kapitel 3), zur Kodierung der Konnektivität (Kapitel 4) und zur topologieerhaltenden Kompression von zweidimensionalen Vektorfeldern (Kapitel 5). Für Volumen, die oft in Form von diskreten Datenpunkten gegeben sind, verwenden wir zur Rekonstruktion und Visualisierung neuartige trivariate Splineräume, die auf

einer bestimmten Tetraederzerlegung definiert sind. Wir studieren die Eigenschaften der neuen Splineräume (Kapitel 7) und entwerfen effiziente Algorithmen zur Rekonstruktion und Visualisierung mittels Darstellung von Isoflächen, jeweils für regulär (Kapitel 8) und irregulär (Kapitel 9) verteilte Datenpunkte.

Summary

With the advent of powerful 3D acquisition technology, there is a growing demand for the modeling, processing, and visualization of surfaces and volumes. The proposed methods must be efficient and robust, and they must be able to extract the essential structure of the data and to easily and quickly convey the most significant information to a human observer. Independent of the specific nature of the data, the following fundamental problems can be identified: shape reconstruction from discrete samples, data analysis, and data compression.

This thesis presents several novel solutions to these problems for surfaces (Part I) and volumes (Part II). For surfaces, we adopt the well-known triangle mesh representation and develop new algorithms for discrete curvature estimation, detection of feature lines, and line-art rendering (Chapter 3), for connectivity encoding (Chapter 4), and for topology preserving compression of 2D vector fields (Chapter 5). For volumes, that are often given as discrete samples, we base our approach for reconstruction and visualization on the use of new trivariate spline spaces on a certain tetrahedral partition. We study the properties of the new spline spaces (Chapter 7) and present efficient algorithms for reconstruction and visualization by iso-surface rendering for both, regularly (Chapter 8) and irregularly (Chapter 9) distributed data samples.

Surfaces

The first part discusses surface data, which are often represented as triangle meshes. As this representation does not provide smoothness, the estimation of the curvature of the represented surface is not straightforward. Curvature estimation is important and often required by shape interrogation techniques, which analyze the data. A new approach for the estimation of the curvature tensor is developed, which is directly applicable to piecewise linear surfaces with a piecewise linear normal field. The basic idea is similar to the well-known Phong shading. Applications of discrete curvature analysis are discussed, leading to new algorithms for the detection of feature lines and computer assisted line-art rendering of shapes.

With the emergence of powerful acquisition techniques, the obtained data sets tend to grow in size, and consequently there is a demand for efficient compression techniques. In this thesis, we specifically address connectivity encoding and vector field compression, i.e., the encoding of the graph structure of the triangulation and the compression of linearly interpolated vector attributes. A divide and conquer algorithm is developed for connectivity encoding. The output is a binary tree data structure, which can also be applied for efficient rendering. The vector field compression problem originates from a flow visualization context, where the topology of a complex flow data set provides a very compact and intuitive view of

the data. Consequently, the compression should ideally preserve the vector field topology. Building upon a theoretical framework, new methods for topology preserving vector field compression are presented. The algorithms are efficient, as it is shown that all decisions during compression can be based on local criteria — despite topology being a global property.

Volumes

The second part addresses the modeling, processing, and visualization of volumetric data. Digital volume data is often given as discrete samples, and a mathematical model is required to reconstruct the data in a feasible way, i.e., to provide a continuous representation enabling evaluation at arbitrary domain points. Two different scenarios are discussed: the reconstruction of structured data, which is laid out on a regular grid, and the approximation of general data, which are distributed over the volumetric domain. The foundations for both settings are piecewise polynomials with respect to certain uniform tetrahedral partitions. The polynomial pieces are represented in Bernstein-Bézier form enabling the use of powerful techniques well-known from computer aided geometric design. The goal is to use splines on these partitions with low (or even lowest) polynomial degree and appropriate smoothness properties for model visualization.

The reconstruction of structured data is based on a suitably chosen space of trivariate, quadratic super splines, i.e., piecewise polynomials in three variables of total degree two. We analyze the smoothness and approximation properties of the space and show that elements in the space can be evaluated efficiently. We also demonstrate the potential of precise iso-surface ray-casting for visualization.

For the approximation of general volumetric data, a new algorithm is developed, which is based on a certain cubic spline. The method is local and hence requires only the solution of small linear systems, so that huge data sets can be processed efficiently. Both approaches use splines which satisfy many smoothness properties. This work shows that such splines provide useful tools with advantageous properties for the various requirements of efficient modeling and visualization.

Zusammenfassung

Mit der Einführung leistungsfähiger Verfahren zur Erfassung von dreidimensionalen Daten stellt sich verstärkt die Frage nach Techniken zur Modellierung, Verarbeitung und Visualisierung von Flächen und Volumen. Solche Methoden müssen effizient und robust sein, gleichzeitig müssen sie es ermöglichen, die wesentliche Struktur der Daten zu extrahieren und somit einem Anwender einfach und schnell die wichtigsten Informationen zu vermitteln. Unabhängig von der speziellen Beschaffenheit der Daten stellen sich dabei die folgenden grundlegenden Probleme: Rekonstruktion von diskreten Meßwerten, Datenanalyse und Datenkompression. Diese Dissertation beschreibt einige neue Lösungsansätze zu diesen Problemen für Flächen (Teil I) und Volumen (Teil II).

Für Flächen verwenden wir die wohletablierte Darstellung als Dreiecksnetze. Wir entwickeln neue Algorithmen zur diskreten Krümmungsanalyse, der Bestimmung von Flächencharakteristika und der Erzeugung von Strichzeichnungen (Kapitel 3), zur Kodierung der Konnektivität (Kapitel 4) und zur topologieerhaltenden Kompression von zweidimensionalen Vektorfeldern (Kapitel 5). Für Volumen, die oft in Form von diskreten Datenpunkten gegeben sind, verwenden wir zur Rekonstruktion und Visualisierung neuartige trivariate Splineräume, die auf einer bestimmten Tetraederzerlegung definiert sind. Wir studieren die Eigenschaften der neuen Splineräume (Kapitel 7) und entwerfen effiziente Algorithmen zur Rekonstruktion und Visualisierung mittels Darstellung von Isoflächen, jeweils für regulär (Kapitel 8) und irregulär (Kapitel 9) verteilte Datenpunkte.

Flächendaten

Der erste Teil der Arbeit behandelt Flächendaten. Flächen werden oft in Form von Dreiecksnetzen dargestellt. Da diese Darstellung keine Glattheit – die stückweise lineare Funktion ist nicht stetig differenzierbar – bietet, ist unklar, wie die Krümmung der dargestellten Fläche berechnet werden soll. Die Krümmungsinformation spielt eine wichtige Rolle in der Bewertung und Analyse von Flächen. Es wird ein neuer Ansatz zur Bestimmung des Krümmungstensors entwickelt, wobei die stückweise lineare Fläche zusammen mit einer stückweise linearen Flächennormalen betrachtet wird. Die grundlegende Idee ähnelt dem bekannten Phong-Shading. Anwendungen der Krümmungsabschätzung führen zu neuen Algorithmen zur Bestimmung von charakteristischen Flächenteilen und zur computergestützten Erzeugung von Strichzeichnungen.

Mit der Einführung von leistungsfähigen Digitalisierungstechniken werden die erzeugten Datensätze meist größer, und folglich gewinnen Kompressionsverfahren an Bedeutung. In dieser Dissertation untersuchen wir speziell die Kodierung der Konnektivität von Dreiecksnetzen sowie die Kompression von stückwei-

se linearen Vektorfeldern. Um die Konnektivität – oder Graphstruktur – von Dreiecksnetzen zu kodieren, wird ein Divide-and-Conquer-Algorithmus entwickelt. Dessen Ausgabe besteht aus einem Binärbaum, der das Dreiecksnetz beschreibt und eine effiziente Darstellung der Fläche ermöglicht. Eine Motivation zur Kompression von Vektorfeldern stammt aus der Strömungsvisualisierung: Hier bietet die Topologie eines komplizierten Datensatzes ein kompaktes und intuitives Hilfsmittel, um den gesamten Datensatz schnell überblicken und interpretieren zu können. Aus diesem Grund sollte eine Kompression die Topologie des Vektorfelds nach Möglichkeit nicht verändern oder verfälschen. Ausgehend von einem theoretischen Rahmen werden neue Methoden zur topologieerhaltenden Vektorfeldkompression vorgeschlagen. Diese Methoden sind vor allem deshalb effizient, weil gezeigt wird, daß im Verlauf alle Entscheidungen auf lokale Kriterien zurückgeführt werden können – obwohl Topologie eine globale Eigenschaft darstellt.

Volumendaten

Der zweite Teil der Arbeit behandelt die Modellierung, Verarbeitung und Visualisierung von Volumendaten. Digitale Volumendaten sind oft als eine Menge von diskreten Datenpunkten gegeben. Mathematische Modelle der Daten erlauben eine sinnvolle Rekonstruktion der Daten, d.h. eine Fortsetzung zwischen den gegebenen Punkten. Es werden zwei verschiedene Szenarien behandelt: die Rekonstruktion von strukturierten Daten, die entlang eines regelmäßigen Gitters liegen, und die Approximation von allgemeinen Daten, die beliebig im Volumen verteilt sind. Als gemeinsame Grundlage für beide Probleme dienen stückweise Polynome, die auf einer regelmäßigen Tetraederzerlegung des Volumens definiert sind. Die einzelnen Polynomstücke werden in Bernstein-Bézier-Form dargestellt, so daß bekannte und leistungsfähige Techniken aus dem Computer Aided Geometric Design Anwendung finden. Ziel ist es, sogenannte Splines mit möglichst niedrigem Polynomgrad und zweckmäßigen Eigenschaften zu verwenden. Die Lösung des ersten Problems, die Rekonstruktion strukturierter Daten, basiert auf einem geeignet gewählten Raum von trivariaten quadratischen Super-Splines, also auf stückweisen Polynomen in drei Variablen vom totalen Grad zwei. Wir analysieren die Glattheits- und Approximationseigenschaften dieses Raums und zeigen, daß Elemente in diesem Raum effizient ausgewertet werden können. Gleichzeitig demonstrieren wir das Potential von exaktem Ray-Casting von Isoflächen für die Visualisierung.

Zur Approximation von allgemeinen Daten wird ein neuer Algorithmus entwickelt, der auf bestimmten kubischen Splines aufbaut. Die vorgestellte Methode arbeitet lokal, so daß nur kleine lineare Gleichungssysteme gelöst werden müssen, was die Verarbeitung von sehr großen Datensätzen erlaubt. Beide Ansätze verwenden trivariate Splines, die viele Glattheitsbedingungen erfüllen. Die vorliegende

Arbeit zeigt, daß solche Splines ein nützliches und vorteilhaftes Hilfsmittel zur effizienten Modellierung und Visualisierung darstellen.

Acknowledgements

This thesis would not have been possible without the help and support of many people. First of all, I would like to thank my supervisor Prof. Dr. Hans-Peter Seidel for his interest in this work, his continuous support, and for providing an excellent research environment. I enjoyed the freedom to work on diverse problems, which gave me an always interesting and challenging insight in the field of geometric modeling and visualization.

I would like to thank the two external reviewers Prof. Dr. Marc Alexa and Prof. Dr. Pierre Alliez.

I would like to thank all my present and former colleagues at the Computer Graphics Group, who make MPI such an exciting place for research. When I tried to compile a list, I realized that it is impossible to name all of them here. As I want to avoid an implicit "ranking", I only mention those explicitly who co-authored some of my publications. These are (in alphabetical order): Mario Botsch, Ioannis Ivrissimtzis, Holger Theisel, Jens Vorsatz, Rhaleb Zayer, and Frank Zeilfelder, and many thanks also to Prof. Dr. Leif Kobbelt and Prof. Dr. Günther Nürnberger. Again, I will not forget all the others, who made the time at MPI so wonderful – regarding to research as well as to other activities.

Finally, I thank my family and especially my parents for their support over all the time.

Contents

List of Figures

Chapter 1
Introduction

The visualization of digital data is a central goal of computer graphics. There is a multitude of diverse flavors, ranging for instance from scientific and information visualization over visualization in computer aided design processes, the creation of photorealistic images to the interactive visualization of virtual worlds in cultural heritage applications or computer games. Every specific task comes with its own particular challenges. However, all share the following fundamental problems: an appropriate digital *model* of the data is required, these models must enable the efficient *analysis* of the data, and the efficient processing of huge data must be supported, commonly incorporating custom tailored data *compression* techniques.

There are many different and versatile types of digital data stemming from a physical acquisition process such as a laser range scan of a real object or a computed tomography or magnetic resonance imaging scan of a human body, or from a simulation such as computational fluid dynamics. In general, we observe that the size of the generated data sets tends to increase over the time with the emerging acquisition techniques. This is independent of the specific type and source of the data. And so is the requirement of appropriate mathematical models, i.e. suitable representations of the diverse discrete, digital data. However, the particular model is closely related to the specific data on the one side and to the typical data processing pipeline on the other side. And even here, it often turns out that there is not *the one* ideal model for any processing step, rather different models offer different advantages and disadvantages. In practice, a unified model is preferred wherever possible to minimize data conversion between intermediate steps, which tends to induce some error hence spoils robustness.

A well-known example for this dilemma in digital geometry processing is the decision on an explicit or implicit surface representation: Explicit representations directly provide a model of the surface, while the implicit representations apply a volumetric model and define the surface as zero-set (or more generally as level-

set or isosurface) of a trivariate (signed-distance) function. In general, the first alternative is easier to handle, the surface can be evaluated and manipulated directly. However, certain operations are expensive and hard to realize: these are modifications which change the genus of the surface, i.e. which add or remove a topological handle. It is easy to imagine that such changes are hard to realize. In contrast, this is much easier for the second alternative, the implicit model, where the surface topology is not encoded explicitly. On the other side, a drawback is that shape typically has to be resampled leading to conversion errors and that sharp features may not be reproduced exactly. In summary, although both models represent a surface, they both have their strengths and weaknesses depending on the specific requirements of the applications.

So far, we argued that there is no ideal model for any purpose, indicating that practical demands search for a good compromise. Still there is the following question: Given a specific data and application, what would the ideal model in this context provide? Or what distinguishes a good model? Many data are given as sets of sample points, e.g. a point cloud from shape acquisition. However, most applications rely on a continuous model of the data, i.e. the model provides a reliable guess in regions between the discrete points. In other words, the model must *reconstruct* the data in a feasible way. For geometric data this means that the original shape should be approximated as good as possible by the model. At the same time, the reconstruction or the generation of the model should be efficient in computation (time and memory), and should provide the robustness to deal with any reasonable distributions of the samples. In addition, contamination with measurement noise should be taken into account and automatically be reduced. Besides the reconstruction, the evaluation of the model is very important and must be efficient and robust. Potential application dependent requirements include, for instance, smoothness needed for high-quality visualization or preservation of special features, which may e.g. be of geometric nature like sharp edges and corners of a shape or consist of more evolved properties.

Of course any useful model must enable the efficient and accurate analysis of the represented data. The analysis is required for the processing and for the visualization of the digital data. It again depends in detail on the specific application, and manifold scenarios exist. Still, this can often be viewed as classification and segmentation process. This recovering of structural information plays an important role, it provides a level of abstraction and is often connected with the identification and extraction of certain features of the data set, which describe local or global properties. In many applications, quality criteria are analyzed, for example whether a surface is smooth enough or in which geometrically complex regions the model should be refined by resampling. Segmentation is also a common building block for the conversion from one model to another one, where the resulting segments are then handled individually. And in many situations, the vi-

sualization of abstract properties derived from an analysis of the data provides a better view of the model, conveying more (but filtered) information with fewer rending primitives. Typical examples are the visualization of complex flow data and non-photorealistic rendering.

The analysis leads to simplification or reduction of complexity. As mentioned earlier, in practice one has to face very large and ever growing data sets. This emphasizes the need for efficient data reduction or compression techniques, especially for storing and transmitting such data. Here, the goal is to transfer one representation of the data into another one, which consumes less storage and is either equivalent to the original representation, i.e. the original can be restored without loss of information, or approximates the original data as close as possible given an appropriate distance metric. Data compression reduces redundancy in the data, while some specified properties are preserved. These properties depend on the specific application imposing an individual challenge and potentially accept loss of (less important) information. The approximation of a surface with fewer parameters or coefficients than used in the original representation, i.e. the projection into a lower dimensional space, is a typical example. In this case, the compression rate or the number of coefficients generally depends on the maximal tolerated approximation error, which can be measured geometrically.

In this work, we address the mentioned problems of data reconstruction, or finding appropriate mathematical models, analysis and compression, and we develop new techniques for the visualization of both surfaces and volumes.

As a fundamental model — for surfaces and volumes — throughout this work we use piecewise polynomials (or splines). This means that the model is composed of many pieces, each of which is a simple mathematical object, namely a bivariate or trivariate polynomial, respectively. For efficiency and simplicity, we strive to keep the polynomial degree as low as possible. In the surface case, this means we apply piecewise linear splines, including triangular meshes. This popular surface model has proven to be extremely flexible and enables the efficient representation of complex objects (see Chapter 2). The situation is different for volumetric models, where additional smoothness properties are required for high-quality visualization. Here, we apply piecewise quadratic (Chapter 8) and piecewise cubic (Chapter 9) polynomials for the reconstruction of gridded data — the samples are laid out on a regular grid — and for the approximation of general, distributed data, respectively. This leads to challenging problems and interesting new algorithms to solve the reconstruction and approximation problem for the visualization of complex data sets.

We analyze surfaces and two-dimensional, piecewise linear vector fields. Thinking in terms of the differential geometry of a shape, it seems natural to compute and to make use of the surface curvature for the analysis, e.g. for segmentation and feature detection or for emphasis in the visualization (Chapter 3).

Similarly, specific features of vector fields are suited for efficient flow visualization, conveying an impression of the whole data set by showing only important parts.

The so-called topological skeleton of a flow field represents such a feature, and we develop new compression algorithms that preserve the topological skeleton while yielding the best compression rates currently available (Chapter 5). The compression algorithms are efficient as we will show that — although topology is a global property of the vector field — all decisions can be made from local criteria. This compression scheme drops less important information while the important features are preserved. Such lossy data encoding is tolerated or even desired for this kind of geometric data, and similarly for the volumetric approximation. The situation is different for other types of data. Here, we consider the connectivity of a triangle mesh, i.e. the information on how the individual triangles are interconnected. We propose and study an encoding which compresses this information by organizing it in a tree data structure and triangle strips, a rendering primitive for efficient visualization.

Finally, we give a brief overview of the following chapters, summarizing the main contributions of this work. Along with that we reference the publications that this thesis is based on. All these articles are published at different conferences and in journals. The thesis is separated into two parts — surfaces and volumes — reflecting two different general data models applied in digital geometry processing and visualization.

- Chapter 2 gives a review of polygonal meshes, as triangle meshes provide the basic model for two-dimensional data, surfaces and vector fields (see also [Kobbelt et al. 2000, Kähler et al. 2001]).

- Chapter 3 analyzes techniques for the estimation of discrete curvature and derives a new algorithm [Rössl and Kobbelt 1999, Theisel et al. 2004b]. We discuss feature detection [Rössl et al. 2000a, Rössl et al. 2001] and line-art rendering [Rössl and Kobbelt 2000, Rössl et al. 2000b] as applications and remark the potential for feature sensitive remeshing [Botsch et al. 2000].

- Chapter 4 presents a tree-based divide-and-conquer approach to connectivity encoding, which leads to an intuitive compression algorithm [Ivrissimtzis et al. 2002, Ivrissimtzis et al. 2003]. The resulting data structure can be exploited for efficient visualization [Rössl et al. 2003b].

- Chapter 5 addresses the compression of piecewise linear vector fields. In this flow visualization context, we develop a theoretical framework and new methods for the efficient compression, with the additional, non-trivial constraint of topology preservation [Theisel et al. 2003b, Theisel et al. 2003a,

Theisel et al. 2004a]. The algorithms are efficient despite of the global constraint, and we report best compression rates currently available. We remark that an alternative application based on these techniques is the comparison of flow fields [Theisel et al. 2003c].

- Chapter 6 provides an introduction to the volumetric setting.
- Chapter 7 introduces trivariate splines on a so-called type-6 tetrahedral partition. The analysis of the spline spaces [Hangelbroek et al. 2004] provides the basis for developing appropriate algorithms.
- Chapter 8 describes a new, efficient model for the reconstruction of gridded volume data based on quadratic super splines [Rössl et al. 2003a, Rössl et al. 2004a] with advantageous approximation properties [Nürnberger et al. 2004c]. Applying only the lowest polynomial degree possible, we enable high-quality visualization.
- Chapter 9 presents a first approach to efficient approximation of huge sets of distributed volumetric samples. The approach is based on piecewise cubic polynomials [Rössl et al. 2004b] which provide an efficient model of the data with potential to automatic denoising and compression. As in the previous chapter we apply a concept of consistent splines which satisfy appropriate smoothness conditions for visualization.
- Chapter 10 summarizes the volume setting.
- Chapter 11 concludes the thesis.

Notation

We briefly summarize on the notation used throughout this work.

- Scalar values/functions and indices are written in italic letters, e.g. a.
- Bold lower case letters $\mathbf{a}$ denote vectors.
- Bold upper case letters $\mathbf{A}$ denote matrices.
- diag$(a_1, \ldots, a_n)$ denotes a diagonal matrix.
- $\times$ denotes the vector product or cross product.
- $(\mathbf{ab})$ is a dot product of two vectors.
- $|.|$ denotes the absolute value of a scalar.
- $\|.\|$ denotes the Euclidian distance or the norm $\|.\|_2$.
  The use of other norms will be indicated.
- $\#$ denotes the cardinality of a finite set.
- span denotes the linear space spanned by a finite set.
- dim denotes the dimension of a linear space.
- $\delta_{\nu,\mu}$ denotes Kronecker's symbol.
- $\frac{\partial f}{\partial \mathbf{r}}$ denotes the directional derivative in direction $\mathbf{r}$.
- D_x, D_{xx} denote the first and second order differential operators, the partial derivatives are written as $f_x := D_x f$, $f_{xx} := D_{xx} f$, respectively.

Part I

Surface Meshes

Chapter 2
Overview of Triangle Meshes

There exists a multitude of models for the representation of digital shapes. Polygonal meshes, are a popular choice, and especially for the representation of highly complex objects they constitute the de facto standard. Among the various reasons for this fact we mention the following. In contrast to parametric surfaces, shapes of arbitrary topology can be handled easily. Increasing shape complexity only means adding more polygons, while the simplicity of the surface description is retained. This makes the model itself and manipulating algorithms flexible and numerically robust. In addition, triangles – as the simplest polygons – are the natural primitive for efficient, hardware accelerated rendering. Regarding the whole processing pipeline from the generation over various editing operations to the rendering, triangle meshes provide a simple, efficient, and unified model, which eliminates costly and error-prone conversion and enables the seamless integration of different software tools.

In this section we provide a brief introduction to the topic, where we explicitly focus on and restrict ourselves to the specific requirements of the subsequent sections. For a more complete overview, we refer to the surveys [Kobbelt et al. 2000, Kähler et al. 2001], additional references can be found in the recent article [Bischoff and Kobbelt 2004].

In the following, we consider only triangular meshes as the simplest type of polygonal meshes, i.e. each individual polygon is triangulated.

Triangle Meshes

A *triangular mesh* is described compactly as a pair $(\mathcal{K}, \mathbf{V})$, where $\mathcal{K}$ is a simplicial complex representing the connectivity of vertices, edges and faces, and $\mathbf{V} = (\mathbf{v}_0, \ldots, \mathbf{v}_n)$ describes the geometric positions of the vertices. Then $i \in \mathcal{K}$ is a vertex of the mesh, $(i, j) \in \mathcal{K}$ means the vertices i and j are connected by a (di-

rected) edge, and $(i, j, k) \in \mathcal{K}$ represents a triangular face. We define the *1-ring neighborhood* of a vertex $i \in \mathcal{K}$ as the set of adjacent vertices $\mathcal{N}_i = \{j|(i, j) \in \mathcal{K}$. The number of neighbors $\#\mathcal{N}_i$ is called the *valence* of the vertex i.

Piecewise linear interpolation

This definition separates the mesh connectivity $\mathcal{K}$ from the geometry information $\mathbf{V}$. The surface mesh is a piecewise linear function with coefficients $\mathbf{V}$. We remark that $\mathbf{V}$ may represent not only positions in the domain but any (additional) attributes. Then we can define piecewise linear functions of these attributes over the triangular domain using the *barycentric coordinates* $\lambda_1, \lambda_2, \lambda_3$ as basis.

Given is a (non-degenerate) triangle $(i, j, k) \in \mathcal{K}$ with vertex positions $\mathbf{x}_i, \mathbf{x}_j, \mathbf{x}_k$ in the (planar) domain. The barycentric coordinates w.r.t. this triangle are the linear polynomials λ_ν, $\nu = 0, 1, 2$, which satisfy the interpolation conditions $\lambda_\nu(\mathbf{x}_\mu) = \delta_{\nu,\mu}, \nu = 0, 1, 2$. The barycentric coordinates are obtained as the solution of a linear system, and it is easy to see that $\lambda_0 + \lambda_1 + \lambda_2 = 1$ (see e.g. [Hoschek and Lasser 1993]).

We will apply barycentric coordinates for linear interpolation of attributes over individual triangles, e.g. for the linear interpolation of vertex normals in the next Chapter 3.

2-manifold

Regarding the connectivity information represented by $\mathcal{K}$, for practical reasons, we restrict ourselves to *bounded two-manifolds*, i.e. the surface has to be disk-like at every inner point (vertex) or topologically equivalent to a half-disk on boundaries. In particular, this rules out edges with more than two adjacent triangles or points, where two triangles meet only at their tips (see for instance [Kobbelt et al. 2000]).

Topology /connectivity

The complex $\mathcal{K}$ represents the graph structure of the mesh and hence its topological properties, including the number of handles of the surface and the existence of boundaries. The *Euler characteristic* of $\mathcal{K}$ is the number $\chi(\mathcal{K}) = V - E + F$ (*Euler's formula*), where V, E, and F denote the number of vertices, edges, and faces in $\mathcal{K}$, respectively. The Euler characteristic of a surface is independent of the particular representation of this surface as a complex. Consider a shape which is topologically equivalent to a sphere with p handles and q boundary loops, i.e. q topological disks removed, then its Euler characteristic is given as $\chi = 2 - 2p - q$.

For instance, the Euler characteristic of a sphere and a torus are two and zero, respectively.

This holds for polygonal surfaces in general. For the special case of triangulated surfaces, one can easily derive the following, well-known relation between the number of vertices and the number of triangles in a mesh: neglecting the boundary vertices, we obtain $F \approx 2V$. This leads to the observation that the average valence of an (interior) vertex is six. Vertices of valence six are commonly called *regular*, and consequently meshes with no (or only few) non-regular vertices are often called (semi-)regular.

The properties listed above will be used in Chapter 4 together with some considerations on data structures, which are reviewed in the following.

Data structures and operators

There are various choices for the representation of triangular meshes, which enable the efficient access of neighborhood information, like the navigation between adjacent triangles and traversal of the 1-ring of a vertex. Among these are the *half-edge* data structures which are of particular interest (see e.g. [Campagna et al. 1998, Kettner 1998] and the references therein), due to their simplicity and the imposed orientation.

Here, every edge is separated into two directed and inversely oriented half-edges. The basic building blocks — which are reflected by the references stored by the particular implementation — are the access of the face that the half-edge is associated to, a map between the source vertex and its excident half-edge, the enumeration of the next and previous half-edge, to navigate counter-clockwise and clockwise around a face, and the access to the inverse half-edge, which provides the opposite face. In the case of triangular meshes, every face consists of exactly three half-edges, which implicitly provides the references to the face as well as to the next and previous half-edge as a function of the edge.

It is straightforward to formulate more complex queries or traversals in terms of the given basic operations: For instance, to traverse the next vertex in the 1-ring, first take the associated excident half-edge, then the opposite of the next edge, and finally access the source vertex of the latter half-edge. In a similar fashion, the associated *fan* of triangles around a vertex or a triangle strip (cf. Section 4.2.1) can be traversed.

Mesh decimation

From the so-called mesh processing-pipeline, we apply *mesh decimation* in Chapter 5. The basic idea of these techniques is to reduce the complexity by clustering and merging or removing entities of the mesh while certain validity and

approximation constraints are satisfied. We refer to the mentioned surveys and [Gotsman et al. 2002] for an overview of different techniques and details of the standard algorithms.

We apply an incremental method for mesh decimation, choosing the half-edge collapse — which collapses the source into the target vertex and hence removes two half-edges and the associated triangles — as the basic topological operator for simplification. Here, to all half-edges a priority value is assigned, which supports the greedy decision on which edge should be collapsed next. The priority usually reflects a distance measure between the original and the simplified surface to enable good approximation. In addition, certain validity checks are applied, which may forbid a particular edge collapse because of either (topological) degeneracies or certain application requirements (see Section 5.3.2).

The basic framework of the incremental mesh decimation algorithm would repeat the following until no more reduction is possible: In each iteration get (and remove) a half-edge from a priority queue and apply the half-edge collapse if allowed. Then re-evaluate and correct the priorities of all half-edges in the affected neighborhood. For efficiency reasons, it is crucial that all decisions and computations of the priority values only depend on a local neighborhood. This is given in the standard setup of geometry simplification (e.g. evaluating triangle normals, error quadrics, or the one-sided Hausdorff distance, see e.g. [Gotsman et al. 2002]). However, we will consider an application where a global property — namely the topology of a piecewise linear vector field — is respected, and it is not trivial to see how this can be based on local decisions.

Chapter 3

Discrete Curvature Computation and Applications

3.1 Background

A variety of shape interrogation techniques require the estimation of the curvature of a surface. Curvature is defined for smooth surfaces, i.e., surfaces which are sufficiently often differentiable. Hence, piecewise linear triangle meshes can not be analyzed directly in a reasonable way. Instead, concepts of differential geometry are applied indirectly assuming that meshes are approximations of smooth surfaces of interest. This leads to discrete differential geometry operators [Meyer et al. 2002] and discrete curvature estimation. Many techniques for this purpose haven been developed within the last decade, and in the following we provide a brief overview. Before, we recall some basic differential geometry for smooth surfaces.

3.1.1 Differential Geometry of Smooth Surfaces

This section shortly reviews differential geometry of surfaces, we refer to a text book, e.g. [Do Carmo 1976], for a comprehensive course.

Let S be a sufficiently smooth surface and $\mathbf{p} \in S$ be a point on this surface. Consider a neighborhood of $\mathbf{p}$ such that $\mathbf{x}(u, v)$ is a local parameterization of S. Then $\mathbf{n} = \frac{\mathbf{x}_u \times \mathbf{x}_v}{\|\mathbf{x}_u \times \mathbf{x}_v\|}$ denotes the unit *normal vector* perpendicular to the surface. The partial derivatives $\mathbf{x}_u$ and $\mathbf{x}_v$ of $\mathbf{x}$ w.r.t. u and v are tangents to S. The bilinear

first and second fundamental forms of $\mathbf{x}$ are defined by the matrices

$$\mathbf{I} = \begin{bmatrix} E & F \\ F & G \end{bmatrix} := \begin{bmatrix} \mathbf{x}_u\mathbf{x}_u & \mathbf{x}_u\mathbf{x}_v \\ \mathbf{x}_u\mathbf{x}_v & \mathbf{x}_v\mathbf{x}_v \end{bmatrix}, \quad \text{and} \tag{3.1}$$

$$\mathbf{II} = \begin{bmatrix} e & f \\ f & g \end{bmatrix} := \begin{bmatrix} \mathbf{x}_{uu}\mathbf{n} & \mathbf{x}_{uv}\mathbf{n} \\ \mathbf{x}_{uv}\mathbf{n} & \mathbf{x}_{vv}\mathbf{n} \end{bmatrix}. \tag{3.2}$$

We note that $\mathbf{II}$ can be expressed alternatively using the identities $\mathbf{x}_{uu}\mathbf{n} = -\mathbf{x}_u\mathbf{n}_u$, $\mathbf{x}_{uv}\mathbf{n} = \mathbf{x}_{vu}\mathbf{n} = -\mathbf{x}_u\mathbf{n}_v = -\mathbf{x}_v\mathbf{n}_u$, and $\mathbf{x}_{vv} = -\mathbf{x}_v\mathbf{n}_v$.

Let $\mathbf{t} = a\mathbf{x}_u + b\mathbf{x}_v$ be a unit vector in the tangent plane in $\mathbf{p}$, which is represented as $\bar{\mathbf{t}} = (a, b)^\top$ in the local coordinate system. The *normal curvature* $\kappa_n(\bar{\mathbf{t}})$ is the curvature of the planar curve that results from intersecting S with the plane through $\mathbf{p}$ which is spanned by $\mathbf{n}$ and $\mathbf{t}$. The normal curvature in direction $\bar{\mathbf{t}}$ can be expressed in terms of the fundamental forms as

$$\kappa_n(\bar{\mathbf{t}}) = \frac{\bar{\mathbf{t}}^\top \mathbf{II}\, \bar{\mathbf{t}}}{\bar{\mathbf{t}}^\top \mathbf{I}\, \bar{\mathbf{t}}} = \frac{ea^2 + 2fab + gb^2}{Ea^2 + 2Fab + Gb^2}$$

The maximal normal curvature κ_1 and the minimal normal curvature κ_2 are called *principal curvatures*, and the associated tangent vectors $\mathbf{t}_1$ and $\mathbf{t}_2$ are called *principal directions*. $\mathbf{t}_1$ and $\mathbf{t}_2$ are perpendicular to each other. The principal curvatures are also obtained as eigenvalues of the *Weingarten curvature matrix* (or second fundamental tensor)

$$\mathbf{W} := \frac{1}{EG - F^2} \begin{bmatrix} eG - fF & fG - gF \\ fE - eF & gE - fF \end{bmatrix}. \tag{3.3}$$

$\mathbf{W}$ represents the Weingarten map or shape operator which gives the directional derivative of the normal, i.e. $\mathbf{W}\bar{\mathbf{t}} = \frac{\partial}{\partial \bar{\mathbf{t}}}\mathbf{n}$. Then the normal curvature is given as

$$\kappa_n(\bar{\mathbf{t}}) = \bar{\mathbf{t}}^\top \mathbf{W} \bar{\mathbf{t}}\,.$$

With a local coordinate system defined by the principal directions $\mathbf{t}_1$ and $\mathbf{t}_2$, $\mathbf{W}$ is a diagonal matrix, or in general

$$\mathbf{W} = \begin{bmatrix} \bar{\mathbf{t}}_1 & \bar{\mathbf{t}}_2 \end{bmatrix} \begin{bmatrix} \kappa_1 & 0 \\ 0 & \kappa_2 \end{bmatrix} \begin{bmatrix} \bar{\mathbf{t}}_1 & \bar{\mathbf{t}}_2 \end{bmatrix}^{-1}. \tag{3.4}$$

Then the normal curvature can be written as

$$\kappa_n(\bar{\mathbf{t}}) = \kappa_n(\phi) = \kappa_1 \cos^2\phi + \kappa_2 \sin^2\phi, \tag{3.5}$$

where ϕ is the angle between $\bar{\mathbf{t}}$ and $\bar{\mathbf{t}}_1$ (Euler's theorem).

The *curvature tensor* $\mathbf{T}$ is expressed as a symmetric 3×3 matrix with the eigenvalues κ_1, κ_2, 0 and the corresponding eigenvectors $\mathbf{t}_1$, $\mathbf{t}_2$, $\mathbf{n}$. $\mathbf{T}$ measures the change of the unit normal with respect to a tangent vector $\mathbf{t}$ independently of the parameterization. It can be constructed as

$$\mathbf{T} \;=\; \mathbf{P}\mathbf{D}\mathbf{P}^{-1}\,,$$

with $\mathbf{P} = [\mathbf{t}_1, \mathbf{t}_2, \mathbf{n}]$ and $\mathbf{D} = \text{diag}(\kappa_1, \kappa_2, 0)$.

The *Gaussian curvature* K and the *mean curvature* H are defined as the product and the average of the principal curvatures, respectively,

$$K \;=\; \kappa_1\kappa_2 \;=\; \det(\mathbf{W})\,, \quad \text{and} \quad H \;=\; \frac{\kappa_1 + \kappa_2}{2} \;=\; \tfrac{1}{2}\text{trace}(\mathbf{W}). \tag{3.6}$$

The mean curvature can alternatively be expressed as the (continuous) average of the normal curvatures

$$H \;=\; \frac{1}{2\pi}\int_0^{2\pi} \kappa_n(\phi)d\phi\,. \tag{3.7}$$

3.1.2 Curvature Tensor Estimation from Discrete Shapes

Estimates of the curvature tensor on polygonal meshes are applied in a variety of applications ranging from the detection of surface defects to the detection of features. Many techniques have been proposed (see, e.g., [Petitjean 2001] for a recent survey), in this section we provide an overview of different approaches.

In order to estimate the curvature tensor at a vertex a certain neighborhood of this vertex is considered, typically its 1-ring. A common approach is to first discretize the normal curvature along edges. Given is an edge (i, j), vertex positions $\mathbf{x}_i$, $\mathbf{x}_j$, and the normal $\mathbf{n}_i$, then

$$\kappa_{ij} \;=\; 2\frac{(\mathbf{x}_j - \mathbf{x}_i)\mathbf{n}_i}{\|\mathbf{x}_j - \mathbf{x}_i\|^2} \tag{3.8}$$

provides an approximation of the normal curvature at $\mathbf{x}_i$ in the tangent direction which results from projecting $\mathbf{x}_i$ and $\mathbf{x}_j$ into the tangent plane defined by $\mathbf{n}_i$. This expression can be interpreted geometrically as fitting the osculating circle interpolating $\mathbf{x}_i$ and $\mathbf{x}_j$ with normal $\mathbf{n}_i$ at $\mathbf{x}_i$ (cf. [Moreton and Séquin 1992]). Alternatively, the equation can be derived from discretizing the curvature of a smooth planar curve (cf. [Taubin 1995]). With estimates κ_{ij} of the normal curvature for all edges incident to vertex i, Euler's theorem (3.5) can be applied to relate the κ_{ij} to the unknown principal curvatures (and principal directions). Then approximates to the principal curvatures can be obtained either directly as functions of

the eigenvalues of a symmetric matrix ([Taubin 1995, Page et al. 2001]) or from solving a least-squares problem ([Moreton and Séquin 1992, Meyer et al. 2002]). Alternatively, Watanabe and Belyaev [2001] apply the trapezoid rule to get a discrete approximation of (3.7), which provides the mean curvature H, the Gaussian curvature K is obtained from a similar integral over κ_n^2, and the principal curvatures are then obtained from (3.6).

Another class of techniques for curvature tensor estimation locally fits a smooth parametric surface patch and then derives the differential quantities from that. This leaves the choice for the surface — typically polynomials of low degree — the geometric quantities to interpolate or approximate — e.g. the vertex positions in a 1-ring neighborhood — and a projection operator to obtain a parameterization — in general the projection into the tangent plane.

A straightforward choice is to consider the quadratic height surface

$$z(x,y) \;=\; \tfrac{1}{2}a_{20}x^2 + a_{11}xy + \tfrac{1}{2}a_{02}y^2 \;,$$

for a local coordinate system spanned by the normal $\mathbf{n}_i$ (in z-direction) and two orthogonal tangent vectors (in x- and y-direction) and with origin $\mathbf{x}_i = \mathbf{0}$ [Goldfeather and Interrante 2004]. Then the parameters a_{20}, a_{11}, and a_{02} obtained as a least-squares solution define the Weingarten matrix as $\mathbf{W} = \begin{bmatrix} a_{20} & a_{11} \\ a_{11} & a_{02} \end{bmatrix}$. This can be interpreted as estimating the normal curvature from parabolas rather than circles (as with (3.8)) and then solving a least-squares system like in [Moreton and Séquin 1992].

Welch and Witkin [1994] apply a quadratic Taylor polynomial of different form, namely

$$\mathbf{f}(u,v) \;=\; \mathbf{f}_u u + \mathbf{f}_v v + \tfrac{1}{2}u^2\mathbf{f}_{uu} + \mathbf{f}_{uv}uv + \tfrac{1}{2}v^2\mathbf{f}_{vv} \;.$$

The coefficients of the local least-squares approximating polynomial are the first and second order partials and hence define the fundamental forms. For robustness reasons, an exponential map is used as projection operator rather than a simple projection to the tangent plane. Numerical tests in [Rössl and Kobbelt 1999] back up this choice, moreover the exponential map can be assumed to be nearly isometric.

Most recently, Goldfeather and Interrante [2004] propose the use of a cubic approximation scheme which takes into account vertex normals in the 1-ring. As the normals themselves are local estimates, this can be interpreted as enlarging the neighborhood to a 2-ring. This leads again to a least-squares problem of finding the coefficients of the cubic height surface

$$z(x,y) \;=\; \tfrac{1}{2}a_{20}x^2 + a_{11}xy + \tfrac{1}{2}a_{02}y^2 + a_{30}x^3 + a_{21}x^2y + a_{12}xy^2 + a_{03}y^3 \;. \quad (3.9)$$

Note that the Weingarten matrix is obtained entirely from the quadratic terms in the same way as before.

In general, least-squares methods may suffer from degenerate cases — even for reasonable geometric configurations — which lead to ill-conditioned system matrices. In [Welch and Witkin 1994] the polynomial basis is successively reduced in such cases. An alternative is to provide more samples e.g. from linear interpolation. Cazals and Pouget [2003] discuss the patch fitting approach from an approximation theory point of view including robustness and numerical issues. For high-quality and consistent estimation of curvatures and their derivatives, Ohtake et al. [2004] apply a (rather expensive) global fitting of an implicit surface to the surface mesh.

In contrast to the previously mentioned techniques, tensor averaging methods estimate the curvature tensor as an average over a certain region of a polyhedral mesh. Cohen-Steiner and Morvan [2003] derive the curvature tensor building upon the theory of normal cycles. This work includes a proof of convergence under certain sampling conditions based on geometric measure theory. The curvature tensor is defined at each point along an edge, and all contributions are integrated over a small region (see also [Alliez et al. 2003]). Most recently, Hildebrandt and Polthier [2004] applied a similar discrete curvature measure.

An interesting technique is proposed in [Rusinkiewicz 2004]: The directional derivatives of the normal $\mathbf{W}_{\mathbf{t}} = \frac{\partial}{\partial \mathbf{t}}\mathbf{n}$ are expressed as finite differences for every edge of a triangle. The resulting system of $3 \times 2 = 6$ equations is set up from the vertex positions (in parameter space) and normals and then solved for the three unknowns of the Weingarten matrix $\mathbf{W}$ in least-squares sense. The tensors which are obtained per triangle are transformed to a common coordinate system to get a per-vertex average over the 1-ring. The algorithm can be applied with only slight modifications to compute curvature derivatives from the prior result.

3.2 Normal Based Estimation of the Curvature Tensor

3.2.1 Central Idea

The approaches listed above have in common that they target at and yield discrete estimations of the curvature tensor in the vertices of the mesh.[1] In [Theisel et al. 2004b], we propose an alternative approach to estimating the curvature tensor: instead of computing it per vertex, we do the estimation per triangle. We consider each triangle of the mesh (together with the normals in its

[1] The first step in [Rusinkiewicz 2004] can be interpreted as an exception.

vertices) independently and compute the curvature tensor as a smooth function on the triangle. The basic idea for doing so comes from the well-known concept of Phong-shading [Phong 1975] (see also e.g. [Foley et al. 1996]): given a triangle of a mesh together with its vertex normals, two linear interpolations are applied. The linear interpolation for the vertices gives the current location, while the linear interpolation of the vertex normals gives the normal for the illumination model. Although a certain error is taken into account — the normal from the piecewise linear surface generally differs from the linearly interpolated normal — this approach has been proven to produce smooth-looking representations of meshes.

Bearing in mind that the curvature tensor of a smooth surface is completely defined by its first order partials and the first order partials of its normals, we can use the idea of Phong shading to get an estimation of the curvature tensor on a single triangle: We use the linear interpolation of the vertices to get the surface and its first order partials, while the normals and its first order partials are obtained from the linearly interpolated vertex normals. Similar to Phong shading, this introduces a certain error which is due to the application of two different linear interpolations. However, we show that this error can compete with the errors of other estimation schemes of the curvature tensor.

Rusinkiewicz [2004] similarly takes into account isolated triangles. However, this recent approach is very different in its central idea and motivation. It defines the curvature tensor as a constant estimate per triangle rather than as piecewise smooth function. And from a technical point of view our algorithm does not require the solution of linear systems or a coordinate transform for averaging around a vertex.

3.2.2 Normal Based Estimation of $\mathbf{T}$

The new approach we present here considers only a single (non-degenerate) triangle with the vertices $\mathbf{x}_0$, $\mathbf{x}_1$, $\mathbf{x}_2$, and the corresponding (not normalized) normals $\mathbf{n}_0$, $\mathbf{n}_1$, $\mathbf{n}_2$. Then we can obtain a point and a normal on the triangle by applying a linear interpolation of $\mathbf{x}_i$ and $\mathbf{n}_i$ respectively. We describe these linear interpolations both in barycentric coordinates $(\lambda_0, \lambda_1, \lambda_2)$ and in local cartesian coordinates (u, v) with the origin $\mathbf{x}_0$ and the base vectors $\mathbf{x}_1 - \mathbf{x}_0$ and $\mathbf{x}_2 - \mathbf{x}_0$:

$$\begin{aligned}
\tilde{\mathbf{x}} &= \tilde{\mathbf{x}}(\lambda_0, \lambda_1, \lambda_2) = \lambda_0\,\mathbf{x}_0 + \lambda_1\,\mathbf{x}_1 + \lambda_2\,\mathbf{x}_2 \\
&= \tilde{\mathbf{x}}(u, v) = \mathbf{x}_0 + u\,(\mathbf{x}_1 - \mathbf{x}_0) + v\,(\mathbf{x}_2 - \mathbf{x}_0) \\
\tilde{\mathbf{n}} &= \tilde{\mathbf{n}}(\lambda_0, \lambda_1, \lambda_2) = \lambda_0\,\mathbf{n}_0 + \lambda_1\,\mathbf{n}_1 + \lambda_2\,\mathbf{n}_2 \\
&= \tilde{\mathbf{n}}(u, v) = \mathbf{n}_0 + u\,(\mathbf{n}_1 - \mathbf{n}_0) + v\,(\mathbf{n}_2 - \mathbf{n}_0).
\end{aligned} \tag{3.10}$$

The conversion between both coordinate systems is a simple affine transformation.

The main idea now is to use $\tilde{\mathbf{x}}$ and $\tilde{\mathbf{n}}$ to get the necessary vectors $\mathbf{x}_u$, $\mathbf{x}_v$, $\mathbf{n}_u$, and $\mathbf{n}_v$ to compute $\mathbf{T}$. We compute the normalized normal $\mathbf{n}$ and its derivatives as

$$\mathbf{n}(u,v) = \frac{\tilde{\mathbf{n}}}{\|\tilde{\mathbf{n}}\|} \quad , \quad \mathbf{n}_u = D_u\mathbf{n} \quad , \quad \mathbf{n}_v = D_v\mathbf{n} \; . \tag{3.11}$$

For the partials of the surface we get

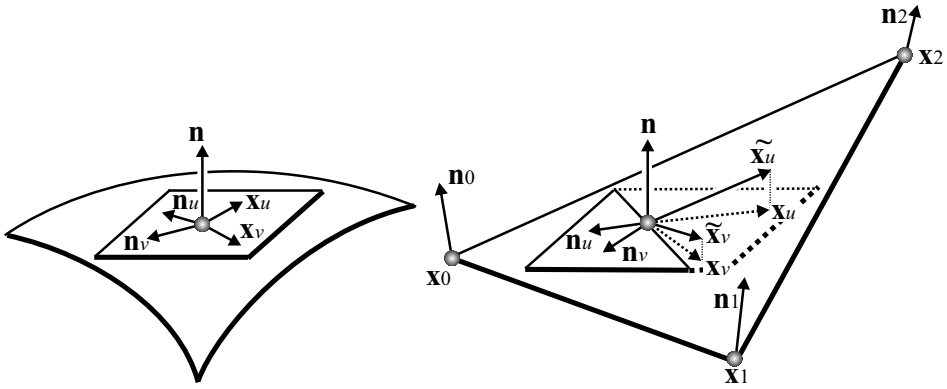
$$\tilde{\mathbf{x}}_u(u,v) = D_u\tilde{\mathbf{x}} = \mathbf{x}_1 - \mathbf{x}_0 \;\; , \;\; \tilde{\mathbf{x}}_v(u,v) = D_v\tilde{\mathbf{x}} = \mathbf{x}_2 - \mathbf{x}_0 \; .$$

$\mathbf{T}$ is completely defined by $\mathbf{x}_u$, $\mathbf{x}_v$, $\mathbf{n}_u$, and $\mathbf{n}_v$ (Figure 3.1a). For these vectors, the following statements hold
(i) $\mathbf{x}_u$, $\mathbf{x}_v$, $\mathbf{n}_u$, $\mathbf{n}_v$ are coplanar, i.e. they are in the tangent plane of $\mathbf{x}$.
(ii) $\mathbf{n}_u\mathbf{x}_v \;=\; \mathbf{n}_v\mathbf{x}_u$.
(To see this, consider the partials of the equations $\mathbf{n}^2 = 1$, $\mathbf{n}\mathbf{x}_u = 0$, and $\mathbf{n}\mathbf{x}_v = 0$.)

In order to fulfill (i), we map $\tilde{\mathbf{x}}_u$ and $\tilde{\mathbf{x}}_v$ into the plane defined by $\mathbf{n}_u$ and $\mathbf{n}_v$:

$$\mathbf{x}_u \;=\; \tilde{\mathbf{x}}_u - (\mathbf{n}\,\tilde{\mathbf{x}}_u)\,\mathbf{n} \quad , \quad \mathbf{x}_v \;=\; \tilde{\mathbf{x}}_v - (\mathbf{n}\,\tilde{\mathbf{x}}_v)\,\mathbf{n}. \tag{3.12}$$

Figure 3.1b gives an illustration. Now we have all ingredients to compute $\mathbf{T}$: We



(a) (b)

Figure 3.1: (a) $\mathbf{x}_u$, $\mathbf{x}_v$, $\mathbf{n}_u$, $\mathbf{n}_v$ completely define $\mathbf{T}$. (b) computing $\mathbf{x}_u$, $\mathbf{x}_v$, $\mathbf{n}_u$, $\mathbf{n}_v$ on a triangle.

compute the elements of $\mathbf{I}$ and $\mathbf{II}$ and from this the Weingarten matrix $\mathbf{W}$ (i.e. apply 3.1-3.3) and then determine the principal curvatures as eigenvalues of $\mathbf{W}$. Doing so, we obtain closed formulations of the Gaussian curvature K and the

mean curvature H in barycentric coordinates:

$$K(\lambda_0, \lambda_1, \lambda_2) = \frac{\det(\mathbf{n}_0, \mathbf{n}_1, \mathbf{n}_2)}{\tilde{\mathbf{n}}^2 \cdot (\tilde{\mathbf{n}}\, \tilde{\mathbf{m}})} \tag{3.13}$$

$$H(\lambda_0, \lambda_1, \lambda_2) = \frac{1}{2} \frac{(\tilde{\mathbf{n}}\, \mathbf{h})}{\|\tilde{\mathbf{n}}\| \cdot (\tilde{\mathbf{n}}\, \tilde{\mathbf{m}})} \tag{3.14}$$

with

$$\begin{aligned} \tilde{\mathbf{m}} &= \mathbf{r}_2 \times \mathbf{r}_0 = \mathbf{r}_0 \times \mathbf{r}_1 = \mathbf{r}_1 \times \mathbf{r}_2 \\ \mathbf{h} &= (\mathbf{n}_0 \times \mathbf{r}_0) + (\mathbf{n}_1 \times \mathbf{r}_1) + (\mathbf{n}_2 \times \mathbf{r}_2) \end{aligned}$$

and

$$\mathbf{r}_0 = \mathbf{x}_2 - \mathbf{x}_1 \quad , \quad \mathbf{r}_1 = \mathbf{x}_0 - \mathbf{x}_2 \quad , \quad \mathbf{r}_2 = \mathbf{x}_1 - \mathbf{x}_0.$$

This way, $\tilde{\mathbf{n}}$ is the (not normalized) linearly interpolated normal as defined before, and $\tilde{\mathbf{m}}$ is the (not normalized) triangle normal (i.e. $\|\tilde{\mathbf{m}}\| = \frac{1}{2}\text{area}(\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2)$).

Appendix A provides a derivation of these formulas and reveals an elegant expression for the Weingarten matrix $\widetilde{\mathbf{W}}$.

3.2.3 Properties of the Estimation of $\mathbf{T}$

We summarize properties if the normal based estimation of $\mathbf{T}$ as described in the previous section:

Theorem 3.1 *(properties of normals based curvature estimation)*
Given is a triangular mesh which approximates a smooth surface, and the curvature tensor $\mathbf{T}$ *is estimated as described in Section 3.2.2. The following properties hold:*

(i) $\mathbf{T}$ *converges to the curvature tensor of the smooth surface when refining the mesh.*

(ii) $\mathbf{T}$ *depends on the length of the vertex normals.*

(iii) In general, $\mathbf{T}$ *is not symmetric, and the estimated principal directions* $\mathbf{t}_1$ *and* $\mathbf{t}_2$ *are not perpendicular.*

(iv) $\mathbf{T}$ *is not continuous across edges of the triangulation.*

In order to show (i), we consider the cubic height surface (3.9). Let the coefficients $a_{20} = \kappa_1$ and $a_{02} = \kappa_2$ be certain constants, $a_{11} = 0$, and a_{30},a_{21}, a_{12}, a_{03} are certain scalar functions of (x, y) describing the higher order terms in the Taylor approximation (3.9). Note that every surface can be locally represented in this form. For this surface, the curvature tensor is well-defined at $(x = 0, y = 0)$:

$$\mathbf{T}(0,0) \;=\; \text{diag}(\kappa_1, \kappa_2, 0)\;.$$

Now we consider a triangulation of (3.9) and repeatedly refine it in the neighborhood of $(x = 0, y = 0)$. In fact, we consider a triangle of the vertices

$$\mathbf{x}_i \;=\; (t\,x_i\;,\; t\,y_i\;,\; z(t\,x_i, t\,y_i))^\top, \quad 0 \le i \le 2\;, \tag{3.15}$$

where (x_0, y_0), (x_1, y_1), (x_2, y_2) are certain constants building a non-degenerate triangle in the domain. Since $\mathbf{x}_0$, $\mathbf{x}_1$, $\mathbf{x}_2$ are on the surface defined by (3.9), we compute $\mathbf{n}_0$, $\mathbf{n}_1$, $\mathbf{n}_2$ as the surface normals of (3.9):

$$\mathbf{n}_i \;=\; (\,-z_x(t\,x_i, t\,y_i)\;,\; -z_y(t\,x_i, t\,y_i)\;,\; 1\,)^\top, \quad 0 \le i \le 2\;.$$

Note that for $t \to 0$ the triangle $(\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2)$ collapses to the single point $(0, 0, 0)^\top$ with the normal $(0, 0, 1)^\top$.

Now we compute $\mathbf{T}(\lambda_0, \lambda_1, \lambda_2)$ as described before for the triangle defined by (3.15) and (3.16). We are interested in the behavior of $\mathbf{T}(\lambda_0, \lambda_1, \lambda_2)$ for $t \to 0$. Applying some algebra yields

$$\begin{aligned} \lim_{t\to 0} \mathbf{T}(1,0,0) = \lim_{t\to 0} \mathbf{T}(0,1,0) &= \lim_{t\to 0} \mathbf{T}(0,0,1) \\ = \text{diag}(\kappa_1, \kappa_2, 0) \;&=\; \mathbf{T}(0,0) \end{aligned} \tag{3.16}$$

and hence proves (i). We refer to Appendix A for a formal proof of (3.16).

Before we discuss the remaining properties (ii)-(iv), we explain our *visualization* of $\mathbf{T}$: Given a triangle $(\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2)$ with vertex normals $(\mathbf{n}_0, \mathbf{n}_1, \mathbf{n}_2)$, we represent K and H as *focal surfaces* (see [Hagen et al. 1992]):

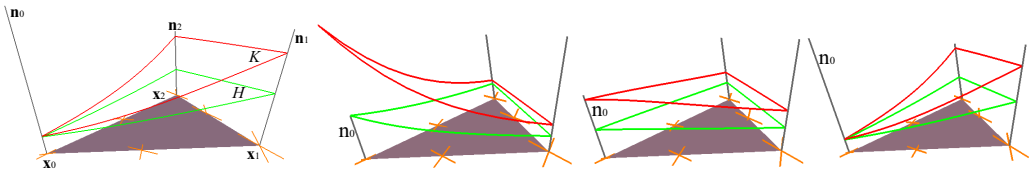
$$\begin{aligned} \mathbf{x}_K(\lambda_0, \lambda_1, \lambda_2) &= \tilde{\mathbf{x}}(\lambda_0, \lambda_1, \lambda_2) + s_K \cdot |K(\lambda_0, \lambda_1, \lambda_2)| \cdot \mathbf{n}(\lambda_0, \lambda_1, \lambda_2) \\ \mathbf{x}_H(\lambda_0, \lambda_1, \lambda_2) &= \tilde{\mathbf{x}}(\lambda_0, \lambda_1, \lambda_2) + s_H \cdot |H(\lambda_0, \lambda_1, \lambda_2)| \cdot \mathbf{n}(\lambda_0, \lambda_1, \lambda_2) \end{aligned}$$

where s_K and s_H are global positive scaling factors controlling the distance between the mesh and the focal surfaces. In order to visualize $\mathbf{x}_K$ and $\mathbf{x}_H$, we only show their boundary curves in a green (for positive K/H) or red (for negative K/H) color.[2] The estimated principal directions $\mathbf{t}_2$ and $\mathbf{t}_2$ at a certain point $\tilde{\mathbf{x}}(\lambda_0, \lambda_1, \lambda_2)$ are visualized as line segments

$$(\tilde{\mathbf{x}} - s_L \mathbf{t}_i\;,\; \tilde{\mathbf{x}} + s_L \mathbf{t}_i), \quad i = 1, 2,$$

[2]Here, we prefer such a geometric representation of K and H to a color coding on the surface (which is the most common approach) because the human eye reacts far more sensitive to small perturbations in shape than in color.

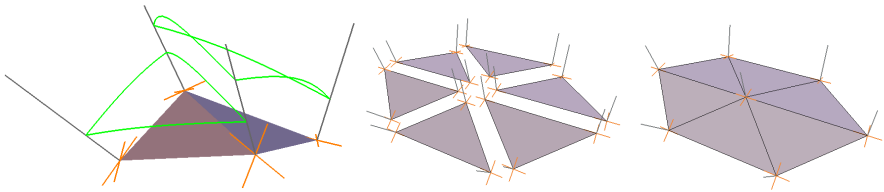
where s_L is a positive global scaling factor. Figure 3.2 (a) shows an example for a single triangle.



(a) (b)

Figure 3.2: (a) A single triangle with visualizations of H, K (focal surfaces) and $\mathbf{t}_1/\mathbf{t}_1$ in six sample points. (b) Changing the length of $\mathbf{n}_0$ changes H, K and $\mathbf{t}_1/\mathbf{t}_2$, the figure shows three configurations. Here K is negative (red surface), and H is positive (green surface).

Property (ii) is obvious from the definitions, where the given vertex normals are *not* normalized. Figure 3.2 (b) illustrates different curvature values for changing the length of one normal $\mathbf{n}_0$.



(a) (b)

Figure 3.3: (a) K across two adjacent triangles is not continuous. (b) Estimating $\mathbf{T}$ at a vertex $\mathbf{x}_i$. Left: Each triangle sharing $\mathbf{x}_i$ gives another $\mathbf{T}$. Right: principal directions of the averaged $\mathbf{T}$.

In order to see why (iii) applies, we recall the definition of the second fundamental form (3.2) and the Weingarten matrix (3.3). For smooth surfaces we have $\mathbf{x}_{uv}\mathbf{n} = \mathbf{x}_{vu}\mathbf{n}$ or $-\mathbf{n}_u\mathbf{x}_v = -\mathbf{n}_v\mathbf{x}_u$. This does not hold here, in fact we get instead

$$\widetilde{\mathbf{W}} = \frac{1}{EG - F^2} \begin{bmatrix} eG - f_1F & f_2G - gF \\ f_1E - eF & gE - f_2F \end{bmatrix},$$

where $f_1 = \mathbf{x}_{uv}\mathbf{n}$ and $f_2 = \mathbf{x}_{vu}\mathbf{n}$. Hence, $\mathbf{T}$ is not symmetric in general, and consequently $\mathbf{t}_1$ and $\mathbf{t}_2$ are not necessarily perpendicular.

(iv) is obvious as we consider triangles independently of each other. Figure 3.3 (a) illustrates this. Both, (iii) and (iv) indicate an error in the estimation of $\mathbf{T}$. However, our numerical tests in the next section show that this error can compete with errors of other estimation techniques, especially if averages are applied to compute $\mathbf{T}$ at a vertex (see Figure 3.3 (b) and next section).

3.3 Evaluation of Curvature Estimation Methods

We evaluate several methods for estimating the curvature tensor to rate the normal based approach described in the previous section. The numerical tests apply discrete, piecewise linear approximations of well-defined smooth parametric surfaces such that the exact curvature tensor is available at every vertex. The comparison between the exact and the estimated quantities gives a measure of the quality of the estimation. We consider a torus and the test surface used by Goldfeather and Interrante [2004][3]. Both surfaces cover a variety of different curvature configurations.

Triangulations of Test Surfaces

Most estimators give good results for rather regular triangulations, while for irregular triangulations their quality tends to drop significantly (see e.g. [Goldfeather and Interrante 2004]). We use two different triangulations for each of the two test surfaces with different types of irregularity. The first triangulation places the vertices on a regular grid in the domain while the triangulation of each grid cell is chosen randomly. This way the resulting mesh contains vertices with varying valences ranging from four to eight. For the second triangulation we apply a perturbation of the grid points in the parameter domain, then a (2D-) Delaunay triangulation is applied. This way we obtain an irregular triangulation where the majority of the vertices reaches the average valence six. Figure 3.4 illustrates and labels the discrete approximations of the surfaces used for the tests. The meshes consist of 20.000 (torus) and 10.000 (Goldfeather) triangles, respectively. Obviously, for the normal based curvature tensor estimation, the quality of $\mathbf{T}$ strongly depends on the quality of the available normals. If the underlying surface is known in an implicit form, the "perfect" normals (both in direction and length) are available by considering the gradient. In case of an available parametric description of the underlying surface, the exact normal direction can be obtained as

[3] In fact, we consider only the middle part of this surface since the joints to the top and bottom parts are not curvature continuous.

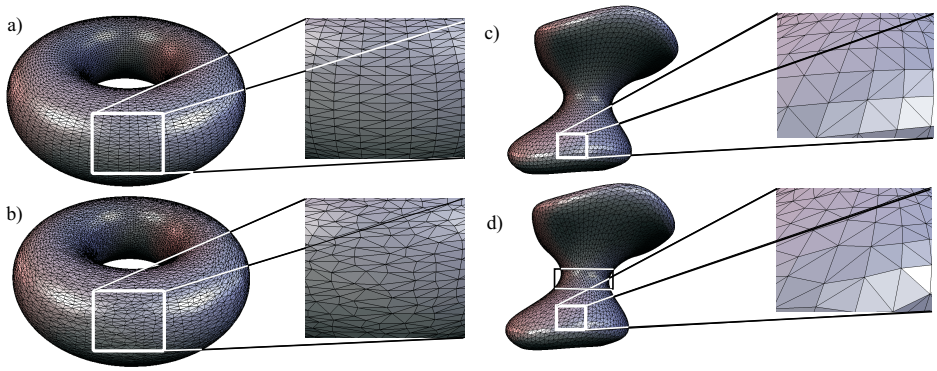


Figure 3.4: The four test triangulations: a) *Torus 1*. b) *Torus 2*. c) *Goldfeather 1*. d) *Goldfeather 2*.

well. However, in most cases they have to be estimated from a discrete approximation. A number of approaches exist (see e.g. [Max 1999, Meyer et al. 2002] and the references therein). For our computations we used the simplest weighting triangle normals by the triangle areas (also applied in [Taubin 1995]), which already provides good results.

Estimation-per-Vertex Setting

In order to find a fair comparison between the described normal based estimation and pre-existing approaches, we carefully have to choose an appropriate setup because of the different nature of the estimators: Other methods yield discrete estimations of $\mathbf{T}$ in the vertices, while our method gives continuous functions of $\mathbf{T}$ inside each triangle (with discontinuities at the boundaries and the vertices). We make both approaches comparable and adapt our method to compute $\mathbf{T}$ in every vertex of the mesh. In general, our method gives n different estimations of $\mathbf{T}$ for a vertex with a valence of n, i.e. one for each attached triangle. We get a unique $\mathbf{T}$ for every vertex from computing the (uniformly weighted) average of all estimations of $\mathbf{T}$ in this vertex. Figure 3.3 (b) illustrates this setting.

Error Measures

We require a distance metric of the estimated $\mathbf{T}$ for the evaluation. We do not consider $\mathbf{T}$ directly but extract the Gaussian curvature, mean curvature and principal directions and provide distances based on these measures. In particular, we compute the distances of the Gaussian curvature between two estimations (or between

one estimation and the exact values) as

$$\text{dist}_K = \frac{1}{N}\sum_{i=1}^{N}\left(K_1(i) - K_2(i)\right)^2 \ ,$$

where N is the number of vertices of the mesh and $K_1(i)$, $K_2(i)$ are estimations of K at the i-th vertex using the two methods to be compared. Distances dist_H between estimates H_1 and H_2 of the mean curvature are defined analog.

We define the distance of estimates $\mathbf{t}_1$, $\mathbf{t}_2$ and $\mathbf{t}_1'$, $\mathbf{t}_2'$ of the principal directions in a vertex i as

$$\text{dist}_P(i) = \tfrac{1}{2}\min\left\{\arccos|\mathbf{t}_1\mathbf{t}_1'| + \arccos|\mathbf{t}_2\mathbf{t}_2'|\ ,\ \arccos|\mathbf{t}_1\mathbf{t}_2'| + \arccos|\mathbf{t}_2\mathbf{t}_1'|\right\}.$$

This gives the average angle deviation between the corresponding directions, and yields

$$\text{dist}_P = \frac{1}{N}\sum_{i=1}^{N}\text{dist}_P(i)$$

for the global distance of the principal directions.

Evaluation

We compare the normal based estimation with three well-established estimation methods which can be considered as being among the most powerful methods which are currently available (cf. [Goldfeather and Interrante 2004]). In fact, we compare the following estimations with the exact curvature values:

- The cubic fitting ([Goldfeather and Interrante 2004]) incorporating estimated normals (in the following called CF),
- the quadratic fitting ([Goldfeather and Interrante 2004]) (QF),
- the quadratic fitting ([Welch and Witkin 1994]) (QT),
- the normal based estimation using the exact (perfect) normals (NP), and
- the normal based estimation using estimated normals by a weighted average of the triangle normals (NE).

For each of the four test meshes we consider five versions: the original one and four with uniform noise added. The noisy versions are created by translating each vertex in normal direction, where the length of the displacement is a chosen randomly in the four intervals $[0, s \cdot \frac{d}{100}]$, where $s \in \{0, \frac{1}{2}, 1, 2, 3\}$ and d is the length of the bounding box diagonal. Figure 3.5 shows error plots for the *torus 1* data

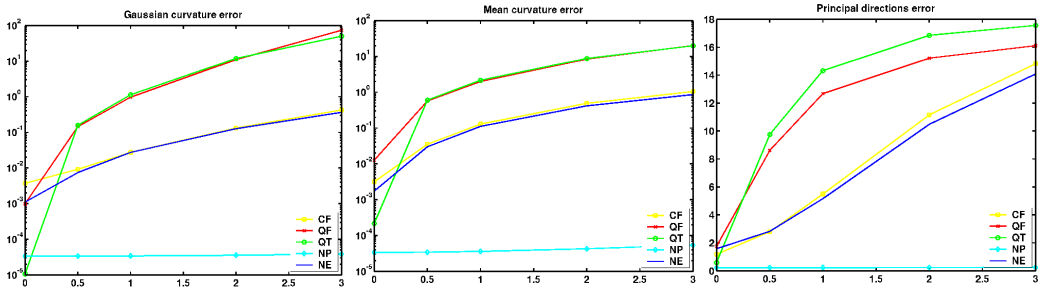


Figure 3.5: *Torus 1*: Error (vertical axes) against amount of noise (horizontal axes).

set. The horizontal axes of the diagrams show the amount of noise $s\,d$, the vertical axes show the errors between the estimation and the exact values for Gaussian curvature, mean curvature and principal directions, respectively. The results for the five different estimation techniques are shown in different colors. The figure indicates that the normal based estimation outperforms the other techniques if the exact normals are available (NP), e.g. if an implicit surface is sampled. If the normals have to be estimated, our method (NE) yields similar results as the cubic fitting (CF). In fact, in most cases NE performs slightly better than CF. The two quadratic fitting methods QF and QT show a rather similar behavior revealing a larger error than NE and CF. This is due to the fact that NE and CF incorporate a 2-ring to estimate $\mathbf{T}$ while QT and QF work only on a 1-ring around the vertex. Figure 3.6 shows the error plot for the torus 2 data set using the same setup as

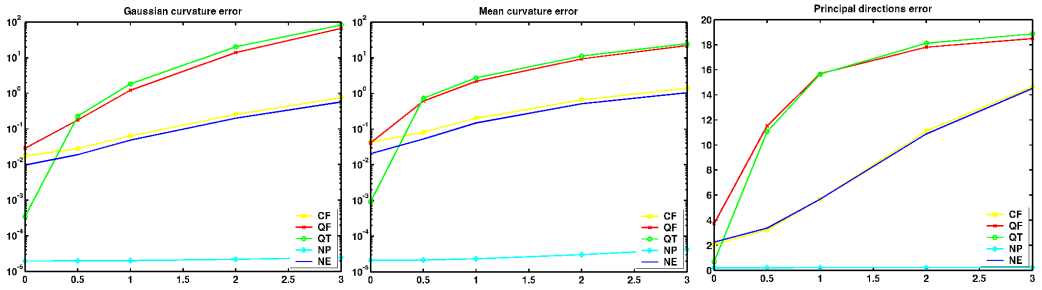


Figure 3.6: Torus 2: Error (vertical axes) against amount of noise (horizontal axes.

for Figure 3.5. We obtain a similar result as for *torus 1*: NP generally performs best, CF and NE have a similar performance (where NE is slightly better), while QT and QF tend to produce the largest error. Figure 3.7 visualizes the Gaussian curvature as focal surfaces over the triangles (see Section 3.2.3) for torus 2 with added noise ($s = \frac{1}{2}$), comparing the exact result with the different estimations. Here, we also compare the normal based estimation for independent triangles to the per-vertex average. The figure clearly shows that NP comes closest to the cor-

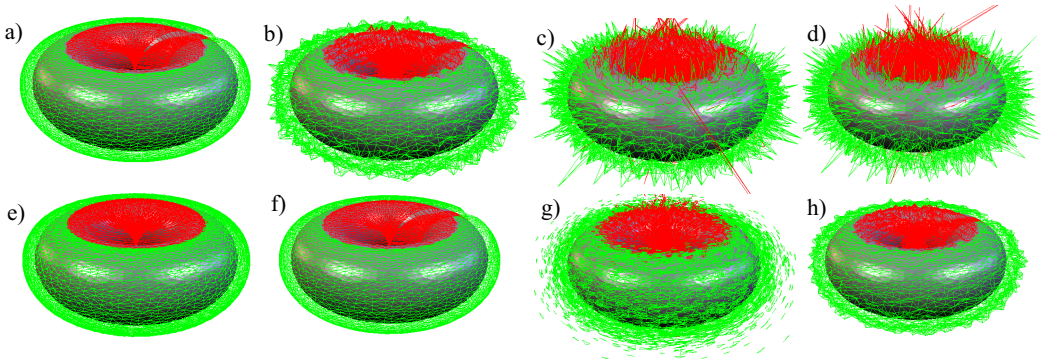


Figure 3.7: Focal surfaces of Gaussian curvature for torus 2: a) exact, b) CF, c) QF, d) QT, e) NP (independent triangles), f) NP (averaged at vertex), g) NE (independent triangles), and h) NE (averaged at vertex).

rect curvature plot, that CF and NE have a similar behavior, and that QF and QT introduce the largest errors. A similar statement holds for Figure 3.8 which shows

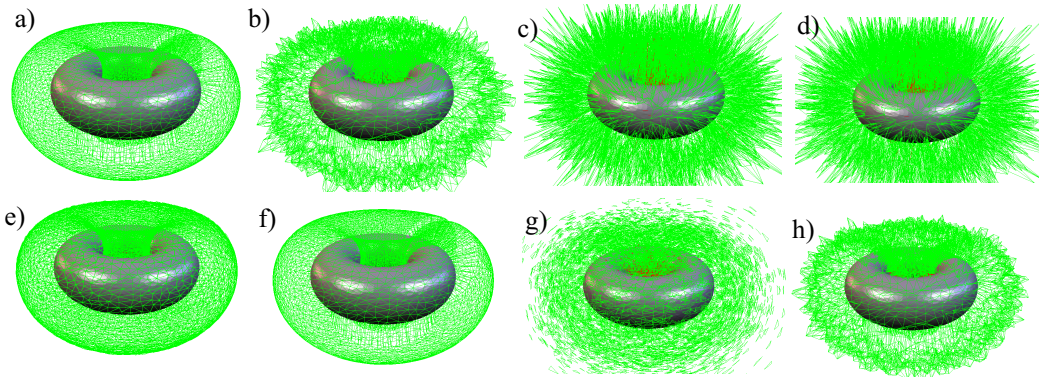


Figure 3.8: Focal surfaces of mean curvature for torus 2: a) exact, b) CF, c) QF, d) QT, e) NP (independent triangles), f) NP (averaged at vertex), g) NE (independent triangles), h) NE (averaged at vertex).

a collection of mean curvature plots for the same data set.

Figure 3.9 shows the error plots for the *Goldfeather 1* data set with the same setup as in Figures 3.5 and 3.6. Figure 3.10 does so for the *Goldfeather 2* mesh. Both plots reveal a rather similar behavior as observed for the tori: NP performs best, NE and CF have similar errors, while QF and QT give the largest error for slightly noisy data. The same trend can be seen in the curvature plots of the *Goldfeather 2* surface with added noise ($s = \frac{1}{2}$). Figures 3.11 and 3.12 show focal surfaces of the Gaussian curvature and the mean curvature, respectively, while Figure 3.13 visualizes the principal directions.

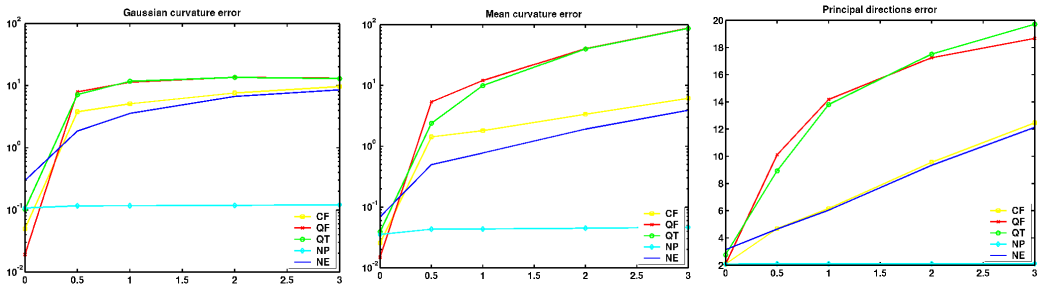


Figure 3.9: *Goldfeather 1*: Error (vertical axes) against amount of noise (horizontal axes).

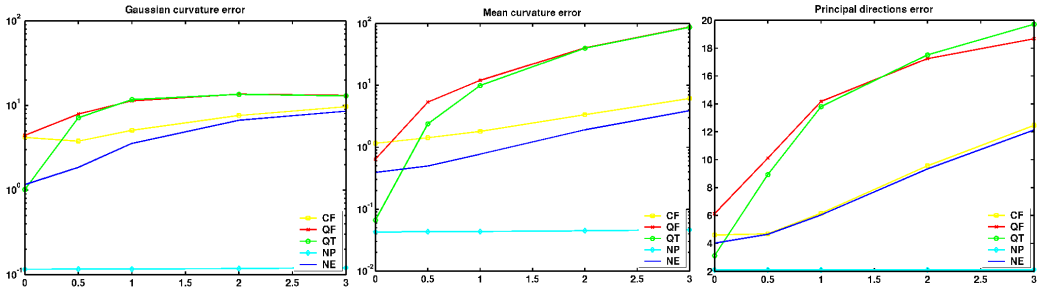


Figure 3.10: *Goldfeather 2*: Error (vertical axes) against amount of noise (horizontal axes).

Finally, we apply our normal based estimation to a large real world data set. Figure 3.14 (a) shows the triangular which was mesh obtained from a range scan of the wooden sculpture *Freezing Old Woman* (*Frierende Alte*, 1937) by the German expressionist sculptor Ernst Barlach (1870–1938). The model consists of about 1.5 million triangles, and it took 2.9 seconds to compute the curvature tensor at each vertex on a 3 GHz Intel Pentium 4 computer. (It was acquired by Hendrik Lensch, Holger Theisel, and Heiko Wanning with kind permisson of Volker Probst, Ernst Barlach foundation, Güstrow.) Figure 3.14 (b) shows a closeup of the focal surface of the mean curvature for the region near the nose and right eye. The principal directions are visualized for a region of two fingertips in Figure 3.14 (c).

3.4 Detection of Feature Lines

A typical application for discrete curvature estimates is the detection of surface features, e.g. for recovering structural information from a digitized shape in reverse engineering. In the following we discuss an approach to surface segmentation based on morphological operators (cf. [Rössl et al. 2000a, Rössl et al. 2001]),

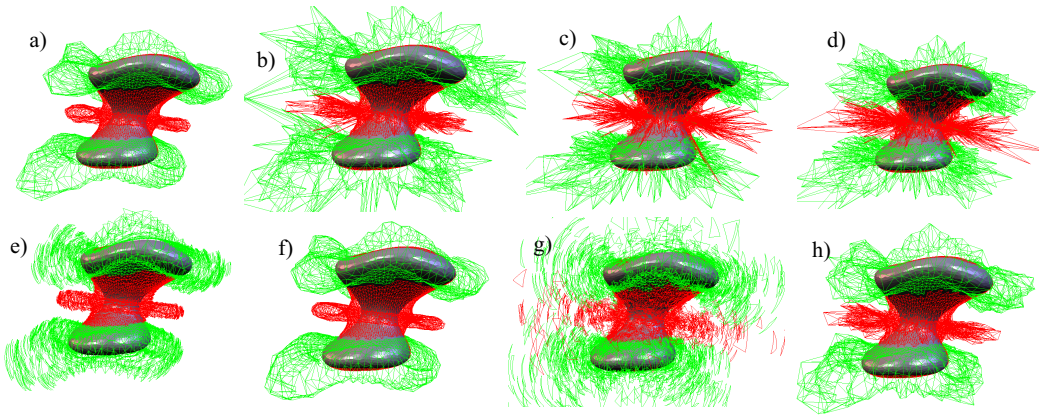


Figure 3.11: Focal surfaces of Gaussian curvature for *Goldfeather 2*: a) exact, b) CF, c) QF, d) QT, e) NP (independent triangles), f) NP (averaged at vertex), g) NE (independent triangles), h) NE (averaged at vertex).

which is effective, efficient and simple in implementation.

3.4.1 Surface Segmentation and Feature Detection

We focus on recovering structural information from densely sampled triangulated surfaces potentially with some stochastical noise as they are typically obtained from digitizing real-world objects. Deriving structural information is also known as *segmentation*, a tool which is crucial for the reverse engineering process where suitable computer models are created from surfaces of physical objects. An overview on this topic is given e.g. in [Várady and Benkő 2000].

There are two generic approaches to surface segmentation. The first one is *region growing* or also called *face-based* approach [Várady et al. 1997]. A small seed region is grown by adding neighboring regions that are similar with respect to some flatness criterion e.g. [Isselhard et al. 1998, Sapidis and Besl 1995]. Besides the bottom-up region growing there is also a top-down approach. In [Várady and Benkő 2000] a sophisticated cascade of surface fitting steps is used that finally decomposes a surface region in the last step when no surface primitive could be determined for this region. An initial segmentation is obtained by a preprocessing step (planar filtering) similar to the second approach below.

This second approach is called *edge-detection* or *edge-based*. Instead of finding explicitly the different surface regions, the boundaries between these regions are estimated e.g. by assuming rapid changes of angular variation at edges [Hoschek and Dankwort 1996]. Edge detection schemes may suffer from noise and poor sampling in sharp surface regions. With our approach, we try to overcome some of these difficulties:

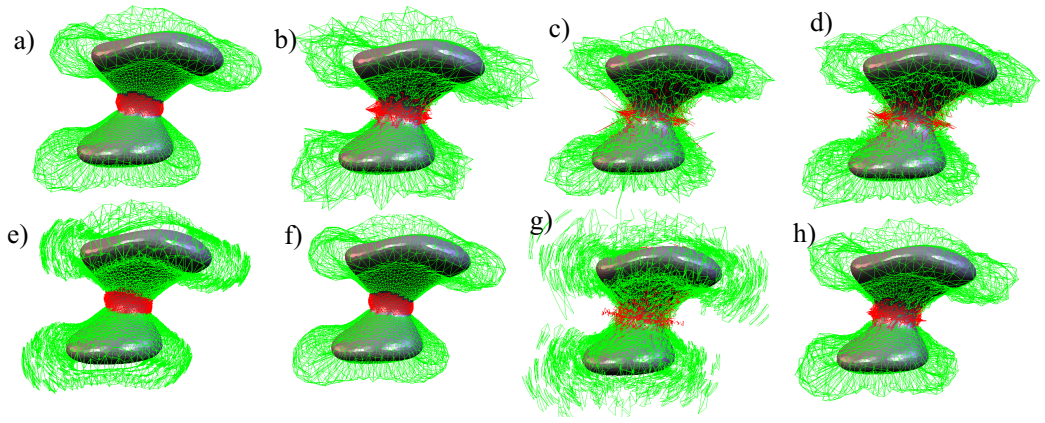


Figure 3.12: Focal surfaces of mean curvature for *Goldfeather 2*: a) exact, b) CF, c) QF, d) QT, e) NP (independent triangles), f) NP (averaged at vertex), g) NE (independent triangles), h) NE (averaged at vertex).

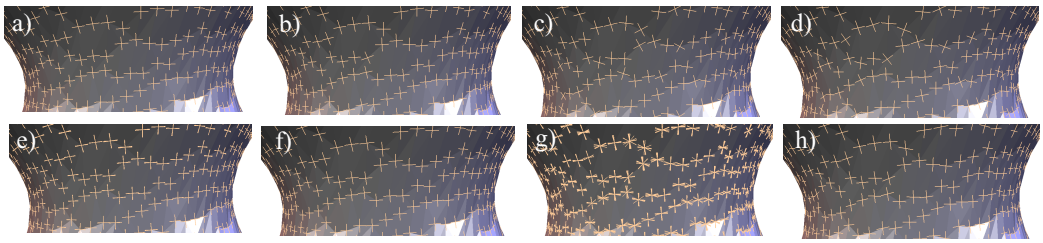
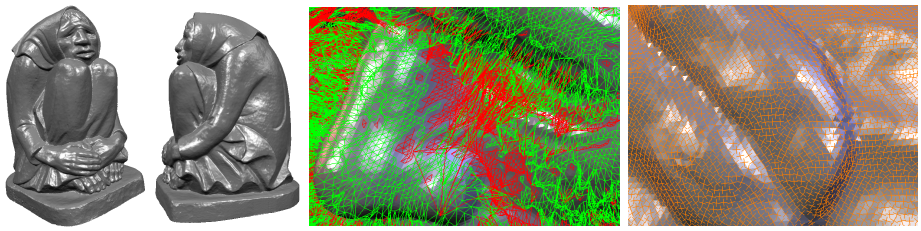


Figure 3.13: Principal direction for *Goldfeather 2*: a) exact, b) CF, c) QF, d) QT, e) NP (independent triangles), f) NP (averaged at vertex), g) NE (independent triangles), h) NE (averaged at vertex).

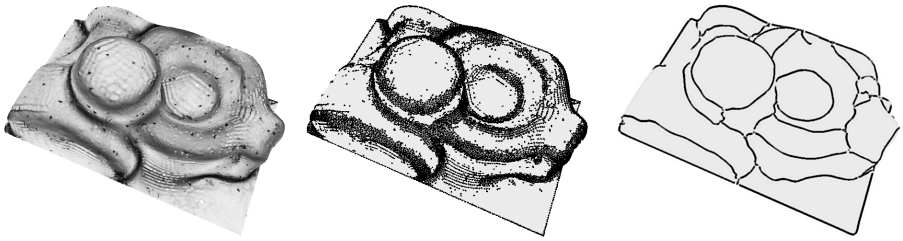
We use an arbitrary edge detection scheme that is based on discrete surface curvature information and a conservative criterion for extracting "feature regions". The extracted regions will not represent the desired boundaries or surface features exactly, but most of these feature lines can be expected to lie inside the detected regions. The wide feature regions are narrowed down to feature lines that approximate the structural information and which can be used as natural boundaries for different surface regions. Figure 3.15 illustrates this procedure.

In order to implement this skeletonization process, morphological operators are generalized from digital image processing to triangle meshes. These operators then work on Boolean valued functions on arbitrary meshes rather than on regular domains. Dilation and erosion are used for reducing noise on the feature regions while a skeletonization operator is introduced for shrinking these regions down to feature lines. For constructing consistent boundary curves, a more sophisticated graph structure is extracted from the feature lines representing the network of



(a) (b) (c)

Figure 3.14: (a) Large data set: Ernst Barlach *Freezing Old Woman* (1.5 million triangles). (b) Focal surface of mean curvature around nose region. (c) Principal directions for region of two fingertips.



(a) (b) (c)

Figure 3.15: The figures show a resampled version of the well-known benchmark object from [Hoschek and Dankwort 1996]. The maximum curvature on the mesh (a) is thresholded to obtain the initial feature regions (b). The (smoothed) feature lines are then extracted from these regions by skeletonization (c).

boundary curves.

The operators used are simple and can be implemented very efficiently because they only rely on basic operations on the triangle mesh data structures. Of course there is a tradeoff between simplicity and the capabilities of our algorithm as will be explained below.

3.4.2 Setup

We assume that the vertex positions are more or less uniformly scattered over the surface such that the edge lengths do not vary too much and that the connectivity is rather regular. This is usually the case for digitized shapes, otherwise the input mesh should be resampled or remeshed in a way that the condition is fulfilled (see

for instance Vorsatz et al. [2001, 2003a, 2003b]).

This is necessary for our morphological operators to work reasonably, as they are *topological*[4] operators. As a consequence, a *geometric* interpretation is only valid if there is some correlation, i.e. edge lengths are approximately constant. Working just on the mesh topology results in a very efficient algorithm, much in the spirit of morphological operators in digital image processing.

Feature regions are detected from discrete curvature analysis, i.e. curvature information is used to classify feature and non-feature vertices. This may be done with a simple but robust thresholding operation on e.g. the maximum curvature at a vertex (cf. Figure 3.15) or a more sophisticated scheme for ridge detection such as [Lukács and Andor 1998, Watanabe and Belyaev 2001]. Either way, the criterion for feature vertices is relaxed such that wide feature regions are extracted.

Consider a triangle mesh with vertices $\{1, \ldots, N\}$ and edges $\mathcal{E} \subset \mathcal{K}$. Then we describe the feature region as the vector $\mathbf{F} \in \{0, 1\}^N$ with

$$\mathbf{F}_i \;=\; \begin{cases} 1, & i \text{ is a feature vertex,} \\ 0, & \text{else.} \end{cases}$$

Vertices i with $\mathbf{F}_i = 1$ are called *marked*. For convenience we introduce an alternative notation and associate $\mathbf{F}$ the set

$$\mathbb{F} := \{i \in \{1, \ldots, N\} \,|\mathbf{F}_i = 1\}\,,$$

with $\overline{\mathbb{F}} := \{1, \ldots, N\} \setminus \mathbb{F}$.

There is only one operation on the triangle mesh that is needed: the enumeration of the 1-neighborhood of a vertex. We define this neighborhood relation as

$$\mathrm{nhd}(i) := \{i\} \cup \{j|(i, j) \in \mathcal{E}\}\;.$$

Here, we restrict ourselves to 1-neighborhoods. A recursive definition of d-neighborhoods is discussed in [Rössl et al. 2001], these enlarged neighborhoods have been applied alternatively to set up local color tables for visualization in [Rössl and Kobbelt 1999].

3.4.3 Morphological Operators and Skeletonization

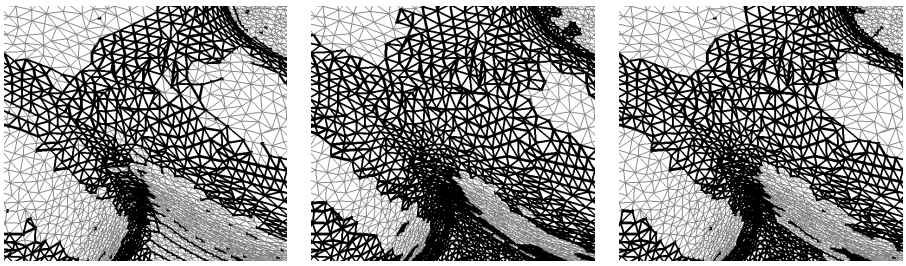
Mathematical morphology is a common tool in digital image analysis. Morphological operators are particularly interesting and often preferred to convolution operators because of their simplicity and the fact, that they can be efficiently implemented in hardware (cf. [Haralick et al. 1987]).

[4]Here and in the following, *topology* refers to the mesh connectivity graph.

We adapt morphological operators to operate on the binary feature vector $\mathbf{F}$. As we are using discrete curvature for setting up the initial feature region, high frequency noise may be a problem. Apart from some prefiltering of the input data, we use the dilation and erosion operators to suppress "classification noise" in $\mathbf{F}$.

The classical definitions of dilation and erosion are based on addition in a (m-dimensional) Euclidean vector space E^m. For instance, the dilation operator generates from two given sets $A, B \subset E^m$ the union $A \oplus B = \{c \in E^m \,|\, c = a + b, a \in A, b \in B\}$. Here A is the image or pattern to be dilated, and B denotes the so called structure element. Note that we are dealing with *binary* values in contrast to gray scale operators as used e.g. in [Hoschek and Dietz 1998] for edge detection.

Such definitions for a vector space E^n cannot be used directly on general triangle meshes. Since the connectivity of the mesh is irregular, there is no reasonable definition for an addition. Therefore we generalize morphological operators for triangle meshes — even though in a limited way, i.e. only for special types of structure elements.



(a) (b) (c)

Figure 3.16: (a) Initial feature region (close up view from Figure 3.15), after dilation (b), and subsequent erosion (c). This illustrates an application of the closing operator.

Dilation and Erosion

We define the *dilation* and *erosion* operators as

$$\begin{aligned} \mathrm{dilate}(\mathbb{F}) &= \{j \,|\, \exists i \in \mathbb{F} : j \in \mathrm{nhd}(i)\} \text{ , and} \\ \mathrm{erode}(\mathbb{F}) &= \{j \,|\, \mathrm{nhd}(j) \subseteq \mathbb{F}\} \text{ .} \end{aligned}$$

This definition resembles the classical one for a Euclidean vector space with the restriction to a disk-like structure element $\{(x, y) \mid -1 \leq x, y \leq 1\}$. (As one single structure element nhd will be employed, we use a unary notation for the operators.) The dilation operator adds vertices to the feature, i.e. $\#\mathrm{dilate}(\mathbb{F}) \geq \#\mathbb{F}$.

So $\mathbb{F}$ is grown in a way roughly preserving its "shape" on the mesh. The dilation operator can therefore be effectively used to fill "holes" of unmarked vertices inside and at the boundary of the feature. In order to reverse the effect of dilation and to, ideally, recover the original shape, the erosion operator is used to "shrink" $\mathbb{F}$ as if it were to cut off undesired branches.

Opening and Closing

As indicated, dilation and erosion remove classification artifacts that may be left even after prefiltering the geometry, but they do not preserve the overall size of the feature. This can be avoided by combining both operators. First eroding and then dilating $\mathbb{F}$ will cut branches while preserving the original shape. The combined operator is called *opening*. Furthermore, small isolated regions of marked vertices are just removed as small branches. By changing the order of the operations we obtain the *closing* operator that fills holes in the interior of the feature and cuts along the boundary.

Opening and closing effectively reduce noise on the feature region $\mathbb{F}$. Figure 3.16 shows the effects of closing. Both operators are very easy to implement, and can be applied efficiently. The resulting feature region approximately preserves the shape of the initial feature, and thus it is still a region rather than lines, i.e. it is too wide to be useful. So further operators are defined for narrowing and extracting feature lines.

Skeletonization

Here, *feature lines* are polygons defined by the triangle edges between feature vertices. A suitable removal criterion (see below) ensures that there are no triangles in the mesh that contain more than two such feature vertices with the exception of regions where multiple feature lines meet.

For the extraction of these feature lines, we use a new skeletonization or thinning algorithm similar to techniques used in digital image processing (see e.g. [Gonzales and Woods 1993]). Mathematically, the skeleton of the feature region can be defined via the medial axis transformation. The skeleton or medial axis of a region R with boundary B contains all points in R that have more than one single closest neighbor in B. This can be imagined as the set of centers of the largest disks that lie completely in R and are not included in any larger disk in R.

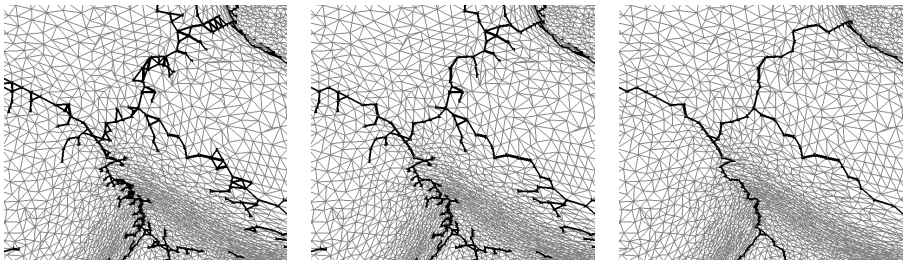
The medial axis transformation is relatively expensive to compute even for a 2D image. This is why iterative algorithms have been developed which efficiently produce a reasonable approximation of the skeleton. These thinning or skeletonization algorithms typically thin the region by peeling off one layer after another until the skeleton finally remains.

We propose a similar technique for the problem of refining the feature regions on a triangle mesh. This can be interpreted as a controlled erosion. So we will obtain the *topological* medial axis rather than the geometric one. Defining an additional criterion, whether a marked vertex may be removed from the feature or not, is the essential difference between the skeletonization and the erosion operator. This guarantees topology preservation. Hence, we define a special class of vertices that must not be removed from the feature vector $\mathbf{F}$:

Consider a clockwise traversal of the $n = \#\mathrm{nhd}(i) - 1$ direct neighbors of vertex i. Let $(v_{i,\nu})$, $0 \leq \nu \leq n$ be the ordered sequence of these neighbors, and let

$$c_i := \sum_{0 \leq \nu < n} |\mathbf{F}_{v_{i,\nu}} - \mathbf{F}_{v_{i,\nu+1 \bmod n}}| \,.$$

We define a vertex i as *complex* iff $\mathbf{F}_i = 1$ and $c_i \geq 4$. The number c_i is called the *complexity* of i.



(a) (b) (c)

Figure 3.17: Results after skeletonize (a), preprune (b) and prune (c).

In order to determine whether a vertex i is complex, one enumerates its neighbors $v_{i,\nu}$ while counting the number c_i of transitions from marked to unmarked vertices and vice versa. The resulting complexity c_i is always even. Complex vertices are either part of the feature line (*arc*, $c_i = 4$) or they belong to a *node*, where several of such lines meet ($c_i > 4$). For practical reasons, it might be useful to define all outer boundary vertices of the mesh as complex, i.e. the boundary of the surface is always part of the feature lines.

We call a vertex $i \in \mathbb{F}$ a *center* if $\mathrm{nhd}(i) \in \mathbb{F}$, a marked vertex with all its neighbors marked. Based on the definitions of complex vertices and centers we define the skeletonization procedure as

Algorithm 3.1 *(skeletonization step)* $\mathrm{skeletonize}(\mathbb{F}) :=$

1. *Let* $\mathbb{F}' := \mathbb{F}$.

2. *For all $i \in \mathbb{F}'$ which are centers w.r.t. $\mathbb{F}'$,*
   enumerate all marked neighbors $j \in \mathbb{F}, (i,j) \in \mathcal{E}$
   If j is neither center nor complex w.r.t. $\mathbb{F}'$ then $\mathbb{F}' := \mathbb{F}' \setminus j$
3. *Output $\mathbb{F}'$.*

The operator $\mathrm{skeletonize}$ is iteratively applied to the feature set $\mathbb{F}$. With every iteration the outmost layer of feature vertices is peeled off. These vertices can be characterized as being part of 1-rings around centers while not being centers themselves, which is equivalent to erosion. By additionally respecting complex vertices, the resulting thin parts of the feature do not vanish, but will form the final feature skeleton. The topology of the feature region is preserved, and connected parts will remain connected.

It is obvious, that $\mathbb{F}' := \mathrm{skeletonize}(\mathbb{F}) \subseteq \mathbb{F}$. If there are either no centers in $\mathbb{F}$ or if all neighbors to center vertices are complex, then $\mathbb{F}' = \mathbb{F}$, and the skeletonization terminates. As the number of centers can only decrease with every iteration, the algorithm always terminates after a finite number of iterations. Note that the number of complex vertices usually increases with every iteration.

The resulting feature contains complex vertices and vertices that cannot be removed, because they do not have a center as neighbor. These are vertices at the end of feature-line-branches and vertices which are close to the feature lines and hence violate the previously demanded property of a maximum of two feature vertices per triangle in "non-node" regions (cf. Figures 3.18/3.19 (a)). Both classes of vertices are disturbing and will be eliminated in the next step.

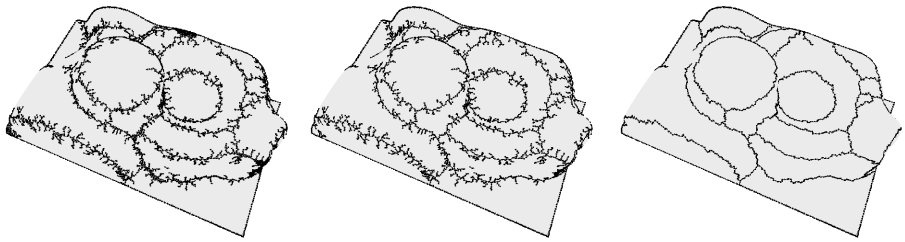
The first so called pre-pruning step will remove the second class of vertices so that the skeleton will consist of feature lines only. Therefore, all non-complex vertices are removed from the feature set unless they are not situated at the end of branches, i.e. they have more than one marked neighbor. Note that vertices are removed iteratively, one after another, where each removal may render other feature vertices complex. This way configurations of complex vertices as shown in Figure 3.19 (a) are handled correctly. If there are no more vertices to be removed the pre-pruning stops.

The resulting feature set represents the feature lines just as demanded (cf. Figures 3.18/3.19 (b)), but there are still small branches that are regarded as unwanted artifacts. The ends of these branches are the only non-complex vertices, i.e. the class of non-complex vertices which was not removed by pre-pruning. Now the definition of an iterative pruning scheme which cuts these branches is simple: in every iteration all non-complex vertices are removed, or $\mathrm{prune}(\mathbb{F}) := \{i \in \mathbb{F} \mid i \text{ is complex}\}$.

Of course this way one cannot clearly distinguish wanted feature lines from unwanted artifacts. A simple heuristic is that unwanted branches are short, say

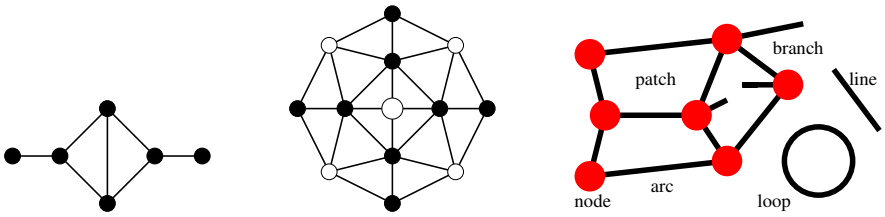
of length $\leq m$ vertices. So prune is iterated m times. In order to prevent open ended feature lines from being shortened, we store the history i.e. the sequence of removed vertices. So after pruning, we can back off and restore these vertices for branches of minimum length m. The back-tracking is started from all open ends (non-complex vertices) which survived the application of prune^m. The separation into two pruning procedures enables the treatment of these feature branches.

The resulting feature lines represent the *topological* medial axis of the initial wide feature regions (cf. Figure 3.18/3.19 (c)). They roughly follow the center of these regions as the mesh is assumed to have very uniform edge lengths allowing a reasonable *geometric* interpretation. In order to use the feature lines in a comfortable way and to be able to implement more high-level operations, some postprocessing will be done for embedding the lines into the mesh structure.



(a) (b) (c)

Figure 3.18: Results after skeletonize (a), preprune (b) and prune (c).



(a) (b) (c)

Figure 3.19: (a) The preprune operator must not change the feature topology, so only one non-complex vertex is removed at a time. (b) The non-complex center vertex is considered to be part of the node. (c) Elements of the high-level graph structure.

3.4.4 Recovering Structural Information

After skeletonization and pruning, $\mathbb{F}$ represents the wanted feature lines as a set of vertices. This set is finally used to construct a graph where the feature vertices are associated with certain elements of that graph. We distinguish between the following elements (cf. Figure 3.19 (c) and Figure 3.15 for an example):

- *Nodes* are clusters of complex vertices i with complexity $c_i > 4$. They are the first elements to be identified by finding a seed vertex for the next node and then iteratively adding adjacent node vertices. This is repeated until no more seed vertices can be found. Prepruning may remove non-complex feature vertices that will cause "holes" in a single node like the center vertex Figure 3.19 (b). Such vertices that have only node-vertices as neighbors are considered to be part of the node.

- *Arcs* are ordered sequences of complex vertices with $c_i = 4$ that start and end from a node. They are traced from node to node.

- *Branches* are just like arcs, but open ended feature lines. They start at a node and end with a non-complex vertex.

- *Loops* are closed feature lines that do not touch any node.

- The remaining feature vertices — if any — are *isolated lines*, i.e. feature lines with two non-complex vertices at the ends.

- All other vertices of the triangular mesh in $\overline{\mathbb{F}}$ can now be grouped into *patches* that are bounded by or include some of the previous elements.

Extracting the different elements in this given order simplifies the implementation. The new data structure allows more advanced operations on the feature, e.g. the feature lines can be edited manually, or small patches can be removed by first marking/filling them and then re-skeletonizing the feature region. Connecting branches automatically is straightforward for two branches in the same patch.

We summarize that the presented technique extracts feature lines from a wide feature region which is defined by a preprocessing like thresholding on estimated discrete curvature quantities. To do so, we apply easy-to-implement and efficient morphological operators and a skeletonization algorithm to suppress "classification noise" and to extract their topological medial axis. This way the topology of the original feature region is preserved, and the result can be used directly for segmentation or as initial guess e.g. for dynamic techniques like [Milroy et al. 1997, Lee and Lee 2001] for optimization. Finally, we note the presented operators are purely based on the mesh connectivity. This makes their

definition and application simple and efficient. However, these properties can be traded for more robustness and accuracy by taking into account geometric information, e.g. by defining nhd in terms of Euclidean or geodesic distances instead of topological distance.

3.5 Line-Art Rendering

Discrete curvature information is commonly used for shape visualization. A demanding case is the rendering of non-photorealistic, artistic line drawings from digital models. Figure 3.20 motivates this setting opposing discrete lines of curvature to an artistic illustration. In the following, we briefly provide some background before we present our contribution. For a complete introduction to non-photorealistic rendering we refer to [Gooch and Gooch 2001, Strothotte and Schlechtweg 2002].

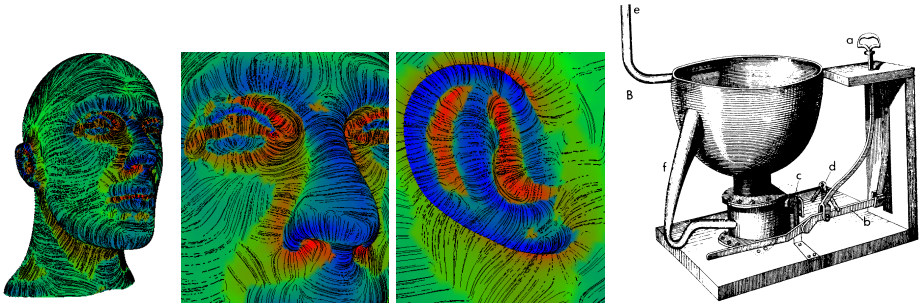


Figure 3.20: Lines of curvature visualize characteristics of a digital shape (with color coded maximum curvature). Similarly, intrinsic geometric properties for hand-crafted line-art drawings used for centuries. The example is a technical sketch from 1778. Such illustrations provide a high density of clearly perceptible information while also satisfying aesthetic standards.

3.5.1 Non-Photorealistic Rendering using Lines

A brief Introduction

In contrast to photorealistic images based on physical simulation, *non-photorealistic rendering* techniques offer various advantages for printed illustrations. The higher level of abstraction that they provide helps to put emphasis on the crucial information of an illustration. Abstraction mechanisms available to designers include levels of detail and focus attention to stress the important parts of a drawing or enriching an image with additional visual information and effects.

Typical applications for the use of non-photorealism are e.g. architectural sketches, illustrations in medical text books or technical manuals, and cartoons. Many of these applications prefer a very specific technique: line rendering. Here, a picture consists of monochrome lines or strokes only. An artist may chose between different line styles or vary line attributes like length, thickness or waviness. There is a broad spectrum of drawing styles, that have been developed in fine arts for centuries. Well known examples are silhouette drawings where just the contours of an object are outlined or more sophisticated techniques like engraved copper plates that have formerly been used for printing or pen-and-ink illustrations.

Apart from the advantage of abstraction or artistic features line drawings also provide some technical benefits: they can be stored resolution independent or come with inherent level-of-detail like features for different resolutions. Besides, such pictures can easily be reproduced on any monochrome media.

Line Drawings

Various computer based techniques for producing line drawings semi-automatically or even automatically have been developed. There are two basic approaches: The first one is image based. Here, the user is assisted in converting a digital grey scale image into a line drawing without having to draw individual strokes. Pnueli and Bruckstein [1994] define a potential field over an image and draw equipotential lines of varying density. Salisbury et al. [1994] use predefined stroke textures to map the tone of the reference image and to produce a pen-and-ink illustration. This method is improved in [Salisbury et al. 1997] in a way that the stroke textures are generated automatically from both a set of reference strokes and an interactively modifiable direction field.

The second approach takes into account the 3D geometry of a scene. Here, an early step was the use of haloed lines [Appel et al. 1979] that give an impression of depth and the use of different line styles for outline and shading [Dooley and Cohen 1990]. Winkenbach and Salesin [1994] apply special stroke textures to render high quality pen-and-ink illustrations from polygonal models. These stroke textures are created for different materials and resemble hand drawn strokes. During shading the strokes are mapped to image space together with the corresponding polygons and are used for tone mapping then. This system was extended to process free-form surfaces in [Winkenbach and Salesin 1996] where isoparametric curves are used for rendering, following [Elber 1995]. Such curves look especially well for surfaces of revolution. Elber [1998] also employs lines of curvature of parametric surfaces and contour lines of implicit surfaces. By precomputing these stokes and applying a simple shader even interactiveness is achieved [Elber 1999]. Praun et al. [2001] take a texture based approach to

hardware-accelerated real-time hatching. Interrante [1997] uses principal direction driven strokes to enhance the visualization of isosurface.

Zorin and Hertzmann [2000] generate a discrete direction field over a triangulated (subdivision) surface, which is based on discrete estimates of the principal directions. In order to reduce noise and to get a pleasant rendering this field is globally optimized in a second step by minimizing some energy functional. For the final picture a silhouette drawing is combined with hatches from the filtered principal directions.

The previously mentioned approaches that respect the geometry of an object are all able to produce "real", continuous stokes, e.g. in Postscript. Besides, there is another class of techniques that are pixel based and produce discrete images. [Saito and Takahashi 1990] introduce the concept of G-buffers that provide additional information per pixel such as depth value, normal vector or the parameter value of a parametric surface. Non-photorealistic images that resemble line drawings can then be generated by applying well known image processing operators. In contrast, [Leister 1994] uses a modified ray-tracing algorithm for emulating engraved copper plates. With processing pixel data it is also possible to take advantage of graphics hardware: [Deussen et al. 1999] generate copper plate like images from intersection lines of OpenGL clipping planes with the object.

Overview of our Approaches

We present two approaches to the generation of line drawings based on discrete curvature. The first approach is described in detail in [Rössl et al. 2000b]. It directly operates on the triangulated surfaces, it proceeds roughly as follows. Similar to the method for parametric surfaces in [Elber 1999], we display lines of curvature. These are streamlines in a piecewise linear vector field defined on the surface by the principal directions for every vertex. Here, the goal is an interactive visualization, i.e. rendering in real time. Therefore, we precompute all possible strokes of a predefined maximum length by first randomly scattering seed points and then integrating streamlines from the seeds. For rendering each frame the view is rendered to the OpenGL z-buffer to solve the visibility problem (render "hidden lines"). Then a simple local lighting model is evaluated associating each seed an intensity value. Based on these values each stroke is rendered to a fraction of its arc-length (including zero length for highlights) to globally approximate the scalar intensity function over the whole surface mesh. In addition, strokes starting near silhouette edges are rendered with higher priority to accentuate the object contours. Figure 3.21 shows examples. Comparing them to illustrations generated with the second approach in Figures 3.25 and 3.26, a high difference in quality is clearly visible.

This second approach (cf. [Rössl and Kobbelt 2000]) is a semi-automatic, hy-

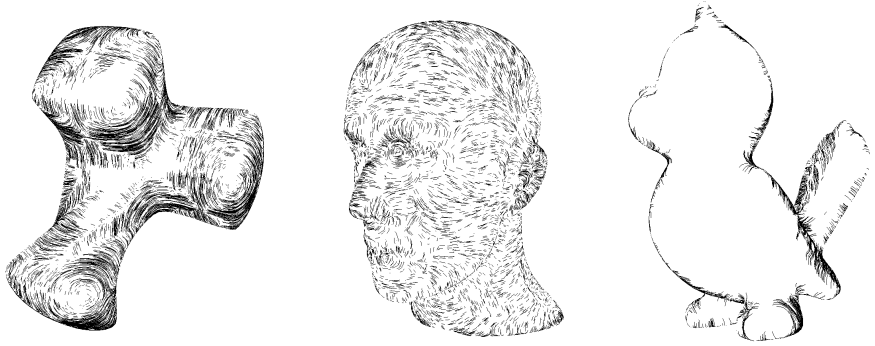


Figure 3.21: Screen shots from of line drawings of three models with 9K, 87K, and 97K triangles, respectively, rendered in real time on a SGI O2 with 225 MHz R10k processor with different shading parameters (see [Rössl et al. 2000b]).

brid approach considering a certain 2D view of the 3D shape resulting in a drawing of continuous strokes. Again, the method examines the geometry of model by approximating the principal directions in every vertex of the given triangle mesh. However, then this data is interpolated across triangles and sampled per pixel for the chosen view. Once all data is grabbed from image buffers the user may modify these buffers as well as an additional (stencil) buffer and interactively place continuous streamlines. Instead of generating all strokes from streamlines, the system lets the user chose a few reference lines and generates strokes by interpolation between these curves. This guarantees that only good, visually appealing strokes that are not disturbed by noise are rendered. For the final Postscript image the silhouette that has been grabbed from an image buffer and converted to a set of polygons is added. The following section describes this interactive system for computer aided generation of line-art drawings in more detail.

3.5.2 Line-Art Rendering of Digital 3D Models

The semi-automatic production of a line-art illustration from a digitized model in [Rössl and Kobbelt 2000] proceeds in several phases:

Image Sampling and Segmentation

We start with a certain view of the model with principal directions and normals given per vertex and defining a piecewise linear vector field each. The linear interpolation for the evaluation of the vector field can be done very elegantly by the underlying rasterization engine which maps the 3D mesh to the frame buffer: vector or scalar valued attributes are encoded as RGB colors and assigned to the mesh vertices. Rendering the object without any shading then generates a frame buffer

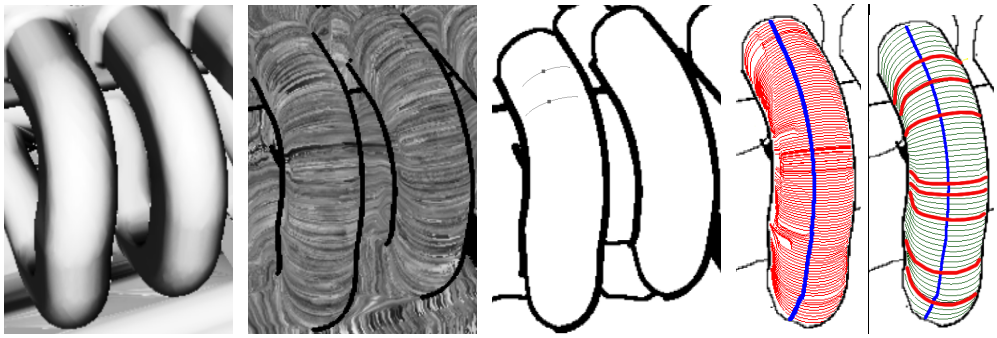
pixel matrix which stores the interpolated values. Arbitrary precision is obtained by appropriate scaling and multi-pass rendering. For vector valued direction attributes a normalization step might be necessary. We render each attribute field separately. In the end we obtain a set of images, one of them containing the (grey-scale) shaded view of the object. The other images show the color coded attribute fields, one for the normal vectors and one for each principal direction. We call the collection of all these images (with identical resolution) the *enhanced frame buffer* with many different values stored for each pixel. We prefer this term to *G-buffer* [Saito and Takahashi 1990] because we interpret pixels as *discrete* samples that are used to (re-)construct *continuous* strokes.

Once the given 3D-object is rendered, the enhanced frame buffer contains all the necessary information for the subsequent steps of the algorithm. Hence, phases two and three entirely work in 2D which reduces their computation costs.

We sampled all relevant geometric information for a certain view of the 3D model in a regular pixel matrix. This simplifies the following steps of the algorithm which now do not have to consider the (possibly complex) topology of the original mesh data. Moreover, most operations can be expressed and implemented as image processing operators which rely on the regular grid structure. As a preprocessing step, we apply simple Gaussian or Median filters to remove high frequency noise from the attribute fields which sometimes emerges from sampling artifacts during the rasterization. From the normal direction field we can easily extract the *silhouette lines* for the object. We do this by extracting the zero-contour from the scalar field defined by the dot product of the normal vectors with the viewing direction. This works for parallel projection as well as for perspective projection. We compute a polygonal approximation of that zero-contour with a two-dimensional variant of the marching cubes algorithm (*marching squares*) [Lorensen and Cline 1987]. The smoothed contour polygons resemble thick, slightly waved hand drawn strokes.

Our goal is to decompose the image into several regions which have a strong coherence in the principal direction fields since these areas are to be rendered by a single set of quasi-parallel hatches. Silhouette lines serve as a reliable detector for the boundary between such regions. In addition, image processing operators can be applied to the sampled data. We believe that automatic algorithms can only provide a rough initial segmentation that is to be refined manually.

The segmentation of the input image is the most sophisticated part of the generation of line-art images. Since the segmentation is usually driven by non-geometric attributes such as functional grouping in technical parts or implicit color and texture information, we allow the user to interactively control the partitioning. In our user interface, we display a LIC image (*Line Integral Convolution*, cf. [Carbal and Leedom 1993]) based on the maximum curvature direction field overlayed with the automatically extracted silhouettes (which serve as initial seg-



(a) (b) (c) (d)

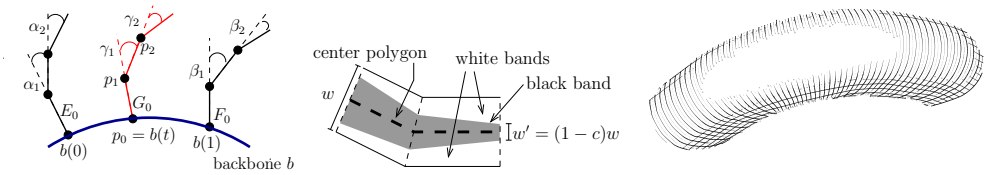
Figure 3.22: To convert the shaded image (a) into a line-art image, it is first partitioned in regions with coherent principal direction fields. A LIC image (b) with overlayed contours supports interactive segmentation. In addition streamlines can be probed, here (c) on the final segmentation. The subsequent hatch generation is based on a fishbone structure (d): The backbone is defined by a curve following the minimum curvature directions and the ribs are computed by tracing along the maximum curvature. In order to reduce the noise in the direction field (left) and to avoid discretization artifacts, the user can define a sparse set of "key-ribs" and a dense set of ribs is constructed by interpolation (right).

mentation). The LIC image gives a good and intuitive perception of regions with coherent flow. The user can now simply draw the segment boundaries on the screen. In practice we observed that the interactive segmentation can be done in several minutes even for moderately complex objects. Figure 3.22 illustrates the segmentation.

We sometimes find regions in the image where no natural direction field for the stroke orientation can be defined. This is true either in the presence of umbilic points where $\kappa_1 = \kappa_2$ or for regions with too much fine detail relative to the sampling density in the frame buffer. In such cases, our interface allows the user to override the directional information in the frame buffer and to locally define a synthetic direction field. With a technique similar to [Ostromoukhov 1999] we use the partial derivatives of bi-linear Coons-patches to generate these synthetic direction fields. In the final line-art image, this fine detail will be taken into account by the tone mapping.

Generating Curvature Guided Pseudo-Parallel Strokes

For every segment with coherent principal direction fields we define a grid of hatches. We build this grid in a *fishbone* fashion by first picking a *backbone* and then arranging the *ribs* in orthogonal direction. To define the backbone, the user



(a) (b) (c)

Figure 3.23: (a) The blending of strokes is based on a decomposition into *orientation* (E_0,F_0) and *characteristic shape* ($\{\alpha_i\}$,$\{\beta_i\}$) and controlled by a parameter $t \in [0,1]$ with the backbone curve b locally parameterized as shown. (b) Strokes of constant width are used, with only a certain portion of a stroke drawn in black while the rest is drawn as two white bands on both sides. (c) Cross hatches are used to enhance the contrast, the figure shows a rotated region of Figure 3.22. Only the black portion of the strokes is drawn to avoid overpainting.

can pick an arbitrary point in the interior of the current segment. Starting at that point we compute the backbone curve by integrating along the *minimum* curvature direction field. On the backbone curve we distribute samples with constant (arc-length) distance. The orthogonal ribs are then computed by starting at those samples and integrating along the *maximum* curvature direction. In some cases, however, this simple rib generation technique fails to produce useful strokelines. Due to the discretization of the direction field, neighboring lines can merge which destroys the global fishbone structure ((Figure 3.22 (d), left). In order to avoid this effect, the user can manually place a few sample points on the backbone from where the *key-ribs* are traced along the maximum curvature directions. In between these key-ribs we uniformly distribute additional *blended ribs*. The blended ribs result from interpolating between the key-ribs. By properly choosing the starting points for the key-ribs, we generate a high quality set of pseudo-parallel strokelines (Figure 3.22 (d), right).

Each strokeline is represented by a polygon with constant edge length h (arc-length parameterization). To uniquely describe the shape of a rib strokeline we need the first polygon edge E_0 and a sequence of angles α_i between successive edges E_i and E_{i+1} (cf. Fig. 3.23 (a)). The complete strokeline can then be reconstructed by starting with the first edge E_0 and adding more edges E_{i+1} with the same length in the direction determined by the angles α_i. In the strokeline representation $[E_0, \{\alpha_i\}]$, the edge E_0 determines the *orientation* and the sequence $\{\alpha_i\}$ determines the characteristic shape. Assume we are given two key-ribs $[E_0, \{\alpha_i\}]$ and $[F_0, \{\beta_i\}]$ which start on the same backbone. Then for every value $t \in [0,1]$

we find a new starting point on the backbone arc between the two key-ribs and the corresponding blended rib is given by $[G_0, \{\gamma_i\} = \{(1-t)\,\alpha_i + t\,\beta_i\}]$ where the orientation G_0 is given by an weighted average of E_0 and F_0 and the characteristic shape is a weighted blend of the two key-ribs. Using this blending technique we can generate very good fishbone type strokelines by prescribing only rather few key-ribs.

After the rib generation, the fishbone structure is given by a set of polygons with unit edge length h. For rendering purposes, the k-th vertex $\mathbf{p}_k^{(l)}$ of the l-th rib $R_l = [G_0, \{\gamma_i\}]$ can be computed as

$$\mathbf{p}_k^{(l)} \;=\; \mathbf{p}_0 + G_0 + h \sum_{i=1}^{k-1} \begin{pmatrix} \cos(\sum_{j=1}^{i} \gamma_j) \\ \sin(\sum_{j=1}^{i} \gamma_j) \end{pmatrix},$$

and similar for negative index k.

The organization of the rib vertices $\mathbf{p}_k^{(l)}$ to a sequence of sequences $[[\mathbf{p}_k^{(l)}]_k]_l$ corresponds to a rectilinear matrix type structure where the vertices of one rib form a row $[\mathbf{p}_k^{(l)}]_k$ and the k-th vertex for all ribs $[\mathbf{p}_k^{(l)}]_l$ forms a column. Hence, it is natural to exploit this structure for the tone mapping. To simplify further processing, we sample the shading values at the locations $\mathbf{p}_k^{(l)}$ and pass a resampled attribute matrix to the tone mapping procedure. In that procedure, a stroke width value $w_k^{(l)}$ is computed for every location $\mathbf{p}_k^{(l)}$. For this we do not need any directional information.

Tone mapping

The last step of the algorithm is the translation of grey value shading information into stroke widths. For this translation several aspects have to be taken into account.

First of all the local brightness of the rendered image obviously depends on the ratio between stroke width and distance between neighboring strokes. If strokes lie more densely, the width of the strokes has to decrease to maintain constant shading. Another possibility is to suppress a part of the strokes and leave the others with their original width. The problem with the suppressing of certain strokelines is that disturbing low-frequency patterns can appear in the final image if the lines are chosen according to a periodic rule. Adapting the distance of neighboring strokes to the local grey value is not an option since that value usually changes along the stroke but the distance cannot be controlled freely.

Many sophisticated techniques have been proposed for the tone mapping in line-art images (see e.g. [Salisbury et al. 1994, Winkenbach and Salesin 1996, Salisbury et al. 1997]). We use a simple and efficient technique which does not

use any global information but still has flexibility to adjust stroke widths and suppress strokes.

The idea is to define strokes to have a constant width w but only a certain portion $w' = (1 - c)\, w$ is drawn in black where $c \in [c_{\min}, c_{\max}] \subset [0, 1]$ is the local grey value (0=black). Restricting the grey values to the interval $[c_{\min}, c_{\max}]$ guarantees a minimum width of the strokes and a minimum width of the white space between strokes. Since strokes have a constant width they partially overlap if neighboring strokes come too close together. If the strokes are painted one after the other then one stroke can delete parts of its neighbors. In the extreme case two non-neighboring strokes can approach so close that all the strokes between them are completely removed.

The technique described so far controls the local tone mapping by adjusting the stroke widths and automatically removes some strokes if their density increases. The remaining question is how to avoid low-frequency patterns in the distribution strokelines. We solve this problem by drawing the strokes in a special order which guarantees that the right strokes are overpainted and the surviving ones are equally distributed.

A sequence of uniformly distributed strokelines (without low frequency patterns) can be generated by drawing every 2^k-th line. If the strokelines become denser we want the strokes from coarser distributions (higher values k) to survive. Hence we have to start drawing the finest level containing the strokes $[2^1\, i]_i$ and then go to coarser and coarser levels $[2^k\, i - 1 + 2^{k-1}]_i$, $k = 2, 3, \ldots$ Here we chose the index offset $-1 + 2^{k-1}$ such that no strokeline appears twice for different values k. The ordering in which the strokeline have to be drawn can easily be computed from their index: in the j-th step the rev(j)-th strokeline is drawn, where rev(j) is the number which has the reverse binary representation of j, i.e., the sequence of binary digits is reversed. Figure 3.24 illustrates the tone mapping.

So far we explained the generation of hatches only along the maximum curvature direction. In some cases the image quality can be improved by also adding hatches in the cross direction since this increases the brightness range (darkest to lightest grey value). Cross hatches are often used to enhance the contrast at shadow boundaries.

Since the cross hatches follow the minimum curvature direction, they are typically less curved than the original hatches. As a consequence the effects of varying strokeline density are less severe and simple stroke width modulation (without strokeline suppression) is usually sufficient for the tone mapping. In our implementation we applied cross hatches in regions of the image where the shading value falls below a prescribed threshold $c_{\min}$. Because one set of hatches has already been painted before the cross hatches are added, we base the stroke width computation on the offset grey values $c' = c - c_{\min}$. In order to avoid overpainting

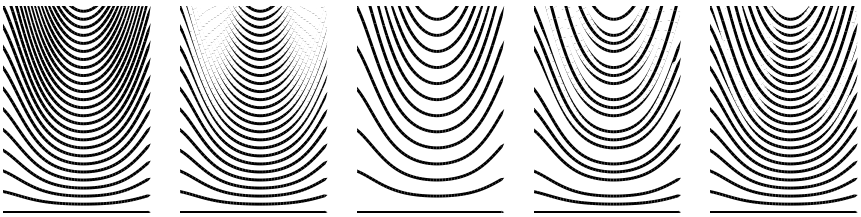


Figure 3.24: The special order in which the strokelines are drawn, guarantees that the surviving lines (the ones that are not overpainted) do not show a low-frequency pattern. Simply painting black strokes with constant width w does not lead to constant color (far left). Drawing the strokes with white bands and constant black fraction (as shown in Figure 3.23 and in sequential top to bottom order leads to low-frequency patterns of suppressed lines (left). The next three images depict our special ordering. The center image shows the lines with index $2\,i$ ($k = 1$) only. In the center right image these lines are partially overpainted by the lines with index $4\,i + 1$ ($k = 2$). In the next step another layer of lines with index $8\,i + 3$ ($k = 3$) is painted (far right). The resulting image has an almost constant shading color which is achieved by suppressing some of the lines in regions where strokelines become denser.

the already existing hatches, we only draw the black portion of the cross hatches (no white band, no strokeline suppression). As stated above this simple cross hatching technique works well because minimum curvature lines usually have a rather straight characteristic shape (see Figure 3.23 (c)).

Examples

Our examples show a technical model and a toy elk. The original triangle meshes consist of about 43K and 30K triangles. The technical part (Figure 3.25) nicely shows the suppression of strokes in the tone mapping process. Cross hatched strokes are used in dark areas. The elk model (Figure 3.26) was generated from range scans of a wooden toy. The strokes at the sphere-shaped wheels have been generated manually as there are no meaningful principle directions defined there. The two images are identical except that the cross hatch threshold $c_{\min}$ was modified.

3.6 Summary

We introduced a new technique for estimating the curvature tensor $\mathbf{T}$ in a triangular mesh which computes $\mathbf{T}$ as a piecewise smooth function on every triangle given (exact or estimated) surface normals. The approach is intuitive and elegant and does not incorporate any parameterization or fitting approaches, moreover the

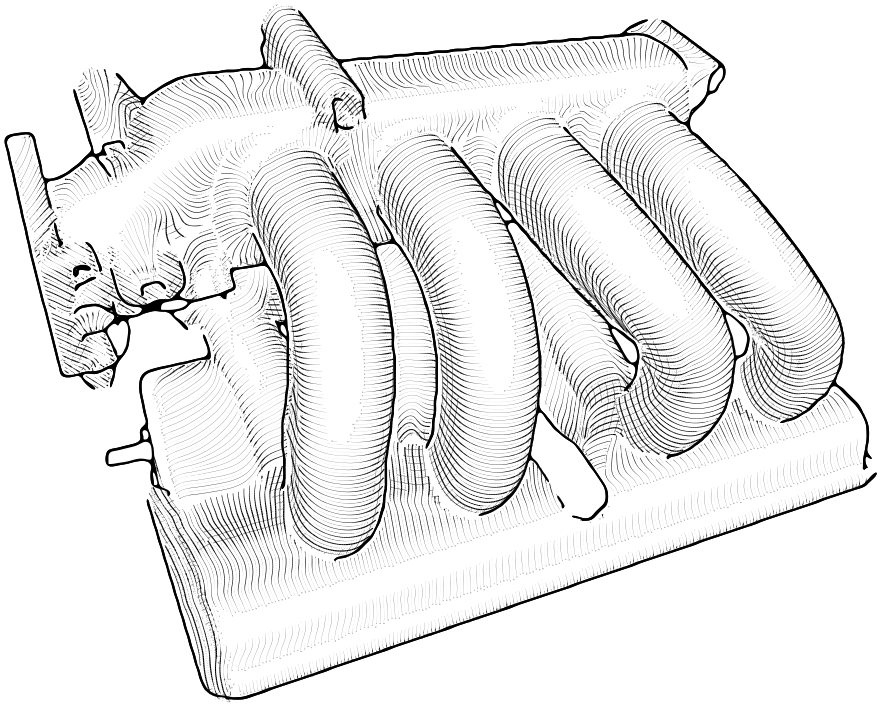


Figure 3.25: A line-art rendering of a technical generated with our system. The view was captured from a model of 43k triangles.

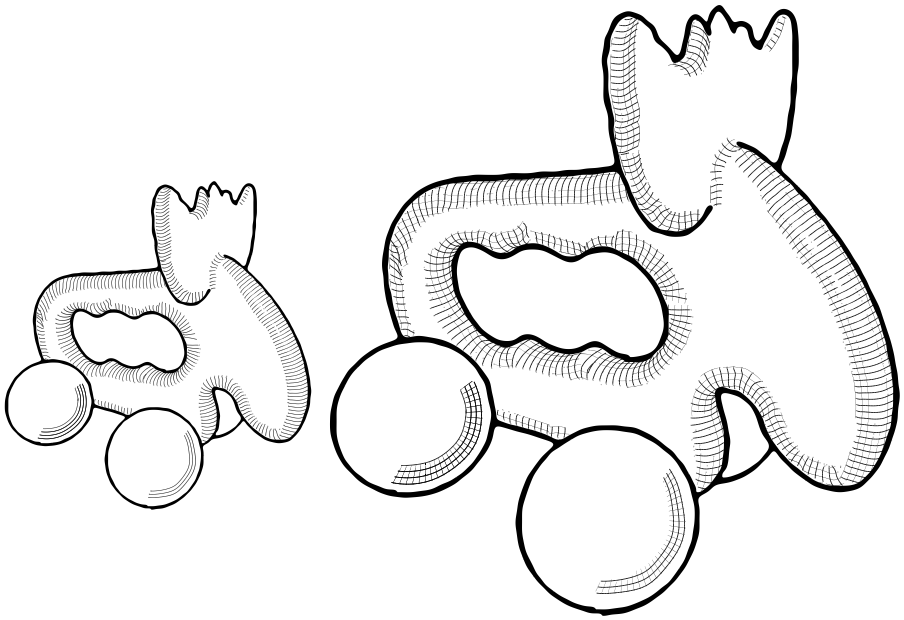


Figure 3.26: Mama elk and baby-elk (30k triangles) with and without cross hatching.

evaluation shows that it competes very well (in general even slightly better than the cubic fitting in [Goldfeather and Interrante 2004] as best competitor) with existing approaches both in terms of approximation errors and efficiency. With exact normals given at the vertices the error drops significantly. Consequently, the approach benefits from more accurate normal estimation.

Possible improvements of the approach would consider (not normalized) estimates of vertex normals with direction *and* length as an additional parameter improving the quality of the curvature estimation. Another possible extension is to apply quadratic interpolation for the normals (see e.g. [Vlachos et al. 2001]) instead of the linear interpolation.

There is a variety of applications for discrete curvature estimation techniques. The proposed morphological operators for feature detection are simple by their nature and efficiently recover structural information extracted from the estimated quantities. While segmentation techniques in general explicitly extract such information from shapes for further, higher-level processing, other applications are interested in the visualization of shape intrinsic properties. Such a scenario is the use of integrated lines of curvature for non-photorealistic rendering. We showed how the discrete estimates of the curvature tensor are used as basic tool for creating high-quality line-art drawings from digital models. In addition, we note that a similar technique provides a basic tool for feature sensitive resampling of surfaces in [Botsch et al. 2000].

In this context we deal with piecewise linear vector fields, where the direction into the maximum normal curvature direction is given for every vertex. Subsequently to the next Chapter 4 on the efficient encoding of mesh connectivity, we discuss the compression of piecewise linear vector fields in Chapter 5.

Chapter 4
Connectivity Encoding

4.1 Background

With the emergence of very large triangle meshes, for instance from the acquisition of complex shapes, the efficient encoding of the arising data attained more and more focus. One basic entity of interest for data compression is the triangle mesh connectivity, which captures the graph structure and hence the topological properties of the mesh, including the genus of the surface and the existence of boundaries.

Certainly, in-core data structures for meshes which enable their manipulation and navigation between adjacent items must trade redundancy for efficiency. However, things are different for storing or transmitting data. Here, more compact representations are preferred, which at the same time should be intuitive and simple. The most common choice for encoding the connectivity of reasonably sized meshes in practice is (still) a shared vertex representation, which consists of a table of vertices and an associated table of triangles, which in turn are triples of indices into the vertex table. (Hence, every vertex is shared among the triangles attached to it.) Nearly every common file format supports this representation — often as the only choice.

This is in contrast to the fact, that techniques for connectivity encoding have been studied intensely for many years, dating back to the theoretical foundations by Tutte [1962] on the enumeration of the planar triangulations. Tutte found generating functions for the number of distinct triangulations over a plane, and studied their asymptotic behavior, and his results established a theoretical upper bound of 3.24 bits per vertex for the encoding of sufficiently large triangle meshes. Brown [1964] removed Tutte's initial condition that two boundary vertices cannot be connected with a non-boundary edge while yielding the same asymptotic behavior of the generating functions. The study of the asymptotic behavior of the number of

polygonal meshes on surfaces of arbitrary topology is still a very active area of mathematics [Arqus and Braud 2000].

In the following, we provide a brief overview of connectivity encoding techniques and in addition refer to the recent surveys [Gotsman et al. 2002, Alliez and Gotsman 2005].

Many diverse techniques have emerged, often with certain advantages over alternative approaches when a particular class of meshes is considered. Among the most prominent techniques are the encoding of the connectivity as a permutation of the vertices [Denny and Sohler 1997], the topological surgery method [Taubin and Rossignac 1998], where a mesh is transmitted as a vertex tree together with the dual face tree, and the Cut-Border Machine [Gumhold and Straßer 1998, Gumhold 1999], which was the first recursive method based on a traversal of the triangles of the mesh.

We remark that there are several methods, which are not strictly comparable, but can also be applied in this setting. Among these are the Face-Fixer [Isenburg and Snoeyink 2000] which encodes polygonal meshes (with arbitrary faces), and many powerful multiresolution techniques, like e.g. [Hoppe 1996, Taubin et al. 1998, Cohen-Or et al. 1999, Isenburg and Snoeyink 1999, Pajarola and Rossignac 2000].

We can classify many encoding techniques in two major branches: *valence-driven* techniques and *Edgebreaker-like* methods, where vertices and triangles are traversed, respectively.

Some of the most efficient techniques are the valence driven methods, which transmit for each vertex its valence together with some special symbols. This method was initiated by Touma and Gotsman [1998] with a deterministic traversal of the mesh vertices. Alliez and Desbrun [2001a] used an adaptive traversal of the vertices, reporting the best compression ratios in practice. The latter method has been expanded to more complex problems such as the progressive transmission of a shape including the geometry data [Alliez and Desbrun 2001b], or the encoding of polygonal meshes, giving again the best reported results when compared with any other similar method. The Angle-Analyzer [Lee et al. 2002] applies geometric attributes to steer the traversal, hence interleaving connectivity and geometry coding.

Finally, another major branch of recently developed techniques is based on the Edgebreaker algorithm. The original Edgebreaker was proposed by Rossignac [1999]. It traverses a tree of the faces of a mesh and for each face returns one symbol from an alphabet of five, determining the adjacencies of that face with the not yet conquered part of the mesh. The method was originally published with a guaranteed worst-case bound of four bits per vertex, which later was improved to 3.67 bits per vertex [King and Rossignac 1999], and to 3.552 bits per vertex [Gumhold 2000].

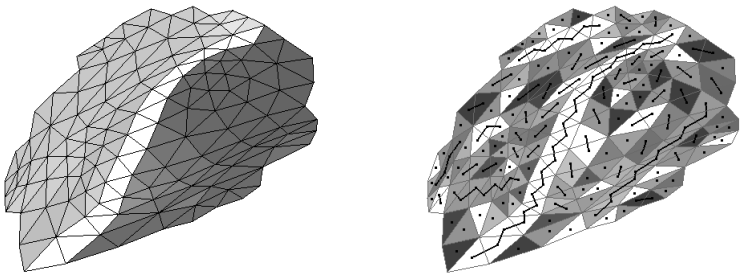


Figure 4.1: A triangle strip divides each mesh into two submeshes (left). The left and the right submeshes are processed the same way recursively defining a binary tree. The result is shown on the right side, the black lines denote the strip connectivity. We can encode a planar triangle mesh as the resulting binary tree with only the strip lengths stored in its nodes.

Numerous other improvements in the efficiency of the technique followed. Rossignac and Szymczak [1999] make the encoding and decoding process linear in time, Isenburg and Snoeyink [2001] further simplify the decoding algorithm especially for meshes of arbitrary topology, while [Szymczak et al. 2001] provides an adaptation of the Edgebreaker for highly regular meshes.

In the following, we present and analyze a divide and conquer approach to connectivity encoding, which is in this second branch of Edgebreaker-like encoding algorithms. In contrast to other approaches, the output is structured as a binary tree, with interesting properties and potential applications.

Outside the mesh compression literature, a method for graph compression was reported in [Deo and Litow 1998], employing an approach similar to the one presented due course. There, a graph is decimated with the use of graph separators, i.e. subgraphs whose removal separates the graph into components of roughly similar size. In a recursive process these components are encoded in a tree data structure. The different setting — especially the nature of the graph as a combinatorial rather than a geometric object — makes much more difficult conclusive answers, something that, as we see in the following, is not the case with triangle meshes.

4.2 A Divide and Conquer Approach

In this section, we present the connectivity encoding approach (cf. [Ivrissimtzis et al. 2002]). The idea is to process a triangle mesh by dividing it into submeshes, which are then processed recursively. Figure 4.1 illustrates the principle.

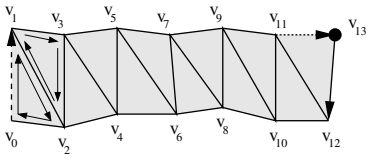


Figure 4.2: A *zig-zag strip* of length 12. The *gate* (v_0, v_1) is marked with a dashed dark arrow, the *left leading directed edge* (v_{11}, v_{13}) is drawn light, the *right* one (v_{13}, v_{12}) is dark, and the *leading vertex* v_{13} is marked with a black dot.

4.2.1 Preliminaries

The basic version algorithm encodes planar triangulations with a single boundary loop. Modifications for triangulations with arbitrary topology, i.e. possibly with handles and holes, are discussed in Section 4.2.5 (In either case, an oriented manifold surface mesh is assumed.) The following terms will be used throughout the section. A *rooted mesh* is a triangle mesh with one of its boundary directed edges marked. The marked boundary edge is the *gate* of the rooted mesh.

A *zig-zag strip* of *length* n is a mesh consisting of $n + 2$ vertices $(v_0, v_1, \ldots, v_{n+1})$, $v_i \in \mathcal{K}$, $i = 0, \ldots, n + 1$, and the triangles $(v_i, v_{i+1}, v_{i+2}) \in \mathcal{K}$ with $0 \leq i < n$ (cf. Figure 4.2). In rooted zig-zag strips, the gate will always be the directed edge (v_0, v_1). Then the vertex $n + 1$ is called the *leading vertex*, and the triangle (v_{n-1}, v_n, v_{n+1}) is the *leading triangle*. Assuming that all the triangles have the same orientation as indicated by the gate, the oriented leading triangle is (v_{n-1}, v_n, v_{n+1}) for odd n and (v_n, v_{n-1}, v_{n+1}) for even n. For odd n, (v_n, v_{n+1}) and (v_{n+1}, v_{n-1}) are the left and the right directed *leading edges*, respectively. For even n, the left and right leading directed edges are (v_{n-1}, v_{n+1}) and (v_{n+1}, v_n), respectively. In both cases the left leading directed edge is the one pointing towards the leading vertex. Figure 4.2 illustrates a rooted zig-zag strip.

4.2.2 Basic Algorithm

The mesh is processed by conquering vertices. In the beginning of the encoding process, we mark the boundary vertices as conquered, all other vertices are not yet conquered. If the mesh has no boundary we remove a single triangle, creating a trivial boundary with three edges. We randomly choose a directed edge on the boundary of the mesh as the initial gate, and we build a zig-zag strip conquering its vertices. The construction of the strip stops when arriving at an already conquered vertex. This happens when the leading vertex of the zig-zag strip reaches either the boundary of the mesh or another vertex of the strip. In both cases the original rooted mesh splits into two rooted submeshes: The left submesh with the left leading directed edge of the strip as gate, and the right submesh with the right

leading edge of the strip as gate. Note that one of these submeshes or even both can be empty. We continue recursively, encoding separately the two submeshes, and the encoding process terminates when all the submeshes are empty.

We organize the data acquired in this process in the form of a binary tree with the strip lengths stored in its nodes. The length of the initial zig-zag strip is stored in the root of the tree, the encoding of the left submesh is the left branch of the tree, and the encoding of the right submesh is the right branch of the tree.

Encoding

Encoding a triangle mesh can be written in pseudo code as

Algorithm 4.1 *(encode preorder)* *encode-preorder* $(gate)$:=

1. $(length, left, right)$ = *conquer-strip* $(gate)$
2. *output* $length$
3. *if* $length$>0
   for $d \in \{left, right\}$: *encode-preorder* (d)

Here, the procedure *conquer-strip* grows a zig-zag strip by iteratively conquering a new leading vertex. The strip starts from the directed edge $gate$, and it stops when its leading vertex is already conquered. The procedure returns a triple $(length, left, right)$, which describes the $length$ of the strip and the two leading edges $left$ and $right$. Then the procedure *encoding-preorder* describes the mesh as a binary tree, where each node stores the length of the associated triangle strip. Note that the pseudo code above does not explicitly describe the construction of a tree data structure. Instead, the lengths are output into a stream such that the resulting code corresponds to a preorder traversal which implicitly represents the tree. Figure 4.3 illustrates an example run of the algorithm for a small mesh. Here, the tree is constructed and shown explicitly, the resulting code is the output sequence 2,1,1,0,0,0,1,0,0 (where length 0 indicates an empty subtree, not shown in the figure).

Decoding

The main idea of recursively partitioning and encoding the mesh is reflected in algorithm 4.1. Conversely, in one recursive step of the decoding process we create a zig-zag strip of specified length and two rooted meshes, and we glue them together. We identify the gate of the left rooted mesh with the left leading edge of the strip, and we glue the left boundary of the strip with the boundary of the mesh stopping at the gate of the strip. Then the same is repeated for the right rooted

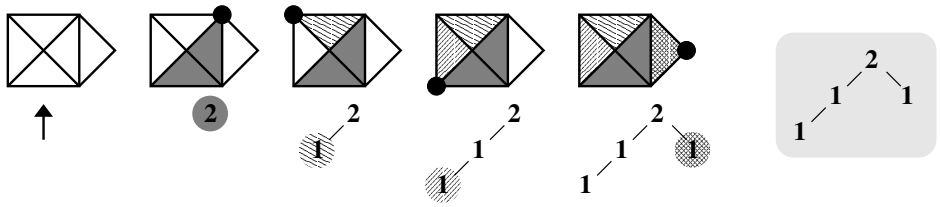


Figure 4.3: Example run of the algorithm. A simple mesh (left) is recursively encoded as a binary tree (right) starting from the gate denoted by the arrow. The four resulting strips are filled with different patterns. The first strip of length two partitions the mesh into two submeshes, all other strips have one or two empty submeshes. The dot denotes the leading vertex of the current strip, i.e. the point where the strip touches a conquered/boundary vertex. For every new strip the corresponding tree state is shown with the new node highlighted.

mesh and the right boundary of the strip. The gate of the new mesh is the gate of the strip.

Given the preorder traversal of the tree as output of Algorithm 4.1, the recursive decoding reads as follows:

Algorithm 4.2 *(decode preorder traversal)* *decode-preorder :=*

1. $length$ = *input ()*
2. *if* $length = 0$ *return empty-mesh*
3. $(gate, left, right)$=*create-strip* $(length)$
4. *for* $d \in \{left, right\}$: $gate_d$=*decode-preorder ()*
5. *for* $d \in \{left, right\}$: *glue-meshes* $(d, gate_d)$

Here, *input* reads an integer length value from the output of Algorithm 4.1. *create-strip* generates a new triangle strip analog to *conquer-strip* but as an individual mesh. The procedure returns the gate and the leading directed edges of this mesh as the triple $(gate, left, right)$. The recursive call to *decode-preorder* creates the left and right submeshes and returns their gates. Finally, the strip boundaries are glued to the two submeshes. The number of boundary edges which are traversed and connected by *glue-meshes* can easily be calculated from the strip's length and its parity, e.g. 6 and 7 for the strip of length 12. Gluing a mesh and *empty-mesh* results in a boundary. For instance in the setting shown in Figure 4.1 (left), we glue starting from the leading edges (top right in the picture) first the submesh in the right hand side (with the *left* leading edge as gate) and then the one on the left of the dividing strip. In both cases gluing stops at the gate of the strip (bottom left). Figure 4.4 shows an example run of the decoding algorithm. Note that some

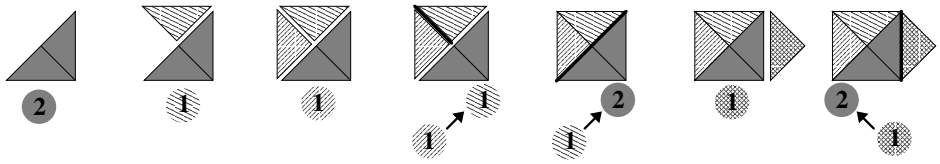
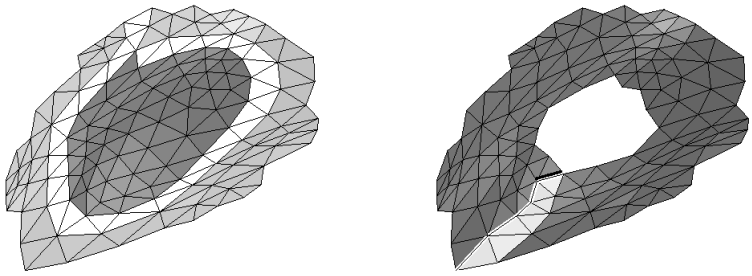


Figure 4.4: For decoding, the tree from Figure 4.3 is traversed in preorder. The strips are created during the top-down traversal of the tree nodes, and submeshes are glued (thick line) when an edge of the tree is followed in bottom-up direction. The corresponding tree nodes and edges are shown below the state of the mesh.



(a) (b)

Figure 4.5: (a) A strip may intersect itself resulting in a loop. (b) Handles or holes prevent the strip from splitting the mesh into two parts. A special *split* code is output instead of the empty left submesh. It references the corresponding edge in the right submesh drawn black. With this information the remaining boundary (white) can be glued.

modification is needed to handle the cases of self intersection, vertices of valence three, and non-planar topology cases.

A strip may intersect not only with the boundary of a submesh but also with itself (or its own boundary) resulting in a loop. Figure 4.5 (a) illustrates this situation. This is the same setting as before but with an inner and an outer submesh. However, the leading vertex now induces some kind of singularity for gluing: When we start gluing from the inner leading edge (light-dark border), we cannot decide, which direction to take once we arrive back at the leading vertex as the outer boundary has not been glued yet. This technical problem can easily be resolved by gluing both — left and right — leading edges first, before gluing along the whole strips (*glue-meshes*). As pointed out earlier, the left and right submeshes may be empty. The following cases can be distinguished (see also Figure 4.6):

- A strip of *length one*, i.e. a single triangle, can have one or both submeshes empty, if one or both leading edges are boundary edges of the mesh, respec-

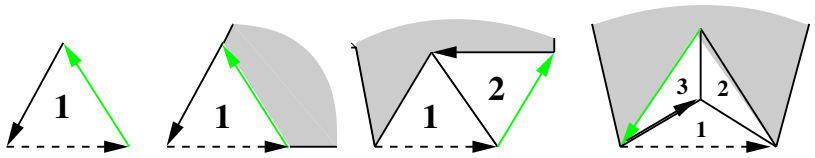


Figure 4.6: From left to right: All strips are entered through the bottom gate (dashed). The first strip of length one has both submeshes empty, while the next strip of length one has only the right submesh empty (dark gate). The strip of length two has the left submesh empty (light gate). The strip of length three has the right submesh empty (dark gate). Note that in the case of a strip with length greater than two the empty submesh is in the interior of the strip and not on the boundary (valence-3 case).

tively.

- A strip of *length two* can only have the left submesh empty, because the non-gate vertex of the first triangle of the strip is not boundary. Otherwise the process of growing the strip would have stopped before.

- In a strip of *length greater than two* both, the leading directed edges are not boundary, and an empty submesh occurs only if the strip intersects itself and the internal submesh is empty. This happens precisely when the strip passes through a vertex of valence three. The parity of the length of the strip determines whether the right or the left mesh is empty.

The latter case of passing a valence-3 vertex has to be checked separately, as the empty submesh does not correspond to boundary. If the strip length is greater than two and either the left or the right submesh is empty, an extra glue operation is needed: Let e be the corresponding edge in the strip (cf. dark edge in triangle 1 in Figure 4.6, right). Then the edges e and either $left$ or $right$, respectively, are glued together. We note that e can be referenced easily by a fixed navigation path back through the strip.

4.2.3 Analysis of the Encoding Algorithm

Next, we give two basic properties about the behavior of the encoding algorithm. The first property addresses the size of the obtained binary tree, and the second one describes the relationship between the structure of the tree and the strip length information stored in its nodes.

Let v and t denote the number of vertices and triangles of the mesh, respectively. And let n be the number of the strips obtained from the divide and conquer algorithm, which is equivalent to the number of nodes in the resulting binary tree.

We remark that t is also obtained as the sum of all strip lengths. Then following properties hold.

Theorem 4.1 *(number of vertices and nodes)*
Let the number of nodes and vertices, n and v, be defined as above. Then we have $n = v - 2$.

To see this, we notice that a strip of length l conquers $l - 1$ vertices, and thus n strips of total length t conquer $t - n$ vertices. The number of vertices to be conquered is v_I, the number of interior vertices of the mesh. Hence, we have $t - n = v_I$, giving $n = t - v_I$. The latter can also be written as $n = v - 2$, which can be seen immediately from applying Euler's formula.

Alternatively, there is an inductive proof, which we sketch as follows. Indeed, the claim is true for a single triangle because then $v = 3$ and $s = 1$. Suppose now that we have two meshes with v_1 and v_2 vertices, respectively, which are encoded by the divide and conquer algorithm in $s_1 = v_1 - 2$ and $s_2 = v_2 - 2$ strips, respectively. Gluing them along a strip will give a new mesh with $v' = v_1 + v_2 - 1$ vertices, because all the vertices of the strip are identified with a vertex from exactly one of the submeshes, except of the leading vertex which is identified with one vertex from both the submeshes. The new mesh is one level above the two submeshes in the recursive process and can be encoded with $s' = s_1 + s_2 + 1$ strips. We have $s' = v' - 2$ completing the inductive step for this case. The four cases with empty submeshes, shown in Figure 4.6, must be treated separately in an analog way, they pose no special difficulty.

The binary tree and the strip length information can be combined in a single data structure, namely a binary tree with positive integer weights assigned to its nodes. The next question is when such a data structure is valid, i.e. when it represents a planar triangle mesh. This leads to

Theorem 4.2 *(validity of a planar mesh encoding)*
Let T be a binary tree with n nodes and let the sum of the strip lengths be t. Then T represents a planar mesh if and only if

(i) $t + 1 \leq 2n$ *holds for T and all its subtrees.*

(ii) The strip lengths of the nodes with only left child are odd, and the strip lengths of the nodes with only right child are either one or even.

In order to prove the above, let b be the number of the boundary vertices of a planar mesh. Applying Euler's formula gives

$$b = 2v - t - 2$$

which from Then 4.1 becomes

$$b \;=\; 2n - t + 2$$

Then, the inequality $3 \leq b$, which holds for the boundary of the mesh, gives $t + 1 \leq 2n$. We note that the inequality $3 \leq b$ is a standard assumption in the literature of meshes, as $b = 2$ would give two boundary vertices connected by two different edges, while $b = 1$ would give a vertex connected with itself.

Condition (ii) describes the exceptional cases where one of the submeshes is empty and two edges of the dividing strip are glued together. Conversely, if the condition (i) holds, there is enough free boundary to perform all the gluing operations, and we get a valid triangle mesh, while condition (ii) guarantees that we can perform the gluing in the exceptional cases as well.

4.2.4 Connection to Edgebreaker

Before we discuss the obtained tree data structure, it is worth having a look at the algorithm in a more general setting. A first observation is that the algorithm, like the Edgebreaker [Rossignac 1999], implicitly induces a traversal of the triangles of the mesh. This traversal can be seen at two levels. The first level is the *traversal of the triangles* of a zig-zag strip, from the root of the strip to the leading edges, and the second level is the *traversal of the tree* that stores these zig-zag strips.

Also, we realize that the algorithm works not only with zig-zag strips but with general strips as well. In fact, any bit stream the encoder and the decoder would agree on, defines a way of building general strips, and thus a variation of the method. Here we use the zig-zag strips as the most natural choice for generating a strip.

The fan strip is another important class of strips. In this case, assuming a preorder traversal of the tree, we get the same traversal of the triangles as the Edgebreaker, and our approach differs only in the interpretation of the obtained data. In fact, we can translate the encoding of the binary tree and the strip lengths into the familiar C, L, E, R, S string of the Edgebreaker and vice-versa. Each strip length n can be written as a string of $n - 1$ C's, and for each node of the tree we use one of the L, R, S, E symbols, depending on whether it has only left child (L), only right child (R), two children (S) or no child (E). We remark that — because our data structure is non-linear, namely, a binary tree rather than a symbol stream — it is not necessary to assume any particular traversal of the tree to interpret the data.

4.2.5 Arbitrary Topology

So far, we considered planar triangulations with a boundary. For arbitrary topology meshes a strip can have the same non-empty submesh on the left and on the right. The simplest example is a planar mesh with a hole in it, which is topologically equivalent to a cylinder, Figure 4.5 (b) illustrates this setting. In this case we need to encode only one branch of the tree and give some additional information on how the boundary of this submesh is glued to the other side of the strip.

This situation can be detected in a simple way during encoding: For every triangle we store the corresponding level of the binary tree when it is conquered. If a leading edge of a strip touches an edge of a triangle that has been conquered at the same or a higher level, we output a special *split*-symbol referencing the corresponding edge for gluing (cf. Figure 4.5 (b)). Similar techniques to deal with arbitrary topology are applied by other algorithms.

The modifications of the algorithms are straightforward: For handling arbitrary topology, the encoding procedure is made aware of the current recursion level, and conquered triangles are tagged with this level as well as a serial number and the index $i \in \{0, 1, 2\}$ of the edge that it has been entered through. This information uniquely identifies the edge. The level test is straightforward, the number and the edge index are coded into the split symbol. If during decoding a split-symbol is read instead of the code of a submesh, then the edge is extracted, and the boundary is glued as if there were two submeshes.

The overhead for the encoding can be characterized as follows. Let g be the genus of the mesh is g, and h the number of holes in it, and v the number of vertices. It is a simple topological fact that the number of the strips which do not separate the mesh is at most $2g + h$. For each such strip we need $O(2 \log v)$ bits to identify the directed edge for gluing, and in the worst case $O(\frac{1}{2} \log v)$ bits for the extra split-symbol.

4.2.6 Reverse Decoding

The proposed encoding method is linear in the number of input triangles, as every triangle is visited once during encoding, and for decoding every node is visited once during the traversal of the tree. In contrast, the original Edgebreaker algorithm requires a preprocessing phase for computing certain offsets into the list of active boundary edges, resulting in an asymptotic worst case time complexity of $O(n^2)$ or $O(n \log n)$ depending on the choice of data structures.

Our approach better compares to the Edgebreaker's Wrap&Zip (cf. [Rossignac and Szymczak 1999]) extension which also guarantees linear time complexity. Here, visiting a node first during top-down traversal applying *create-strip* can be interpreted as the wrapping phase, while *glue-meshes* before leaving

the current recursion level does the zipping. In the analogue notation (see previous section), the C operation in our method would extend the strip while L, R, S, E steer the gluing.

There are always at most three meshes, the dividing strip and the two submeshes, that have to be taken into account by the procedure *decode-preorder*, and the active boundary loops can be implicitly determined from navigation through the meshes. Still, strips are generated early before the recursion and glued later. This leads to an effective two-pass algorithm. Even though the tree itself is only traversed once, every strip is touched twice by *create-strip* (top-down) and *glue-meshes* (bottom-up, see also Figure 4.4).

By using a different tree traversal (cf. [Rössl et al. 2003b]) we can adapt the decoding algorithm for a bottom-up reconstruction in the spirit of Isenburg and Snoeyink's [2001] Spirale Reversi decoding. Assume that the code is given as a postorder traversal of a tree with the strip length and the tree connectivity $c \in \{L, R, S, E\}$ for each node. The standard *iterative* algorithm for constructing a tree from this stream is modified in a way that it will construct the mesh instead, resulting in Algorithm 4.3.

Algorithm 4.3 *(decode postorder traversal)* *decode-postorder () :=*
while $(length, c) =$*input_length_and_LRSE () :*

1. $(gate, left, right)$=*create-strip* $(length)$
2. *case* $c = L$: $gate_{left}$=*pop (), glue-meshes* $(left, gate_{left})$
3. *case* $c = R$: $gate_{right}$=*pop (), glue-meshes* $(right, gate_{right})$
4. *case* $c = S$: *for* $d \in \{left, right\}$: *glue-meshes* $(d$,*pop ())*
5. *push* $(gate)$

An initially empty stack of gates into submeshes is required. For every symbol that is read from the input stream a triangle strip is created. The gate of this strip is just pushed onto the stack in case of a leaf node (E), whereas else either one (L, R) or two (S) gates are popped from the stack and their submeshes are glued to the current strip whose gate is then pushed. Figure 4.7 shows an example for this reverse decoding. Arbitrary topology is handled as described in the previous section.

The concept of reverse decoding from [Isenburg and Snoeyink 2001] translates easily to our algorithm. This results in an iterative, single-single pass decoding algorithm. In the following section, we discuss our tree data structure and the different options for traversal in the context of data compression.

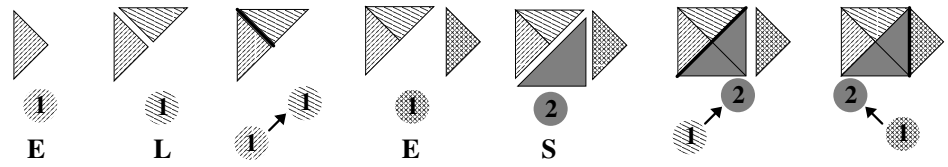


Figure 4.7: Reverse decoding from a postorder traversal of the tree from Figure 4.3. The input from encoding this traversal is `1E,1L,1E,2S`.

4.3 Tree-based Data Structures

In this section, we study the encoding algorithm while considering the output tree data structure and the compression of these data (cf. [Ivrissimtzis et al. 2003]). With the divide and conquer approach, the encoding of triangle mesh connectivity can be split into two separate but closely related subtasks: the *encoding of the binary tree*, and the *encoding of the strip lengths* stored in its nodes. We study these two encodings, first separately and then in their interrelation and provide experimental results. For simplicity, we assume planar meshes for the analysis. However, arbitrary topology can be treated in a similar manner, taking into account the special output symbols and the changes of the Euler's characteristic they introduce.

4.3.1 Binary Tree Encodings

The binary tree is one of the most popular structures for storage and maintenance of data, and thus, many basic related problems have been widely studied. For a brief comparative study of different binary tree encodings see e.g. [Katajainen and Mäkinen 1990] and [Mäkinen 1991].

In our context, the relevance of the tree encoding methods becomes apparent with the observation that any binary tree can correspond to a planar mesh. Indeed, if all the strip lengths are equal to one, then by Theorem 4.2 every binary tree gives a valid planar mesh. The number of all the binary trees with n nodes is given by the Catalan number C_n

$$C_n = \frac{(2n)!}{n!(n+1)!} \sim \frac{4^n}{\sqrt{\pi} n^{3/2}} \tag{4.1}$$

(see e.g. [Graham et al. 1994]). Thus, there is an asymptotic bound of two bits per node.

Some of the existing methods for binary tree encoding enumerate all the trees with n nodes so that each tree can be represented by an integer number. Obviously, such methods achieve optimal compression ratios, but the encoding and

decoding cost is very high, and especially for the large trees we use here this cost is prohibitive. Other encoding methods traverse the tree, transmitting one letter for each node. These methods are separated into two categories: methods which use a fixed alphabet and methods using an alphabet depending on the number of the nodes n.

The most common fixed alphabet encoding uses four letters, say $\{L, R, S, E\}$ to make the analogy with the Edgebreaker clearer (see Section 4.2.4). Each letter determines if the corresponding node has only left, only right, both or none children. The cost is $2n$ bits, which, asymptotically achieves the bound of two bits per node given by (4.1). By Theorem 4.1 this is also equal to two bits per vertex $(2b/v)$.

Another well-known fixed alphabet encoding are Zaks' sequences which use the two letters alphabet $\{0, 1\}$. The encoding process first transforms the binary tree into a complete binary tree by appending new leaves wherever possible, i.e. two new leaves at any old leaf and one new leaf at any single child node. Then we traverse the nodes of the complete binary tree in preorder, transmitting a 1 symbol if the node is internal and a 0 symbol if the node is a leaf. There are standard algorithms deciding when a given sequence of 0's and 1's is the Zaks' sequence of a tree.

Concerning compression, the size of the tree code can be further reduced with entropy coding such as arithmetic coding (see e.g. [Moffat et al. 1998]). In this case the compression ratio also depends on the traversal of the tree. Note that there are traversals, like the inorder, which do not work with the above four or two letter encodings, because we can not reconstruct the tree without some additional information.

The variable alphabet encodings most often use the letters of the set $\{0, 1, \ldots, n\}$. When it comes to compression issues they have the problem that it is more difficult to find a sharp guaranteed upper bound which can be trivially found for a fixed alphabet. Nevertheless, many times they are more flexible, and there are standard algorithms determining the validity of a code, making it easy to eliminate the transmission of redundant information.

An example of variable length encoding is the weight sequence (see e.g. [Mäkinen 1991]). There, the letter corresponding to a node is the number of the nodes of its left subtree. The inductive argument to show that the method works is as follows. If we know the number of the nodes of the tree and the number of the nodes of the left subtree, we can find the number of nodes of the right subtree by subtraction, and we can continue this process recursively until we reach the leaves. The weight sequence encoding of a tree is of special interest because by Theorem 4.1 the number of the nodes n is related to the number of the vertices v of the mesh. For this reason, the weight sequence is a useful intermediate representation of the binary tree corresponding to a mesh.

Mesh	#V	#F	HC	FIXED	F&AC	A&D
David 1	315	586	3.96	3.59	3.94	2.96
David 2	1512	2924	3.43	3.69	3.56	2.88
David 3	6035	11820	3.20	3.69	3.39	2.70
David 4	24085	47753	3.09	3.71	3.18	2.52
Dinosaur	14070	28136	3.00	3.73	3.13	2.25
Fandisk	6475	12946	2.49	3.51	1.95	1.02
Mannequin 1	428	839	3.63	3.69	3.79	2.51
Mannequin 2	11703	23402	1.91	3.33	1.06	0.37
Venus	8268	16532	3.27	3.71	3.46	2.71
Max-Planck	100086	199996	2.42	3.51	1.42	n/a

Table 4.1: Results from connectivity encoding as described in Section 4.3.2. The first three columns show the name of the data sets, the number of vertices, and the number of triangles. The remaining columns provide the compression rates in bits per vertex for using plain Huffman coding (HC), the fixed alphabeth (FIXED), and the latter with subsequent arithmetic coding (F&AC). The last column compares to Alliez and Desbrun [2001a], who report the best compression rates in practice.

4.3.2 Weighted Binary Tree Encoding — A First Approach

Before we continue to study and exploit the relationships between the binary tree and the strip lengths associated as weights of nodes, we report on some experimental results from a straightforward approach taken in [Ivrissimtzis et al. 2002]. We encode the tree along with the nodal weights in an intervowen fashion: A preorder traversal of the tree is assumed, and each node is assigned an integer $4n + (b_l b_r)_2$, where n denotes the associated strip length and the two-bit binary number $(b_l b_r)_2$ represents the tree connectivity (and hence refers to one of L, R, S, E). This will increase the number of symbols only slightly due to the fact that empty submeshes, e.g. $b_l = 0$, do not occur arbitrarily as discussed before. Table 4.1 provides results for a number of well-known benchmark meshes from Alliez and Desbrun [2001a] and compares to the results from their best-performing valence-driven approach.

The compression rates are given results in bits per vertex for three variants: for the Huffman-coded $4n + (b_l b_r)_2$ symbols (*HC*), for an alternative, fixed symbol table (*FIXED*, guaranteeing $< 4b/v$, see next Section 4.3.3), and the latter with arithmetic coding applied (*F&AC*, [Moffat et al. 1998], bit model) to the output. The binary trees are traversed in preorder.

In the following sections, we will separate the encoding of the strip lengths from the encoding of the binary tree and study them in more detail.

4.3.3 Strip Lengths Encodings

The encoding of the strip lengths is equivalent to the encoding of $v-2$ numbers summing up to t. The range of the numbers is from one up to the length of the largest strip we create. The largest strip has always length less than $\lfloor \frac{v}{2} \rfloor$, still this is not a sharp bound with any practical use.

The simplest way to encode such a sequence of numbers is to use a word of k bits consisting of $k-1$ 1's followed by a 0 to represent the number k. The total cost in this case is t bits, and the total cost of encoding the mesh — including the tree — is bounded by $t+2v-4 < 4v$. It is worth noting here that the guaranteed performance of $4b/v$ for this simplest encoding method, can not be improved without going deeper into the study of the relationship between the encoding tree and the corresponding strip-lengths.

Another, very popular method to encode a sequence of numbers is the Huffman code (see e.g. [Storer 1992]). Each number is assigned a unique code, and the number of bits we spend on each code depends on the probability of each number to appear in the sequence. After the Huffman coding, we can use arithmetic coding to further exploit any existing entropy. We remark that the compression achieved with the arithmetic coding depends on the order the strip lengths are transmitted, i.e. on the particular traversal of the tree. In addition, we remark that the situation is different to that in Section 4.3.1 as now the tree is given, hence any traversal works. For a survey of different tree traversals see for instance [Berztiss 1986].

If we assume that the number of boundary vertices of the mesh is relatively small, the number of the vertices — and thus, the number of the tree nodes — is about half the number of the triangles due to Euler's formula. Therefore, the average strip length is near two, and the entropy of strip lengths largely depends on the number of strips with length two. Our experiments show that the more regular — i.e. almost all vertices have valence six — a mesh the more strips of length two occur in its encoding. For an intuitive explanation of the latter fact, consider Figure 4.1: A large strip passing through a regular area of the mesh creates a regular boundary with vertices of valence four, and this in turn creates a lot of strips with length two. This observation partly justifies the choice of the leading directed edges of the dividing strip as the gates of the two submeshes. An alternative deterministic choice of the gates, for example near the middle of the dividing strip, would increase the length of the strips near the root of the tree but the result would be worse in terms of compression ratios, because the total entropy would decrease.

The Table 4.2 shows the experimental results for preorder and postorder encoding of the tree with a four- and a two-letter alphabet, and preorder, inorder, postorder and level order traversal in the encoding of the strip lengths, for a variety of meshes. In [Ivrissimtzis et al. 2002] we used preorder traversals for both,

Mesh	$t_{pre,1}$	$t_{pre,2}$	$t_{post,2}$	$s_{pre,H}$	s_{pre}	s_{post}	s_{in}	s_{level}
David 1	2.46	2.46	2.46	1.40	2.08	2.08	2.03	2.08
David 2	2.11	2.10	2.11	1.43	1.66	1.67	1.66	1.67
David 3	2.03	2.03	2.03	1.46	1.58	1.60	1.53	1.62
David 4	2.05	2.06	2.06	1.46	1.51	1.53	1.39	1.55
Dinosaure	2.04	2.06	2.05	1.45	1.52	1.54	1.37	1.55
Fandisk	1.52	1.50	1.50	1.51	1.23	1.24	0.75	1.57
Mannequin 1	2.34	2.34	2.34	1.36	1.89	1.91	1.87	1.91
Mannequin 2	0.92	0.91	0.90	1.30	0.84	0.83	0.34	1.21
Venus	2.04	2.05	2.05	1.49	1.60	1.61	1.55	1.63
Max-Planck	1.20	1.16	1.16	1.52	1.08	1.06	0.67	1.39

Table 4.2: Results from binary tree and strip lengths encoding for different tree traversals. The trees (t) are encoded separately from the strip lengths (s). The suffixes indicate pre-, post-, in-, and level-oder traversal, the 1-alphabet (Zaks) or 2-alphabet. For $s_{pre,H}$ only Huffman coding is applied, arithmetic coding and a fixed alphabet is used in all other columns.

the tree and the strip lengths, and we transmitted them in an interwoven fashion, i.e. at each node the code of the strip length is followed immediately after the tree code. Comparing the results there (see Table 4.1), even with the most favorable combination of separate transmission of tree and strip lengths file, we see that they are better. That means that there is a lot of entropy in the blend of tree codes and strip lengths, and this entropy is exploited by the arithmetic coder.

Next we assume that the tree code and the strip length code of a node are sent one after the other. This means that we assume the same traversal for both the tree code and the strip lengths code. We study the relationship between the two encodings, and we see how we can save information from the tree code using information from the strip-lengths and vice versa.

4.3.4 Tree First Transmission

For this section, we assume that the code of the tree is sent first, using the $\{L, R, S, E\}$ alphabet, and the code of the strip length follows. Some simple observations can reduce the amount of information we have to send for the strip lengths.

A first observation is that because of Theorem 4.2, condition (ii), the weight of an R-node is either one or an even number, while the weight of an L-node is always odd. Another observation, from the same Theorem, condition (i), is that all the leaves have strip length one, and do not need encoding. Going one step further into the study of this situation, we realize that if an R is just above a leaf then,

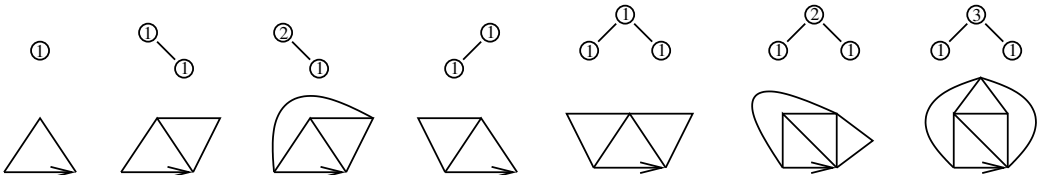


Figure 4.8: The encoding trees and the corresponding meshes near the leaves. Note that some of the rooted meshes are isomorphic as un-rooted meshes.

by condition (i), the corresponding strip length is either one or two and can be encoded with a single bit. Similarly, an L-node just above a leaf can only have a strip length one, and we do not need to transmit any information. Figure 4.8 shows the encoding near the leaves and the corresponding meshes. The above observations are the simplest instances of a more general feature of our approach, namely that every node of the tree represents a gluing operation, and Theorem 4.2 gives a simple criterion to determine when such a gluing operation is legal. Therefore, we can treat these instances in a unified way by determining at each node all legal gluing operations and sending only the necessary information for the decoder to distinguish between them. This assumes a postorder reconstruction of the mesh.

Checking the criterion of the special gluing operations is straightforward. By Theorem 4.2, condition (ii), if the node is of L-type, then the set of possible strip lengths is restricted to odd numbers. In contrast, if the node is of R-type, the set of possible strip lengths consists of the even numbers and the one. For a criterion to check for regular gluing operations, i.e. Theorem 4.2, condition (i), we require for each node the number of nodes of its subtree and the sum of the strip lengths of that subtree.

This information can be held in an auxiliary data structure, where each node of the tree is assigned a pair of integers (n_j, w_j), the number of nodes n_j and the sum of strip lengths w_j of its subtree. Note that the integers w_j are essentially the weight sequence of the tree (see Section 4.3.1). When we process a new node, we first find its children and assume that the integers assigned to them are (n_l, w_l) and (n_r, w_r). Then

$$n_j = n_l + n_r + 1 \quad \text{and} \quad w_j = w_l + w_r + s \ ,$$

where s is the strip length corresponding to $\mathbf{n}_j$. Then, Theorem 4.2, condition (i), gives

$$w_l + w_r + s + 1 \leq 2(n_l + n_r + 1) \ ,$$

and hence

$$s \leq 2n_l + 2n_r - w_l - w_r + 1 \ .$$

This way we find explicitly the set of all the strip lengths giving a legal gluing operation at a node. And instead of transmitting the actual strip length, we send

an offset determining its position in the set of all legal values. If the set contains only one element, i.e. for the set $\{1\}$ we send no information. If the set has two elements, i.e. this is either $\{1, 2\}$ or $\{1, 3\}$, we send only one bit. In all the other cases, it is better to resort to Huffman encoding of the offsets rather than using an ad hoc code for every particular set, because of the significantly higher frequencies of the short strips.

The algorithm we just described can also be used as a test for the validity of a code, e.g. with the original encoding of the output of Algorithm 4.1 as proposed in [Ivrissimtzis et al. 2002], were the actual strip lengths are transmitted. In this case, we proceed as above and find the set of all legal strip lengths corresponding to a node, and we check whether the current value of s is included in that set. If this happens for every node then all the gluing operations are legal, and the code describes a valid mesh. If there is a node with strip length not included in the set, then the decoding process will break at that point. Of course the testing algorithm must be coupled with an algorithm checking the legality of the corresponding tree code.

4.3.5 Strip-lengths First Transmission

In contrast to the previous section, suppose now that we first transmit the strip-length of a node and then the tree code corresponding to it. From Theorem 4.2, condition (ii), even strip lengths correspond to an S- or an R-node, while odd strip lengths greater than one correspond to an S- or an L-node. Therefore, we need a single bit for the tree code of the nodes with strip length greater than one.

Table 4.3 shows some more results obtained with the encoding techniques described in Sections 4.3.4 and 4.3.5. This setting is similar to the initial experiment in Section 4.3.2. Depending on the mesh the results here tend to be slightly better.

4.3.6 Valence-3 Vertices

From Theorem 4.2, and the above discussion of its implications, it is apparent that the vertices with valence three are very characteristic. Their peculiarity arises from the fact that a zig-zag strip collapses to itself only when passing through a valence-3 vertex (see Section 4.2.2). Equivalently, it is the only case when a strip of length greater than two can correspond to an R- or an L-node. Therefore, in many cases it may pay off to have an initial preprocessing step clearing the mesh from its valence-3 vertices. A similar strategy to improve mesh regularity and efficiency of algorithms was also proposed by Alliez and Desbrun [2001b] in the context of progressive coding.

After the clearance step we proceed as described in Sections 4.3.4 and 4.3.5, separating the case of tree code transmission first from the case of strip length

Mesh	pre	post,H	post
David 1	3.94	3.30	3.94
David 2	3.53	3.35	3.53
David 3	3.36	3.40	3.36
David 4	3.13	3.39	3.14
Dinosaure	3.09	3.38	3.10
Fandisk	1.90	3.42	1.94
Mannequin 1	3.79	3.29	3.78
Mannequin 2	1.02	3.27	1.07
Venus	3.42	3.40	3.42
Max-Planck	1.38	3.43	1.42

Table 4.3: The table shows results for transmitting the tree (2-alphabet) and indices of valid strip lengths in an interwoven fashion in pre- and postoder, the latter with Huffman coding (*post,H*) only and arithmetic coding (*post*), respectively.

transmission first. Sending the tree code first, we know that an L-node can only store a strip length equal to one, because any greater strip length would create a valence-3 vertex, and therefore, we do not need to send any strip length information. Similarly, a strip length corresponding to an R-node is either one or two, and it is encoded in a single bit. On the other hand, if we first transmit the strip length of a node, then any length greater than two corresponds to an S-node, and we do not need any extra tree code. A strip length equal to two, corresponds to either an S or an R-node and we need a single bit for the tree code.

4.4 Stripification: An Application to Rendering

So far, we discussed the divide and conquer connectivity encoding and the resulting tree data structure from a data compression point of view. In this section we show how this structure can be applied for efficient rendering of triangle meshes (cf. [Rössl et al. 2003b]).

Here, a common technique is to decompose a mesh into a set of triangle strips. The reason is, that then instead of sending three vertices per triangle to the graphics hardware, only $l + 2$ vertices have to be transmitted for a strip of length l. The triangles of a strip are represented as a sequence of vertices just as illustrated in Figure 4.2. For a penalty of one additional vertex — which is sent twice and introduces an empty or degenerate triangle — the orientation of the strip can be swapped, allowing so called generalized strips. This is the representation sup-

ported by OpenGL (cf. [Woo et al. 1997]).

If the cost for rendering a mesh is expressed as the total numbers of vertices which have to be transmitted to the graphics hardware, these are $\sum_{i=1}^{v-2}(l_i+2) = 2(v-2)+t \approx 2t$ vertices using the strips from our encoding method versus $3t$ vertices without strips but using individual triangles. We can improve this guaranteed saving of $\frac{1}{3}$ of the cost by extending the strips, i.e. concatenating every parent strip in the tree with one of its children. Here, either the left or the right child is the natural choice depending on the parity of the strip length, and a swap-penalty is introduced for the other one. While traversing the tree in preorder, we do a greedy decision for every node and chose always this natural child for extending the current strip. A new strip is started beginning from the other child node.

In fact we can deduce an upper bound $c_{\max}$ for the cost using this kind of extended strips. Consider a basic cost of $t+2$, i.e. the mesh is – hypothetically – encoded as one single strip. Let s, e, l, r denote the total numbers of the nodes of type S, E, L, R, respectively. For every S-node we have two children. The strip is concatenated to the the child strip in the natural zig-zag direction without any penalty (zero cost). We have to start a new strip from the other child which costs two vertices for initialization. This yields a total cost of $2s$ for all S-nodes. An L- or an R-node might or might not cost a penalty vertex depending on the orientation. For the upper bound we assume to have *always* the extra cost, i.e. $l+r$ vertices in total. An E-node terminates a strip and does not induce any cost. We now have

$$c_{\max} = (t+2) + 2s + l + r$$

but $s = e-1$ (relation between splits and leaves in a binary tree) and $s+e+r+l = n = v-2$ (total number of nodes from Theorem 4.1), and we get a guaranteed 50% saving from

$$c_{\max} = (t+2) + s + (e-1) + l + r \;=\; t+v-1 \;\approx\; 3v \;\approx\; \frac{3}{2}t$$

On average, we may assume that only for half of the L- and R-nodes the swap-penalty must be paid.

Empirically, we get an average saving of $> 57\%$ (always between 56% and 59%) for the models used in [Ivrissimtzis et al. 2002]. Figure 4.9 shows typical results of the stripification. It compares also to stripifications resulting from the well-known optimization method of Evans et al. [1996] — we used their *STRIPE V2.0* implementation — which provide some few percent more savings. Evans' method applies heuristics to find a good stripification, as finding the optimal strips is known to be a NP-complete problem. In a similar context, Isenburg [2000] presents an algorithm for encoding and compressing the connectivity along with a stripification, e.g. the one generated by *STRIPE*, of a polygonal mesh.

(a) (b)

Figure 4.9: Examples for stripification for rendering. (a) 3219 vs. 7401 cost (57% saving), 131 strips (661 swaps). (b): 255935 vs. 599988 cost (57% saving), 10711 strips (49477 swaps). The corresponding cost from *STRIPE* are 2992 (60%) and 232264 (61%) vertices, respectively.

In contrast, our encoding algorithm does not do any (expensive) optimization, nor does it encode a certain stripification, which requires the storage of some extra amount of data. Using the recursive encoding, we obtain a stripification that is implicitly defined by the resulting binary tree data structure and that does not have to be encoded explicitly. Still, this stripification comes with a guarantee and is competitive when compared to others generated by more sophisticated algorithms, and it thus can be exploited for efficient rendering. We remark that, we considered the quality of this stripification only using a simple cost measure. More sophisticated measures are applied e.g. in [Bogomjakov and Gotsman 2002].

4.5 Summary

We presented algorithms for the encoding and decoding of the connectivity of a triangle mesh. The encoding works recursively following the divide and conquer principle, which makes its description intuitive. The output of the algorithm is naturally organized as a binary tree with positive integer weights assigned to its nodes, and which features additional properties used for the further analysis. The algorithms can take advantage of different tree traversal strategies, and we showed how the post order traversal can be exploited for reverse decoding yielding an effective single pass algorithm. We studied the tree data structure, different

traversals as well as the coupling of the tree structure with the node weights in the context of data compression. The derived properties reveal some redundancy in the code which can be either stripped for compression or used for validity checks.

We remark that there is a connection to the Edgebreaker encoding. From this observation, we can improve the reported four bits per vertex to obtain a guaranteed upper bound of 3.585 bits per vertex as compression ratio. In order to see this, assume the simplest encoding in Section 4.3.3 and associate every 1 with a C-symbol and each 0 and one of L, R, S, R (see Section 4.2.4), yielding a total number of one bits per C three bits for the latter symbols. The number of C's is $t - v + 2$, and the number of L, E, R, S is $v - 2$. Any non-empty string of C symbols denotes a strip of length greater than one, and — using arguments from the previous sections — it can be followed by only two of the L, E, R, S, depending on the parity of the number of C's. With this correspondence established, we obtain the guaranteed upper bound following [Gumhold 2000]. In practice, much better results are achieved, especially for very regular meshes.

Besides compression, we show that the generated tree data structure offers potential to other applications such as efficient rendering. In this context, we show that the encoding algorithm generates a partition into generalized triangle strips at no extra cost, and that this stripification competes well with results of more expensive optimization algorithms.

Connectivity data is only one part of surface meshes which come with additional (geometric) attributes such as positions and surface normals. For the efficient compression of certain attributes, certain traversals may be favored. For instance zig-zag strips may be of advantage for vertex normals, which are supposed to have low variation along the strip. In general, connectivity and attributes can be considered in an interwoven fashion as applied by Lee et al. [2002], i.e. the attribute data steers the construction of the strip. The proposed framework provides the flexibility to include such considerations.

Clearly, the size of attribute data may dominate the total size of a data set. Consequently, there are many applications which require a (potentially lossy) compression of these attributes, and there are a variety of algorithms dealing with the most common attributes like vertex positions. A typical example is mesh decimation under the preservation of the overall shape of the surface. In the next section, we present a framework for the compression of vector valued attributes — i.e. of a piecewise linear vector field. This is particularly challenging and interesting due to the constraint of preserving the topology of the vector field, which is a global property — in contrast to a distance measure to a reference shape as in the example before.

Chapter 5
Vector Field Compression

5.1 Background

In the previous section, we discussed the compression of the connectivity of triangle meshes, now we shift the focus to attribute data associated with the vertices. Here, a particular interesting and challenging problem is the compression of vector fields under certain global constraints: A vector is assigned to every vertex, this defines a piecewise linear vector fields over the triangular domain. The goal is to find a representation which can be stored more compactly, i.e. a piecewise linear function with considerably fewer coefficients, while the topology of the original vector field is preserved. The demand and applications for such compression techniques originate from flow visualization. We give a brief overview in the following.

Flow visualization is one of the most important subfields of scientific visualization. From its very beginning, flow visualization had to face the problem of dealing with large and complex data — usually far more complex than a human is able to process, or than computers can transmit and process in acceptable times. Thus most of the flow visualization techniques are somehow involved with compressing and simplifying the flow data, either by visualizing only important parts of the data or by extracting features which contain the most relevant information about the vector field. We refer to Post et al. [2002] for an overview of flow visualization techniques.

One of the most important features of a vector field is its *topological skeleton* which has been introduced as a visualization tool in [Helman and Hesselink 1989]. The topological skeleton of a vector field essentially consists of a collection of critical points and special stream lines called separatrices which separate the flow into areas of different flow behavior. The attractiveness of the topological skeleton as a visualization tool lies in the fact that

even a complex flow behavior can be expressed (and visualized) by using only a limited number of graphical primitives.

In the original work of Helman and Hesselink [1989], only first order critical points were considered, i.e. critical points with a non-vanishing Jacobian. Based on an eigenvector analysis of the Jacobian matrix, these critical points were classified into sources, sinks, centers and saddles. Furthermore, the only separatrices which were considered started from the saddle points in the directions of the eigenvectors of the Jacobian matrix there. In addition, separatrices from detachment and attachment point starting at zero-flow boundaries were considered.

In the following years, this concept of the topological skeleton of a vector field has been extended in several ways. Scheuermann et al. [1998] treat the appearance of higher order critical points. These critical points are characterized by a number of sectors of similar flow behavior around them, namely parabolic sectors, elliptic sectors, and hyperbolic sectors [Firby and Gardiner 1982]. In [de Leeuw and van Liere 1999a], separatrices starting from boundary switch points are considered to separate regions of different inflow/outflow behavior across the boundary of the flow. In [Wischgoll and Scheuermann 2001] the appearance and detection of closed stream lines is treated. Trotts et al. [2000] introduce critical points at infinity to obtain additional separatrices. Kenwright et al. [1999] consider separation and attachment lines as additional topological features. Bajaj et al. [1998] treat the topology of scalar fields for visualization purposes. First approaches for visualizing 3D topological skeletons are in [Globus and Levit 1991].

Flow data sets tend to be large and complex. This fact has motivated intensive research in *simplifying* and *compressing* vector fields. For both challenges, topological concepts have been applied.

Topological simplification techniques apply if the topological complexity of the data set is high and if it is known that certain topological features are due to noise. De Leeuw and van Liere [1999a] collapse critical points by using area metrics. The same authors [1999b] propose two methods: an implicit method (apply a global smoothing over the vector field), and an explicit method (collapse appropriate adjacent first order critical points). In [Tricoche et al. 2000], clusters of first order critical points are merged to a higher order critical point. Tricoche et al. [2001] collapse pairs of critical points under preservation of the underlying grid structure of the vector field. Westermann et al. [2001] analyze the curvature normal of certain time surfaces to obtain a topology-preserving smoothing of a vector field. The simplification of the topology of scalar fields (which can be considered as a special case of vector field topology) is treated in [Edelsbrunner et al. 2001] and [Bajaj and Schikore 1998]. All these topology simplification algorithms mentioned above focus on reducing the number of critical points and do not explicitly treat separatrices. However, since a high number of separatrices starts and ends in

critical points, a reduction of the number of critical points also reduces the number of separatrices.

Compression techniques for vector fields are motivated by the necessity of transmitting large flow data sets over networks with low bandwidth, or by the goal to produce visualizations of the data in low-end machines with a small main memory. For these cases the consideration of compressed vector fields makes the process of visual analysis of the flow data more efficient and is sometimes the only way to process the data in reasonable time at all. Simple compression techniques ([Heckel et al. 1999], [Garcke et al. 2000], [Telea and van Wijk 1999]) are based on distance functions which locally compare the vectors of the vector fields without including topological issues. Since the topological skeleton has proven to describe the vector field in a compact way, it is a natural approach to search for compression techniques which are based on the topology of the vector fields. [Lodha et al. 2000] is a first approach to compress a vector field under the consideration of preserving the characteristics of critical points. Based on a distance measure of vector fields which compares the present critical points, a compression is carried out until the difference of original and compressed vector field exceeds a certain threshold. In [Theisel 2002] a method is introduced which does not only preserve the critical points but also the behavior of the separatrices. This is achieved by extracting the topological skeleton and reconstructing it by a new piecewise linear vector field. For rather simple topologies, this reconstructed piecewise linear vector field turns out to be a compressed version of the original one. However, if the topology of the vector field becomes more complex, the compression ratio drops and might become even negative.

As the main contribution of this chapter, we introduce a new method of topology preserving compression of piecewise linear vector field (cf. [Theisel et al. 2003a]). Contrary to pre-existing compression methods, the method guarantees that the topology of original and compressed vector field coincides both for critical points and for the connectivity of the separatrices. We show that even under these strong conditions we achieve high compression ratios for vector fields with complex topologies. In this context, we study modifications of the topology preserving algorithm (cf. [Theisel et al. 2004a]) as well as topological simplification (cf. [Theisel et al. 2003b]). The experimental results on large data sets with complex topology illustrate and verify the theoretical framework and show the efficiency of the proposed techniques.

5.2 Theoretical Framework

In this section we give the theoretical background of our topology preserving compression algorithm (cf. [Theisel et al. 2003a]). The main contribution of this sec-

tion is to show in Theorem 5.1 that for a *local modification* of the vector field it can be decided by a *local analysis* whether the topology is affected by the modification. This property is not evident, since the topology of a vector field is a *global* feature, and local modifications of the vector field may change the global behavior. To show this property, we have to describe the concept of vector field topology and local modifications in a formal way. Section 5.2.1 gives a formal description of the topology of a 2D vector field. Section 5.2.2 discusses the concept of topological equivalence of vector fields. Section 5.2.3 treats the impact of local modifications to the topology of a vector field. Section 5.2.4 discusses further concepts and extensions of the topology of a 2D vector field from the viewpoint of topology preserving local modifications.

5.2.1 The Topology of a 2D Vector Field

Let $\mathbf{D} \subset \mathbb{E}^2$ be a closed point set which is bounded by n continuous boundary curves $\mathbf{F}_i$ $(i = 0, .., n-1)$. $\mathbf{D}$ serves as the domain of the vector field. Normally, $\mathbf{D}$ is bounded by only one curve, but we particularly allow domains with "holes", i.e. with more than one boundary curve. Then a *vector field* $\mathbf{v}$ is a continuous map $\mathbf{v}: \quad \mathbf{D} \rightarrow \mathbb{R}^2$. We define $\mathbf{C}(\mathbf{v})$ as the *set of all critical points* of $\mathbf{v}$:

$$\mathbf{C}(\mathbf{v}) = \{\mathbf{x} \in \mathbf{D} \ : \ \mathbf{v}(\mathbf{x}) = (0,0)^T\} \tag{5.1}$$

and assume that $\mathbf{C}(\mathbf{v})$ is a finite set, i.e., all critical points are isolated. Also, we assume that no critical point lies on one of the boundaries of $\mathbf{D}$. Furthermore, let $\mathbf{v}$ be differentiable in a neighborhood of each critical point, which gives that the Jacobian matrix

$$\mathbf{J}_\mathbf{v}(\mathbf{x}) \ = \ (\mathbf{v}_{x_1}(\mathbf{x}), \mathbf{v}_{x_2}(\mathbf{x}))$$

exists for each $\mathbf{x} \in \mathbf{C}(\mathbf{v})$ (cf. [Helman and Hesselink 1989]). Here we assume that all critical points are first order critical points, i.e. $\det(\mathbf{J}_\mathbf{v}(\mathbf{x})) \neq 0$ holds for all $\mathbf{x} \in \mathbf{C}(\mathbf{v})$. Based on an analysis of $\mathbf{J}_\mathbf{v}$, first order critical points can be classified into *saddles*, *sources*, *sinks*, and *centers* (cf. Helmann and Hesselink [1989, 1991]). For the description of the topology, saddle points are of particular interest. A saddle $\mathbf{s} \in \mathbf{C}(\mathbf{v})$ is characterized by $\det(\mathbf{J}_\mathbf{v}(\mathbf{s})) < 0$.

Let $\mathbf{F}_i$ be a boundary curve of $\mathbf{D}$. Then $\mathbf{v}$ partitions $\mathbf{F}_i$ into a number of consecutive regions of heterogenous outflow and inflow behavior of the vector field (cf. [de Leeuw and van Liere 1999a]). A point $\mathbf{b} \in \mathbf{F}_i$ is called *boundary inflow point* iff

$$\exists\, \varepsilon_0 > 0 \ \ \forall\, \varepsilon > 0 \ : \ \varepsilon < \varepsilon_0 \Rightarrow (\mathbf{b} + \varepsilon\, \mathbf{v}(\mathbf{b})) \in \mathbf{D}$$

i.e., the flow enters the domain $\mathbf{D}$ there. A point $\mathbf{b} \in \mathbf{F}_i$ is called *boundary outflow point* iff

$$\exists\, \varepsilon_0 > 0 \ \ \forall\, \varepsilon > 0 \ : \ \varepsilon < \varepsilon_0 \Rightarrow (\mathbf{b} + \varepsilon\, \mathbf{v}(\mathbf{b})) \notin \mathbf{D}$$

i.e., the flow leaves the domain $\mathbf{D}$ there. A point $\mathbf{b} \in \mathbf{F}_i$ is called *boundary switch point* iff in every ε-neighborhood of $\mathbf{b}$ both boundary inflow and outflow points exist. For the case of differentiable boundary curves, boundary switch points are characterized by a flow direction parallel to the tangent direction of the boundary curve in this point. Let $\mathbf{B}(\mathbf{v})$ be the *set of all boundary switch points* of $\mathbf{v}$.

In the following we assume that $\mathbf{B}(\mathbf{v})$ is a finite set. We consider the boundary switch points on the boundary curve $\mathbf{F}_i$. The points on $\mathbf{F}_i$ which are located between adjacent boundary switch points (while traversing in counterclockwise direction on $\mathbf{F}$) have a similar inflow/outflow behavior: all of them are either boundary inflow points, or all of them are boundary outflow points. We call the set of all points on $\mathbf{F}$ between two adjacent boundary switch points a *boundary inflow region* or *boundary outflow region*. Figure 5.1a gives an illustration.

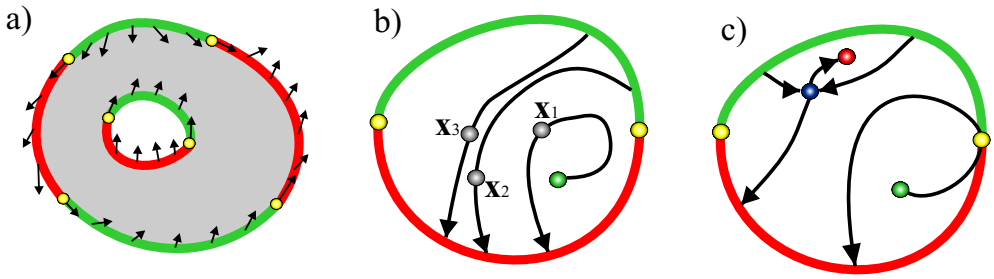


Figure 5.1: a) The domain $\mathbf{D}$ (gray area) is bounded by two boundary curves. The outer boundary curve has four boundary switch points (yellow) which produce two boundary inflow regions (green line) and two boundary outflow regions (red line). The inner boundary curve has two boundary switch points producing one boundary inflow region and one boundary outflow region. b) Three points $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3$ and their stream lines; $\mathbf{x}_2$ and $\mathbf{x}_3$ are stream line equivalent while $\mathbf{x}_1$ is not stream line equivalent to $\mathbf{x}_2$ or $\mathbf{x}_3$. c) Topological skeleton of a vector field consisting of two boundary switch points (yellow), one boundary outflow region (red line), one boundary inflow region (green line), one saddle point (blue point), one source (green point), one sink (red point), and the separatrices (black lines).

A *stream line* of $\mathbf{v}$ is a curve in $\mathbf{D}$ with the property that for every point on the curve the tangent direction coincides with the direction of $\mathbf{v}$ in this point (cf. [Helman and Hesselink 1989]). A particular stream line can be obtained by picking a point $\mathbf{x} \in \mathbf{D}$ and integrating both in forward and backward direction until the stream line either ends in a critical point or leaves $\mathbf{D}$ in a boundary inflow/outflow point (for now we assume that no circulating behavior of a stream line is present). Note that through every non-critical point there is exactly one stream line. Furthermore, stream lines cannot intersect each other (except in critical points). These facts can be used to classify points in $\mathbf{D}$ with respect to the behavior of the stream line through them.

Let $\mathbf{x} \in \mathbf{D}$, and let $\mathbf{S}_\mathbf{v}(\mathbf{x})$ be the stream line through $\mathbf{x}$. $\mathbf{S}_\mathbf{v}(\mathbf{x})$ can be integrated both in forward and backward direction until it ends in a critical point or leaves $\mathbf{D}$ in a boundary inflow/outflow point. We define

Definition 5.1 *Two points* $\mathbf{x}_1, \mathbf{x}_2 \in \mathbf{D}$ *are* stream line equivalent *concerning* $\mathbf{v}$ *(written* $\mathbf{x}_1 \sim^s_\mathbf{v} \mathbf{x}_2$*) if the stream lines through* $\mathbf{x}_1$ *and* $\mathbf{x}_2$ *end in the same critical point or inflow/outflow region, for both forward and backward integration.*

Figure 5.1b gives an illustration of Definition 5.1.

The relation $\sim^s_\mathbf{v}$ partitions $\mathbf{D}$ into sectors of similar flow behavior. It is known (cf. [Helman and Hesselink 1991, de Leeuw and van Liere 1999a]) that these sectors are separated by a number of particular stream lines called *separatrices*. The set of separatrices can be constructed by considering the stream lines originating from the boundary switch points (both by backward and forward integration), and certain stream lines starting from saddle points.

Every boundary switch point $\mathbf{b}$ is related to two separatrices: one separatrix is obtained by forward integration from $\mathbf{b}$, the other by backward integration. From a saddle point $\mathbf{s}$, four separatrices are constructed: for both eigenvectors of $\mathbf{J}_\mathbf{v}(\mathbf{s})$ in forward and backward direction. This means, we construct separatrices from the four points

$$(\mathbf{s} + \varepsilon\, \mathbf{j}_1)\ ,\ (\mathbf{s} - \varepsilon\, \mathbf{j}_1)\ ,\ (\mathbf{s} + \varepsilon\, \mathbf{j}_2)\ ,\ (\mathbf{s} - \varepsilon\, \mathbf{j}_2)$$

where $\mathbf{j}_1, \mathbf{j}_2$ are the two eigenvectors of $\mathbf{J}_\mathbf{v}(\mathbf{s})$, ε is a very small positive number, and the integration direction is "away from the saddle point". This is justified by the fact that for $\det(\mathbf{J}_\mathbf{v}(\mathbf{s})) \neq 0$, $\mathbf{v}$ is governed by a first order approximation in a neighborhood of $\mathbf{s}$. Using this system of separatrices, each separatrix is uniquely defined by its starting point and its integration direction.

After introducing the concepts above, we can define the topological skeleton of a vector field as the collection of critical points, boundary switch points, and separatrices. Figure 5.1c illustrates an example.

5.2.2 Topologically Equivalent Vector Fields

To compare the topology of vector fields, the concept of topological equivalence of two vector fields $\mathbf{v}$ and $\mathbf{w}$ over the same domain $\mathbf{D}$ has to be introduced. Several ways of doing this are possible. One rather restrictive way is to demand that $\mathbf{v}$ and $\mathbf{w}$ coincide in all critical points, their Jacobian matrices, and all separatrices. Since for topologically complex vector fields the separatrices lie rather densely in the domain, this definition tends to allow only identical vector fields to be equivalent.

On the other hand, a very loose definition of topological equivalence is to demand the coincidence of the structure of the topology graph (i.e., corresponding

critical points may vary both in location and in Jacobian, as far as their connectivity of the separatrices coincides.)

Here we want to use a compromise of the above-mentioned definitions:

Definition 5.2 *The vector fields* $\mathbf{v}$ *and* $\mathbf{w}$ *over the domain* $\mathbf{D}$ *are* topologically equivalent *(written* $\mathbf{v} \sim_{\mathbf{T}} \mathbf{w}$*) iff the following conditions hold:*

i. $\mathbf{C}(\mathbf{v}) = \mathbf{C}(\mathbf{w})$

ii. $\forall \mathbf{x} \in \mathbf{C}(\mathbf{v}) \ : \ \mathbf{J}_{\mathbf{v}}(\mathbf{x}) = \mathbf{J}_{\mathbf{w}}(\mathbf{x})$

iii. $\mathbf{B}(\mathbf{v}) = \mathbf{B}(\mathbf{w})$

iv. Corresponding separatrices in $\mathbf{v}$ *and* $\mathbf{w}$ *end in the same critical point or boundary inflow/outflow region.*

Concerning this definition, $\mathbf{v}$ and $\mathbf{w}$ have identical critical points and boundary switch points. Thus $\mathbf{v}$ and $\mathbf{w}$ also have the same number of separatrices. A separatrix in $\mathbf{v}$ corresponds to a separatrix in $\mathbf{w}$ if they start in the same point with the same integration direction.

This definition of topologically equivalent vector fields is rather restrictive to critical points and boundary switch points while giving some freedom to the separatrices. We made this choice because it ensures that topologically equivalent vector fields always have a zero distance concerning all known topology based vector field metrics (cf. [Lavin et al. 1998, Batra et al. 1999, Theisel and Weinkauf 2002]).

5.2.3 Local Modifications of the Topology

In this section we analyze the effect of local modifications of the vector field and its topology. Since the topology of a vector field is a *global* feature, local modifications can change the topology everywhere in its domain. Imagine for instance the creation or removal of critical points in the modified area which may affect the separatrices far away from this area.

Nevertheless, we show in this section that it can be decided entirely by a *local* analysis in the area to be modified, whether or not a local modification of the vector field will change its topology.

Let $\mathbf{v}$ and $\mathbf{w}$ be two differentiable vector fields over the domain $\mathbf{D}$, and let $\mathbf{D}' \subset \mathbf{D}$ be a closed subdomain which is bounded by one closed curve $\mathbf{F}'$ (thus assuming that $\mathbf{D}'$ does not have any holes). Furthermore, let $\mathbf{D}'' = (\mathbf{D} \setminus \mathbf{D}') \cup \mathbf{F}'$, and let $\mathbf{v}$ and $\mathbf{w}$ differ only inside $\mathbf{D}'$, i.e.

$$\forall \mathbf{x} \in \mathbf{D}'' : \mathbf{v}(\mathbf{x}) = \mathbf{w}(\mathbf{x}). \tag{5.2}$$

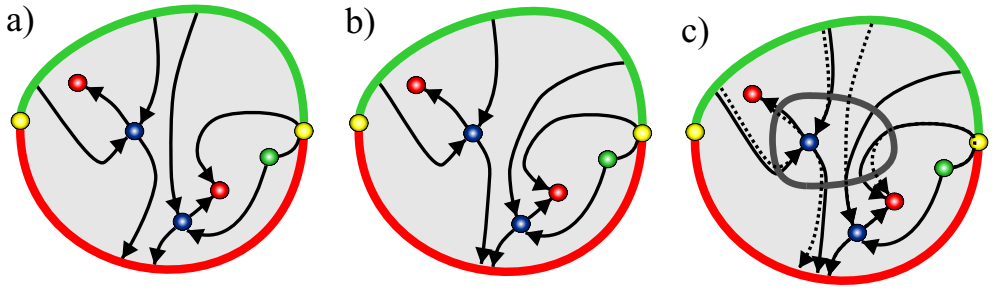


Figure 5.2: a) Example of a vector field $\mathbf{v}$ and its topological skeleton. b) Example of a vector field $\mathbf{w}$ and its topological skeleton. c) Overlay of a) and b): $\mathbf{v}$ and $\mathbf{w}$ differ only inside the area $\mathbf{D}'$ (marked by the inner ring). Stream lines which start in $\mathbf{D}''$ coincide in $\mathbf{v}$ and $\mathbf{w}$ until they enter $\mathbf{D}'$. Since they generally leave $\mathbf{D}'$ in different points on $\mathbf{F}'$, they have different paths in $\mathbf{D}''$after passing through $\mathbf{D}'$.

Figure 5.2 illustrates an example.
We search for local conditions for $\mathbf{v} \sim_{\mathbf{T}} \mathbf{w}$. To do so, we collect a number of points on $\mathbf{F}'$. Let $\mathbf{P}_{\mathbf{F}'}(\mathbf{v})$ be the set of all intersection points of all separatrices of $\mathbf{v}$ with $\mathbf{F}'$, and let

$$\mathbf{Q}_{\mathbf{F}'}(\mathbf{v}) = \mathbf{P}_{\mathbf{F}'}(\mathbf{v}) \cup \mathbf{B}(\mathbf{v}|_{\mathbf{D}'}) ,$$

where $\mathbf{v}|_{\mathbf{D}'}$ denotes the vector field $\mathbf{v}$ restricted to the domain $\mathbf{D}'$. Assuming $\mathbf{Q}_{\mathbf{F}'}(\mathbf{v}) \subset \mathbf{F}'$ to be a finite set, we can formulate

Theorem 5.1 *The vector fields* $\mathbf{v}$ *and* $\mathbf{w}$ *fulfilling (5.2) are topologically equivalent (*$\mathbf{v} \sim_{\mathbf{T}} \mathbf{w}$*) if the following conditions hold:*

1. *Every separatrix of* $\mathbf{v}$ *and* $\mathbf{w}$ *intersects* $\mathbf{F}'$ *at most once, i.e. has at most one entry point and one exit point with* $\mathbf{F}'$.

2. $\mathbf{v}|_{\mathbf{D}'} \sim_{\mathbf{T}} \mathbf{w}|_{\mathbf{D}'}$.

3. *The corresponding points in* $\mathbf{Q}_{\mathbf{F}'}(\mathbf{v})$ *and* $\mathbf{Q}_{\mathbf{F}'}(\mathbf{w})$ *are in the same circular order on* $\mathbf{F}'$ *(while traversing counterclockwise around* $\mathbf{F}'$*).*

From (5.2) and condition 2 of Theorem 5.1 it follows that $\mathbf{v}$ and $\mathbf{w}$ have the same critical points and boundary switch points. Hence, the separatrices of $\mathbf{v}$ and $\mathbf{w}$ have a one-to-one correspondence concerning starting point and integration direction. This and condition 1 of Theorem 5.1 gives that there is also a one-to-one correspondence between $\mathbf{P}_{\mathbf{F}'}(\mathbf{v})$ and $\mathbf{P}_{\mathbf{F}'}(\mathbf{w})$: a point $\mathbf{x}_1 \in \mathbf{P}_{\mathbf{F}'}(\mathbf{v})$ corresponds to $\mathbf{x}_2 \in \mathbf{P}_{\mathbf{F}'}(\mathbf{w})$ if their creating separatrices correspond, and both are either entry points or both are exit points of $\mathbf{D}'$ while integrating in integration direction. This

unique correspondence between $\mathbf{P}_{\mathbf{F}'}(\mathbf{v})$ and $\mathbf{P}_{\mathbf{F}'}(\mathbf{w})$ (and therefore also between $\mathbf{Q}_{\mathbf{F}'}(\mathbf{v})$ and $\mathbf{Q}_{\mathbf{F}'}(\mathbf{w})$) justifies the formulation of condition 3 of Theorem 5.1.

Theorem 5.1 means that for checking the topological equivalence of $\mathbf{v}$ and $\mathbf{w}$, we simply have to collect the points of $\mathbf{Q}_{\mathbf{F}'}(\mathbf{v})$ and $\mathbf{Q}_{\mathbf{F}'}(\mathbf{w})$, find the corresponding pairs, and compare their order on $\mathbf{F}'$.

To prove Theorem 5.1 we have to show that the conditions 1–3 of Theorem 5.1 yield the conditions i–iv of Definition 5.2. Conditions i–iii of Definition 5.2 follow directly from (5.2) and condition 2 of Theorem 5.1. Only iv of Definition 5.2 remains to be shown. To do so, we construct the topological skeleton of the vector fields $\mathbf{v}|_{\mathbf{D}''} = \mathbf{w}|_{\mathbf{D}''}$ as shown in Figure 5.3a. This gives $\mathbf{P}_{\mathbf{F}'}(\mathbf{v}|_{\mathbf{D}''}) = \mathbf{P}_{\mathbf{F}'}(\mathbf{w}|_{\mathbf{D}''})$. Furthermore, we construct the topological skeleton of the vector fields $\mathbf{v}|_{\mathbf{D}'}$ and

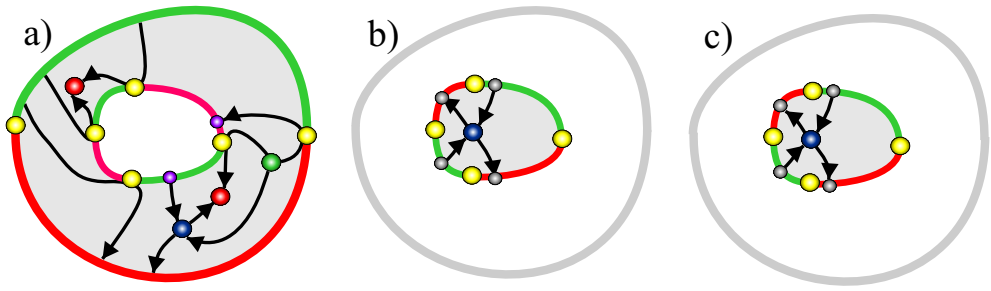


Figure 5.3: a) Example vector field $\mathbf{v}|_{\mathbf{D}''} = \mathbf{w}|_{\mathbf{D}''}$ (grey area) and its topological skeleton. b) $\mathbf{v}|_{\mathbf{D}'}$ and its topological skeleton: $\mathbf{P}_{\mathbf{F}'}(\mathbf{v}|_{\mathbf{D}'})$ consists of the marked points on the boundary of the grey area. c) $\mathbf{w}|_{\mathbf{D}'}$ and its topological skeleton.

$\mathbf{w}|_{\mathbf{D}'}$, and compute $\mathbf{P}_{\mathbf{F}'}(\mathbf{v}|_{\mathbf{D}'})$ and $\mathbf{P}_{\mathbf{F}'}(\mathbf{w}|_{\mathbf{D}'})$. Figures 5.3b and 5.3c illustrate this. Condition 2 of Theorem 5.1 ensures that there is a one-to-one correspondence between $\mathbf{P}_{\mathbf{F}'}(\mathbf{v}|_{\mathbf{D}'})$ and $\mathbf{P}_{\mathbf{F}'}(\mathbf{w}|_{\mathbf{D}'})$. Also, we have

$$\begin{aligned} \mathbf{P}_{\mathbf{F}'}(\mathbf{v}|_{\mathbf{D}''}) \cup \mathbf{P}_{\mathbf{F}'}(\mathbf{v}|_{\mathbf{D}'}) &\subseteq \mathbf{Q}_{\mathbf{F}'}(\mathbf{v}), \\ \mathbf{P}_{\mathbf{F}'}(\mathbf{w}|_{\mathbf{D}''}) \cup \mathbf{P}_{\mathbf{F}'}(\mathbf{w}|_{\mathbf{D}'}) &\subseteq \mathbf{Q}_{\mathbf{F}'}(\mathbf{w}). \end{aligned}$$

This and condition 3 of Theorem 5.1 ensure that corresponding points of $\mathbf{P}_{\mathbf{F}'}(\mathbf{v}|_{\mathbf{D}''}) \cup \mathbf{P}_{\mathbf{F}'}(\mathbf{v}|_{\mathbf{D}'})$ and $\mathbf{P}_{\mathbf{F}'}(\mathbf{w}|_{\mathbf{D}''}) \cup \mathbf{P}_{\mathbf{F}'}(\mathbf{w}|_{\mathbf{D}'})$ are in the same order on $\mathbf{F}'$.

The joint of the topological skeletons of $\mathbf{v}|_{\mathbf{D}'}$ and $\mathbf{v}|_{\mathbf{D}''}$ does not yield the topological skeleton of $\mathbf{v}$ yet. In fact, all separatrices which intersect $\mathbf{F}'$ end there. Figure 5.4 illustrates this. In order to complete the topological skeleton of $\mathbf{v}$, we have to do the following constructions:

A. continue the integration of all separatrices of $\mathbf{v}|_{\mathbf{D}'}$ if they leave $\mathbf{D}'$ in a point on $\mathbf{F}'$,

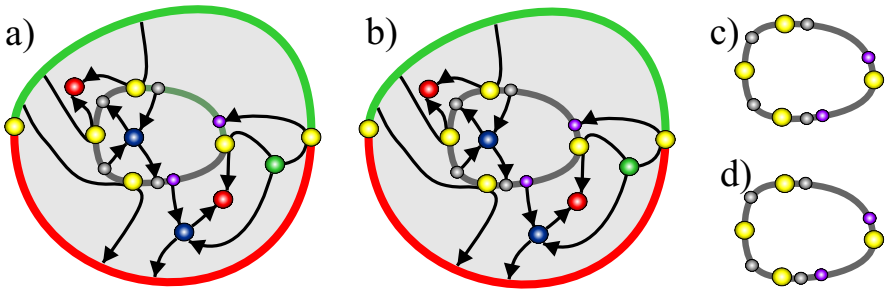


Figure 5.4: a) Joining the topological skeletons of $\mathbf{v}|_{\mathbf{D}'}$ and $\mathbf{v}|_{\mathbf{D}''}$ does not yield the topological skeleton of $\mathbf{v}$, since the separatrices of $\mathbf{v}|_{\mathbf{D}'}$ and $\mathbf{v}|_{\mathbf{D}''}$ end when crossing $\mathbf{F}'$. b) Joining the topology of $\mathbf{w}|_{\mathbf{D}'}$ and $\mathbf{w}|_{\mathbf{D}''}$; c) $\mathbf{P}_{\mathbf{F}'}(\mathbf{v}|_{\mathbf{D}''}) \cup \mathbf{P}_{\mathbf{F}'}(\mathbf{v}|_{\mathbf{D}'})$ consists of the marked points on $\mathbf{F}'$. d) $\mathbf{P}_{\mathbf{F}'}(\mathbf{w}|_{\mathbf{D}''}) \cup \mathbf{P}_{\mathbf{F}'}(\mathbf{w}|_{\mathbf{D}'})$.

B. continue the integration of all separatrices of $\mathbf{v}|_{\mathbf{D}''}$ if they leave $\mathbf{D}''$ in a point on $\mathbf{F}'$.

(The similar statement holds for the vector field $\mathbf{w}$.) Concerning A mentioned above, let $\mathbf{S}_1$ be a separatrix in $\mathbf{v}|_{\mathbf{D}'}$, and let $\mathbf{S}_2$ be the corresponding separatrix in $\mathbf{w}|_{\mathbf{D}'}$. This means that $\mathbf{S}_1$ and $\mathbf{S}_2$ start from the same point in the same integration direction. Let $\mathbf{S}_1$ intersect $\mathbf{F}'$ in the point $\mathbf{x}_1$, and let $\mathbf{S}_2$ intersect $\mathbf{F}'$ in $\mathbf{x}_2$. In general, $\mathbf{x}_1$ and $\mathbf{x}_2$ differ. Nevertheless, from condition 3 of Theorem 5.1 we know that $\mathbf{x}_1$ and $\mathbf{x}_2$ are located between the same adjacent points of $\mathbf{P}_{\mathbf{F}'}(\mathbf{v}|_{\mathbf{D}''}) = \mathbf{P}_{\mathbf{F}'}(\mathbf{w}|_{\mathbf{D}''})$. This means that $\mathbf{x}_1$ and $\mathbf{x}_2$ are not separated by a separatrix in $\mathbf{v}|_{\mathbf{D}''} = \mathbf{w}|_{\mathbf{D}''}$. Hence the integration of the stream lines in $\mathbf{D}''$ starting from $\mathbf{x}_1$ and $\mathbf{x}_2$ ends in the same critical point or boundary inflow/outflow region: the separatrices $\mathbf{S}_1$ and $\mathbf{S}_2$ in the whole domain $\mathbf{D}$ are corresponding.

Concerning B mentioned above, let $\mathbf{S}_1$ be a stream line in $\mathbf{v}|_{\mathbf{D}''}$, let $\mathbf{S}_2 = \mathbf{S}_1$ be the corresponding stream line in $\mathbf{w}|_{\mathbf{D}''}$, and let $\mathbf{x} \in \mathbf{P}_{\mathbf{F}'}(\mathbf{v}|_{\mathbf{D}'})$ be the intersection of $\mathbf{S}_1$ and $\mathbf{S}_2$ with $\mathbf{F}'$. We integrate $\mathbf{S}_1$ in $\mathbf{D}'$ from $\mathbf{x}$ until $\mathbf{S}_1$ leaves $\mathbf{D}'$ in a point $\mathbf{x}_1 \in \mathbf{F}'$. Also, we integrate $\mathbf{S}_2$ in $\mathbf{D}'$ from $\mathbf{x}$ until $\mathbf{S}_2$ leaves $\mathbf{D}'$ in a point $\mathbf{x}_2 \in \mathbf{F}'$. Figure 5.5 illustrates this. Since $\mathbf{x}_1 \in \mathbf{Q}_{\mathbf{F}'}(\mathbf{v})$, $\mathbf{x}_2 \in \mathbf{Q}_{\mathbf{F}'}(\mathbf{w})$, and condition 3 of Theorem 5.1, $\mathbf{x}_1$ and $\mathbf{x}_2$ are located between the same adjacent points of $\mathbf{P}_{\mathbf{F}'}(\mathbf{v}|_{\mathbf{D}''}) = \mathbf{P}_{\mathbf{F}'}(\mathbf{w}|_{\mathbf{D}''})$ on $\mathbf{F}'$. Then the same argumentation as in A gives that $\mathbf{S}_1$ and $\mathbf{S}_2$ are corresponding in the whole domain $\mathbf{D}$, which proves Theorem 5.1.

5.2.4 Extensions of the Topology Concept

In Section 5.2.1 we made a number of simplifying assumptions about the considered vector fields in order to keep the proof of Theorem 5.1 simple. Although many practical vector fields (including our examples in the next section) fulfill

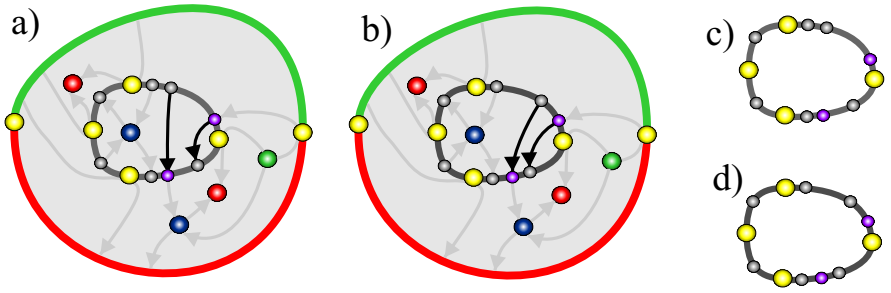


Figure 5.5: a) Continue integrating separatrices of $\mathbf{v}|_{\mathbf{D}''}$ if they enter $\mathbf{D}'$. b) Continue integrating separatrices of $\mathbf{w}|_{\mathbf{D}''}$ if they enter $\mathbf{D}'$. c) $\mathbf{Q}_{\mathbf{F}'}(\mathbf{v})$ consists of the marked points. d) $\mathbf{Q}_{\mathbf{F}'}(\mathbf{w})$ consists of the marked points.

these assumptions, there are vector fields with different topological structures. In this section we discuss the applicability of Theorem 5.1 to these vector fields:

- Zero flow boundaries:
  Zero flow boundaries are present if, for instance, the flow around certain solids is simulated. Instead of boundary switch points, so called attachment and detachment points divide the boundary curve there. The proof of Theorem 5.1 was only based on the fact that the topological skeleton provides a complete partition into areas of similar flow behavior. If this condition is fulfilled by the additional consideration of attachment and detachment points, Theorem 5.1 can also be applied for vector fields with zero-flow boundaries.

- Closed separatrices:
  Vector fields may have additional closed stream lines as separatrices. These separatrices do not have a unique starting point. Moreover, condition 1 of Theorem 5.1 is not fulfilled if such a separatrix intersects $\mathbf{D}'$. Hence the conditions of Theorem 5.1 are not fulfilled if closed separatrices enter $\mathbf{D}'$.

- Higher order critical points:
  If higher order critical points are present, the regions of similar flow behavior around them are separated by a number of separatrices. Since these separatrices have a unique starting point, Theorem 5.1 can be applied.

5.3 Compressing the Vector Field

In this section we apply the results of section 5.2 to build an algorithm for topology preserving compression of 2D vector fields. The algorithm works on a piecewise linear original vector field. This means, given is a triangulation of the domain

with velocity information in every vertex. This way the vector field can be considered as a triangular mesh; the problem of compressing the vector data set is thus converted to a mesh decimation problem.

Techniques for solving this simplification problem have been studied extensively during the last decade, see e.g. [Gotsman et al. 2002] for a recent and comprehensive survey. We choose the so called *half-edge collapse* as basic removal operator. It collapses a vertex $\mathbf{p}_0$ into its neighbor $\mathbf{p}_1$ along the directed edge $(\mathbf{p}_0, \mathbf{p}_1)$ (see Figures 5.7c and 5.7d for an example). Before a half-edge collapse is applied, we have to make sure that it does not change the topology of the vector field. We describe the algorithm for doing so in Section 5.3.2. Section 5.3.1 describes the necessary data structures for the algorithm. Section 5.3.3 describes the whole compression algorithm.

5.3.1 The Data Structure

As a preprocess of the algorithm, we have to extract the topology of the original piecewise linear vector field and store it in an appropriate data structure. This data structure is essentially a half-edge data structure representing a triangular mesh with vector information at each vertex. In addition, each triangle has information about present critical points, boundary switch points and separatrices. If a critical point appears inside the triangle, its location and classification are stored with the triangle. If there is a boundary switch point in a boundary triangle, its location is stored with the triangle. Starting from the boundary switch points and saddle points, the separatrices are integrated over the vector field. This way a separatrix usually passes through a number of triangles. For each of these triangles the following items are stored:

- ID of the separatrix
- entry point into the triangle (or starting point of the integration if the separatrix origins inside the triangle) in integration direction
- exit point out of the triangle (or critical point inside the triangle where the separatrix ends) in integration direction

Note that the integration direction is not necessarily the flow direction of the vector field. Instead, the integration direction is always “away from” its originating saddle point or boundary switch point.

5.3.2 Controlled Half-Edge Collapse

Based on the mesh data structure described above, we can apply the results of Theorem 5.1 to locally check whether a half-edge collapse changes the topology

of the vector field. Let $\mathbf{p}_0, \mathbf{p}_1$ be two vertices which are connected by an edge, let $\mathbf{p}_1, ..., \mathbf{p}_n$ be the 1-ring around $\mathbf{p}_0$, and let $\mathbf{t}_1, ..., \mathbf{t}_n$ be the (counterclockwise ordered) triangles around $\mathbf{p}_0$ [1]. Figure 5.6a illustrates this setting. A half-edge

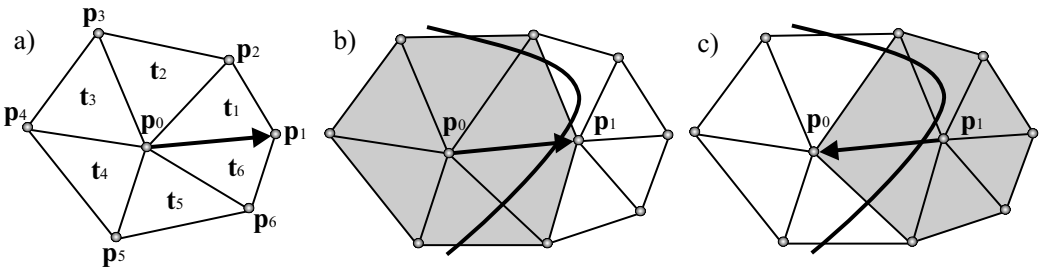


Figure 5.6: a) A half-edge collapse $\mathbf{p}_0 \rightarrow \mathbf{p}_1$ only affects the vector field inside the triangles $\mathbf{t}_1, ..., \mathbf{t}_n$. b) Example of an invalid half-edge collapse $\mathbf{p}_0 \rightarrow \mathbf{p}_1$ because one separatrix enters $\mathbf{D}'$ (marked grey) twice. c) For the same configuration as b), the half-edge collapse $\mathbf{p}_1 \rightarrow \mathbf{p}_0$ may be allowed because the separatrix enters $\mathbf{D}'$ (marked grey) only once.

collapse $\mathbf{p}_0 \rightarrow \mathbf{p}_1$ only affects the vector field inside the triangles $\mathbf{t}_1, ..., \mathbf{t}_n$. This is the area $\mathbf{D}'$ of Theorem 5.1 in which local modifications of the vector field take place.

Now we can describe the algorithm to check whether a half-edge collapse changes the topology of a vector field:

Algorithm 5.1 *(controlled half-edge collapse)*

1. *Check if there are critical points inside* $\mathbf{D}' = (\mathbf{t}_1, ..., \mathbf{t}_n)$.
   If so, stop and prohibit *the half-edge collapse.*
2. *Collect all separatrices which pass through* $\mathbf{D}'$. *For each separatrix, store entry point and exit point of* $\mathbf{D}'$ *in a cyclic list* L_1 *which is ordered according to the order of the line segments of the closed polygon* $((\mathbf{p}_1, \mathbf{p}_2), ..., (\mathbf{p}_{n-1}, \mathbf{p}_n), (\mathbf{p}_n, \mathbf{p}_1))$.
3. *If a separatrix enters* $\mathbf{D}'$ *more than once,* stop and prohibit *the half-edge collapse.*
4. *Compute the boundary switch points of the vector field on the polygon* $((\mathbf{p}_1, \mathbf{p}_2), ..., (\mathbf{p}_{n-1}, \mathbf{p}_n), (\mathbf{p}_n, \mathbf{p}_1))$*, insert these points to* L_1.
5. *Simulate the half-edge collapse* $\mathbf{p}_0 \rightarrow \mathbf{p}_1$ *while storing the original configuration (to allow an undo of the half-edge collapse).*

[1] In order to avoid confusion with the vector field $\mathbf{v}$, we denote vertices by $\mathbf{p}$, here.

6. *Apply linear interpolation of the vector field inside the new triangles* $(\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3), (\mathbf{p}_1, \mathbf{p}_3, \mathbf{p}_4), ..., (\mathbf{p}_1, \mathbf{p}_{n-1}, \mathbf{p}_n)$).
Check whether there are critical points inside one of the new triangles. If so, stop and prohibit *half-edge collapse.*

7. *Construct a new cyclic ordered list* L_2 *of points on the polygon* $((\mathbf{p}_1, \mathbf{p}_2), ..., (\mathbf{p}_{n-1}, \mathbf{p}_n), (\mathbf{p}_n, \mathbf{p}_1))$ *consisting of the following points:*

   (a) *all boundary switch points of step 4 of the algorithm*

   (b) *the entry points of all separatrices to* $\mathbf{D}'$

   (c) *integrate the stream lines starting from all points of step 7b of this algorithm inside* $\mathbf{D}'$ *until they reach the boundary again; store the exit points in* L_2.

8. *Undo simulated half-edge collapse* $\mathbf{p}_0 \rightarrow \mathbf{p}_1$

9. *Compare the cyclic order of the points in* L_1 *and* L_2.
If the corresponding points do not have the same cyclic order in L_1*and* L_2*, stop and prohibit the half-edge collapse.*

10. Stop and allow the half-edge collapse .

Figure 5.7 illustrates this algorithm where an edge collapse is allowed. Figure 5.8 shows an example where the algorithm prohibits an edge collapse.
Algorithm 5.1 needs some remarks:

- The collected points in the lists L_1 and L_2 correspond to $\mathbf{Q}_{\mathbf{F}'}(\mathbf{v})$ and $\mathbf{Q}_{\mathbf{F}'}(\mathbf{w})$ of Theorem 5.1. Moreover, steps 1 and 6 of algorithm 5.1 ensure condition 2 of Theorem 5.1, and step 2 ensures condition 1 of Theorem 5.1. Hence Theorem 5.1 proves the correctness of algorithm 5.1.

- The entire algorithm works locally on the 1-ring around $\mathbf{p}_0$.

- If an edge collapse is impossible because of a re-entry of a separatrix, an edge collapse of an adjacent edge (or of the opposite half-edge) might still be possible. Figures 5.6b and 5.6c show an example.

5.3.3 The Compression Algorithm

The vector field compression is achieved by applying a mesh reduction to the triangulated domain of the piecewise linear vector field. A standard algorithm is adapted to this specific problem. Its basic topological operator is the *controlled* half-edge collapse from Section 5.3.2. The mesh reduction algorithm can now be sketched as follows:

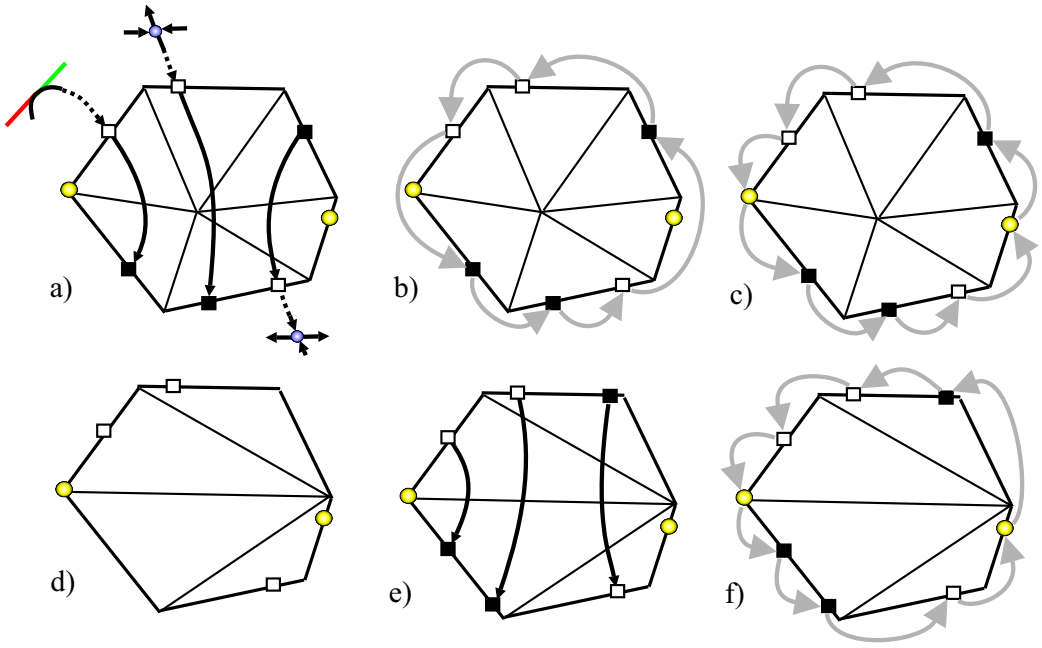


Figure 5.7: Example of algorithm 5.1. a) Three separatrices passing through $\mathbf{D}'$, and two boundary switch points (yellow) are present. The empty boxes are the entry points of the separatrices into $\mathbf{D}'$ (in integration direction), the solid boxes describe the exit points. b) Cyclic list L_1 (grey arrows) after step 2. c) L_1 after step 4. d) Collecting points of new list L_2 after half-edge collapse: after step 7a and 7b. e) Integrate new stream lines (7c). f) Cyclic list L_2 after step 7c. The edge collapse is allowed, since the corresponding points in L_1 and L_2 (shown in c) and f)) are in the same order.

Algorithm 5.2 *(Topology preserving vector field compression)*
Repeat...

1. Initialization. *For all directed edges* $(\mathbf{p}_i, \mathbf{p}_j)$*:*
   - *(a) Apply* Algorithm 5.1 *to check whether half-edge collapse is allowed.*
   - *(b) If allowed: evaluate priority and put* $(\mathbf{p}_0, \mathbf{p}_1)$ *into priority queue* **PQ**.
2. Iterative removal. **While PQ** *not empty*
   - *(a) Get and remove* $(\mathbf{p}_i, \mathbf{p}_j)$ *from* **PQ**.
   - *(b) If half-edge collapse* $(\mathbf{p}_i, \mathbf{p}_j)$ *allowed: apply half-edge collapse.*
   - *(c) Update topology data structure.*
   - *(d) Re-apply* Algorithm 5.1 *to all edges incident to* $\mathbf{p}_j$ *and to* $\mathbf{p}_j$*'s 1-ring, and update* **PQ** *accordingly.*

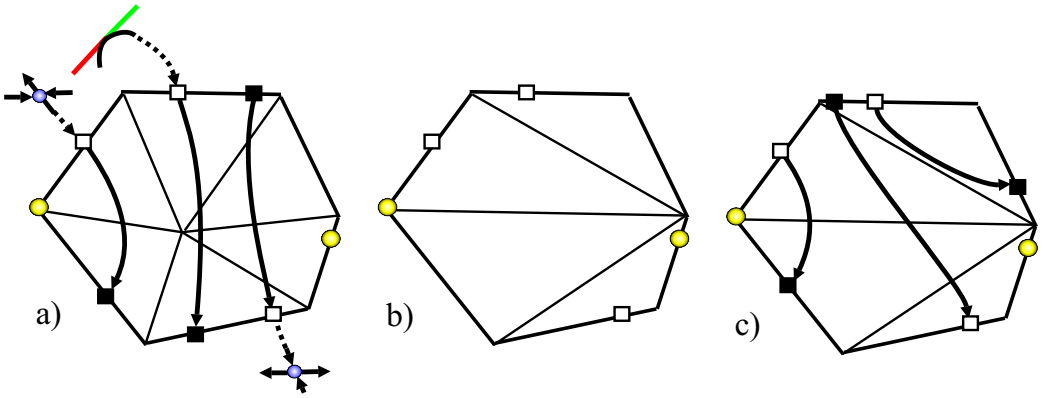


Figure 5.8: Another example of algorithm 5.1. a) Three separatrices passing through $\mathbf{D}'$, and two boundary switch points (yellow) are present: L_1 consists of the marked points on the boundary. b) Points of L_2 after step 7b. c) Points of L_2 after step 7c. The edge collapse is not allowed, since the corresponding points in L_1 and L_2 (shown in a) and c)) are in different order.

. . . **until** *no more collapses possible.*

Here, the inner loop reflects the standard mesh reduction algorithm. The outer loop that causes repeated reinitialization reflects the fact that local changes may have global impact and may thus allow collapses that have been prohibited before. The update to the topology data structure (step (2c) of algorithm 5.2) is the most expensive part of the algorithm. If a half-edge collapse is carried out, the topology data structure (described in Section 5.3.1) has to be updated: all separatrices which pass through the 1-ring around $\mathbf{p}_0$ have to be reintegrated from their entrance point into the 1-ring.

The whole process is a greedy optimization driven by a priority queue. In a 3D setup the priority of a collapse would be some kind of quality measure as e.g. distance to the original surface. In the 2D case we are left with an additional degree of freedom. A natural choice would be to locally apply some difference measure for flow fields (see e.g. [Garcke et al. 2000], [Heckel et al. 1999]). We do not consider advanced quality measures for the following. Instead, we apply a simple heuristic to gradually coarsen the mesh and merely assign priorities proportional to edge lengths, preferring short edges for collapse.

5.4 Modifications of the Topology Preserving Compression Algorithm

We provide two modifications to algorithm 5.2 in order to better evaluate the topology preserving compression (cf. [Theisel et al. 2004a]). Both modifications affect the notion of the equivalence of topological skeletons (see Section 5.2.2, Definition 5.2). While the first modification applies a stronger concept of topological equivalence, the second modification relaxes the original concept:

The original concept of topological equivalence requires that the skeletons have the same critical points (both location and Jacobian matrices) and that the corresponding separatrices end in the same critical points or inflow/outflow regions.

We define the two alternative equivalence concepts as follows:

Equivalence concept 1 *(strong equivalence) Two topological skeletons are equivalent iff both, their critical points and their separatrices are identical.*

Note that this is a rather strong concept: $\mathbf{v}_1$ and $\mathbf{v}_2$ are supposed to have the same critical points (including the Jacobian matrices) and the same separatrices. The original Definition 5.2 relaxes this concept 1 by allowing that corresponding separatrices have different paths (as long as they end in the same critical point or inflow/outflow region). In the following concept 2, we further relax this condition and also allow the critical points to move, as long as they do not merge or change their classification.

Equivalence concept 2 *(relaxed equivalence) The topological skeletons of* $\mathbf{v}_1$ *and* $\mathbf{v}_2$ *are equivalent iff there is a one-to-one map between the critical points of* $\mathbf{v}_1$ *and* $\mathbf{v}_2$*, such that saddles are mapped to saddles, sources to sources, and sinks to sinks, and corresponding separatrices of* $\mathbf{v}_1$ *and* $\mathbf{v}_2$ *end in corresponding critical points or inflow/outflow regions.*

Based on these two alternative concepts of topological equivalence of vector fields, we construct two new compression algorithms. In both cases, the algorithm framework remains the same as for the original algorithm 5.2, because exchanging the equivalence concept only affects steps 1a and 2d. This is in fact the core of the compression method, namely the algorithm which decides whether a particular half-edge collapse changes the topology of the whole vector field. So instead of applying the controlled half-edge collapse according to algorithm 5.1, we provide the modifications below.

A compression algorithm which preserves equivalence concept 1 can easily be formulated as

Algorithm 5.3 *(controlled half-edge collapse, concept 1)*

1. *If one of the triangles* $\mathbf{t}_1, ..., \mathbf{t}_n$ *contains a critical point or a part of a separatrix,* stop and prohibit *the half-edge collapse.*
2. *Simulate the half-edge collapse* $\mathbf{p}_0 \rightarrow \mathbf{p}_1$.
3. *If one of the new triangles contains a critical point,* stop and prohibit *the half-edge collapse.*
4. Stop and allow the half-edge collapse.

This algorithm is justified in the fact that any local modification will change the location of a critical point or a separatrix. Hence, only in regions without any topological features, a half-edge collapse can be allowed.

Now we want to modify algorithm 5.1 to handle equivalence concept 2. To do so, we have to compare the critical points in $\mathbf{D}'$ before and after the half-edge collapse to check if some of the critical points collapsed. We get the following

Algorithm 5.4 *(controlled half-edge collapse, concept 2)*

1. *Extract and store the critical points inside* $\mathbf{D}' = (\mathbf{t}_1, ..., \mathbf{t}_n)$. *If there is more that one saddle, or if there is more than one source/sink, then* stop and prohibit *the half-edge collapse.*
2. *– 5.* as in algorithm 5.1
6. *Apply linear interpolation of the vector field inside the new triangles* $(\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3), (\mathbf{p}_1, \mathbf{p}_3, \mathbf{p}_4), ..., (\mathbf{p}_1, \mathbf{p}_{n-1}, \mathbf{p}_n))$. *Check the new triangles for critical points.*

   If the number of saddles does not coincide with step 1, or if the number of sources/sinks does not coincide with 1, stop and prohibit *the half-edge collapse.*

   If there is one saddle, integrate its four separatrices until they leave $\mathbf{D}'$. *Store the four exit points into a new cyclic list* L_2 *of points on the polygon* $((\mathbf{p}_1, \mathbf{p}_2), \ldots, (\mathbf{p}_{n-1}, \mathbf{p}_n), (\mathbf{p}_n, \mathbf{p}_1))$.
7. *Insert the following points into* $L_2 \ldots$ — substeps as in algorithm 5.1
8. *– 10.* as in algorithm 5.1

Figures 5.9a and 5.9b illustrate an example of algorithm 5.4 where the half-edge collapse is not allowed. Figures 5.9c — 5.9e show an example with an allowed half-edge collapse.

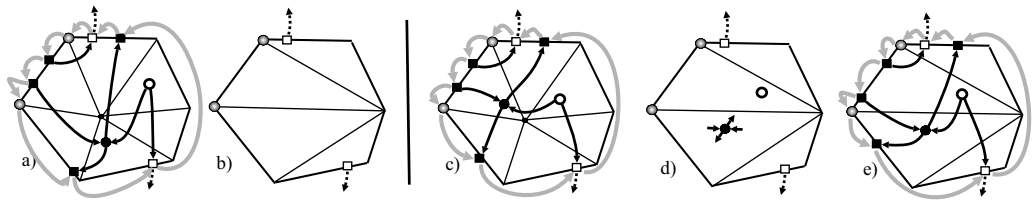


Figure 5.9: Controlled half-edge collapse for equivalence concept 2. a) 1-ring containing one saddle (solid circle) and one source (hollow circle). Three of the four separatrices created by the saddle leave the 1-ring while one ends in the source. In addition, two separatrices enter the region from outside (hollow boxes): one ends in the source, the other leaves the region. b) Simulated half-edge collapse removes the critical points: half-edge collapse is not allowed. c) Another example of a 1-ring containing one saddle (solid circle) and one source (hollow circle). d) Simulated half-edge collapse gives two new critical points: one saddle and one sink. e) Cyclic list L_2 after step 7 of algorithm 5.2. e) The half-edge collapse is allowed.

5.5 Topological Simplification and Topology Preserving Compression

In this section we combine topology simplification and topology preserving compression of 2-dimensional vector fields [Theisel et al. 2003b]. Although these two techniques are somehow opposite approaches which had been developed independently of each other, for real life data sets both problems appear simultaneously. In fact, a data set may have unimportant topological features due to noise, but has to be compressed under preservation of the important topological structures.

This motivates the development of a compression technique which preserves important topological structures but removes the unimportant ones. The basic algorithm is as follows:

Algorithm 5.5 *(combining topological simplification and topology preservation)*

1. *Extract the complete topological skeleton (i.e., all critical points, boundary switch points and separatrices) of the original vector field.*

2. *Assign a weight $w \in [0,1]$ to every critical point and every separatrix. This weight describes the importance of the critical point or the separatrix: the higher the weight, the more important the feature is.*

3. *Pick a threshold $w_0 \in [0,1]$ which makes the distinction between important ($w \geq w_0$) and unimportant features ($w < w_0$).*

4. *Apply a compression of the vector field which ensures the preservation of the important topological features.*

As a result of this algorithm, we obtain a compressed version of the original vector field in which the important topological features are preserved. Step 1 is straightforward, step 3 is subject of interaction to steer the process, and step 4 essentially applies algorithm 5.2 (with a slight modification). In the following, steps 2 and 4 will be discussed in detail.

5.5.1 Creating a system of importance weights

The problem of creating a system of importance weights for the topological features (step 2 of algorithm 5.5) is strongly related to the problem of topological simplification of vector fields: in both approaches important features have to be depicted. However, we need to compute the importance of both critical points and separatrices. Since all pre-existing topology simplification algorithms only consider critical points, we have to introduce a new algorithm.

The main problem to provide a system of weights is to make it topologically consistent for every threshold w_0. This means, for every $w_0 \in [0, 1]$ the subskeleton consisting of all critical points and separatrices with a weight larger or equal w_0 must describe a valid topological structure. Thus, the following conditions have to be fulfilled:

- Fulfill the index theorem for the whole vector field: the sum of the indices of all important critical points must be constant and independent of w_0.
- Consistency of separatrices: every separatrix must start in a saddle or boundary switch point, every separatrix must end in a source/sink or leave the domain of the vector field, from every saddle exactly four separatrices and emanating.

In order to fulfill the first condition, we first group the critical points to pairs such that each pair consists of a saddle and a non-saddle (i.e., source or sink). Then both points of a pair are assigned the same weight. The next Section 5.5.2 discusses this step in detail. In order to fulfill the second condition, a system of weights for the separatrices has to be found as well as the initial weights of the pairs of critical points have to be updated. Section 5.5.3 presents the details of this step.

5.5.2 Coupling Critical Points and Finding Initial Weights

Several approaches to couple critical points are reported in the literature. De Leeuw and von Liere [1999b] use the Euclidean distance of the critical points

to couple them, Tricoche et al. [2001] demand that critical points building a pair have a common separatrix. However, these coupling strategies do not always yield unique solutions especially in areas containing many critical points. Moreover, they do not provide an importance weight of a critical point. Because of this, we propose an alternative approach which is based on the concept of feature flow fields [Theisel and Seidel 2003] which originally had been developed to track critical points in time-dependent vector fields. Given a 2D vector field $\mathbf{v}$, we construct a vector field $\mathbf{v}_s$ by applying a very strong Laplacian smoothing which only keeps the boundary of the domain of $\mathbf{v}$ unchanged. This operator can be expected to smooth out many topological features of $\mathbf{v}$. In fact, we expect $\mathbf{v}_s$ to have significantly less critical points than $\mathbf{v}$. Now we consider the time-dependent vector field $\mathbf{v}(\mathbf{x},t) = (1-t)\mathbf{v} + t\mathbf{v}_s$ in the time interval $0 \leq t \leq 1$. In order to track the behavior of the critical points of $\mathbf{v}$ over the time, we construct a 3D vector field $\mathbf{f}$ in such a way that the paths of the critical points of $\mathbf{v}$ over time correspond to stream lines of $\mathbf{f}$. In [Theisel and Seidel 2003] it is shown that

$$\mathbf{f}(\mathbf{x},t) = \begin{pmatrix} \det(\mathbf{v}_{x_2}, \mathbf{v}_t) \\ \det(\mathbf{v}_t, \mathbf{v}_{x_1}) \\ \det(\mathbf{v}_{x_1}, \mathbf{v}_{x_2}) \end{pmatrix}. \tag{5.3}$$

To check if a critical point $\mathbf{x}_0$ in $\mathbf{v}$ has a partner critical point, we simply integrate the stream line of $\mathbf{f}$ starting from $(\mathbf{x}_0, 0)$. Doing so, the integration direction has to be chosen such that the stream line initially moves inside the time slab $[0,1]$. Since $\mathbf{v}$ and $\mathbf{v}_s$ coincide in the boundaries of their domains, the stream line of $\mathbf{f}$ leaves the valid domain either in a point $(\mathbf{x}_1, 0)$ or in a point $(\mathbf{x}_1, 1)$. In the first case it can be shown ([Theisel and Seidel 2003]) that $\mathbf{x}_1$ is a critical point of $\mathbf{v}$ with an index opposite to $\mathbf{x}_0$: $\mathbf{x}_0$ and $\mathbf{x}_1$ become a couple, and their common weight w is determined by the maximal t-value of the stream line of $\mathbf{f}$ between $(\mathbf{x}_0, 0)$ and $(\mathbf{x}_1, 0)$. If the stream line of $\mathbf{f}$ ends in a point $(\mathbf{x}_1, 1)$, $\mathbf{x}_0$ does not have a partner critical point in $\mathbf{v}$, it gets the weight 1. Figure 5.10 gives an illustration.

5.5.3 Making the weights consistent

Given the partner relation and the initial weights of the critical points from the algorithm described above, we now have to adjust these weights and find weights for the separatrices in such a way that this system is consistent for every threshold w_0. This means, the following conditions have to be fulfilled:

- Two critical points which are partners must have the same weight. This ensures the index theorem of the vector field for every w_0.
- The four separatrices starting from a saddle point must have the same weight as their creating saddle.

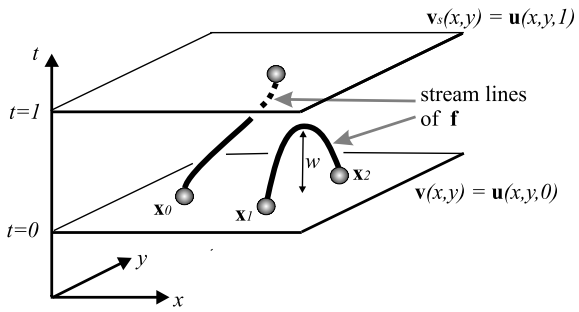


Figure 5.10: Three critical points $\mathbf{x}_0$, $\mathbf{x}_1$, $\mathbf{x}_2$ of $\mathbf{v}$ and their stream lines of the feature flow field $\mathbf{f}$. $\mathbf{x}_0$ gets the weight 1 since it does not have a partner critical point in $\mathbf{v}$. $\mathbf{x}_1$ and $\mathbf{x}_2$ are detected to be partners and are assigned with the common weight w.

- If a separatrix ends in a source or a sink, the weight of this source/sink must not be smaller than the weight of the separatrix. This ensures that if a separatrix is considered to be important, its ending source/sink is important as well.

We start with the following initial weights: every critical point is assigned the weight from the previous Section 5.5.2, every separatrix starting from a boundary switch point gets a weight of 1, and every separatrix starting from a saddle gets a weight of 0. Then we iteratively correct the weights which contradict to the conditions above:

Algorithm 5.6 *(consistent importance weights)*

Iterate until the above conditions re fulfilled:

1. *If two partner critical points have a different weight, they are set to the maximum of both weights.*

2. *If a separatrix starting from a saddle has a smaller weight than the saddle, its weight is set to the weight of the saddle.*

3. *If a separatrix ends in a source/sink and the weight of the source/sink is smaller than the weight of the separatrix, the weight of the source/sink is set to the weight of the separatrix.*

Obviously, the termination of this algorithm is ensured: in the worst case the algorithm stops when all weights reach the value 1. Section 5.6.3 shows results of applying algorithm 5.6.

5.5.4 The compression algorithm

To compress the vector field, we use an adapted version of algorithm 5.2. The algorithm needs some modifications, since now we distinguish between important and unimportant critical points and separatrices: Algorithm 5.2 is designed to not allow any changes of critical points. Here, this condition is relaxed such that only the presence of an important critical point (i.e. a critical point with a weight above the threshold) prohibits the collapse. If unimportant critical points are inside a 1-ring, they are allowed to move, change and even collapse with other unimportant critical points. —- In fact, a collapse of unimportant critical points is even desired, since it increases the chances to find more allowed half-edge collapses.

If a 1-ring contains unimportant critical points and they do not disappear during the simulated half-edge collapse, the test is very similar to algorithm 5.1. If a saddle point is present, its new location after the half-edge collapse has to be extracted, and the new separatrices starting from it have to be integrated until they end in a source/sink or leave the 1-ring. Figure 5.11 illustrates this situation.

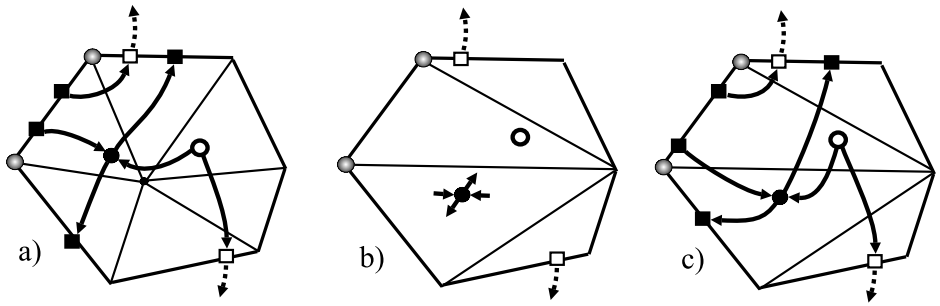


Figure 5.11: a) 1-ring containing an unimportant saddle (solid circle) and an unimportant source (hollow circle). Three of the four separatrices created by the saddle leave the 1-ring while one ends in the source. In addition, two separatrices enter the region from outside (hollow boxes): one ends in the source, the other one leaves the region. b) Simulated half-edge collapse under preservation of the entry points of the separatrices and extracting the new locations of the inner critical points. c) Integrating the separatrices gives a similar cyclic order of the corresponding points like in a): the half-edge collapse does not change the topology.

If the half-edge collapse leads to a collapse and disappearing of a saddle and a non-saddle (both unimportant critical points), this test applied in algorithm 5.2 has to be modified such that the separatrices coming from the saddle are not considered for the test. Figure 5.12 gives an illustration.

Note that, although we could distinguish between important and unimportant critical points in the algorithm, this distinction cannot be done for separatrices. In fact, all separatrices passing through a 1-ring have to be equally treated as impor-

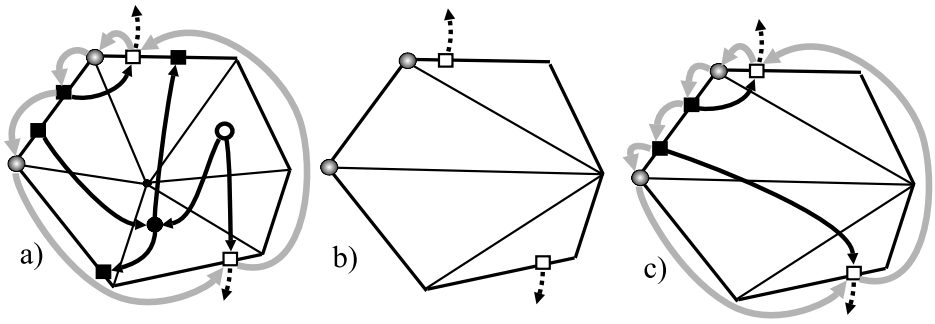


Figure 5.12: a) 1-ring containing two unimportant critical points. b) if a half-edge collapse makes the critical points disappear, the separatrices starting from the saddle point are not considered for comparing the cyclic order (grey arrows) in a). c) reintegrating the remaining separatrices and comparing the cyclic order with a): here the half-edge collapse changes the topology because the cyclic orders differ.

tant. This is due to the fact that the theoretical foundation of the algorithm 5.1 is based on the assumption that the complete topological information about the vector field is present for the local analysis. However, by applying half-edge collapses which remove unimportant critical points, unimportant separatrices are removed as well.

So far, the decision which of the allowed half-edge collapses is carried out first was done by a greedy optimization driven by a priority queue where the priority is based on the edge length, preferring short edges to collapse. For algorithm 5.5, we modify the priority function by assigning half-edge collapses which remove unimportant critical points and separatrices with a higher priority. This strategy is based on the observation that the removal of unimportant separatrices increases the chances that half-edge collapses at other locations become possible.

Results are summarized in Section 5.6.3.

5.6 Results

In the following we present results from vector field compression. All tests were performed on an 1.7 GHz Intel Xeon processor for a moderate and a fairly complex data set.

5.6.1 Test Data Sets

We applied our compression algorithm to two test data sets. The first data set *Greifswalder Bodden* describes (the perpendicular of) the flow of a bay area of

data set	triangles	cp	bsp	sep	Figures
1: Greifswalder Bodden	14,086	72	44	168	5.13a,c,e
2: skin friction	12,726	338	34	714	5.14a,c,e; 5.15a,c,e; 5.16a

Table 5.1: Test data sets used for vector field compression. The columns denote the data set and the number of triangles, critical points (cp), boundary switch points (bsp), and separatrices (sep), respectively. The last column refers to Figures showing the original data sets.

the Baltic Sea near Greifswald (Germany). The data set was created by the Department of Mathematics, University of Rostock (Germany). The data was given as an incomplete flow data set on a regular 115×103 grid. Each grid cell is split in two triangles such that a regular triangulation consisting of 14,086 triangles (see Figure 5.13a) is obtained as the domain of the piecewise linear vector field. Figure 5.13e shows the topological skeleton of the vector field while Figure 5.13c shows its LIC image. This flow data set consists of 71 critical points, 44 boundary switch points, and 168 separatrices.

The second test data set *skin friction* describes the skin friction on a face of a cylinder which was obtained by a numerical simulation of a flow around a square cylinder. The data set was generated by R.W.C.P. Verstappen and A.E.P. Veldman of the University of Groningen (the Netherlands) and provided by Wim de Leeuw. The same data set has been analyzed in [de Leeuw and van Liere 1999a] and [Lodha et al. 2000]. The data was given on a rectangular 102×64 grid with varying grid size. After splitting the grid cells uniformly we get a triangular domain consisting of 12,726 triangles (see Figure 5.14a). As we can see in this picture, all triangles there tend to be large and thin. Figure 5.14c shows the LIC image of the vector field while Figure 5.14e shows its topological skeleton. This vector field consists of 338 critical points, 34 boundary switch points, and 714 separatrices. Therefore, it can be considered as a vector field of both large size and complex topology.

Table 5.1 summarizes facts on the test data sets.

5.6.2 Topology Preserving Vector Field Compression

Applying compression algorithm 5.2 to test data set 1 (Greifswalder Bodden), we obtained a new piecewise linear vector field which consists of 660 triangles. Figure 5.13b shows the triangular domain of the compressed vector field. Figure 5.13d shows the LIC image, and Figure 5.13f shows the topological skeleton of the compressed vector field. Note that the topological skeletons of original and compressed vector field (Figures 5.13e and 5.13f) are equivalent concerning Definition 5.2. The compression ratio is 95.3%, and the complete compression

data sets	algorithm 5.2		algorithm 5.3		algorithm 5.4	
	△	ratio	△	ratio	△	ratio
1: Greifswalder Bodden	4,944	64.9%	660	95.3%	374	97.3%
2: skin friction	10,680	16.1%	2,153	83.1%	1,071	91.6%

Table 5.2: Results of topology preserving vector field compression. The table compares the three algorithms, the columns show the number of triangles, the compression ratio, and the run time, respectively.

algorithm took 280 seconds. The algorithms 5.3 and 5.3 reduce the number of triangles to 4,944 (64.9%) and 374 (97.3%), respectively. Figure 5.17 shows the triangulations and the topological skeletons.

After applying our compression to the test data set 2 (skin friction) algorithm, we obtained a vector field with the triangular domain shown in Figure 5.14b. This domain consists of 2,153 triangles which gives a compression ratio of 83.1%. Figure 5.14d shows the LIC image of the compressed vector field, Figure 5.14f shows the topological skeleton. The complete compression algorithm took 299 seconds.

Because of the high complexity of this data set, the visualizations in Figure 5.14 appear to be cluttered. Therefore, we present two magnifications of the skin friction data set in figures 5.15 and 5.16. The larger rectangle in Figure 5.14c denotes the magnified area considered in Figure 5.15. The smaller rectangle of figure 5.14c is magnified in Figure 5.16. In the pictures, the characteristics of the critical points are color coded: blue for a saddle, red for a source, and green for a sink. Note that the topological skeletons of the original (Figure 5.14e) and the compressed vector field (Figure 5.14f) are equivalent, even though separatrices in the compressed flow tend to be very close together.

Figure 5.18 shows the triangulations and the topological skeletons after applying algorithms 5.3 and 5.4. The number of triangles is 10,680 (16.1%) and 1,071 (91.6%), respectively.

We remark — especially the skin friction data set — tends to have many separatrices running very close to each other. This can be explained with the presence of attachment and separation lines (cf. [Kenwright et al. 1999]). For these cases, in practice a half-edge collapse may be forbidden due to numerical instabilities. We solve this technical problem, clustering the entry points of separatrices at the 1-ring of a vertex: entry points of separatrices which are very close to each other[2] are set to the same entry point and thus have the same exit point as well.

Table 5.2 summarizes results for both data sets and the topology preserving compression algorithms. Comparing the results, we observe that the equivalence

[2]We used $\frac{1}{1000}$ of the length of the respective boundary edge in the ring.

concept 1 (and algorithm 5.3) yields very poor compression ratios. This is due to the fact that only the presence of a critical point or a segment of a separatrix in a triangle prevents it from taking part in an edge collapse. The resulting triangulations visualize this restriction: in Figure 5.17 (a) there are rather large flow regions which have been simplified, the dense triangulations follow the skeleton. However, Figure 5.18 (a) shows that this is infeasible for complex data sets, there are hardly any regions which can be simplified. While algorithm 5.3 is the easiest to implement, in general it cannot considered to be useful in practice.

The original algorithm 5.2 guarantees that the topological skeletons of the original and the compressed vector field coincide in the critical points and in the connectivity of the separatrices. It achieves high compression ratios even for topologically complex data sets. We compare these compression ratios to existing topology preserving compression techniques. [Lodha et al. 2000] analyzed the skin friction data set, but uses a significantly less strict definition of topology. In fact, small changes of the Jacobian of critical points are permitted, and separatrices are not considered at all. Even with this less strict topology concept, the compression ratio under topology preservation (37%) is significantly less than achieved with algorithm 5.2. Theisel [2002] uses a topology concept similar to the one used here, i.e., the complete Jacobian matrix of the critical points as well as the connectivity of the separatrices are considered. The compression algorithm presented there can only be applied for data sets with a rather simple topology. For vector fields with a topological complexity as in the examples, very low or even negative compression ratios have to be expected.

We conclude that algorithm 5.2 gives significantly better compression ratios than known existing topology-preserving compression techniques when applied to topologically complex data sets.

A further, significant reduction of the number of triangles by a factor of approximately 50% is provided by algorithm 5.4 (equivalence concept 2). The visualization of the topological skeleton shows that the critical points change their locations, but these changes tend to be limited to the neighborhood of the original critical points.

5.6.3 Combining Topological Simplification and Topology Preserving Compression

We show results of applying algorithm 5.5 for the complex data set 2 (skin friction), first analyzing the assigned importance weights. After finding pairs of critical points and attaching them with weights as described in Section 5.5.2, we analyzed the distribution of the weights on all critical points. The solid curve in Figure 5.19 shows the result of this analysis. It turns out that approximately 22%

of all critical points have a very low weight (between 0 and 0.1) while only approximately 2% of the critical points have a very high weight (between 0.9 and 1). The distribution between these extreme values is approximately linear.

In the next step we initialized the separatrices and made the weights consistent as explained in Section 5.5.3. Then the distribution of the weights of the critical points is shown in the dotted line in Figure 5.19. It turns out that in general the weights are higher than before making them consistent: only 11% of the critical points have a low weight between 0 and 0.1, but 17 % have a high weight between 0.9 and 1. This corresponds to the fact that the algorithm 5.6 actually consists of a number of weight increments until all contradictions are removed.

The dotted line in Figure 5.19 also shows that the distribution between very low and very high weights has a high deviation: 3% of the weights are between 0.6 and 0.7, 24% are between 0.7 and 0.8, and 7% are between 0.8 and 0.9. This shows that the algorithm 5.6 tends to build clusters of critical points with identical weights.

The slash-dotted line in Figure 5.19 shows the distribution of the weights of the separatrices after making the weights consistent. This distribution corresponds approximately to the distribution of the critical points after making them consistent. This holds because a separatrix and its creating saddle correspond in their weights. Only for very high weights the percentage of separatrices is bigger than the percentage of critical points, because the weights of the separatrices starting from boundary switch points have been initialized to 1.

Figure 5.20 shows the important topological features of the data set for different thresholds w_0. The upper image ($w_0 = 0$) shows the complete topological skeleton of the data set while the lowest image ($w_0 = 1$) shows only the topological features with a weight of 1. Here it turns out that most of the important features are close to the shorter boundary edges of the domain of the vector field. A reason for this lies in the Laplacian smoothing on which the algorithm is based upon: since this smoothing leaves the boundary of the domain untouched, features in regions close to the boundary tend to get higher weights.

Figure 5.21c shows the result of our algorithm 5.5 for $w_0 = 0$. Here we ended up with 1,805 triangles giving a compression ratio of 85.5%. The computing time was 415 seconds. Since the threshold $w_0 = 0$ means that all critical points and separatrices are considered to be important, the algorithm coincides with the complete topology preserving algorithm 5.2. The difference in the obtained compression ratio is due to fact that in the new version we allow a clustering of the entry points on a 1-ring as described in Section 5.5.4. Figure 5.21d shows the result of our algorithm for $w = 0.5$. Here we obtained 1,367 triangles and a compression ratio of 89.2% at a computing time of 424 seconds. For $w = 1$, we achieved 1,040 triangles (compression ratio of 91.8%) at a computing time of 472 seconds. This is shown in Figure 5.21e.

Figure 5.22 shows the the topological skeletons for the data sets in the same order as in Figure 5.21. Figure 5.22 illustrates the topological skeletons together with an underlying LIC image. The order of the images corresponds to Figures 5.21 and 5.22. Figure 5.23 shows a magnification of the simplified and compressed data sets for $w_0 = 0$, $w_0 = 0.5$ and $w_0 = 1$.

Figures 5.21–5.23 show that for the test data set 2 our approach can significantly decrease the number of obtained triangles (and thus increase the compression ratio) in comparison to algorithm 5.2, if important and unimportant topological features are treated in a different way as explained in section 5.5.4. The algorithm has full control over important features, since they are guaranteed to be preserved. However, the algorithm does not have control over the unimportant features, since they may move and disappear during the compression. In fact, the example has shown that parts of the unimportant features disappear while others only change their location. It would be useful to have a compression algorithm which removes all unimportant topological features, but we have to consider this as an unsolved problem.

5.7 Summary

We presented new algorithms for the compression of piecewise linear vector fields. The particular challenge and the main difference to other work in this field is imposed by the constraint of topology preservation. We provide a sound theoretical framework and show that preservation of this global property can be ensured by applying only local tests. The latter enables the application of adapted mesh decimation techniques for the compression. This is the first method which guarantees topology preservation, and the results confirm the effectiveness and efficiency of the method. In fact, the compression rates are significantly better that for previous work under considerably weaker constraints.

We shows how the initial concept of topological equivalence can be modified, to either stronger or relaxed constraints, and develop alternative algorithms based on these concepts. The experimental results show that the compression rate can be improved significantly, taking into account only the graph structure of the topology, and hence relaxing positional constraints. Alternatively, we studied the simplification of the topology by introducing a system of consistent importance weights. The resulting algorithm can be applied in the compression context as discussed, moreover it is interesting and useful on its own.

So far, the main criterion to rate the results is the size of the obtained data set. Other measures might be useful and interesting for the comparison, especially in the presence of topological simplification. A new approach to the topological comparison of vector fields was developed in [Theisel et al. 2003c]. The proposed

metric is based on the concept of feature flow fields, incorporating both, the characteristics and the distribution of the critical points. It can be evaluated efficiently with reasonable small computation times even for topologically complex vector fields.

We note that although only vector fields over planar triangulations have been considered, this is no general restriction of our algorithm. In fact, the approach can be applied to vector fields on arbitrary triangular domains, i.e. on vector fields defined on surfaces. This requires a modification of the mesh decimation core to additionally respect the geometry of the shape, just as the familiar geometry simplification setting. Here, it would be interesting to study the interplay between vector field compression techniques and mesh simplification techniques.

In summary, we presented new algorithms for the compression of vector valued attributes defined over a triangulation. Before, we have been studying the generation related attributes in the context of discrete curvature approximation and applications. And we have addressed the compression of the triangulations per se, namely connectivity encoding. The topology preserving compression concludes the first part of this work on surface meshes. The second part will discuss volumetric data.

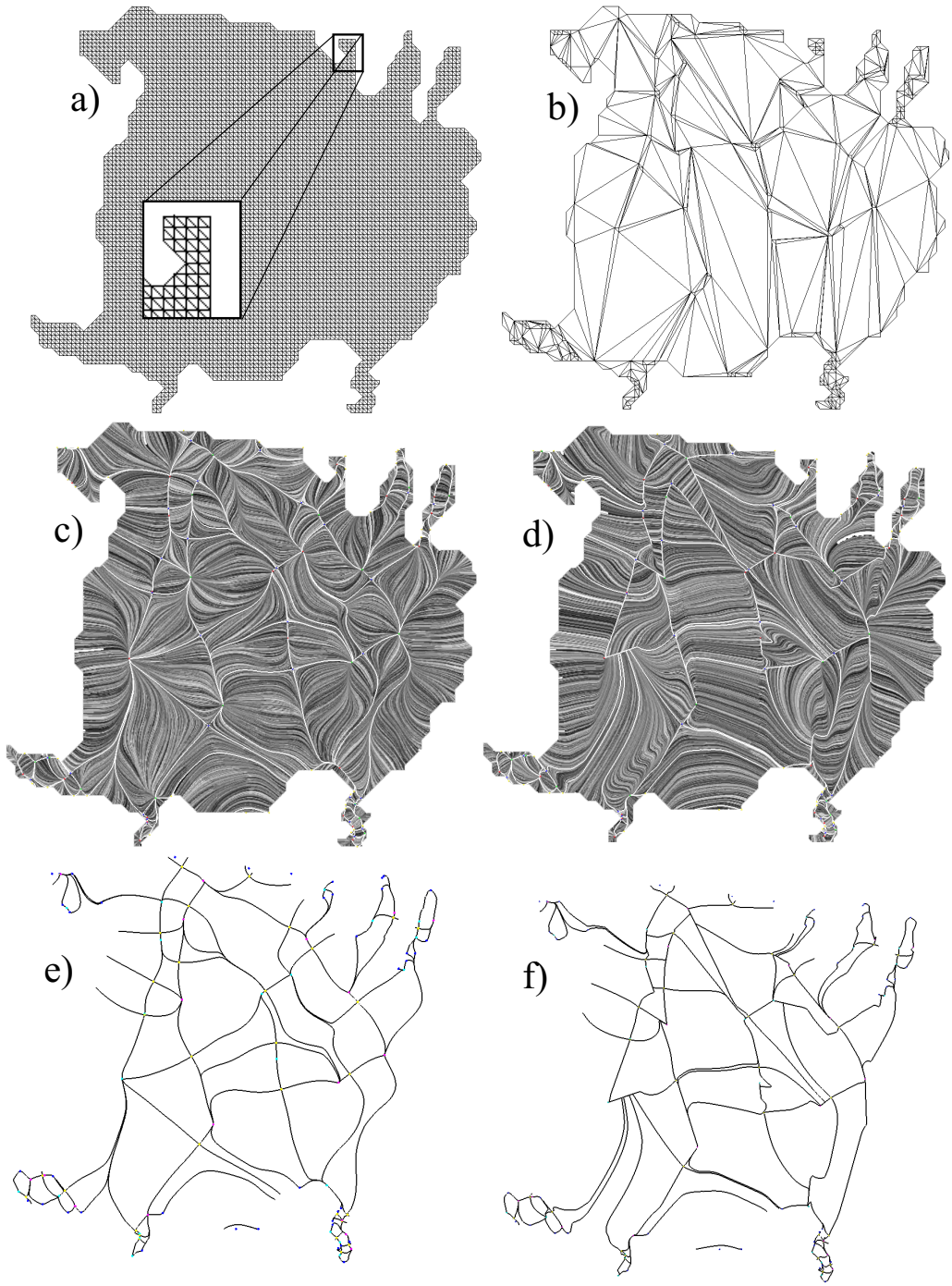


Figure 5.13: Test data set 1 (Greifswalder Bodden), compressed with algorithm 5.2. a) Triangular domain of the original data set. b) Triangular domain of the compressed data set. c) LIC of original data set. d) LIC of compressed data set. e) topological skeleton of original data set. f) topological skeleton of compressed data set.

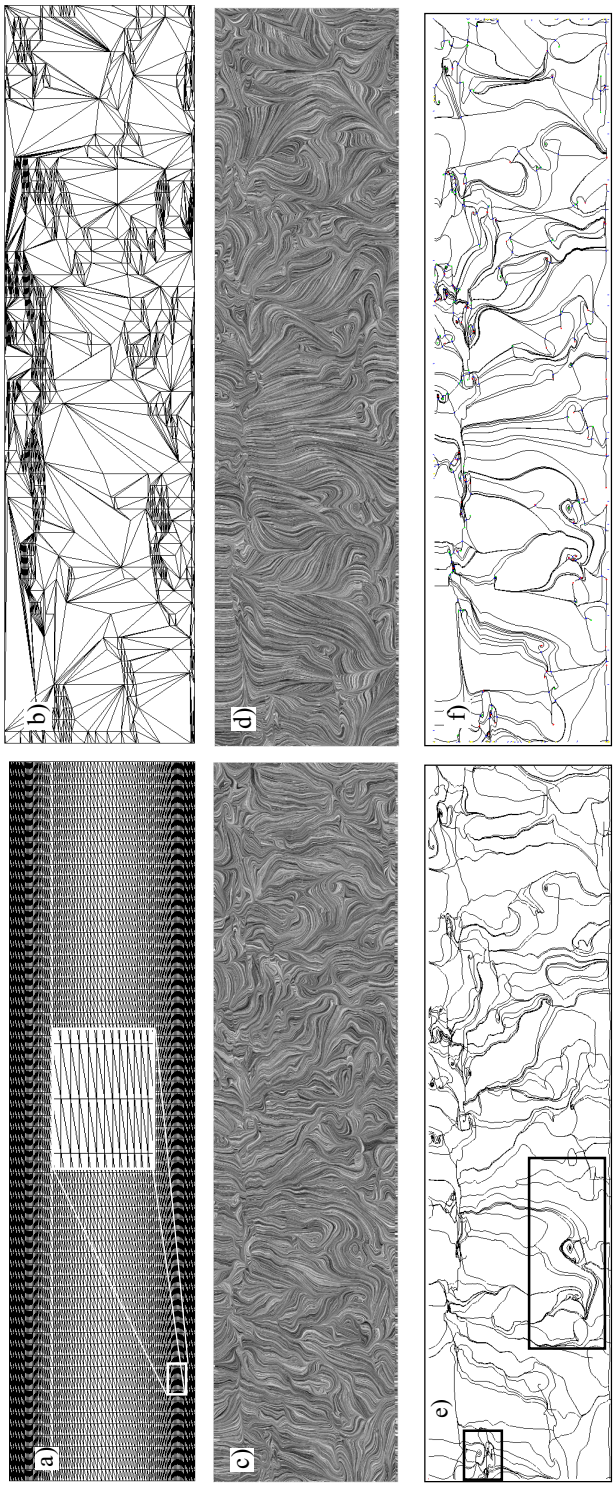


Figure 5.14: Test data set 2 (skin friction), compressed with algorithm 5.2. *Please rotate page.* a) Triangular domain of the original data set. b) Triangular domain of the compressed data set. c) LIC of original data set. d) LIC of compressed data set. e) Topological skeleton of original data set. f) Topological skeleton of compressed data set.

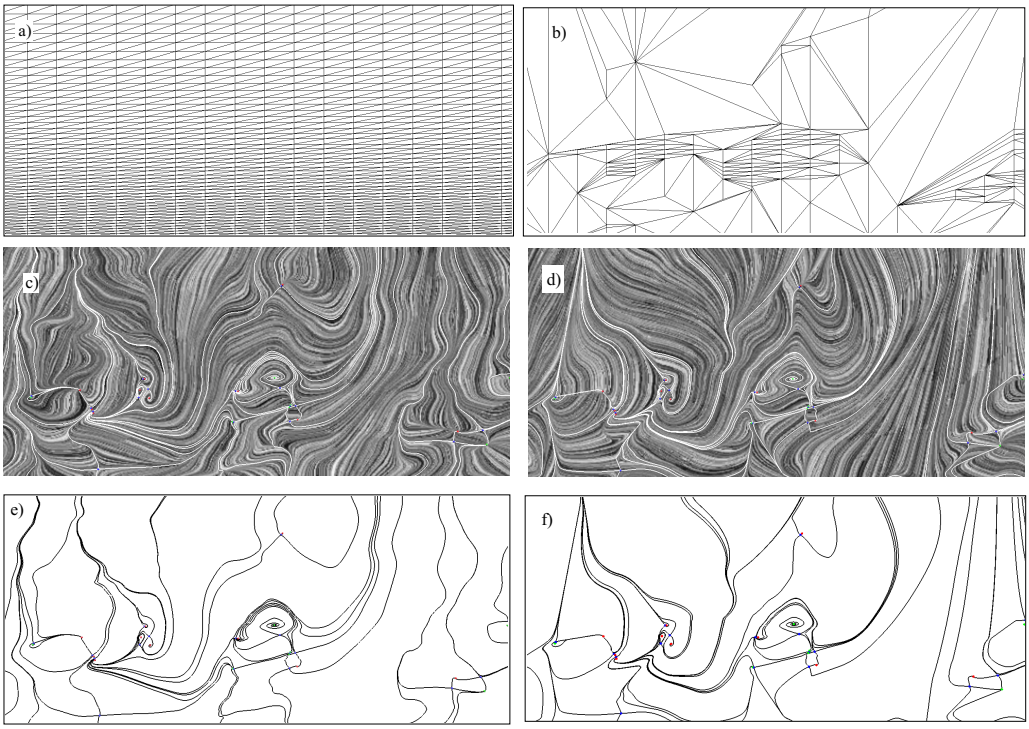


Figure 5.15: Magnification of test data set 2, compressed with algorithm 5.2 (from Figure 5.14). a) Triangular domain of the original data set. b) Triangular domain of the compressed data set. c) LIC of original data set. d) LIC of compressed data set. e) Topological skeleton of original data set. f) Topological skeleton of compressed data set.

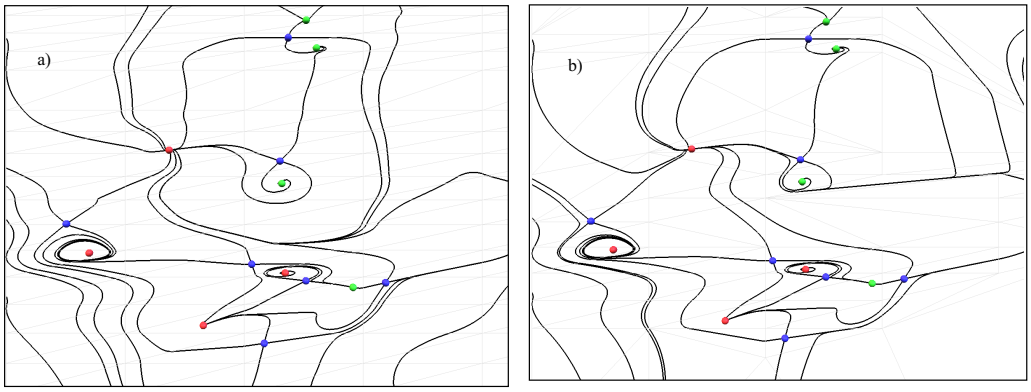


Figure 5.16: Magnification of test data set 2, compressed with algorithm 5.2 (from Figure 5.14). a) Topological skeleton and underlying grid of original data set. b) Topological skeleton and underlying grid of compressed data set.

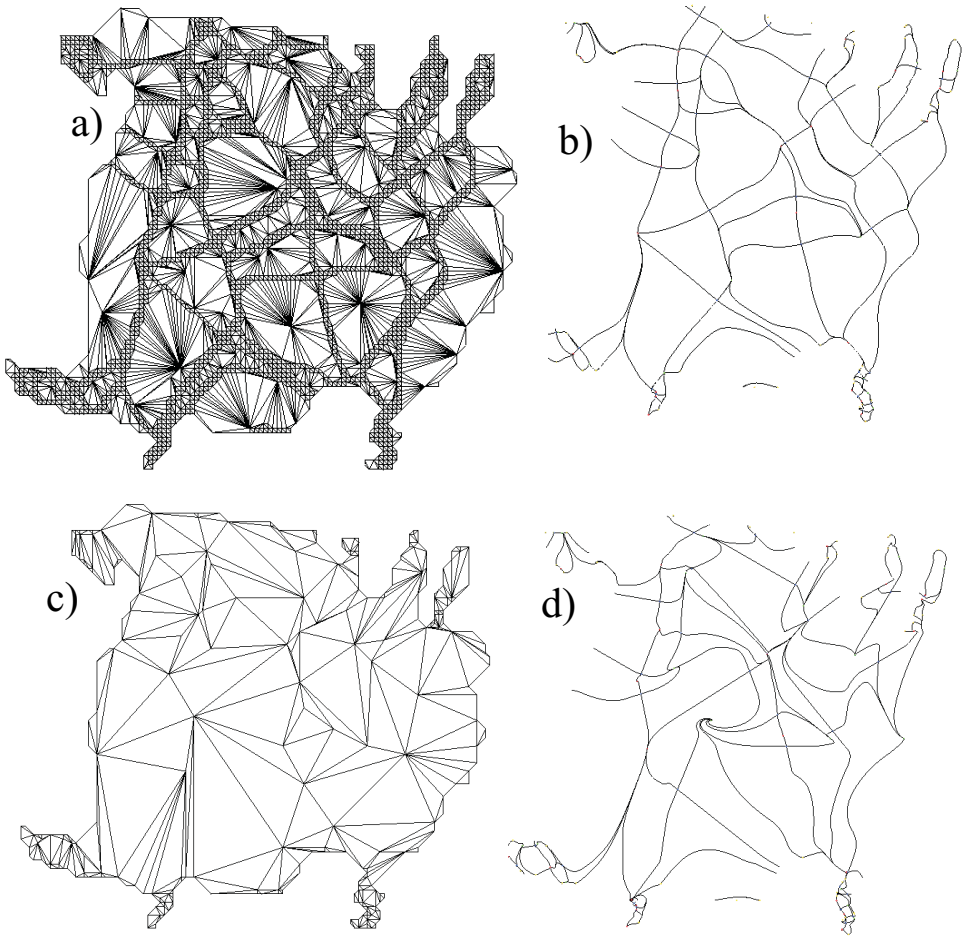


Figure 5.17: Compression of test data set 1 (Greifswalder Bodden) with modified algorithms 5.3 and 5.4. Triangular domain a) and topological skeleton b) after compression with algorithm 5.3). The skeleton is identical to Figure 5.13 due to equivalence concept 1. c) and d) show the same after compression with algorithm 5.4.

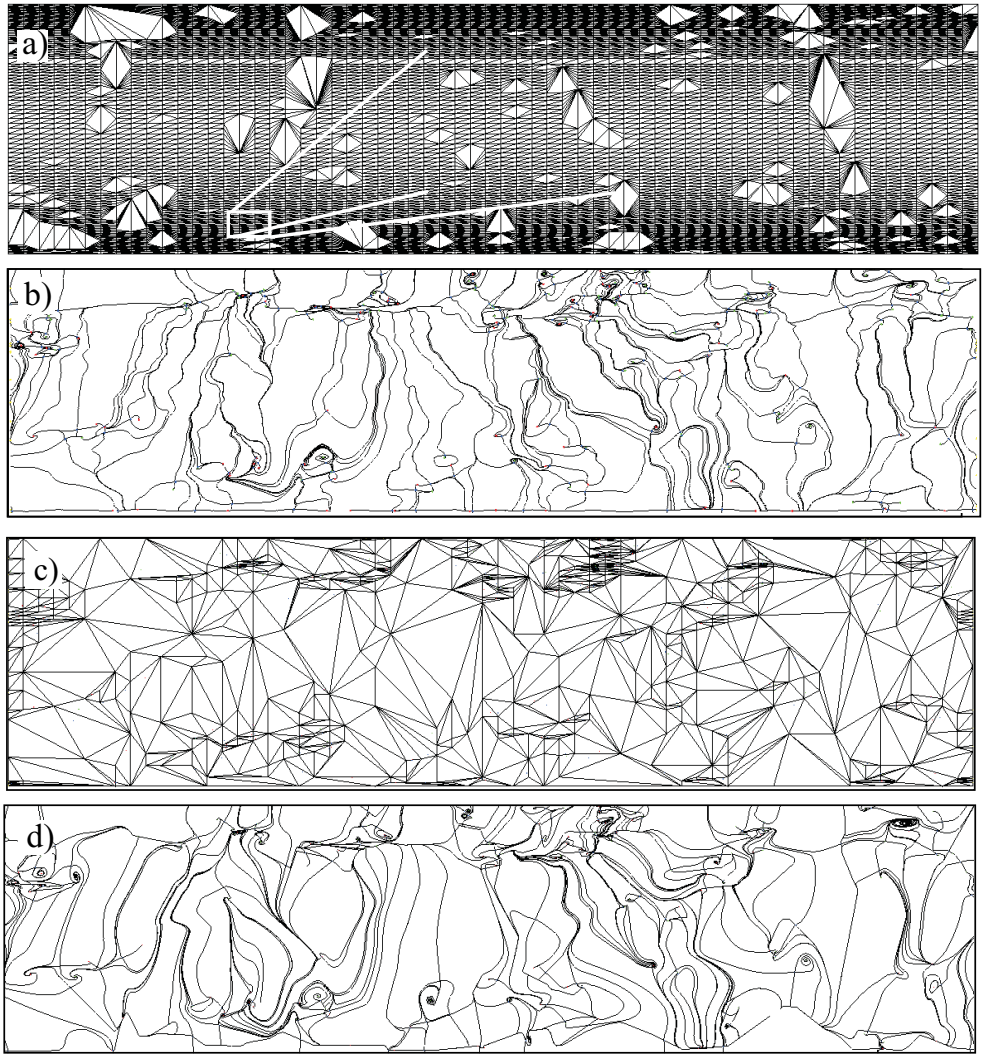


Figure 5.18: Compression of test data set 2 (skin friction) with modified algorithms 5.3 and 5.4. Triangular domain a) and topological skeleton b) after compression with algorithm 5.3). The skeleton is identical to Figure 5.13 due to equivalence concept 1. c) and d) show the same after compression with algorithm 5.4.

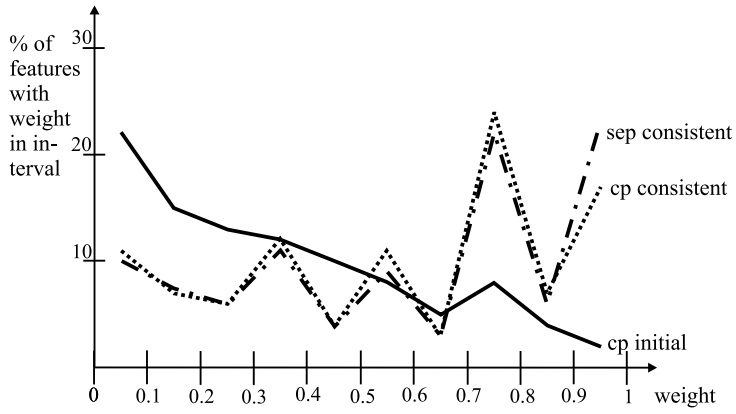


Figure 5.19: Distribution of the importance weights for data set 2 (skin friction). The figure shows the percentages of the features with a weight in the interval $(\frac{i}{10}, \frac{i+1}{10})$ for $i = 0, .., 9$. Solid line: critical points after initializing. Dotted line: critical points after making the weights consistent. Dash-dotted line: separatrices after making the weights consistent.

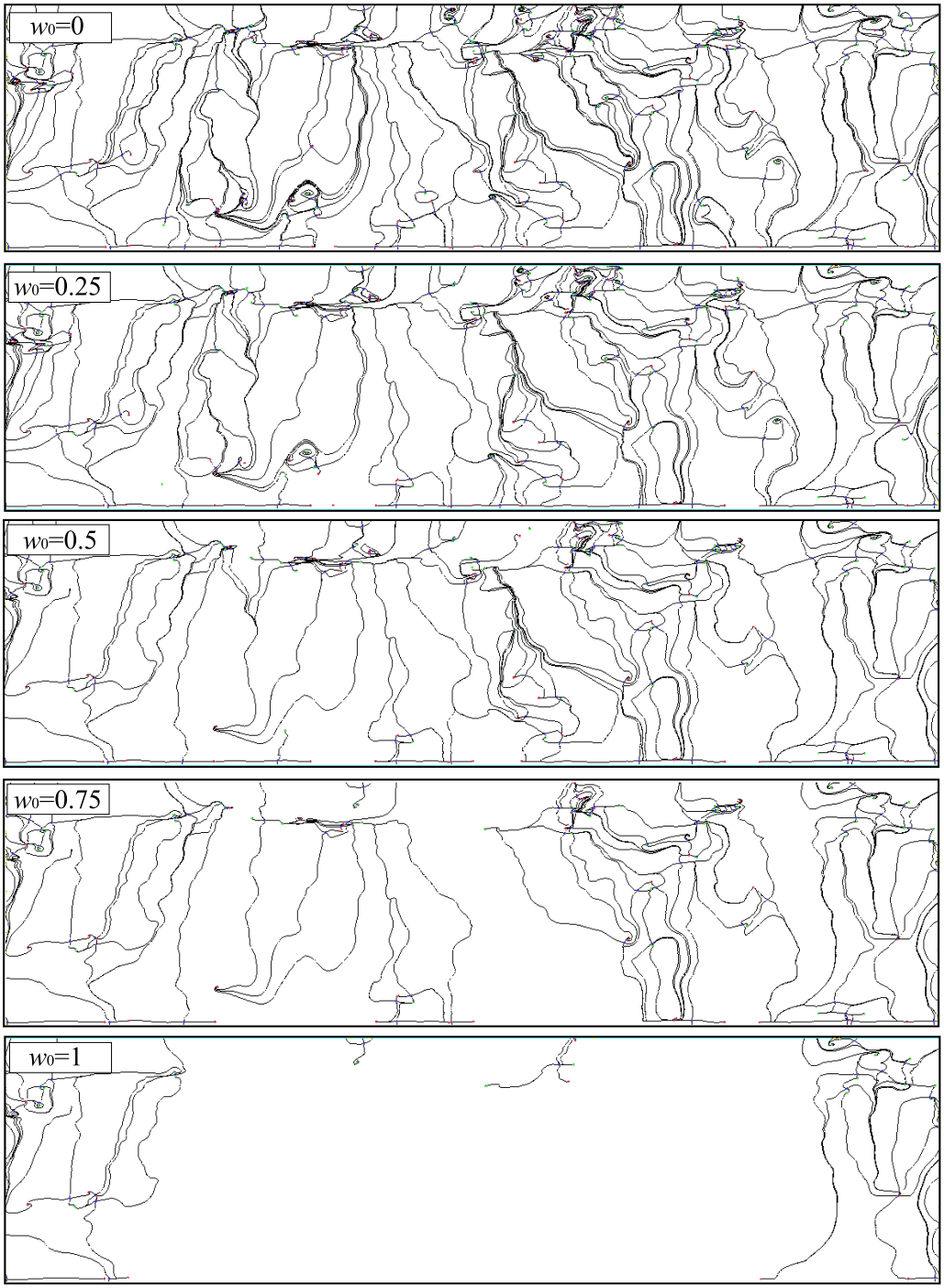


Figure 5.20: Important topological features for different thresholds w_0. The upper image ($w_0 = 0$) shows the complete topological skeleton.

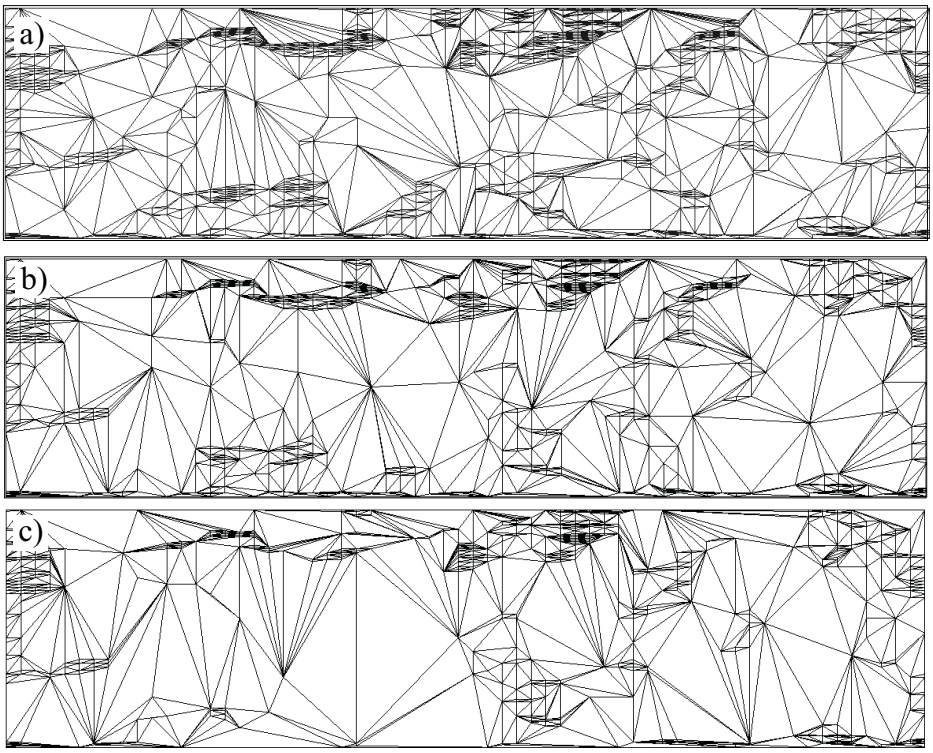


Figure 5.21: Triangular domains after the application of algorithm 5.5 with different thresholds w_0: a) $w_0 = 0$ (1,805 triangles). b) $w_0 = 0.5$ (1,367 triangles). c) $w_0 = 1$ (1,040 triangles).

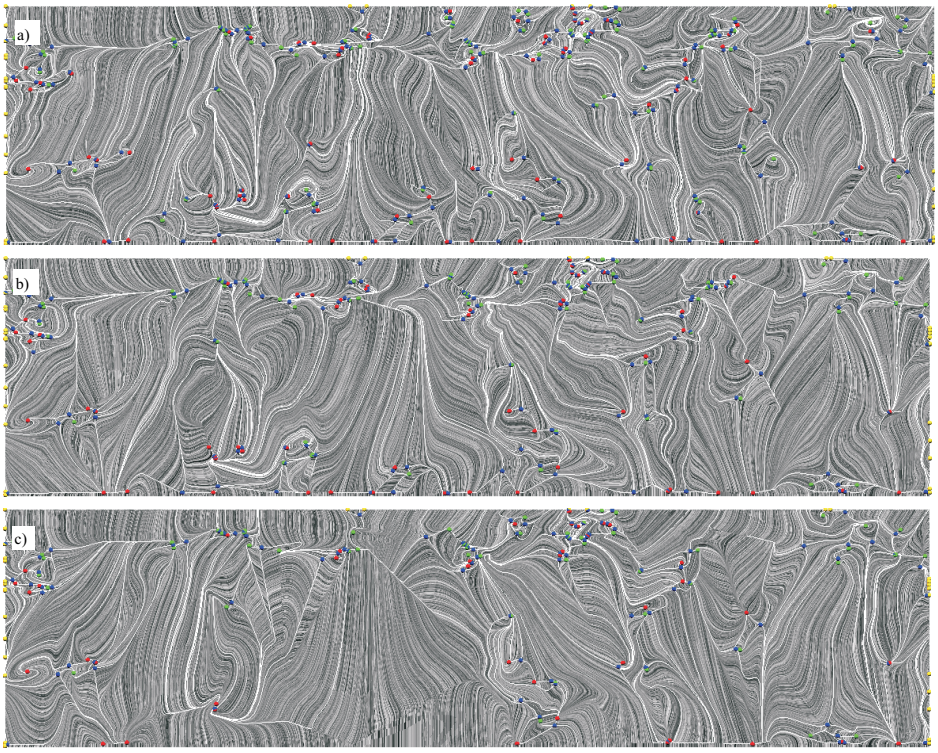


Figure 5.22: Topological skeleton and LIC-like image after the application of algorithm 5.5 with different thresholds w_0: a) $w_0 = 0$. b) $w_0 = 0.5$. c) $w_0 = 1$.

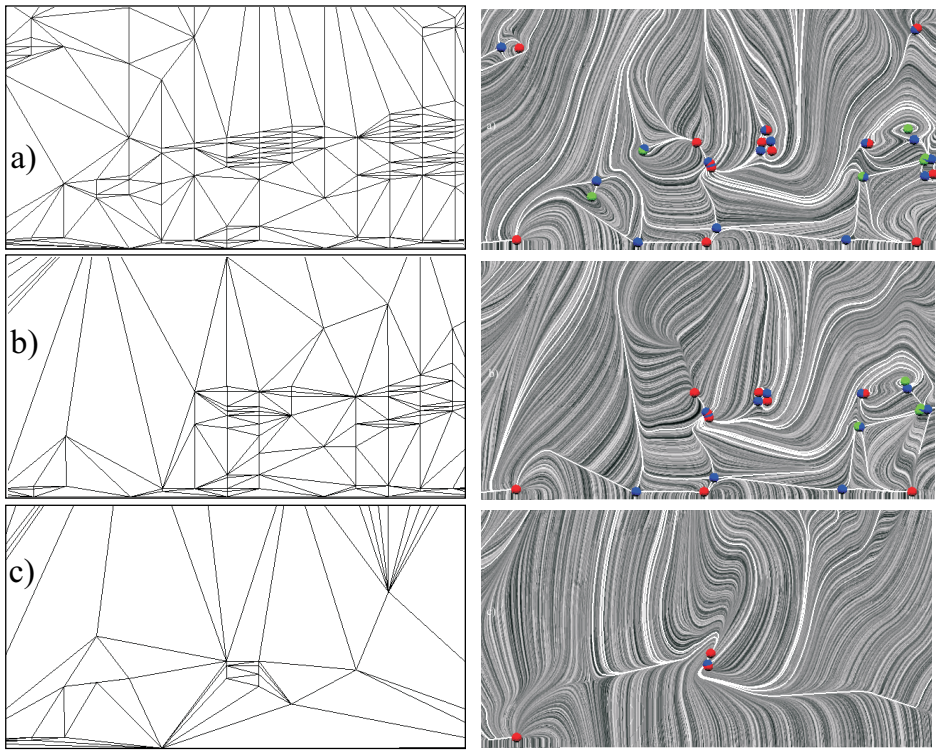


Figure 5.23: Underlying triangular grid and topological skeleton on LIC-like image (magnification) after the application of algorithm 5.5 with different thresholds w_0. a) $w_0 = 0$. b) $w_0 = 0.5$. c) $w_0 = 1$.

Part II

Volumes

Chapter 6
Volumetric Data

In the first part, we considered the visualization of two-dimensional objects, namely surfaces and (2d-)vector fields. In the second part, we now approach volumetric representations or data given in a three-dimensional domain. Generally, things get more involved when adding another dimension. Here, we focus on the fundamental problem of finding efficient models for such volumetric data.

Two different scenarios are of particular interest: the reconstruction of structured data, where the samples are placed along a regular grid, and the approximation of general data, where the sample points are distributed in the domain. For both, the goal is roughly to find a function in three variables that assigns each point in the domain a value related to the values of nearby samples.

There is a multitude of techniques for the visualization of volumetric data (see e.g. [Chen et al. 2000] and the references therein). The demands on the model and its evaluation for high-quality visualization are similar for different techniques: efficient evaluation of function values and partial derivatives, where appropriate smoothness and approximation conditions must be satisfied.

We achieve this by using trivariate splines, i.e. piecewise polynomials which are defined w.r.t. a partition of the volumetric domain. Similar to a triangulation as a partition of the plane, a tetrahedral complex is applied now, and the polynomials are defined w.r.t. tetrahedral cells of the complex. In contrast to the proposed techniques for the two-dimensional setting, which made use of piecewise linear functions, the demand for appropriate smoothness in volume visualization enforces a higher polynomial degree. A particular challenge is to develop models, which fulfill all requirements and at the same time apply only low polynomial degrees. We remark that the constructions generally gets easier with higher polynomial degrees as more degrees of freedom are provided for satisfying smoothness and approximation conditions. Moreover, the construction of splines does not easily translate from the univariate case

to higher dimensions, and it is not obvious what is the best generalization combining simple and elegant construction with computational efficiency. We consider the Bernstein-Bézier representation of the polynomial pieces and interpret the spline as a net of their coefficients, where certain coefficients are variable while others are determined by continuity and smoothness conditions. This is a common approach in multivariate spline theory (see e.g. [Chui 1988, Davydov and Zeilfelder 2003, Hangelbroek et al. 2004, Kohlmüller et al. 2003a, Kohlmüller et al. 2003b, Lai and Méhauté 2003, Nürnberger et al. 2003a, Nürnberger and Zeilfelder 2000, Schumaker and Sorokina 2004a, Zeilfelder 2002, Nürnberger and Zeilfelder 2003]).

In Chapter 7, we explain these general considerations in more detail, reviewing Bernstein-Bézier techniques and introducing splines w.r.t. so-called type-6 tetrahedral partitions.

Building upon this, we develop a new model for the reconstruction of gridded volume data with quadratic super splines in Chapter 8. The approximating splines are determined in a natural and completely symmetric way by averaging local data samples, such that appropriate smoothness conditions are automatically satisfied. This new approach enables efficient reconstruction and visualization of the data. As the piecewise polynomials are of the lowest possible total degree two, and we can efficiently determine exact ray intersections with an isosurface for ray-casting. Moreover, the optimal approximation properties of the derivatives enable the simply sampling of the necessary gradients directly from the polynomial pieces of the splines.

In Chapter 9, a new algorithm for the efficient approximation of huge general volumetric data sets is developed, which is based on cubic trivariate splines. Similar as in recent bivariate approximation approaches (cf. [Davydov and Zeilfelder 2003, Haber et al. 2001]), the splines are automatically determined from the discrete data as a result of a two-step method (see [Schumaker 1976]), where local discrete least squares polynomial approximations of varying degrees are extended by using natural conditions, i.e. the continuity and smoothness properties which determine the underlying spline space. The main advantages of this approach with linear algorithmic complexity are as follows: no tetrahedral partition of the volume data is needed, only small linear systems have to be solved, the local variation and distribution of the data is automatically adapted, noisy data are automatically smoothed.

Both approaches to processing regular and general data are related but different, and we provide the specific background for the two problems individually. However, we emphasize that both methods share the idea of using *consistent* splines on tetrahedral partitions, which satisfy many smoothness conditions. According to our knowledge, these are the first approaches in the literature, where it is shown that such splines provide useful tools with advantageous properties for

the various requirements of efficient visualization.

Chapter 7

Trivariate C^1-Splines on Type-6 Tetrahedral Partitions

We construct splines as piecewise polynomials w.r.t. a certain tetrahedral partition of the volumetric domain. This chapter reviews trivariate polynomials in Bernstein-Bézier form, introduces the type-6 tetrahedral partitions, and discusses the construction and evaluation of C^1-splines w.r.t. such partitions. These are the basic building blocks for the subsequent chapters on the reconstruction and approximation.

7.1 Trivariate Polynomials and Bernstein-Bézier Form

In the following, we briefly recall some notation. We apply *Bernstein-Bézier techniques* which are well established tools from CAGD (see e.g. [Hoschek and Lasser 1993, Prautzsch et al. 2002]). The related *Bernstein-Bézier form* of the polynomial pieces plays a key role for multivariate splines.

For any integer q, we call

$$\mathcal{P}_q = \text{ span}\{x^i y^j z^k : \ i,j,k \geq 0, \ i+j+k \leq q\}$$

the $\binom{q+3}{3}$ dimensional space of *trivariate polynomials of total degree* q.

Given a (non-degenerate) tetrahedron $T = [v_0, v_1, v_2, v_3]$ in $\mathbb{R}^3$ with vertices v_0, v_1, v_2, and v_3, the linear polynomials $\lambda_\nu \in \mathcal{P}_1, \ \nu = 0, \ldots, 3$, with the interpolation property

$$\lambda_\nu(v_\mu) = \delta_{\nu,\mu}, \ \mu = 0, \ldots, 3,$$

are called the *barycentric coordinates w.r.t.* T. It is easy to see that for any point $u \in \mathbb{R}^3$, the barycentric coordinates $\lambda_\nu(u) \in \mathbb{R}, \ \nu = 0, \ldots, 3$, of u are uniquely

determined as the solution of the 4×4 linear system

$$\lambda_0(u) \begin{pmatrix} v_0 \\ 1 \end{pmatrix} + \lambda_1(u) \begin{pmatrix} v_1 \\ 1 \end{pmatrix} + \lambda_2(u) \begin{pmatrix} v_2 \\ 1 \end{pmatrix} + \lambda_3(u) \begin{pmatrix} v_3 \\ 1 \end{pmatrix} = \begin{pmatrix} u \\ 1 \end{pmatrix}. \tag{7.1}$$

Every polynomial $p \in \mathcal{P}_q$ can be written in its *Bernstein-Bézier representation* as

$$p = \sum_{i+j+k+l=q} a_{i,j,k,l}\, B^{q,T}_{i,j,k,l}, \tag{7.2}$$

where

$$B^{q,T}_{i,j,k,l} = \tfrac{q!}{i!j!k!l!}\, \lambda_0^i \lambda_1^j \lambda_2^k \lambda_3^l \in \mathcal{P}_q, \qquad i+j+k+l = q,$$

are the *Bernstein polynomials of degree q w.r.t. T*.

Each *Bernstein-Bézier coefficient* $a_{i,j,k,l} \in \mathbb{R}$ of p is associated with the *domain point* (or *Bézier point*)

$$\xi^T_{i,j,k,l} = (iv_0 + jv_1 + kv_2 + lv_3)/q,$$

and the *set of domain points in T* is denoted by

$$\mathcal{D}_{q,T} = \{\xi^T_{i,j,k,l} : \; i+j+k+l = q\}.$$

Figure 7.1 illustrates the Bernstein-Bézier representation of a quadratic polynomial, the ten coefficients are indicated by white dots (and the ray intersection is discussed in Section 8.5).

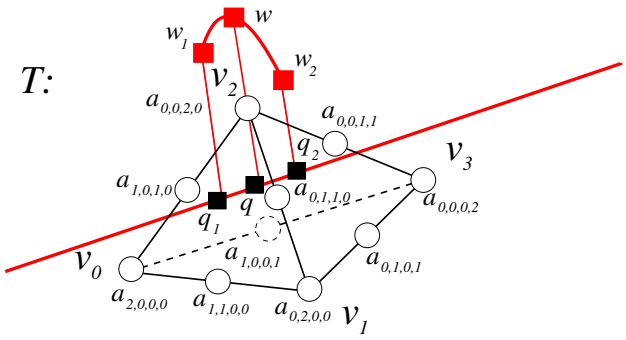


Figure 7.1: The ten Bézier points (white dots) of a quadratic polynomial inside a tetrahedron T are associated with the Bernstein-Bézier coefficients. The restriction of this trivariate polynomial piece to an arbitrary ray (red line) is a quadratic, univariate polynomial (red curve) which is uniquely determined by the values (red boxes) at three points (black boxes), see Section 8.5.

The Bernstein-Bézier representation allows for the efficient and stable evaluation of a polynomial p by applying the *de Casteljau algorithm* (cf.

[de Casteljau 1963], see also [Farin 1986, Hoschek and Lasser 1993]), which is based on the recurrence relation of the Bernstein polynomials

$$B^{q,T}_{i,j,k,l} \;=\; \lambda_0 B^{q-1,T}_{i-1,j,k,l} + \lambda_1 B^{q-1,T}_{i,j-1,k,l} + \lambda_2 B^{q-1,T}_{i,j,k-1,l} + \lambda_3 B^{q-1,T}_{i,j,k,l-1}$$

(where terms with negative indices are set to zero). The trivariate version of this algorithm to determine the value $p(u) = a^{[q]}_{0,0,0,0}$ at a point $u \in T$ reads as follows:

Algorithm 7.1 *(de Casteljau Algorithm)*
For $\ell = 1, \ldots, q$ *and* $i+j+k+l = q-\ell$ *compute*

$$a^{[\ell]}_{i,j,k,l} \;=\; \lambda_0(u)\; a^{[\ell-1]}_{i+1,j,k,l} + \lambda_1(u)\; a^{[\ell-1]}_{i,j+1,k,l} + \lambda_2(u)\; a^{[\ell-1]}_{i,j,k+1,l} + \lambda_3(u)\; a^{[\ell-1]}_{i,j,k,l+1},$$

where $a^{[0]}_{i,j,k,l} = a_{i,j,k,l},\ i+j+k+l = q.$

Figure 7.2 illustrates the evaluation of a quadratic polynomial with the de Casteljau algorithm.

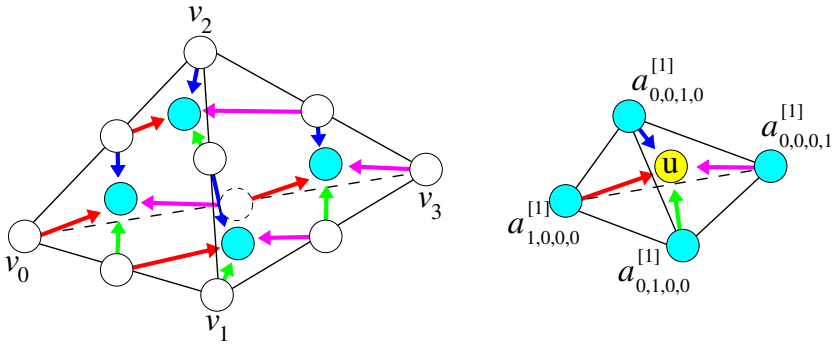


Figure 7.2: Evaluation of a polynomial piece p at a point u (yellow dot) with the de Casteljau algorithm. The configuration is the same as in Figure 7.1, the white dots show the Bézier points associated with the coefficients $a_{i,j,k,l} = a^{[0]}_{i,j,k,l},\ i+j+k+l = 2$. The four new coefficients $a^{[1]}_{i,j,k,l},\ i+j+k+l = 1$, on level $\ell = 1$ indicated as cyan dots are determined from affine combinations by weighting with the barycentric coordinates of u w.r.t. $T = [v_0, v_1, v_2, v_3]$. The arrows show which coefficients are involved, the different colors represent the barycentric coordinates $\lambda_i(u)$. For $\ell = 2$ the final result $p(u) = a^{[2]}_{0,0,0,0}$ (yellow dot) is computed in the same way recursively. Note that the intermediate coefficients $a^{[1]}_{i,j,k,l}$ (cyan dots) define the directional derivatives $(\frac{\partial p}{\partial \varsigma_\nu})(u)$.

The de Casteljau algorithm simultaneously computes *partial derivatives* of p: Let ς_ν be a vector in the direction of the edge $v_{\nu+1} - v_0$ of T with length $\|v_{\nu+1} - v_0\|$, then the partial derivative of p in direction of ς_ν denoted by $\frac{\partial p}{\partial \varsigma_\nu} \in \mathcal{P}_{q-1}$ is given as

$$\frac{\partial p}{\partial \varsigma_\nu} = q \sum_{i+j+k+l=q-1} \left(a_{i,j+j_\nu,k+k_\nu,l+l_\nu} - a_{i+1,j,k,l}\right) B^{q-1,T}_{i,j,k,l}, \qquad (7.3)$$

or

$$\frac{\partial p}{\partial \varsigma_\nu}(u) = q(a^{[q-1]}_{0,j_\nu,k_\nu,l_\nu} - a^{[q-1]}_{1,0,0,0}), \tag{7.4}$$

where $(j_\nu, k_\nu, l_\nu) = (\delta_{\nu,\mu})^2_{\mu=0}$, $\nu = 0, \ldots, q$ (see e.g. [Hoschek and Lasser 1993]). Hence, there exist unique α_0, $\alpha_1, \alpha_2 \in \mathbb{R}$ such that, for instance,

$$\frac{\partial p}{\partial x} = \alpha_0 \, \frac{\partial p}{\partial \varsigma_0} + \alpha_1 \, \frac{\partial p}{\partial \varsigma_1} + \alpha_2 \, \frac{\partial p}{\partial \varsigma_2}. \tag{7.5}$$

This allows for the efficient computation of the *gradient*

$$(\nabla p)(u) = ((\tfrac{\partial p}{\partial x})(u), (\tfrac{\partial p}{\partial y})(u), (\tfrac{\partial p}{\partial z})(u)). \tag{7.6}$$

Degree raising methods express a degree q polynomial

$$p = \left(\sum_{i+j+k+l=q} a_{i,j,k,l} B^{q,T}_{i,j,k,l} \right) (\lambda_0+\lambda_1+\lambda_2+\lambda_3)^\nu = \sum_{i+j+k+l=q+\nu} a^*_{i,j,k,l} B^{q+\nu,T}_{i,j,k,l},$$

as a polynomial of degree $q + \nu$, where

$$a^*_{I,J,K,L} = \sum_{i+j+k+l=q} \frac{\binom{I}{i}\binom{J}{j}\binom{K}{k}\binom{L}{l}}{\binom{q+\nu}{q}} a_{i,j,k,l}, \tag{7.7}$$

and $I + J + K + L = q + \nu$ (see e.g. [Hoschek and Lasser 1993]).

7.2 Type-6 Tetrahedral Partitions Δ

We call a set of tetrahedra Δ a *tetrahedral partition* of a finite polyhedral domain $\Omega \subseteq \mathbb{R}^3$ if the intersection of any two different tetrahedra from Δ is a common vertex, common edge or common triangle, and the union of all tetrahedra from Δ is equal to Ω.

We consider tetrahedral partitions Δ of the unit cube domain $\Omega = [0,1] \times [0,1] \times [0,1] \subseteq \mathbb{R}^3$ which are obtained as follows. Given an integer n, we first use $n+1$ parallel planes in each of the three space dimensions, and subdivide Ω into n^3 subcubes denoted by

$$Q_{i,j,k} = [i, i+1]\, h \times [j, j+1]\, h \times [k, k+1]\, h,$$

for $i, j, k = 0, \ldots, n-1$, where $h = \frac{1}{n}$ is the length of the edges. This defines a *cube partition* $\diamondsuit$[1].

[1]The results can be extended to more general cubic domains, see [Hangelbroek et al. 2004] and Figure 9.1.

We split each of the n^3 cubes Q into six (Egyptian) pyramids by connecting its center point v_Q with the four vertices of every face of Q. Then, we insert both diagonals in these six faces of Q and connect their intersection points with v_Q. This subdivides each of the six pyramids in Q into four tetrahedra, forming a natural, uniform tetrahedral partition Δ of Ω, where every cube $Q \in \diamondsuit$ contains 24 congruent tetrahedra. A more intuitive way to describe this *type-6 tetrahedral partition* Δ is to say that Δ is the tetrahedral partition obtained by slicing Ω with the *six planes* which contain opposite edges of Ω. In [Carr et al. 2001a] the above construction is called a *face-centered 24-fold subdivision* of the cubes.

Figure 7.3 illustrates the construction of Δ. The partition Δ is a generalization of the four-directional mesh which is well-known in the bivariate setting (cf. [Chui 1988, Davydov and Zeilfelder 2003, Haber et al. 2001]). The relation to the bivariate setting is shown in Figure 7.4.

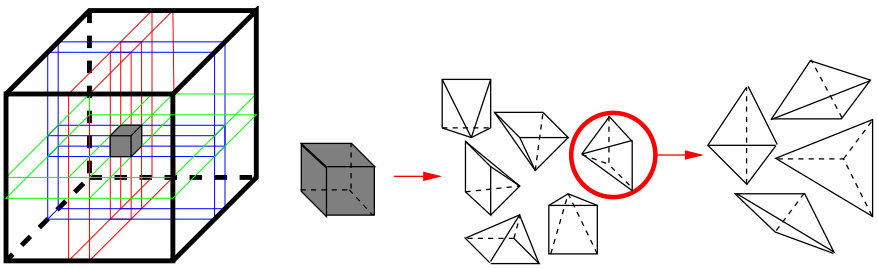


Figure 7.3: The tetrahedral partition Δ is obtained by uniformly subdividing each cube of $\diamondsuit$ into 24 tetrahedra. Since six planes are needed, Δ is called a *type-6 tetrahedral partition*.

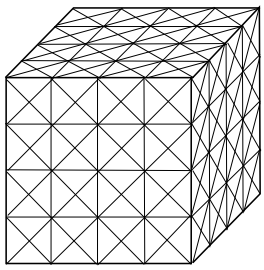


Figure 7.4: The intersections of Δ with planes parallel to the three coordinate planes are four-directional meshes which are well-known from the bivariate setting.

It is easy to see that for this uniform tetrahedral partition Δ, we have

$$\begin{aligned} T_\Delta &= 24\,n^3, \\ F_\Delta &= 48\,n^3 + 12\,n^2, \\ E_\Delta &= 29\,n^3 + 18\,n^2 + 3\,n, \\ V_\Delta &= 5\,n^3 + 6\,n^2 + 3\,n + 1, \end{aligned} \tag{7.8}$$

for the number of tetrahedra T_Δ, the number of triangular faces F_Δ, the number of edges E_Δ, and the number of vertices V_Δ of Δ, respectively.

7.3 Trivariate C^1-Splines on Δ

7.3.1 Preliminaries: C^r-Splines on Δ

Given a tetrahedral partition Δ of Ω, we set

$$\mathcal{S}_q^r(\Delta) = \{s \in C^1(\Omega) : s|_T \in \mathcal{P}_q \text{ for all tetrahedra } T \in \Delta\} \tag{7.9}$$

for the space of *trivariate C^r-splines of degree q w.r.t. Δ*.

The coefficients $a_{\xi^T_{i,j,k,l}}(s) := a_{i,j,k,l}(s|_T),\ i+j+k+l=q$, of $s \in \mathcal{S}_q^0(\Delta)$ in the representation (7.2) of its polynomial pieces $s|_T \in \mathcal{P}_q,\ T \in \Delta$, are uniquely associated with the *domain points in* Ω

$$\mathcal{D}_{q,\Delta} = \bigcup_{T\in\Delta} \mathcal{D}_{q,T}. \tag{7.10}$$

Following [Alfeld et al. 1987], we call $\mathcal{M} \subseteq \mathcal{D}_{q,\Delta}$ a *determining set for a linear subspace* $\mathcal{S}$ of $\mathcal{S}_q^0(\Delta)$, if setting the coefficients $a_\xi(s),\ \xi \in \mathcal{M}$ of a spline $s \in \mathcal{S}$ to zero, implies that $s \equiv 0$. A determining set $\mathcal{M}$ is called *minimal determining set for* $\mathcal{S}$, if no determining set for $\mathcal{S}$ with fewer elements than $\mathcal{M}$ exists. Equivalently, $\mathcal{M}$ is a minimal determining set, if the following property holds: if we set the coefficients $a_\xi(s),\ \xi \in \mathcal{M}$, of a spline $s \in \mathcal{S}$ to arbitrary values, then all its coefficients $a_\xi(s),\ \xi \in \mathcal{D}_{q,\Delta}$ are uniquely determined, i.e. s is uniquely determined. If $\mathcal{M}$ is a minimal determining set for $\mathcal{S}$, then it is obvious that $\#\mathcal{M}$ coincides with the dimension $\dim \mathcal{S}$ of $\mathcal{S}$.

In the following, we consider *continuous splines* in S_q^0 and *C^1-splines* in S_q^1 on Δ.

7.3.2 C^1-Smoothness Conditions

A convenient description for *C^1-smoothness* of neighboring polynomials (i.e. polynomials defined on tetrahedra which have a common triangular face) in Bernstein-Bézier form can be found in [de Boor 1987, Chui 1988, Farin 1986].

Let $p = s|_T$ be given on T as in (7.2), and set $\tilde{p} = s|_{\tilde{T}} \in \mathcal{P}_q$ for a neighboring polynomial on $\tilde{T} = [v_0, v_1, v_2, \tilde{v}_3]$ with Bernstein-Bézier coefficients $\tilde{a}_{i,j,k,l},\ i + j + k + l = q$. Then s is a *continuous spline on* $T \cup \tilde{T}$, if the coefficients associated with the common triangular face $\mathcal{T} = [v_0, v_1, v_2]$ coincide, i.e. $a_{i,j,k,0} = \tilde{a}_{i,j,k,0}$ for all $i + j + k = 2$. And s is *C^1-smooth across $\mathcal{T}$* iff, in addition,

$$\tilde{a}_{i,j,k,1} \;=\; \lambda_0(\tilde{v}_3)\, a_{i+1,j,k,0} + \lambda_1(\tilde{v}_3)\, a_{i,j+1,k,0} + \lambda_2(\tilde{v}_3)\, a_{i,j,k+1,0} + \lambda_3(\tilde{v}_3)\, a_{i,j,k,1},$$

for all $i + j + k = q - 1$.

Analogously to the univariate and bivariate cases, for each condition of this form there is the geometric interpretation that five points in $\mathbb{R}^4$ lie in the same (three-dimensional) hyper-plane, in general. The fourth components of these points are the Bernstein-Bézier coefficients while the first three components are the associated Bézier points. In general, there are five coefficients involved for every single smoothness condition. If one or even two of the barycentric coordinates vanish at $\tilde{v}_3$, the number of involved coefficients is four and three, respectively. For instance, this holds if $\tilde{v}_3$ lies in the plane that contains the triangle $[v_0, v_1, v_3]$ and if $\tilde{v}_3$ lies on the line that contains the edge $[v_0, v_3]$, respectively. In these cases, the smoothness conditions *degenerate* to lower dimensional conditions known from the bivariate and univariate setting, respectively. Figure 7.5 illustrates a degenerated and the general case.

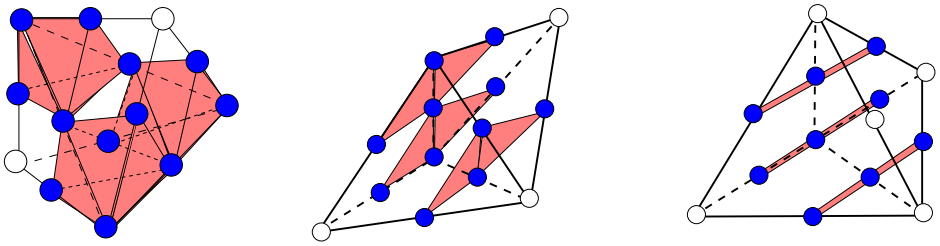


Figure 7.5: Smoothness conditions of neighboring quadratic polynomials across the common triangular face of two tetrahedra. The domain points associated with coefficients involved in the smoothness conditions are shown as blue dots. In general, there are five coefficients involved in each of the three conditions (left). If exactly one barycentric coordinate vanishes at the point $\tilde{v}_3$, then four coefficients are involved in every smoothness condition (middle). If two barycentric coordinates vanish at the point $\tilde{v}_3$, i.e. three vertices lie on a line, then the conditions even degenerate to univariate smoothness conditions (right).

For the type-6 tetrahedral partition Δ defined in the previous Section 7.2, there are three cases, i.e. we have to consider neighboring tetrahedra lying in

- two different cubes of $\diamondsuit$,

- the same pyramid (cf. Figure 7.3) of a cube,
- two different pyramids of a cube.

We observe that the smoothness conditions for the first two cases (inter-cube and intra-pyramid) degenerate to simple univariate conditions involving three coefficients (Figure 7.5, left), while in the third case (inter-pyramid) the smoothness conditions are of the general form involving five coefficients (Figure 7.5, right). Due to symmetry, there are *essentially two conditions* with always the same barycentric coordinates involved, and one can easily see that enforcing smoothness over the common triangular face of neighboring tetrahedra in Δ is to apply one of these two simple averaging formulae[2] illustrated in Figure 7.6.

Figure 7.6: C^1-smoothness conditions on a type-6 tetrahedral partition Δ. The conditions are shown as the stencils of the two averaging rules which apply on Δ, where the outlined Bernstein-Béziercoefficient is determined from the solid ones. The left stencil corresponds to the degenerate, univariate case (inter-cube and intra-pyramid, cf. Figure 7.5 (left)). The right stencil corresponds to the non-degenerate, general case (inter-pyramid, cf. Figure 7.5 (right)).

According to their computational simplicity and numerical stability, C^1-splines are spaces of particular interest in multivariate spline theory and CAGD. On the other hand, the structure of C^1-splines is much more complex than in the case of continuous splines and many open question currently exist for the overall smooth spaces, even in the bivariate case, see the results and methods presented in [Beatson and Ziegler 1985, Chui and He 1986, Dagnino and Lamberti 2004, Jeeawock-Zedek 1994, Nürnberger and Zeilfelder 2001, Morgan and Scott , Powell 1974, Powell and Sabin 1977, Sablonnière 1987, Sablonnière 2003b, Sablonnière 2003a, Schumaker 1984], for instance. This is in striking contrast to the univariate theory. Clearly, the complexity of the spline spaces even increases for three variables as indicated in the following section.

[2]For the ease of notation, we do not explicitly describe the application of the stencils (including their translation and rotation) here. A formal characterization of all C^1-smoothness conditions on Δ is given in [Hangelbroek et al. 2004, Nürnberger et al. 2004c].

q	dim $\mathcal{S}_q^1(\Delta)$	dim $\mathcal{S}_q^0(\Delta)$	dim $\mathcal{S}_q^{-1}(\Delta)$
1	4	$5\,n^3+6\,n^2+3\,n+1$	$96\,n^3$
2	$3\,n^2+9\,n+4$	$34\,n^3+24\,n^2+6\,n+1$	$240\,n^3$
3	$6\,n^3+24\,n^2+18\,n+4$	$111\,n^3+54\,n^2+9\,n+1$	$480\,n^3$
4	$39\,n^3+66\,n^2+27\,n+4$	$260\,n^3+96\,n^2+12\,n+1$	$840\,n^3$
5	$120\,n^3+132\,n^2+36\,n+4$	$505\,n^3+150\,n^2+15\,n+1$	$1344\,n^3$
6	$273\,n^3+222\,n^2+45\,n+4$	$870\,n^3+216\,n^2+18\,n+1$	$2016\,n^3$
7	$522\,n^3+336\,n^2+54\,n+4$	$1379\,n^3+294\,n^2+21\,n+1$	$2880\,n^3$
8	$891\,n^3+474\,n^2+63\,n+4$	$2056\,n^3+384\,n^2+24\,n+1$	$3960\,n^3$
9	$1404\,n^3+636\,n^2+72\,n+4$	$2925\,n^3+486\,n^2+27\,n+1$	$5280\,n^3$

Table 7.1: Comparison of dimensions of splines on type-6 tetrahedral partitions Δ for low polynomial degrees q.

7.3.3 Dimension of C^1-Splines on Δ

Given an arbitrary tetrahedral partition $\widetilde{\Delta}$, the dimension of $\mathcal{S}_q^0(\widetilde{\Delta}),\ q\geq 1$, is easy to determine (cf. [Alfeld et al. 1992]). In this case, it is obvious that $\mathcal{D}_{q,\widetilde{\Delta}}$ is a MDS for $\mathcal{S}_q^0(\widetilde{\Delta})$ and a straight forward computation shows that

$$\dim \mathcal{S}_q^0(\widetilde{\Delta}) = \tbinom{q-1}{3}\,T_{\widetilde{\Delta}} + \tbinom{q-1}{2}\,F_{\widetilde{\Delta}} + (q-1)\,E_{\widetilde{\Delta}} + V_{\widetilde{\Delta}}, \qquad q\geq 1,$$

For a type-6 tetrahedral partition Δ of a cubic domain, (7.8) and some elementary computations imply

$$\dim \mathcal{S}_q^0(\Delta) = (4q^2+1)\,q\,n^3 + 6\,q^2\,n^2 + 3\,q\,n + 1, \qquad q\geq 1.$$

More complex arguments are needed to determine the degrees of freedom of C^1-smooth splines. We summarize the results on the dimension of C^1-splines on Δ.

Theorem 7.1 *The dimension of $\mathcal{S}_q^1(\Delta)$ is given by*

$$3\,n^2+9\,n+4, \qquad \textit{if } q=2,$$

and

$$(4q^3-24q^2+53q-45)\,n^3+6\,(2q^2-7q+7)\,n^2+9\,(q-1)\,n+4, \qquad \textit{if } q\geq 3.$$

Table 7.1 compares the dimensions of splines on Δ for low polynomial degrees q.

The proof of Theorem 7.1 is complex, and we refer to [Hangelbroek et al. 2004], here. It roughly proceeds as follows: First, minimal determining sets for C^1-splines on a particular cube cell $Q_{i,j,k} \in \diamondsuit$ is

constructed. Then, a minimal determining set for the whole C^1-spline space is constructed step by step. This is done inductively by considering tetrahedra of the partition in an appropriate order. In each step, the remaining degrees of freedom are determined.

The construction of the minimal determining set gives some deeper insight on the structure of the spaces. We apply these results in the remaining sections to develop appropriate tools for efficient approximation in these spline spaces for high-quality visualization.

7.4 Evaluation of Trivariate Splines on Δ

In this section we discuss some practical issues of the efficient evaluation of the spline s w.r.t. Δ. Given a point $u \in \Omega$ we want to compute the value $s(u)$ and the gradient $(\nabla s)(u)$. In order to apply the de Casteljau algorithm (see Section 7.1), which computes both quantities simultaneously for a polynomial piece $s|_T$, we need to identify the supporting tetrahedron $T \in \Omega$ and local barycentric coordinates. Hence, we have to determine the cube $Q \in \Omega$ with $u \in Q$, the tetrahedron $T \in Q$ with $u \in T$, and the barycentric coordinates $\lambda_\nu(u), \nu = 0, \ldots, 3,$, of u w.r.t. T.

We apply a uniform scaling by $\frac{1}{h}$ such that $\tilde{u} = (\tilde{x}, \tilde{y}, \tilde{z}) := \frac{1}{h}u$. The indices of the cube Q with $u \in Q$ are found by simple rounding of the coordinates of $\tilde{u}$, i.e. we have $Q = Q_{[\tilde{x}],[\tilde{y}],[\tilde{z}]}$, where $[b]$ denotes the maximal integer $\leq b$. The uniformity of Δ allows a translation of $\tilde{u}$ such that the remaining computations can be performed for (the tetrahedral partition of) the unit cube $Q_0 = [-\frac{1}{2}, \frac{1}{2}]^3$, hence from now we may assume that $\tilde{u} \in Q_0$.

For finding the tetrahedron which contains q, we use the observation mentioned in Section 7.2 that the partition of Q_0 in 24 congruent tetrahedra is obtained by slicing with the six planes

$$P_\nu(x, y, z) = 0, \qquad \nu = 0, \ldots, 5, \tag{7.11}$$

where

$$\begin{aligned} P_0(x, y, z) &= x + y, \quad P_1(x, y, z) = x - y, \quad P_2(x, y, z) = x + z, \\ P_3(x, y, z) &= x - z, \quad P_4(x, y, z) = y + z, \quad P_5(x, y, z) = y - z. \end{aligned}$$

The orientation of $\tilde{u}$ with respect to these planes is determined by performing one addition to compute $P_\nu(\tilde{u})$ followed by a sign check for each of the six planes. This gives a 6-bit binary code for the orientation of $\tilde{u}$ and the tetrahedron $T \subseteq Q_0$ with $\tilde{u} \in T$ is found by a simple table lookup. The whole operation requires six additions, six sign checks and five bit shifts.

For determining the barycentric coordinates $\lambda_\nu(\tilde{u}),\ \nu = 0, \ldots, 3$, of $\tilde{u}$ with respect to T, the vertices of $T = [v_0, v_1, v_2, v_3]$ are organized such that v_0 is the origin, v_1 and v_2 are two corner vertices of Q_0, and v_3 is the intersection point of the diagonals in a face of Q_0. We use the precomputed general solution of the system (7.1). For instance, if $v_1 = (-\frac{1}{2}, -\frac{1}{2}, -\frac{1}{2})$, $v_2 = (\frac{1}{2}, -\frac{1}{2}, -\frac{1}{2})$, and $v_3 = (0, 0, -\frac{1}{2})$, we have

$$\lambda_0(\tilde{u}) = 1 + 2\,\tilde{z},\ \lambda_1(\tilde{u}) = -\tilde{x} - \tilde{y},\ \lambda_2(\tilde{u}) = \tilde{x} - \tilde{y},$$

and $\lambda_3(\tilde{u}) = 2\,(\tilde{y} - \tilde{z})$. Similar expressions for the barycentric coordinates involving five of the variable factors from the ordered list

$$L = (L_\nu)_{\nu=0}^{5} = [\tilde{x},\ \tilde{y},\ \tilde{z},\ -\tilde{x},\ -\tilde{y},\ -\tilde{z}]$$

are obtained for the other 23 tetrahedra in Q_0. We exploit this simple fact for generating another lookup table with 24 entries of the precomputed solutions. If T is the tetrahedron from the above example, its entry E in this table is given by

$$E = [\ (2)\ |\ (3,4)\ |\ (0,4)\ ],$$

with the interpretation that the barycentric coordinates of $\tilde{u}$ with respect to T are computed from L by setting

$$\lambda_0(\tilde{u}) = 1 + (L_2 + L_2),\ \lambda_1(\tilde{u}) = (L_3 + L_4),\ \lambda_2(\tilde{u}) = (L_0 + L_4),$$

and $\lambda_3(\tilde{u}) = 1 - \lambda_0(\tilde{u}) - \lambda_1(\tilde{u}) - \lambda_2(\tilde{u})$. These are seven additions and five essential table lookups. In this way, we determine the barycentric coordinates of $\tilde{u}$ (and hence of u) while avoiding expensive rotation or transformation operations, without explicit branching over 24 cases, and without performing any multiplication (not even by -1).

Given the barycentric coordinates of u, we apply the trivariate version of de Casteljau algorithm (see Section 7.1) to evaluate the polynomial piece, which requires a total number of $4\binom{q+3}{4}$ multiplications and $3\binom{q+3}{4}$ additions, in general. We focus on quadratic (cubic) polynomials. Then these are 20 (60) multiplications and 15 (45) additions. If one or even two barycentric coordinates of u vanish, then this algorithm degenerates to its bivariate and univariate versions, respectively. In these cases, u lies in the interior of a triangular face on T or on an edge of T, and the number of necessary arithmetic operations reduces to $12+8$ and $6+3$ ($30+20$ and $12+6$), respectively. The bivariate case occurs in ray casting algorithms (see Section 8.5).

The de Casteljau algorithm simultaneously computes the partial derivatives in direction of the edges of the tetrahedron. The gradient is obtained easily from (7.3) or (7.4) and the basis transformation (7.5). Again, we use a lookup table for precomputed numbers α_μ for the different tetrahedra.

Chapter 8

Reconstruction of Volume Data with Quadratic Super Splines

There are different setups for the problem of reconstruction of volume data. The reconstruction problem is less complex if the data is structured so that the samples are arranged on a regular three-dimensional grid (compared to unstructured, general data as discussed in Section 9. For instance, CT or MRI sensors, seismic applications or results from numerical simulations typically generate this type of gridded data which is subsequently visualized by volume rendering. In Rössl et al. [2003a, 2004a], we propose new models of such gridded volume data, namely quadratic, trivariate super splines on uniform tetrahedral partitions which yield efficient approximations that can be evaluated and implemented easily. This method builds upon the trivariate spline spaces discussed in the previous sections. We describe the precise smoothness and approximation properties of the quasi-interpolating splines and provide comparisons with some alternative approaches to reconstruction based on piecewise polynomials (linear continuous, trilinear continuous, quadratic continuous, and triquadratic smooth splines). These comparisons include the visual appearance of the reconstruction for very few data samples of a well-known benchmark together with a detailed discussion on the approximation aspect that is of fundamental importance for any method.

8.1 Background

Reconstruction of volume data has been an active area of research for the last decades and many different models have been proposed. General reconstruction of a discrete sampling is well studied in signal processing, and Fourier analysis leads to optimal models. However, optimal models are often not feasible in

practice as they are based on global properties. In order to keep computational costs low, *local* methods have been studied extensively (cf. [LaMar et al. 1999, Marschner and Lobb 1994, Mitchell and Netravali 1988, Möller et al. 1998]). It turns out that local (piecewise) polynomial constructions are often preferred due to their simplicity, efficiency and satisfying reconstructions.

In this context, the simplest model is a piecewise constant approximation based on the closest sample or on some averaging of nearby samples. The next natural model is to use trivariate linear polynomials $\sum_{i+j+k\leq 1} a_{i,j,k}\ x^i y^j z^k$, where $a_{i,j,k} \in \mathbb{R},\ i+j+k \leq 1$. Using these functions, a *tetrahedral partition* Δ is needed, and the model becomes a linear *trivariate spline on* Δ (see [Bonneau et al. 1996, Carr et al. 2001a, Gerstner and Rumpf 2000, Grosso et al. 1997], for instance). More sophisticated models are needed if gradient information is required, e.g. for high quality shading. One of the most popular models in volume visualization is to use trilinear interpolants $\sum_{i,j,k=0,1} a_{i,j,k}\ x^i y^j z^k$, where $a_{i,j,k} \in \mathbb{R},\ i,j,k = 0,1$, i.e. piecewise polynomials with *total degree* three of specific type (cf. [Marschner and Lobb 1994, Parker et al. 1998], and the references therein). Approaches of this type often use central differences of the surrounding data samples to faithfully determine the gradient at a given point location.

Alternatively, models satisfying smoothness properties have been constructed. In this case, the necessary gradient information is directly available from the model, but the data stencil needed for the reconstruction generally increases. In some of the local approaches listed above, the models are based on tricubic splines (also known as cubic filters), i.e. piecewise polynomials of the form $\sum_{i,j,k=0}^{3} a_{i,j,k}\ x^i y^j z^k$, where $a_{i,j,k} \in \mathbb{R},\ i,j,k = 0,\ldots,3$. These are special polynomials of total degree nine. Recently, smooth approximation models using triquadratic tensor splines have been proposed to further reduce the polynomial degree. In this case, the data stencil consists of 27 grid points, and 27 coefficients of the form $a_{i,j,k} \in \mathbb{R},\ i,j,k = 0,1,2$ determine each polynomial piece. These methods are based on the piecewise monomial representation (cf. [Barthe et al. 2002, Mora et al. 2001]) or on the B-spline expansion of tensor splines (cf. [Thévenaz and Unser 2001]), and the total degree of the polynomial pieces is six.

Moreover, reconstructions with high smoothness are discussed in [Möller et al. 1998, Thévenaz and Unser 2001], a mathematical framework using NURBS was developed in [Martin and Cohen 2001], and trivariate Coons patches were proposed in [Holliday and Nielson 2000]. In the A-Patch methods (cf. e.g. [Bajaj 1997, Bajaj et al. 1995, Dahmen 1989, Dahmen and Thamm-Schaar 1993] and the references therein) the zero-sets of trivariate piecewise polynomials are used for surface construction. The literature shows that designing an appropriate model for the visualization of volume data is always a compromise

between computational efficiency and visual quality, where the most successful methods are based on local reconstructions. For further information on the field, we refer to the recent books [Bajaj 1999, Chen et al. 2000], the surveys [Brodlie and Wood 2001, Kaufman 2000, Meissner et al. 2000, Nielson 2000, Theußl et al. 2003] and the references therein.

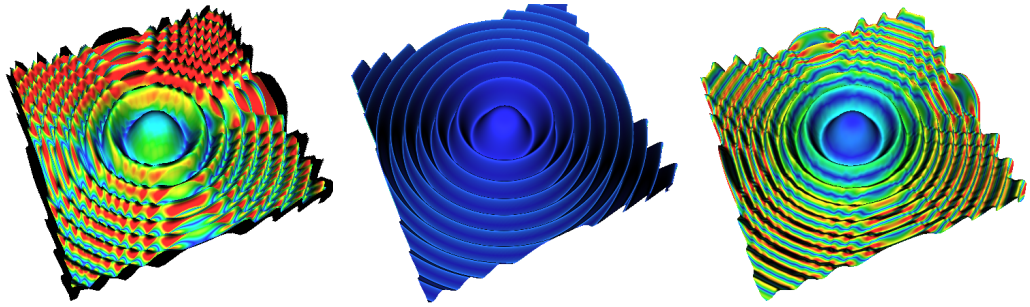


Figure 8.1: Isosurfaces of the synthetic Marschner-Lobb benchmark [Marschner and Lobb 1994] (41^3 samples, isovalue $\frac{1}{2}$, see Section 8.6.1). The approximation error to the original function (see (8.9)) in the uniform norm is color coded (from red≥ 0.075 to blue=0) for the standard trilinear model (left) and our new quadratic super splines (right). The center image shows a visually perfect reconstruction using our model for $(4 \times 41)^3$ samples. The maximum error is 0.0065 (center) compared to 0.088 (right) which illustrates that the quasi-interpolating spline yields nearly optimal approximation order.

8.2 Overview of the Approach

We develop a new approach to efficiently visualize gridded volume data using a *local spline model*. In contrast to the existing approaches, the splines used here are piecewise polynomials of *lowest possible total degree*, namely, the polynomial pieces have the form $\sum_{i+j+k\leq 2} a_{i,j,k}\, x^i y^j z^k$, where $a_{i,j,k} \in \mathbb{R}$, $i+j+k \leq 2$. This means that the total degree is *two*. The *quadratic splines* are defined with respect to a tetrahedral partition Δ. Hence their polynomial pieces are given on tetrahedra. Splines of this natural type have not yet been studied in the context of local volume data reconstruction. Based on our theoretical investigations of the structure concerning smooth trivariate splines of arbitrary degree (cf. Section 7.3.3, [Hangelbroek et al. 2004], and the references therein) and the facts known for bivariate splines (see [Nürnberger and Zeilfelder 2000, Zeilfelder 2002] and the references therein), we choose an appropriate uniform tetrahedral partition Δ (see Section 7.2) and design a *super spline model* which we show to be appropriate for efficient volume visualization. We develop a natural and completely symmetric re-

construction method for these trivariate splines. Their coefficients are computed locally and directly by *repeated averaging* of the given data, while appropriate smoothness properties necessary for the visualization are automatically satisfied. As a non-standard phenomenon the derivatives of the splines yield optimal approximation order for smooth data, while the theoretical error of the values is nearly optimal because of the averaging (see [Nürnberger et al. 2004c]). We take advantage of the (trivariate) *Bernstein-Bézier representation* of the quadratic polynomial pieces. This piecewise representation allows us to exploit the Bernstein-Bézier techniques (see Section 7.1) to efficiently represent, compute, evaluate and visualize our volume spline model.

Our approach allows efficient and high-quality visualization of volume data, which we illustrate by rendering isosurfaces of well-known synthetic and measured test data sets using *ray-casting*. Along an arbitrary ray the quasi-interpolating splines are univariate piecewise quadratics and consequently their exact intersection for a prescribed isovalue can be easily determined in an analytic and precise way by solving *quadratic equations*. Note that all the methods described above (except for those based on constant and linear trivariate splines) need to solve higher order equations through either approximative numerical methods, or – for cubic and quartic equations – implementation of Cardano's and Ferrari's exact formulae, respectively (cf. [Schwarze 1990]). Finally, the gradient, necessary for quality shading, is determined efficiently in our method using Bernstein-Bézier techniques. This direct sampling from the polynomial pieces is motivated by the optimal approximation properties of the derivatives of our new spline model, our examples illustrate the resulting visual quality.

8.3 Reconstruction with Quadratic Super Splines

In the following we are interested in consistent splines which satisfy many smoothness conditions — such splines are called *super splines*. The space of *quadratic super splines with respect to* Δ is defined by

$$\begin{aligned}\mathcal{S}_2(\Delta) = \{\, s \in C(\Omega) :\ & s|_T \in \mathcal{P}_2, \text{ for all } T \in \Delta, \text{ and} \\ & s \text{ is smooth at } v, \text{ for all } v \text{ vertices of } \diamondsuit \,\}.\end{aligned}$$

In our approximation method described below we use quasi-interpolating splines from $\mathcal{S}_2(\Delta)$ which posses many additional natural smoothness properties. Mathematically speaking, this means that we deal with appropriate subspaces of $\mathcal{S}_2(\Delta)$, where the number of free parameters is considerably lower.

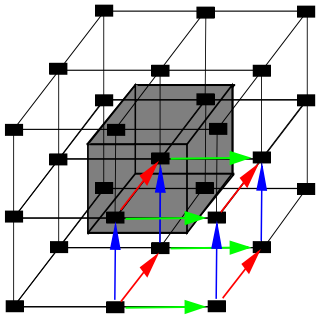


Figure 8.2: In each cube $Q \in \diamondsuit$ (grey) the splines are reconstructed by using a stencil of 27 data samples (black boxes). The derivative of the splines at the grid points of $\diamondsuit$ in each of the three space directions are determined as the average of four differences. For instance, the x derivative at the lower right vertex of Q is obtained from averaging the differences illustrated by the green arrows. Similarly, the y derivative and z derivative at this point are obtained by averaging the differences associated with the red and blue arrows, respectively.

Given gridded volume data, i.e. data points of the form $(\frac{2i+1}{2}, \frac{2j+1}{2}, \frac{2k+1}{2}) \in \mathbb{R}^3$, with corresponding data values $f_{i,j,k} \in \mathbb{R}$, $i,j,k = -1,\ldots,n$, the outline of our reconstruction method is as follows: The coefficients in the piecewise representation (7.2) of the reconstruction s from $\mathcal{S}_2(\Delta)$ are determined by repeated averaging of the data values. First, for every vertex v of $\diamondsuit$, we determine the Bernstein-Bézier coefficients of s close to v by using an averaging of the data values at the center points of the eight cubes which have v as a common vertex. This uniquely determines the value $s(v)$ and the three derivatives $(\frac{\partial s}{\partial x})(v)$, $(\frac{\partial s}{\partial y})(v)$, and $(\frac{\partial s}{\partial z})(v)$. Then, we use repeated averaging of the Bernstein-Bézier coefficients associated with these nodal values at the eight vertices of every cube $Q \in \diamondsuit$ to uniquely determine the 65 coefficients of $s|_Q$ (cf. Table 7.1 for $q = 2, n = 1$) in its piecewise representation (7.2) while satisfying additional natural appropriate smoothness conditions. Hence, similarly to [Barthe et al. 2002, Mora et al. 2001], where a triquadratic tensor spline model is used as a reconstruction, the 27 data values at the centers of the cubes which have a non-empty intersection with Q are needed to reconstruct $s|_Q$ for every cube $Q \in \diamondsuit$ (see Figure 8.2). Note that although in one variable our method would coincide with these approaches, this is completely different in the multivariate case. An illustration of our reconstruction is given in Figure 8.3, where we show the Bézier points from one face of the three different layers within an arbitrary cube Q of $\diamondsuit$. Here, the different colors indicate the order of determining the corresponding coefficients of the splines, as described below. The details of our natural and completely symmetric reconstruction are as follows. Let $Q \in \diamondsuit$ be an arbitrary cube. For every edge e of Q, we

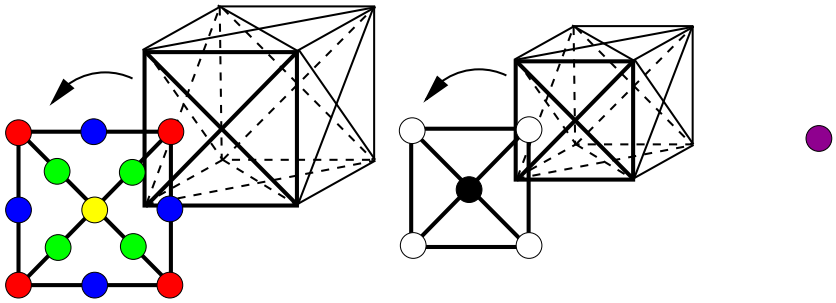


Figure 8.3: A cube is decomposed into three layers of Bézier points. The colored dots show the Bernstein-Bézier coefficients associated with the Bézier points of s for one of the six pyramids — i.e. four tetrahedra — in a cube on each layer. For the whole cube there are 50 coefficients on the outer layer (left), 14 coefficients on the middle layer (center), and the center coefficient (right). These 65 coefficients are determined layer by layer in the following order: blue (a_e), red (a_v), green (a_{m_1}), yellow (a_d), white (a_c), black (a_g), magenta (a_{v_Q}).

first determine the Bernstein-Bézier coefficient a_e associated with the Bézier point at the midpoint of e (blue dot in Figure 8.3). We do this by averaging the four data values $f_0,\ f_1,\ f_2,\ f_3$, which correspond to the data points at the center points of the four cubes in $\diamondsuit$ which share a common edge e, i.e. we set

$$a_e = \tfrac{1}{4}\,(f_0 + f_1 + f_2 + f_3). \tag{8.1}$$

We then determine the Bernstein-Bézier coefficient a_v associated with the Bézier point at every vertex v of Q (red dot in Figure 8.3). This is done by choosing two edges e_1 and e_2 with endpoint v which lie on the same line segment of $\diamondsuit$ and by averaging the two Bernstein-Bézier coefficients $a_{e_1},\ a_{e_2}$, i.e. we set $a_v = \frac{1}{2}\,(a_{e_1} + a_{e_2})$. Note that a_v is uniquely determined and independent from the chosen line segment of $\diamondsuit$ since for each of the three possible choices of edges e_1 and e_2 with endpoint v, we obtain due to uniformity

$$a_v = \tfrac{1}{8}\,(f_0 + f_1 + f_2 + f_3 + f_4 + f_5 + f_6 + f_7), \tag{8.2}$$

where $f_0, \ldots, f_7$, are the data values at the center points of the eight cubes with vertex v. Moreover, the derivatives $(\frac{\partial s}{\partial x})(v)$, $(\frac{\partial s}{\partial y})(v)$, and $(\frac{\partial s}{\partial z})(v)$ are uniquely determined in an automatic way. For instance, it follows from a standard relation (cf. [Farin 1986]) of $(\frac{\partial s}{\partial x})(v)$ with the two Bernstein-Bézier coefficients associated with the points $v = (i, j, k)$ and $(\frac{2i+1}{2}, j, k)$ that

$$\begin{aligned}(\tfrac{\partial s}{\partial x})(v) \;\; = \tfrac{1}{4} \;\; & (f_{i,j-1,k-1} - f_{i-1,j-1,k-1} + f_{i,j,k-1} - f_{i-1,j,k-1} + \\ & f_{i,j-1,k} \;\; - f_{i-1,j-1,k} \;\; + f_{i,j,k} \;\; - f_{i-1,j,k})\,.\end{aligned}$$

This means that $(\frac{\partial s}{\partial x})(v)$ is determined as an average of four simple differences which approximate the derivative in x-direction. Similar interpretations hold for $(\frac{\partial s}{\partial y})(v)$ and $(\frac{\partial s}{\partial z})(v)$ (see Figure 8.2). Note that in contrast to the standard central differences approach for approximating derivative information in volume graphics, similarly as in [Barthe et al. 2002, Mora et al. 2001], no information from an intermediate data sample is lost here.

We proceed by setting the remaining five Bernstein-Bézier coefficients associated with points on each of the six faces of Q. Let $\mathcal{F}$ be a square face of Q, d the point where the two diagonals in $\mathcal{F}$ intersect, and $m_1,\ m_2$, two midpoints of edges in the interior of $\mathcal{F}$ which lie on the same diagonal in $\mathcal{F}$. The Bernstein-Bézier coefficient a_{m_1} associated with the Bézier point m_1 (green dot in Figure 8.3) is determined by averaging the two Bernstein-Bézier coefficients $a_{e_1},\ a_{e_2}$, where e_1 and e_2 are the edges of $\mathcal{F}$ from $\diamondsuit$ which intersect at the vertex of Q closest to m_1, i.e. we set

$$a_{m_1} = \tfrac{1}{2}\,(a_{e_1} + a_{e_2}). \tag{8.3}$$

We determine the coefficient a_{m_2} analogously. Then, we set the Bernstein-Bézier coefficient a_d associated with the Bézier point d (yellow dot in Figure 8.3) as

$$a_d = \tfrac{1}{2}\,(a_{m_1} + a_{m_2}).$$

It is well known in bivariate spline theory (see e.g. [Davydov and Zeilfelder 2003, Nürnberger et al. 2003a, Nürnberger and Zeilfelder 2000]) that a_d is uniquely determined, independently of the two possible choices for m_1 and m_2. Moreover, this setting implies the smoothness within the faces of Q. In particular, the directional derivative $(\frac{\partial s}{\partial \varsigma})(d)$ is uniquely determined, where $\varsigma \neq 0$ is an arbitrary vector in three-dimensional space which lies in the plane through the origin parallel to $\mathcal{F}$.

We proceed by setting the remaining 15 Bernstein-Bézier coefficients associated with points from the interior of Q. First, let c be a midpoint of an edge of Δ which connects the center v_Q with a vertex v of Q, and let e be the common edge of any two faces $\mathcal{F},\ \mathcal{F}^*$ of Q with vertex v. Moreover, let m_1 and m_1^* be the midpoints of the edges in the interior of $\mathcal{F}$ and $\mathcal{F}^*$ with endpoint v, respectively. Using the same notation as above, the Bernstein-Bézier coefficient a_c associated with c (white dot in Figure 8.3) is determined by

$$a_c = (a_{m_1} + a_{m_1^*}) - \tfrac{1}{2}\,(a_v + a_e). \tag{8.4}$$

We note that it follows from Section 7.3.2 (see also [Hangelbroek et al. 2004]) that this setting (together with (8.3)) *now* guarantees that $s \in \mathcal{S}_2(\Delta)$. Moreover, a_c is uniquely determined independent of the three possible choices of $\mathcal{F}$ and

$\mathcal{F}^*$. The coefficient a_g associated with the midpoint g of the edge which connects the intersection point d of the diagonals of a face $\mathcal{F}$ of Q with v_Q (black dot in Figure 8.3) is now determined by setting

$$a_g = \tfrac{1}{4}\,(a_{c_0} + a_{c_1} + a_{c_2} + a_{c_3}),$$

where $c_0, \ldots, c_3$, are the midpoints of the edges which connect the vertices of $\mathcal{F}$ with v_Q. It remains to determine the Bernstein-Bézier coefficient a_{v_Q} at the center v_Q of Q (magenta dot in Figure 8.3). This done by setting

$$\begin{aligned} a_{v_Q} \quad = \quad & \tfrac{1}{3}\,(a_{g_0} + a_{g_1} + a_{g_2} + a_{g_3} + a_{g_4} + a_{g_5}) \\ & -\tfrac{1}{8}\,(a_{c_0} + a_{c_1} + a_{c_2} + a_{c_3} + a_{c_4} + a_{c_5} + a_{c_6} + a_{c_7}), \end{aligned} \tag{8.5}$$

where $g_0, \ldots, g_5$, are the midpoints of the edges which connect the intersection point of the diagonals of the six faces of Q with v_Q, and $c_0, \ldots, c_7$, are the midpoints of the eight edges which connect the vertices of Q with v_Q.

The above settings for a_g and a_{v_Q} are motivated by the fact that they are the average of two and twelve smoothness conditions, respectively, which would have been satisfied simultaneously by an overall smooth spline (cf. [Hangelbroek et al. 2004]), and hence the approximation properties of the model are preserved by an argument of weak-interpolation type, for instance (cf. e.g. [Nürnberger and Zeilfelder 2003]). Now all the coefficients of the spline s are set appropriately. The computation of the 65 coefficients for a single cube $Q \in \diamondsuit$ of $s|_Q$ requires 66 multiplications with constants and 121 additions. The implementation of the model is straightforward. Finally we note that a close inspection shows that the resulting quadratic quasi-interpolating spline s is smooth for all points on the faces of any $Q \in \diamondsuit$, and s yields nearly optimal approximation order while the (piecewise) derivatives of s yield optimal approximation order for data coming from smooth functions.

8.4 Smoothness and Approximation Properties

From Section 7.3.3 we know that the degrees of freedom of the quadratic C^1-spline space w.r.t. Δ is only $3n^2+9n+4$. Unfortunately, this shows that quadratic C^1-splines on Δ do not have enough degrees of freedom to provide appropriate tools for the efficient approximation of three-dimensional data. One reason for this is that the quadratic C^1-splines have to simultaneously satisfy a huge number of smoothness conditions while on the other hand the number of coefficients involved is extremely low.

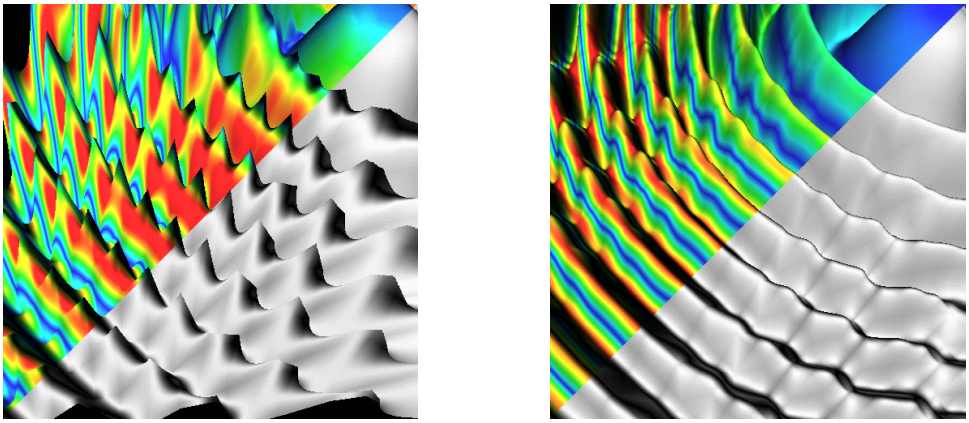


Figure 8.4: Zoomed regions of Figure 8.1 (same color code for upper image parts): Trilinear (left) and our quadratic (right) reconstruction.

One main motivation for our approach was to find a volume reconstruction method which results in approximating, quadratic splines s on Δ while the essential smoothness properties needed for the visualization process are satisfied (see the results from Section 8.6.4). The basic idea presented here is to relax some of the C^1-smoothness conditions which would have been satisfied by an overall quadratic C^1-spline on Δ and to replace *some* of them by other useful conditions, i.e. averages of smoothness conditions (cf. the setting of a_g and a_{v_Q} described in the previous Section 8.3). Due to the simple repeated averaging rules, the coefficients of the splines s can be computed efficiently by using a local procedure involving only small data stencils. We note that the corresponding averaging rules are chosen carefully such that many smoothness conditions are automatically satisfied while certain approximation properties of the quasi-interpolant s are guaranteed. Hence, our approach can be understood as a suggestion for finding a satisfying compromise between visualization and approximation quality using trivariate, quadratic splines.

The coefficients a_g and a_{v_Q} in (8.4) and (8.5) are set as averages of two and twelve smoothness conditions, respectively. To see this, assume the coefficients are indexed as shown in Figure 8.5, and a_{c_i}, $0 \leq i \leq 7$, are already determined as described in the previous Section 8.3. Then for a_{g_0} the two (univariate, intra-pyramid) smoothness conditions

$$a_{g_0} = \tfrac{1}{2}(a_{c_0} + a_{c_2}) \quad \text{and} \quad a_{g_0} = \tfrac{1}{2}(a_{c_1} + a_{c_3})$$

apply. The average of these two conditions gives equation (8.4), and the same arguments apply for the remaining coefficients a_{g_i}, $1 \leq i \leq 5$. In order to show that (8.5) is an average of twelve smoothness conditions which are used to determine the center coefficient a_{v_Q}, we consider all conditions in which a_{v_Q} is involved.

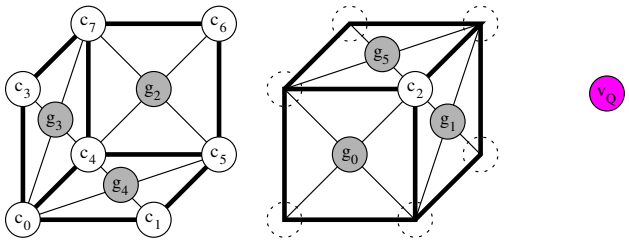


Figure 8.5: The 15 coefficients a_{c_i}, a_{g_j}, a_{v_Q}, $0 \le i \le 7$, $0 \le j \le 5$, of the two inner cube layers from Figure 8.3. The figure labels the Bernstein-Bézierpoints with the indices of the respective coefficients. The coefficients a_{g_j} and a_{v_Q} are determined from averages of two and twelve smoothness conditions, respectively.

These are the twelve (inter-pyramid) conditions

$$
\begin{aligned}
a_{v_Q} &= (a_{g_0} + a_{g_4}) - \tfrac{1}{2}(a_{c_0} + a_{c_1}), & a_{v_Q} &= (a_{g_1} + a_{g_4}) - \tfrac{1}{2}(a_{c_1} + a_{c_5}),\\
a_{v_Q} &= (a_{g_2} + a_{g_4}) - \tfrac{1}{2}(a_{c_5} + a_{c_4}), & a_{v_Q} &= (a_{g_3} + a_{g_4}) - \tfrac{1}{2}(a_{c_4} + a_{c_0}),\\
a_{v_Q} &= (a_{g_0} + a_{g_1}) - \tfrac{1}{2}(a_{c_1} + a_{c_2}), & a_{v_Q} &= (a_{g_5} + a_{g_1}) - \tfrac{1}{2}(a_{c_2} + a_{c_6}),\\
a_{v_Q} &= (a_{g_2} + a_{g_1}) - \tfrac{1}{2}(a_{c_6} + a_{c_5}), & a_{v_Q} &= (a_{g_0} + a_{g_5}) - \tfrac{1}{2}(a_{c_2} + a_{c_3}),\\
a_{v_Q} &= (a_{g_3} + a_{g_5}) - \tfrac{1}{2}(a_{c_3} + a_{c_7}), & a_{v_Q} &= (a_{g_2} + a_{g_5}) - \tfrac{1}{2}(a_{c_7} + a_{c_6}),\\
a_{v_Q} &= (a_{g_0} + a_{g_3}) - \tfrac{1}{2}(a_{c_3} + a_{c_0}), & a_{v_Q} &= (a_{g_2} + a_{g_3}) - \tfrac{1}{2}(a_{c_4} + a_{c_7}),
\end{aligned}
$$

and summation leads to

$$
12\, a_{v_Q} \;=\; 4 \sum_{0 \le i \le 5} a_{g_i} \;-\; \tfrac{3}{2} \sum_{0 \le i \le 7} a_{c_i}\ ,
$$

which is equivalent to (8.5). The careful choice of these rules enables high-quality visualization and nearly optimal approximation.

In the following we summarize smoothness and approximation properties. The complete proofs are provided in [Nürnberger et al. 2004c] (see also Appendix B).

The spline $s \in \mathcal{S}_2(\Delta)$ satisfies the essential smoothness properties needed for the visualization process.

Theorem 8.1 *(smoothness) The spline $s_f \in \mathcal{S}_2^0(\Delta)$ is in $C^1(v)$ for all points v in Ω of the form $v = (ih, y, z)$ or $v = (x, ih, z)$ or $v = (x, y, ih)$, where $i \in \{0, \ldots, n\}$ and $x, y, z \in [0, 1]$.*

This can be shown by reconstruction from samples $f_{i,j,k}, i, j, k \in \{-1, 0, 1\}$ and checking the individual C^1-conditions. Hence, s is not only smooth at the vertices of $\diamondsuit$ but at all points in $\mathbb{R}^3$ from the planes $x = i,\ i = 0, \ldots, n,\ y =$

$j,\ j = 0, \ldots, n$, and $z = k,\ k = 0, \ldots, n$. Moreover, our approximation approach can be understood as a quasi-interpolation as well as a Hermite-interpolation type method.

Concerning the approximation properties, the splines s yield *nearly optimal approximation order*, while its derivatives yield *optimal approximation order* of smooth functions f. More precisely:

Theorem 8.2 *(approximation properties) Let $f \in C(\Omega^*)$. Then the following statements hold.*
(i) If $f \in W_\infty^2(\Omega^*)$, then we have for all $T \in \Delta$,

$$\begin{aligned} &\|(f - s_f)\|_T \leq 5\ |f|_{2,\infty,\Omega^*}\ h^2, \\ &\|D_x^\alpha D_y^\beta D_z^\gamma (f - s_f)\|_T \leq 111\ |f|_{2,\infty,\Omega^*}\ h, \qquad \alpha + \beta + \gamma = 1. \end{aligned}$$

(ii) If $f \in W_\infty^3(\Omega^*)$, then we have for all $T \in \Delta$,

$$\begin{aligned} &\|D_x^\alpha D_y^\beta D_z^\gamma (f - s_f)\|_T \leq 97\ |f|_{3,\infty,\Omega^*}\ h^2, \qquad \alpha + \beta + \gamma = 1, \\ &\|D_x^\alpha D_y^\beta D_z^\gamma (f - s_f)\|_T \leq 422\ |f|_{3,\infty,\Omega^*}\ h, \qquad \alpha + \beta + \gamma = 2. \end{aligned}$$

Here, we denote by $W_\infty^m(\Omega^*)$ the usual *Sobolev space* with the semi-norm

$$|f|_{m,\infty,\Omega^*} = \sum_{\alpha+\beta+\gamma=m} \|D_x^\alpha D_y^\beta D_z^\gamma\ f\|_{\Omega^*},$$

where $D_x^\alpha D_y^\beta D_z^\gamma\ f,\ \alpha + \beta + \gamma = m$, denote the m-th derivatives of a function $f \in W_\infty^m(\Omega^*)$. Appendix B sketches the proof of the theorem.

The above is a non-standard mathematical phenomenon. The key to show (ii) is the following quasi-interpolation property of the spline s_f:

Theorem 8.3 *(reproduction of polynomials) The following statements hold.*
(i) If $p \in$ span$\{1, x, y, xy, xz, yz\}$, then $s_p \equiv p$.
(ii) If $p \in$ span$\{x^2, y^2, z^2\}$, then $s_p \equiv p + \frac{1}{4}h^2$.
(iii) If $p \in \mathcal{P}_2$, then $s_p \equiv p + c$, where c is some constant.

(i) and (ii) can be shown by sampling the polynomials and reconstruction as described in Section 8.3. Note that the construction of the splines is completely symmetric, hence it is sufficient to consider the polynomials 1, x, xy, and x^2. (iii) is an immediate consequence of (i) and (ii): As the derivative $D_x^\alpha D_y^\beta D_z^\gamma\ p$, $\alpha + \beta + \gamma = 1, 2$, of the quadratic polynomial $p \in \mathcal{P}_2$ is either a linear or a constant polynomial, it follows that

$$D_x^\alpha D_y^\beta D_z^\gamma\ s_p \equiv D_x^\alpha D_y^\beta D_z^\gamma\ (p + c) \equiv D_x^\alpha D_y^\beta D_z^\gamma\ p \equiv s_{D_x^\alpha D_y^\beta D_z^\gamma\ p}.$$

In other words, we may swap the order of the differential and the reconstruction operator.

8.5 Visualization: Isosurface Rendering by Precise Ray-Casting

A visualization technique for volume data frequently used in computer graphics is *rendering isosurfaces* from a given reconstruction model. *Ray-casting* is an image-space technique to compute particular views of these surfaces (see e.g. [Foley et al. 1996]). Other methods such as the marching cubes algorithm are described e.g. in [Brodlie and Wood 2001, Chen et al. 2000, Lorensen and Cline 1987]. Ray-casting considers the model along arbitrary rays r,

$$r = r(t): \quad t \mapsto q_0 + t\, r_0, \qquad t \geq 0, \tag{8.6}$$

where the goal is to find the smallest (intersection) parameter $t^* \geq 0$, such that the model along r coincides with a prescribed *isovalue*. Here, $q_0 \in \mathbb{R}^3$ is the position of the viewer and $r_0 \in \mathbb{R}^3$ is the (normalized) viewing direction determined as the difference of the current pixel position in the projection plane and q_0. Therefore $q^* = r(t^*)$ is the point closest to the viewer position, where the model intersects the *isosurface*. A standard *ray-casting algorithm* generates rays through all pixel positions, examines the model along each ray in order to find the closest intersection point q^* with the isosurface, and (if q^* exists) finally evaluates the gradient for proper shading of the isosurface at the current pixel position.

In order to show the potential of our method for efficient visualization of volume data, we apply ray-casting on the reconstruction model $s \in \mathcal{S}_2(\Delta)$ from Section 8.3. In the following, we focus on the specific advantages of our model in contrast to other reconstructions, namely the efficient and exact computation of the intersection point q^* of s along r, and the effective determination of exact gradient information at q^*. Since the approximation s along r is a quadratic univariate spline, and by the choice of the underlying space $\mathcal{S}_2(\Delta)$, it follows that these computations can be made by solving a very simple equation and applying the tools described in Sections 7.1 and 7.4. This uniquely distinguishes our approach from the previously developed methods.

Let r be an arbitrary ray as in (8.6), and let us assume that $Q \in \diamondsuit$ lies within the current region of interest when casting r through Ω. This means that r intersects Q at two points. In the following, we call these points enter and exit points of Q, respectively. We must then process all the tetrahedra in Q which intersect r. A naive approach would be to intersect r with the six cutting planes from (7.11) and to obtain a sequence of all intersection points with the tetrahedra in Q by sorting the (non-negative) ray parameters. In order to avoid unnecessary computations, we first determine a tetrahedron T_0 in Q from the enter point of Q as described in Section 7.4. The intersected face of T_0 is axis aligned. In this case, the second

intersection point of r with T_0 lies in another non-axis aligned face of T_0.

The three candidate faces lie in one of the six cutting planes from (7.11). If needed, we analogously determine another tetrahedron in Q containing the second intersection point from T_0, and proceed similarly. We eventually iterate until r meets the tetrahedron which contains the exit point of Q. As Q is sliced by six planes, previously computed results can be reused here, and we calculate at most six intersection parameters at a cost of two additions and one division each.

Given r as in (8.6) and a prescribed isovalue which we may assume to be zero, for the current tetrahedron $T \in \Delta$, we have to determine the closest point $q^* \in T$ to the viewer, where the trivariate polynomial piece $p = s|_T \in \mathcal{P}_2$ vanishes along r, and we have to find out quickly when such a point q^* does not exist in T. Let $q_1 = r(t_1)$ and $q_2 = r(t_2)$, where $t_1 < t_2$, be two intersection points of r with T. Then, the restriction of p to the line segment $[q_1, q_2]$ is a quadratic, univariate polynomial (see Figure 7.1). It is therefore obvious that we only have to consider a quadratic equation, whose roots can be found in an analytic way with only small computational effort. For setting up the necessary equation, we first compute the values $w_1,\ w$, and w_2 of p at the three points $q_1,\ q = \frac{q_1+q_2}{2}$, and q_2 in T, i.e. $w_1 = p(q_1),\ w = p(q)$, and $w_2 = p(q_2)$. This is done by applying de Casteljau's algorithm from Section 7.1. We access the 10 coefficients of p via an index table into the 65 coefficients for the whole cube. Since the points q_1 and q_2 both lie within a triangular face of T, we first perform the bivariate version of the de Casteljau algorithm twice. The third run of the algorithm is done for the point q. This is the only run which is of trivariate type, in general. We use some previously computed results such that the total number of required operations reduces to 15 multiplications and 13 additions. Note that except for the tetrahedron T_0 (containing the enter point of a cube Q) the above bivariate version of de Casteljau's algorithm has to be performed only once per tetrahedron, since the second intersection point q_2 of T becomes a point of type q_1, when we move on to the adjacent tetrahedron. The intersection point q^* is now determined as follows. Using a precomputation of Newton's interpolation form, we find the unique quadratic polynomial on an appropriate interval $[0, \delta]$, which interpolates the three values $w_1,\ w$, and w_2 at the points $0,\ \frac{\delta}{2}$ and δ. From this, we obtain the quadratic equation

$$\alpha\, \tau^2 + \delta\, \beta\, \tau + \delta^2\, \gamma = 0, \qquad \tau \in [0, \delta], \tag{8.7}$$

where $\alpha = 2\,(w_1 + w_2 - 2\,w),\ \beta = 4\,w - 3\,w_1 - w_2$, and $\gamma = w_1$. Hence, once $w_1,\ w$ and w_2 are determined, the equation (8.7) is set up by using 10 additions. If (8.7) degenerates to a linear equation, i.e. $\alpha = 0$, we obtain

$$t^* = t_1 + \tfrac{\delta}{2}\,(\tfrac{w_1}{w_1 - w})\,(t_2 - t_1).$$

Otherwise, we get

$$t^* = t_1 + \tfrac{\delta}{2\alpha}\,\left(-\beta \pm \sqrt{\beta^2 - 4\,\alpha\gamma}\right)\,(t_2 - t_1), \tag{8.8}$$

and we choose the (smaller) solution in $[t_1, t_2]$ to fix q^*, if it exists. The latter is not the case if $\beta^2 - 4\,\alpha\gamma < 0$, or, otherwise, if the solution(s) from the above equations do not lie in $[t_1, t_2]$. Note that depending on α, β and γ the solution can always be determined in a numerically stable way, switching to another formula of same type, if needed. The necessary arithmetic operations are at most 5 multiplications, 6 additions and one square root evaluation.

Still, in the worst case, all the tetrahedra $T \subseteq Q$ along r have to be processed in order to check if $s|_Q$ is *not* intersected by the isosurface. We can easily accelerate this process by applying a quick conservative test on whether s restricted to r cannot intersect the isosurface locally in a tetrahedron or in a cube. If $p = s|_T$ is given in the form (7.2), then we check $\sigma\ a_{i,j,k,l} > 0,\ i + j + k + l = 2$, where $\sigma \in \{-1, 1\}$. If this sign criterion is satisfied, then we do not have to consider T and can skip it because of the well-known *convex hull property* of the Bernstein-Bézier form. A similar test can be applied to all the 65 coefficients of $s|_Q$, where the minimum and maximum coefficients can be precomputed and stored for each cube, e.g. in a min-max-octree for optimized ray-casting with eventually varying isovalues.

Once an intersection point $q^* = r(t^*)$ is found, we determine the gradient $(\nabla p)(q^*)$ as defined in (7.6) following Section 7.4. A well-known result from differential geometry shows that the normal vector n^* at q^* is given by $n^* = (\nabla p)(q^*)/\|(\nabla p)(q^*)\|$. The normal n^* is required for shading computations, e.g. using the standard Phong illumination model. The results given in Section 8.6.4 show that the isosurfaces are visually smooth due to the high quality normals obtained from the local gradients from our spline model.

The evaluation of roots along a ray is exact and inexpensive for quadratic polynomials, non-trivial for cubics [Schwarze 1990] (trilinear) and analytically impossible for degree six polynomials (triquadratic), i.e. a numerical root finding algorithm must be applied. In addition, the univariate quadratic polynomials allow efficient integration by applying quadrature formulae and evaluation of the extreme values along a ray. The necessary computations can be performed in a straightforward way by following the method from Section 8.5.

8.6 Results

In this section we present results which on the one side confirm the theoretical properties of our reconstruction method on synthetically generated data sets. On

the other side we show that the method enables efficient visualization of real-world data.

8.6.1 Synthetic Benchmarks

We apply the following smooth test functions to obtain synthetic data for numerical tests.

The *Marschner-Lobb* test function (cf. [Marschner and Lobb 1994]) is frequently used as benchmark in volume visualization. It is defined as $ml : [-1,1] \times [-1,1] \times [-1,1] \mapsto \mathbb{R}$ with

$$ml(v) \;=\; (1 - \sin\frac{\pi z}{2} \;+ \alpha\,(1 + \cos(2\pi f_M \cos\frac{\pi r}{2}))/(2(1+\alpha)), \quad (8.9)$$

for all $v = (x,y,z) \in [-1,1] \times [-1,1] \times [-1,1]$, where $r := \sqrt{x^2+y^2}$, $\alpha = 1/4$ and $f_M = 6$. The function is extremely oscillating and therefore a difficult test for any efficient three-dimensional reconstruction method. This concerns in particular the cases when only very few data is taken and simultaneous derivative approximation plays a role. Figures 8.1 and 8.4 show isosurfaces of ml, the sampling on the $41 \times 41 \times 41$ grid is near the Nyquist rate of the function ml, which makes the reconstruction a challenging test for any approximation method.

As a second test we use the smooth trivariate test function of exponential type (cf. [Holliday and Nielson 2000])

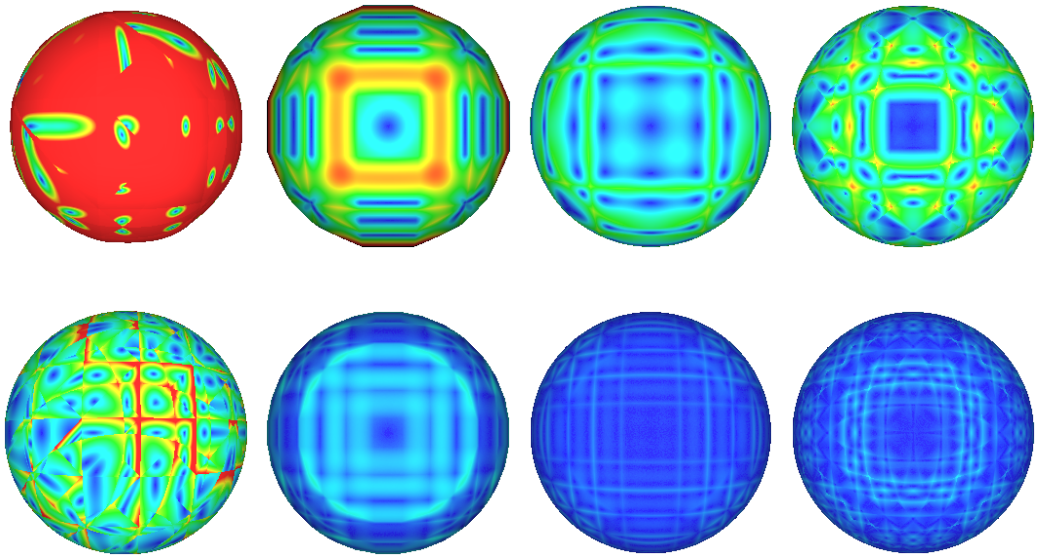
$$\begin{aligned} f_{\text{test}}(v) \;&=\; \tfrac{1}{2}\, e^{-10((x-\frac{1}{4})^2+(y-\frac{1}{4})^2))} \\ &+\; \tfrac{3}{4}\, e^{-16((x-\frac{1}{4})^2+(y-\frac{1}{4})^2+(z-\frac{1}{4})^2))} \\ &+\; \tfrac{1}{2}\, e^{-10((x-\frac{3}{4})^2+(y-\frac{1}{8})^2+(z-\frac{1}{2})^2))} \\ &-\; \tfrac{1}{4}\, e^{-20((x-\frac{3}{4})^2+(y-\frac{3}{4})^2))}, \end{aligned} \quad (8.10)$$

for $v = (x,y,z) \in [-\frac{1}{2},\frac{1}{2}] \times [-\frac{1}{2},\frac{1}{2}] \times [-\frac{1}{2},\frac{1}{2}]$. This function does not have the same extreme oscillatory behavior as the Marschner-Lobb function. Figure 9.4 shows different isosurfaces of f_{test}.

8.6.2 Numerical Tests on the Approximation

In order to illustrate the approximation and smoothness properties of the splines, we give some numerical examples.

We first consider the Marschner-Lobb benchmark (8.9). We compute the quasi-interpolating splines s_{ml} as described in Section 8.3 for decreasing side length h of the cubes and consider different kinds of errors. The numerical results



(a) (b) (c) (d)

Figure 8.6: Isosurfaces showing the error of the respective models' gradient for the (a) piecewise quadratic, continuous spline on Δ_F (cf. Section 8.6.3), (b) trilinear, (c) for triquadratic reconstruction and (d) for our quadratic super spline model. For the top row 8^3 samples are applied in contrast to 16^3 gridded data points for the bottom row. The samples are taken from a spherical function $f(x,y,z) = ||(x,y,z)||$, $(x,y,z) \in [-\frac{1}{2},\frac{1}{2}]^3$. The angular deviation from the perfect gradient ∇f is color coded (from red$\geq 1^\circ$ to blue=0°) on the isosurface $f(x,y,z) = 0.4$. As expected, the underlying grid structure imposes visible artifacts for this extreme diagram.

are given in Table 8.1. The first column of this table contains h, the remaining columns contain different types of errors. Here, we denote by

$$err^{ml}_{data} := \max\{|(ml - s_{ml})((2i+1, 2j+1, 2k+1)\, h/2)| : \; i,j,k = 0,\ldots,n-1\}$$

the maximal error at the gridded data points. In addition, the maximal error in the uniform norm is given as

$$err^{ml}_{max} := \max\{|(ml - s_{ml})(v)| : \; v = (x,y,z) \in \Omega\}.$$

The latter error is computed approximatively as follows. We choose 10 uniformly distributed points in each tetrahedron of Δ, and denote the set of all points from Ω chosen in this way by $\mathcal{V}$. Then the error err^{ml}_{max} is approximatively given as the maximal error at the points from $\mathcal{V}$. Note that we did the same test with higher numbers of (scattered) points, and the results were similar. Moreover, we give the

h	err^{ml}_{mean}	err^{ml}_{rms}	err^{ml}_{max}	err^{ml}_{data}
1/16	0.0637338	0.0763573	0.1794933	0.0886842
1/32	0.0469557	0.0548679	0.1202057	0.0923545
1/64	0.0175995	0.0206885	0.0395824	0.0368751
1/128	0.0049393	0.0058288	0.0105322	0.0103358
1/256	0.0012735	0.0015042	0.0026710	0.0026593

Table 8.1: Approximation of the Marschner-Lobb test function ml by s_{ml}.

h	$err^{D_x ml}_{mean}$	$err^{D_x ml}_{rms}$	$err^{D_x ml}_{max}$	$err^{D_x ml}_{data}$
1/16	4.0446910	5.1182602	12.3920158	10.0356594
1/32	3.2655525	4.3084310	12.8719495	12.5411960
1/64	1.2419448	1.6807952	5.6377440	5.6073735
1/128	0.3483834	0.4733294	1.6051045	1.5936699
1/256	0.0896466	0.1220021	0.4144013	0.4115753

Table 8.2: Approximation of the first derivative $D_x ml$ of ml by $D_x s_{ml}$.

(approximative) average error

$$err^{ml}_{mean} := \tfrac{1}{\#\mathcal{V}} \sum_{v \in \mathcal{V}} |(ml - s_{ml})(v)|,$$

where $\#\mathcal{V}$ denotes the cardinality of $\mathcal{V}$, and the (approximative) root mean square error,

$$err^{ml}_{rms} := \sqrt{\tfrac{1}{\#\mathcal{V}} \sum_{v \in \mathcal{V}} ((ml - s_{ml})(v))^2}.$$

The results in Table 8.1 show that the quasi-interpolating splines yield approximation order two, since in each step the error decreases by about the factor four.

h	$err^{D_x^2 ml}_{mean}$	$err^{D_x^2 ml}_{rms}$	$err^{D_x^2 ml}_{max}$	$err^{D_x^2 ml}_{data}$
1/16	329.2754617	460.6656652	1381.6373930	1140.3801860
1/32	285.6398127	405.5817230	1504.0177006	1008.4225786
1/64	149.7005648	217.5784357	1188.2312690	427.7408473
1/128	70.6715715	107.0387996	656.3764722	239.3863193
1/256	34.0443422	53.1077465	335.8704193	123.7781568

Table 8.3: Approximation of the second derivative $D_x^2 ml$ of ml by $D_x^2 s_{ml}$.

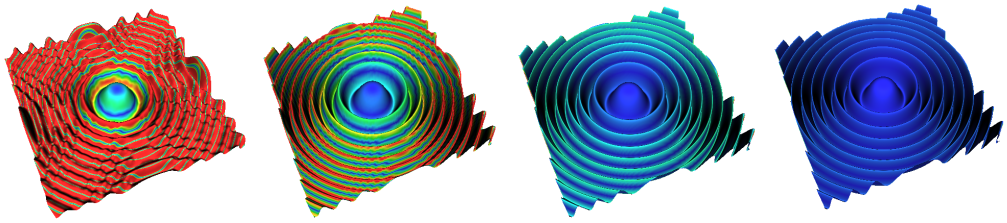


Figure 8.7: Reconstruction of the Marschner-Lobb test function with quadratic super splines for different regular sampling grids. The pictures show isosurfaces (from left, isovalue $\frac{1}{2}$) for the grid spacings $h = 2^{-5}, h = 2^{-6}, h = 2^{-7}$, and $h = 2^{-8}$ with the approximation error to the original function in the uniform norm color coded (from red≥ 0.03 to blue$= 0$). For $h = 2^{-8}$ the maximum error of the spline to the function is 0.00267.

Figure 8.7 visualizes the approximation error for the reconstruction of ml from different sample grids. Moreover, Table 8.2 and 8.3 show that the first derivative $D_x ml$ of ml behaves in the same way (i.e. is optimal), and that for the second derivative $D_x^2 ml$ the error decreases by about the factor two, respectively. This confirms the theoretical results presented in Theorem 8.2. Note that we did the same test with the remaining first and second derivatives and the results were similar. Moreover, we remark that in our method we do not use any derivatives of ml at prescribed points — only functional values are needed. Hence, using the data of a smooth function, the derivatives are simultaneously approximated by the splines. This has to do with the theoretical observation that our method automatically generates approximative derivatives from the given gridded data (see Section 8.3).

In a second test we use the test function (8.10). In Table 8.4 , we summarize the results of our computations for these test functions, where we use analogous notations for the different errors as in the above test. In addition, we compute the analogous errors for the derivatives $D_x^\alpha(s_{f_i} - f_i),\ i = 1, 2,\ \alpha = 1, 2$, which are given in Tab. 8.5 and 8.6, respectively. Again, this confirms the result in Theorem 8.2: the derivatives of the approximating splines s_{f_i} converge optimally to the derivatives of f_i for h tending to zero.

Further numerical tests are provided in [Nürnberger et al. 2004c].

8.6.3 Comparison to Other Methods

We compare to some alternative, straightforward reconstruction methods based on piecewise quadratic polynomials, like [Marschner and Lobb 1994, Parker et al. 1998, Barthe et al. 2002,

h	$err_{mean}^{f\text{test}}$	$err_{rms}^{f\text{test}}$	$err_{max}^{f\text{test}}$	$err_{data}^{f\text{test}}$
1/16	0.0038309	0.0066271	0.0429973	0.0430021
1/32	0.0009200	0.0016158	0.0109885	0.0109880
1/64	0.0002248	0.0003975	0.0027622	0.0027620
1/128	0.0000555	0.0000985	0.0006915	0.0006914
1/256	0.0000138	0.0000245	0.0001729	0.0001729

Table 8.4: Approximation of the function f_{test} by $s_{f_{\text{test}}}$.

h	$err_{mean}^{D_x f\text{test}}$	$err_{rms}^{D_x f\text{test}}$	$err_{max}^{D_x f\text{test}}$	$err_{data}^{D_x f\text{test}}$
1/16	0.0203633	0.0333122	0.2238992	0.1562143
1/32	0.0050603	0.0083125	0.0603967	0.0393916
1/64	0.0012533	0.0020636	0.0152990	0.0098670
1/128	0.0003114	0.0005133	0.0038459	0.0024764
1/256	0.0000776	0.0001280	0.0009622	0.0006192

Table 8.5: Approximation of $D_x f_{\text{test}}$ by $D_x s_{f_{\text{test}}}$.

h	$err_{mean}^{D_x^2 f\text{test}}$	$err_{rms}^{D_x^2 f\text{test}}$	$err_{max}^{D_x^2 f\text{test}}$	$err_{data}^{D_x^2 f\text{test}}$
1/16	0.6418473	0.9323099	6.6045684	5.8388519
1/32	0.3094677	0.4531853	3.3131101	3.0231247
1/64	0.1524176	0.2239878	1.6240159	1.5090936
1/128	0.0757079	0.1114145	0.8012538	0.7509042
1/256	0.0377399	0.0555706	0.3980345	0.3742364

Table 8.6: Approximation of $D_x^2 f_{\text{test}}$ by $D_x^2 s_{f_{\text{test}}}$.

Mora et al. 2001, Thévenaz and Unser 2001]. We remark that these methods have the same theoretical approximation order (cf. Theorem 8.2 (i)) for the error, hence up to a constant scaling numerical tests would look similar. However, we use lower degree piecewise polynomials, and it is not always clear whether these methods provide an error bound for the derivatives as given in Theorem 8.2 (ii).

Considering the existing methods in the literature on visualization, higher approximation orders can be obtained by using piecewise polynomials of degree nine (see e.g. [Marschner and Lobb 1994, LaMar et al. 1999, Mitchell and Netravali 1988, Möller et al. 1998]) assuming that the necessary approximative derivatives possess proper weak-interpolation properties (see [Nürnberger et al. 2003b]).

A straightforward method would use piecewise quadratic, continuous splines (with no smoothness properties) interpolating at all Bézier points of a Freudenthal (6-fold) tetrahedral partition Δ_F of a cube partition (see e.g. [Carr et al. 2001a]).

This obviously yields optimal approximation order for the quadratic splines as well as for its derivatives, assuming that the data comes from a C^3-function. A visual comparison for the Marschner-Lobb benchmark of linear, trivariate splines on Δ, quadratic, continuous trivariate splines on Δ_F and triquadratic C^1-splines, i.e. approximations by piecewise polynomials of total degree six, is provided in Figure 8.8. Figure 8.9 illustrates that all models provide effective approximation. (Both figures use the same color code as Figure 8.1). Figure 8.6 visualizes the quality of the gradients. Note that the Freudenthal partition Δ_F is defined with respect to cubes with edge length $2h$, i.e. there are data points given on the edges and on the faces of the cubes in order to guarantee the optimal approximation properties. Also, this partition requires the choice of a major diagonal and is hence not symmetric which might be one reason for the direction dependent artifacts in the reconstruction.

8.6.4 Visualization of Volume Data with Quadratic Super Splines

We applied our new reconstruction by quadratic super splines to a number of well-known volume data sets. The figures show the visualization of isosurfaces using classical perspective ray-tracing as previously outlined. All local calculations such as evaluation and intersection are performed efficiently. However, our overall ray-casting algorithm is not yet tuned for speed and not competitive to more sophisticated systems like e.g. [Barthe et al. 2002, Parker et al. 1998] which may even aim towards interactive frame rates (see [Wald and Slusallek 2001, Wald 2004] for a recent survey). There are numerous optimizations of the general ray-casting

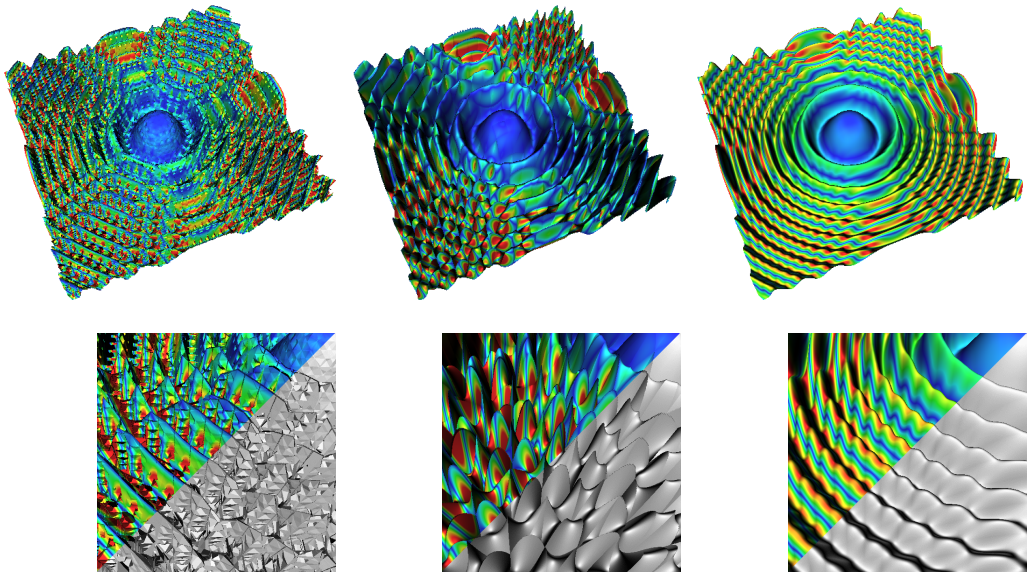


Figure 8.8: Alternative reconstructions of the Marschner-Lobb test function from 41^3 samples. Each row shows the isosurfaces (isovalue $\frac{1}{2}$) on the overall domain and a zoomed region for comparison with Figures 8.1 and 8.4 (same color code), respectively. From left: piecewise *linear*, continuous reconstruction on Δp; piecewise *quadratic*, continuous reconstruction on Δ_F (both: interpolation); *triquadratic* C^1 reconstruction (approximation).

algorithm, any optimization can be combined with our model in a straightforward way with a direct benefit in ray-casting performance. In particular, this includes hierarchical space partitioning or efficient cube traversal by an object-order ray-casting algorithm as applied for triquadratic tensor spline models (cf. [Barthe et al. 2002, Mora et al. 2001]).

We perform a simple preprocessing of the data for a given isovalue, precomputing all cubes and keeping only the relevant ones in memory, i.e. those which potentially intersect the isosurface (typically only some few percent for our experiments). This allows us to provide timings for the construction of a single cube and to estimate a faithful lower bound for more sophisticated preprocessing as the generation of a min-max-octree. All runtimes are measured on a 2.8GHz Intel Xeon CPU, where we observe $0.27\mu s$ for the construction of the spline on a single cube (Section 8.3) plus an average of $0.13\mu s$ for the convex hull tests to determine the relevance of a cube (Section 8.5). We report per frame timings for quadratic reconstruction (average $38.7\mu s$ per ray), as well as the isovalues and the percentages of relevant (and precomputed) cubes for the Figures 8.10, 8.11 8.12, and 8.13 which are rendered into a 512×512 viewport. For all respective figures, we computed high-quality, non-local gradients on the trilinear model (see below)

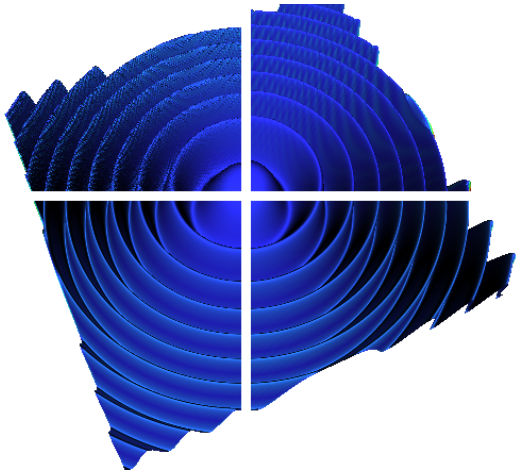


Figure 8.9: Different reconstructions of the Marschner-Lobb test function from $(4 \times 41)^3$ samples for comparison with Figure 8.1 (center). Top left quadrant: piecewise *linear*, continuous reconstruction on Δ. Top right: piecewise *quadratic*, continuous reconstruction on Δ_F (both: interpolation). Bottom right: *triquadratic* C^1 reconstruction. Bottom left: our quadratic super spline model (both: approximation).

to ensure a fair visual comparison. The difference between the models becomes most visible for high frequency areas (e.g. arteries, leaves of the bonsai) with a feature size of only few samples. Figures 8.1 and 8.4 show a synthetic benchmark. And the experiment visualized by Figure 8.6 emphasizes the quality of the gradients.

Regarding the number of floating point operations, our quadratic approach is close to the simple trilinear interpolation and much cheaper than a triquadratic model. The same is true for the computation of the gradients. However, as the trilinear model does not satisfy smoothness conditions, local gradient evaluation is inexact for general data, while the costs for better gradients such as using central differences from evaluation in six neighboring cells (as used here) is more expensive. The price for our approach is a slight overhead of point location in a tetrahedron and the requirement of 65 coefficients instead of 27 (triquadratic) or working directly on the data (trilinear).

For our experiments we only store a fraction of the cubes in memory. However, it is clear that for the complete spline (even though not necessary for the visualization) far less than $65n^3$ coefficients are needed. In this case computation time can be balanced against storage and memory bandwidth depending on the application by precomputing and storing only certain coefficients which allow for faster local reconstruction. For instance, precomputing and storing only the coefficients on the vertices (a_v) and edges (a_e) of the cubes results in a total memory requirement of $4n^3$ coefficients (the original data is not needed anymore).

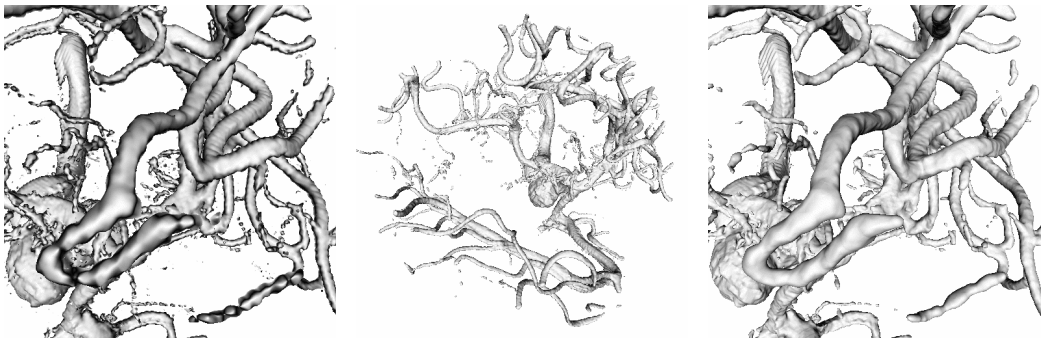


Figure 8.10: Isosurface of the *aneurism* data set (courtesy of Philips Research, Hamburg, Germany; 256^3 samples, isovalue 50). Trilinear (left) and our quadratic (center and right) reconstruction ($12.7s$ and $11.7s$, 0.66% relevant cubes).

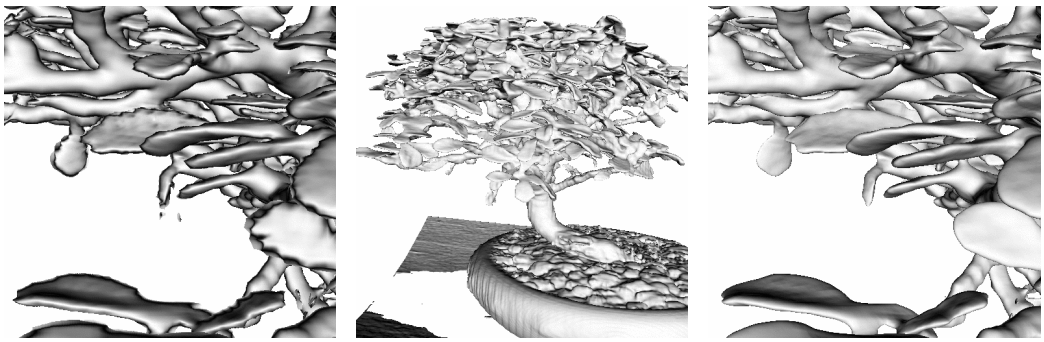


Figure 8.11: Isosurface of the *bonsai* data set (courtesy of Stefan Röttger, VIS, University of Stuttgart, Germany; 256^3 samples, isovalue 40). Trilinear (left) and our quadratic (center and right) reconstruction ($7.1s$ and $9.1s$, 3.1% relevant cubes).

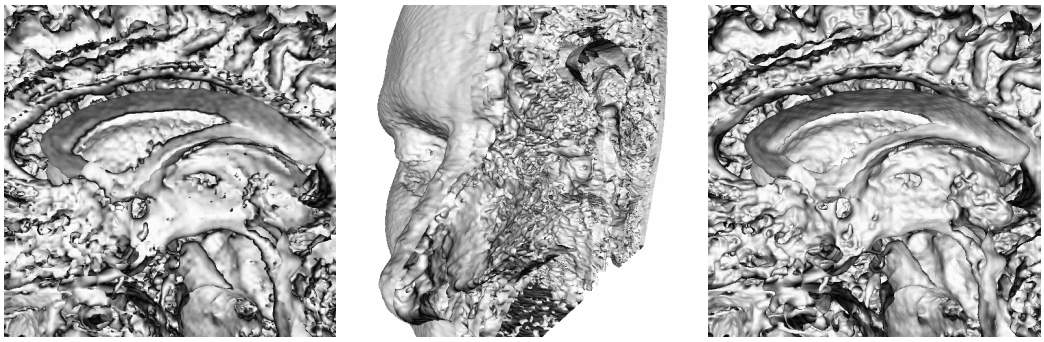


Figure 8.12: Isosurface of an MRI scan of a human head; $256 \times 195 \times 107$ samples). Trilinear (left) and our quadratic (center and right) reconstruction ($2.4s$ and $1.97s$, 14.99% relevant cubes).

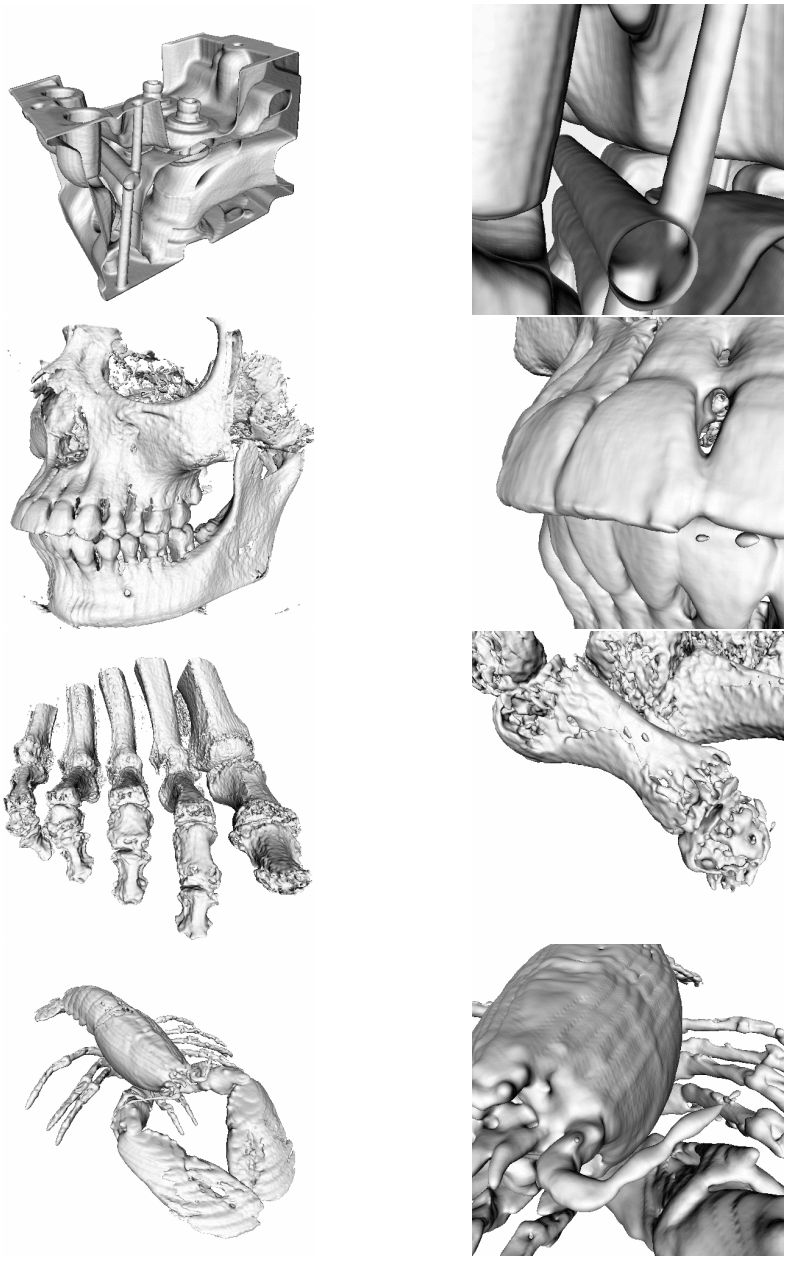


Figure 8.13: More isosurfaces rendered from our quadratic model. *engine* courtesy of General Electric ($141 \times 198 \times 110$ samples, isovalue $c = 80$, $2.5s$ (full) and $1.95s$ (close-up), 9.7% relevant cubes). *skull* courtesy of Siemens Medical System, Forchheim, Germany (256^3, $c = 40$, $4.3s$ and $5.41s$, 5.2%). *foot* courtesy of Philips Research, Hamburg, Germany (256^3, $c = 90$, $8.7s$ and $6.8s$, 2.37%). *lobster* courtesy of SUNY Stony Brook ($301 \times 324 \times 56$ samples, isovalue $c = 40$, $4.2s$ (full) and $4.47s$ (close-up).

Chapter 9

Approximation of General Volumetric Data

9.1 Background

General *data fitting* is an important problem in many scientific areas and applications. The general goal in this field is to efficiently compute suitable models which approximate given sets of discrete data of different types. This gets challenging for very large data sets with arbitrarily distributed data samples possibly contaminated with some noise resulting from measurement. A very well-known and important example in *computer graphics*, *approximation theory* and *numerical analysis* is the bivariate problem of surface approximation, i.e. fitting the height data at given points which are arbitrarily distributed over a planar domain. The literature shows that even in this case it is a complex task to find appropriate methods which satisfy (almost all) requirements of efficient and exact fitting for data of general type. In the following sections we go a step further and consider fitting of *general volumetric data*, i.e. we assume that sets of discrete points are arbitrarily distributed in a volume domain and some associated (scalar) density values at the points are given, and we are interested in finding a suitable non-discrete approximating model of the data, which allows a convenient further processing (cf. [Rössl et al. 2004b]).

It is obvious that the most important property of any fitting method should be that it approximates well, i.e. the values evaluated from the model should be close to the data values at the given points. Besides this main point of good approximation quality there are a number of additional requirements which should be ideally satisfied by an approximation method. In brief, some of these requirements are: efficient computation, evaluation and representation of the models, applicability to reasonable distributions of general data, the models should satisfy smoothness

conditions for high quality visualization, the models should have the potential to automatically reduce noise in contaminated data and for automatic data reduction. Depending on the specific applications this list of desirable properties may even be increased.

Due to the importance of data fitting in the different fields of application there exists a vast literature on this topic. We list some of the methods although we are aware that this list is far from being totally complete. An approach is to use radial basis functions and related hybrid and Shepard-like methods (see e.g. [Buhmann 2000, Franke and Hagen 1999, Lodha and Franke 1999, Schaback 2000]). Recently, such methods have been tuned towards surface reconstruction from volumetric data (see e.g. [Carr et al. 2001b, Dinh et al. 2001, Ohtake et al. 2003, Turk and O'Brien 2002]), which has typical applications in computer graphics and reverse engineering. Other methods are based on different types of splines. We mention local and global methods based on tensor product splines in three variables and the simplex spline approach [Pfeifle and Seidel 1995, Pfeifle and Seidel 1996]. If the data is structured (for instance, as a result from some local nearest neighbor estimation, quantization type or gridding algorithm), then the usage of tensor product splines and related methods is often straightforward (see e.g. [Foley 1986, Martin and Cohen 2001, Nürnberger 1989, Park and Lee 1997] and the references therein).

In the following, we also use local spline models, based on piecewise *cubic* polynomials which are defined w.r.t. type-6 tetrahedral partitions of the three-dimensional domain and satisfy smoothness conditions. Due to their mathematical complexity currently there are only a few papers on trivariate splines and many open questions concerning these spaces exist to date (see [Alfeld 1984, Chui 1988, Hangelbroek et al. 2004, Lai and Méhauté 2003, Schumaker and Sorokina 2004a, Schumaker and Sorokina 2004b, Worsey and Farin 1987] and the references therein). Nevertheless, we show here that the trivariate splines provide the necessary potential to be useful tools for solving volume fitting problems. For further information on the topic of data fitting we refer to [Alfeld 1990, Lancaster and Šalkauskas 1986, Schumaker 1976, Zeilfelder 2002] and the references therein.

For developing the method presented here, we approach the problem from two sides. On the one side we benefit from the techniques described in the previous sections on the approximation of gridded data with quadratic super splines. On the other hand, it has only been recently that algorithms for the efficient interpolation and approximation of general bivariate data sets appearing in certain real-world settings have been developed which take many of the above requirements into account. These methods (see [Davydov and Zeilfelder 2003, Haber et al. 2001, Kohlmüller et al. 2003a, Nürnberger et al. 2004a, Nürnberger et al. 2004b,

Nürnberger et al. 2003a, Nürnberger and Zeilfelder 2004], and the survey article [Nürnberger and Zeilfelder 2000] as well as the references therein) are based on *bivariate splines*, i.e. piecewise polynomials satisfying smoothness conditions which are defined w.r.t. planar and three-dimensional triangulations. In fact, the method presented here is the first generalization of the recent bivariate fitting methods [Davydov and Zeilfelder 2003, Haber et al. 2001] to the more complex trivariate setting and therefore falls into the class of *spline extension methods*. Roughly speaking, this *two-step approach* (see [Schumaker 1976]) works as follows. In a first step, we independently compute trivariate polynomial approximations to appropriate local portions of the data directly in its Bernstein-Bézier form. This can be done by imposing a *checkerboard-coloring* (see [Nürnberger et al. 2001]) to the uniform type tetrahedral partition associated with the splines. Following the ideas in [Davydov and Zeilfelder 2003, Haber et al. 2001], we adapt the degree of the local polynomials to the local variation and distribution of the data as well as for the type of data. This makes this step stable and robust and provides some smoothing of the noisy data if this is necessary. In contrast to earlier methods known from the literature based on trivariate piecewise polynomials, we directly use these local polynomial approximants *as pieces* of the trivariate splines. In the second step, these local pieces are glued together using the continuity and smoothness conditions which define the underlying spline space. In this way, the splines are defined on the whole volumetric domain as a result of building extensions of the local representants of the data.

The complexity of this general algorithm is linear in the number of (reasonably distributed) data points. In brief, its main advantages are as follows. No computation or storage for a tetrahedral partition of (a subset) of the data points is needed. Only small linear systems have to be solved - this can be done independently and in parallel, and therefore enables the handling of huge data sets (i.e. the number of data points is of order $\mathcal{O}(10^6)$). The computation, evaluation and representation of the approximating splines is efficient due to the exploitation of the Bernstein-Bézier techniques. The algorithm provides an insensitiveness concerning data contaminated with moderate noise. Moreover, only basic operations and tools available in standard numerical libraries are applied. These facts ensure not only the efficiency of the method but also the simplicity of its implementation.

9.2 Overview of the Algorithm

We briefly sketch the basic idea of the algorithm. The method is based on cubic splines w.r.t. a type-6 tetrahedral partition Δ, i.e. piecewise polynomials of total degree three which satisfy continuity and smoothness conditions across the com-

mon triangular faces of neighboring tetrahedra. Basically, the method consists of two steps. In the first step, we use a checkerboard coloring and choose a subset of tetrahedra for which we compute local least squares polynomial approximations of varying degrees for small portions of the data close to the respective tetrahedron. On the set of these (pairwise distinct) tetrahedra, we define the spline to be equal to the local polynomial approximations. In the second step, we use the conditions of the underlying spline space to uniquely extend the polynomial pieces obtained in the first step to a consistent spline on the whole domain. Hence, the algorithm completely follows the basic ideas known from approximation method of the bivariate setting ([Davydov and Zeilfelder 2003, Haber et al. 2001]). What is new here, is that we generalize these methods to the more complex trivariate setting on Δ. Comparing with [Davydov and Zeilfelder 2003, Haber et al. 2001], we observe that in the trivariate setting the second step (extension to a consistent spline) becomes more complex, while the first step (local polynomial approximation) essentially coincides. For the second step, we skip a few of the smoothness conditions and replace others by some different natural conditions in this first method dealing with general volumetric data. One motivation for doing this comes from the observations on the reconstruction of gridded data with quadratic super splines. Below we show that the whole approach applies only basic computations and averaging operations and therefore the algorithm is simple and straightforward to implement. Moreover, we note that we use cubics, because according to our experience these spaces provide at least some of the additional flexibility (comparing with quadratics) needed for the efficient approximation of arbitrarily distributed, three-dimensional data.

9.3 Consistent Cubic Splines on Δ

The approximation method is based on cubic splines, i.e. piecewise polynomials of total degree three which are defined w.r.t. the same type-6 tetrahedral partitions Δ as used for reconstruction of gridded data. For ease of explanation we choose again a cubic domain $\Omega = [0,1]^3 \subseteq \mathbb{R}^3$ which is decomposed into $n \times n \times n$ cubes, although more general domains are possible, which are decomposed into cubes (cf.cFigure 9.1). Given a set of discrete *volumetric data points* $\mathcal{X} = \{\mathbf{x} = (x_\nu, y_\nu, z_\nu) \in \Omega : \ \nu = 1, \ldots, N\} \subseteq \mathbb{R}^3$ with associated *functional (scalar) values* $f_{\mathbf{x}} \in \mathbb{R}$, $\mathbf{x} \in \mathcal{X}$, we set $n = \lfloor \sqrt[3]{N/10} \rfloor$ and cover the domain Ω with cubes $Q_{i,j,k}$, $i,j,k = 0, \ldots, n-1$, of edge length $h = 1/n$. This choice of n ensures that the number of degrees of freedom of the spline space approximately coincides with the number of scattered data points. On the other hand, this is just a reasonable heuristic choice which performed well for many of the computational examples presented in Section 9.5. We also note that for automatic data reduction

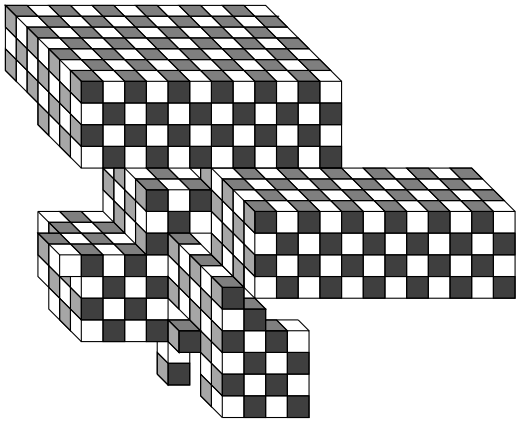


Figure 9.1: Example of a more general domain Ω than the unit cube. The domain is decomposed into uniform cubes which are colored *black* and *white*.

different choices of n might be advantageous. In addition, we need a ring of *border cells* surrounding the union $\diamondsuit$ of the cubes $Q_{i,j,k}$ to completely determine the approximating spline on Ω.

We impose a checkerboard coloring (a concept introduced in the context of local Lagrange interpolation by bivariate splines in [Nürnberger et al. 2001]) to the cubes from $\diamondsuit$: Cubes $Q_{i,j,k}$ where $i+j+k$ is even are called *black cubes*, while the rest of the cubes are called *white cubes*. For every black cube we choose the same tetrahedron, e.g. always the front facing one in the bottom pyramid (cf. Figures 7.3 and 9.2), and call these in the remainder of this paper the *black tetrahedra* of Δ.

We will define a piecewise cubic spline on Δ with polynomial pieces defined over every tetrahedron. As shown before, the C^1-smoothness for the polynomial pieces of the splines on *two adjacent* tetrahedra of the type-6 tetrahedral partition Δ are relatively simple to describe by using the piecewise Bernstein-Bézier form. On the other hand, if we consider the *complete partition* Δ, these conditions connect the coefficients of an overall C^1-smooth spline in a highly non-trivial way, because for each (interior) tetrahedron T of Δ the conditions have to be satisfied simultaneously across all the four triangular faces of T - and they can obviously not be considered independently. This observation is contrasted to the situation of splines in one variable, in the sense that for smooth multivariate spline spaces of low (and lowest possible) polynomial degree, one can sometimes observe that the splines have to simultaneously satisfy a huge number of smoothness conditions, while on the other hand the number of coefficients involved is relatively low. From Section 7.3.3 we know that in the particular case of C^1 cubics on Δ the number of degrees of freedom of the spline spaces fits into the formula $6\,n^3+24\,n^2+18\,n+4$, which shows that the spaces allow to deal with trivariate data, in principle.

On the other hand, some basic observations motivated by the bivariate approximation methods in [Davydov and Zeilfelder 2003, Haber et al. 2001] (see also [Schumaker and Sorokina 2004b], Remark 7.3) seem to indicate that this number is too small for designing a local approximation method with optimal properties using the overall C^1-smooth cubic splines. As noted above, this makes the extension step of the below algorithm more complex and different to the bivariate case. More precisely, in this first approach for local extension described below we use cubic splines on Δ with about $10\ n^3 + \mathcal{O}(n^2)$ free parameters (i.e. the 20 coefficients associated with the complete set of domain points in each of the $n^3/2$ black tetrahedra are chosen), where most of the C^1-smoothness properties (but *not all*) are satisfied and *only few* of them are skipped or replaced by other useful conditions, so that the local approximation of the data is preserved. An additional motivation for proceeding this way comes from the results on piecewise quadratic reconstructions from gridded volume data. We note, that it can be easily seen from the description of the extension step in Section 9.4.2 in conjunction with the specific form of the stencils of the averaging rules representing smoothness conditions (cf. Figures 7.5 and 7.6) that the resulting splines satisfy a huge number of smoothness conditions including those essential for certain visualization purposes and therefore have an almost similar behavior as mathematically smooth functions. In addition, this is confirmed by the computational examples given in the results section.

9.4 Approximation Method

We use the basic ideas from the two-step methods [Davydov and Zeilfelder 2003, Haber et al. 2001] and adapt them to the trivariate setting. With the uniform tetrahedral partition Δ defined, in the first step, we determine least squares polynomial approximations for small, local portions $\mathcal{X}_{loc}$ of the given data from $\mathcal{X}$. Computing the trivariate local polynomials p_{loc} with $p_{loc}(\mathbf{x}) \approx f_{\mathbf{x}},\ \mathbf{x} \in \mathcal{X}_{loc}$, can be done by using the same basic principles as described in [Davydov and Zeilfelder 2003, Haber et al. 2001] (see also [Davydov 2002]) and applying them in a straightforward way to the trivariate setting. In contrast, finding methods for extending the local trivariate polynomial pieces to a consistent cubic spline (second step) are more difficult and different to the two-dimensional case. In the approach described below, we mainly concentrate on the important aspects of computational efficiency.

9.4.1 Local Polynomial Approximation

In the first step, we determine least squares polynomial approximations for a huge number of small, local portions of the given data points $\mathcal{X}$ and the associated values $f_{\mathbf{x}},\ \mathbf{x} \in \mathcal{X}$. To do this, only the black tetrahedra (cf.Section 9.3) in Δ are considered. More precisely, for each such tetrahedron T, we choose an appropriate subset $\mathcal{X}_{loc} = \mathcal{X}_{loc}^T$ of $\mathcal{X}$ containing data points which are close to T, and compute $p_{loc} = p_{loc}^T$, trivariate polynomials of degree $d \in \{0, \ldots, 3\}$ in its Bernstein-Bézier form w.r.t. T, such that the error

$$\sum_{\mathbf{x} \in \mathcal{X}_{loc}^T} (p_{loc}^T(\mathbf{x}) - f_{\mathbf{x}})^2 \tag{9.1}$$

becomes minimal. To do this algorithmically, for each of the black tetrahedra $T \in \Delta$ we choose an initial sphere centered at the barycenter of T such that the volumetric domain is completely covered by the union of these spheres. Then, we collect the data points within each such sphere. The finding of these points can be done efficiently by initially sorting the data points from $\mathcal{X}$ into an appropriate uniform grid data structure. Following [Davydov and Zeilfelder 2003, Haber et al. 2001], we analogously balance the number of data points distributed within a particular sphere depending on the local distribution of data points. This is done either by *thinning* or *increasing the radius* of the spheres. In this way, for each black tetrahedron T, we obtain a local portion $\mathcal{X}_{loc}^T$ of the data which is contained in an appropriate sphere $\mathcal{S}_{loc}^T$. Then, we determine the local polynomial p_{loc}^T on T which approximates the data values $f_{\mathbf{x}}$ at the points $\mathbf{x} \in \mathcal{X}_{loc}^T$ in the above *discrete least squares sense*. We solve the arising system of linear equations by computing the *singular value decomposition (SVD)*. Since the corresponding observation matrices are of moderate size (the polynomial degree p_{loc}^T and the cardinality of $\mathcal{X}_{loc}^T$ are both small) this can be done in a fast and robust way. In addition, as is well-known the SVD allows to check if this system is well-conditioned or not (in our current implementation, we follow [Haber et al. 2001] at this point, although we are aware that this can be improved). If the latter case appears (i.e. there is some hidden redundancy in $\mathcal{X}_{loc}^T$), we proceed by dropping the polynomial degree d and consider a new system for polynomials of degree $d-1$, and iterate this process until either the system is well-conditioned or the polynomial degree is zero. This procedure is initialized with $d = 3$. If the resulting polynomial is of lower degree we can rewrite it as a cubic polynomial by applying (successive) *degree raising* (7.7)). This local approximation procedure is analogous to the bivariate case [Davydov and Zeilfelder 2003, Haber et al. 2001] (see also [Davydov 2002]), and provides numerical stability of the approximation part of the algorithm. Figure 9.9 shows a visualization of a single local polynomial approximant.

9.4.2 Spline Extension

The approximation step described in the previous subsection determines the polynomial pieces of the approximating spline on the set of all black tetrahedra. More precisely, for each such tetrahedron T, we set the *approximating spline* to be *equal to* the local polynomial approximation p_{loc}^T, i.e. the 20 Bernstein-Bézier coefficients of the spline piece in (7.2) coincide with the 20 Bernstein-Bézier coefficients of p_{loc}^T. Now, in this second step, we show how to compute the Bernstein-Bézier coefficients of the approximating spline on the remaining tetrahedra of Δ which are *not black* and have a non-empty intersection with Ω. As noted above this can be understood as an extension of the local approximating pieces obtained in the first step, where we use the continuity and many smoothness conditions.

For ease of explanation, we proceed by considering only a black and a white cube of the partition $\diamondsuit$ (see Section 7.2). Figures 9.2 and 9.3 show the domain points associated with the coefficients of the polynomial pieces defined on tetrahedra in a black and a white cube, respectively. These two cubes represent all interior cubes of $\diamondsuit$. As coefficients of the outermost layer (called layer 3) coincide with those of neighboring cubes, we only show the inner layers of the white cube. We use these figures to explain how the remaining coefficients are determined. The coefficients are computed step by step and this is done locally, i.e. simultaneously for all the cubes in $\diamondsuit$. To understand the below description, it may help to simultaneously think of what happens in each step to the imaginary neighbors of the black and the white cube (which have a common edge or a common vertex with these cubes). We denote the coefficients by a_i, $i = 0, \ldots, 239$, their indices i are shown in the diagram. The indices i are chosen to represent the order in which the a_i are determined, i.e. for $i < j$ the coefficient a_i is computed before (or eventually simultaneously with) a_j, so the value of a_j may depend on the value of a_i. We decided to give the following description of the second step, because it follows the method we basically used to implement it — obviously a pure mathematical description could be done shorter — but would require additional notation.

Since the spline is already determined on the black tetrahedra, it follows that the coefficients $a_0, \ldots, a_{19}$ as well as $a_{20}, \ldots, a_{25}$ are already uniquely determined. For the latter coefficients this can be seen by taking into account that there are other black cubes which share a common edge or a vertex with the black cube of consideration. These initial coefficients (resulting from the approximation procedure of the first step) are marked yellow. We will determine the main part of the remaining coefficients by applying the simple averaging rules connected with the smoothness conditions (cf. Figures 7.5 and 7.6). The given sequential order implicitly defines the appropriate rule, structural ambiguities do not impose any over-determination since in this case we skip the smoothness conditions or replace by appropriate averages, which are non-standard rules. Only some few a_i

will be treated in such a specific, unusual way — on the other hand this makes the whole construction possible. The light green coloring gives a hint on finding the coefficients $a_{26}, \ldots, a_{147}$, which are computed before the first non-standard rule is applied.

Using intra-pyramid rules gives $a_{26}, \ldots, a_{37}$, e.g. $a_{26} = 2a_6 - a_2$. An inter-pyramid condition determines $a_{38} = a_{12} - a_6 + \frac{1}{2}(a_3 + a_{26})$, then $a_{39} = 2a_{38} - a_{26}$, and $a_{40} = \frac{1}{2}(a_2 + a_{39})$. An inter-cube condition gives $a_{41} = 2a_{38} - a_{12}$ in the neighboring white cube. We continue this way and apply the smoothness conditions to obtain $a_{42}, \ldots, ...a_{92}$. Hence, we determine all coefficients around the cube vertices, i.e. the spline becomes smooth at all vertices of $\diamondsuit$. Note that as we "walk around" a vertex, we consider the already determined coefficients in the neighboring cubes, i.e. for the diagram we imagine a continuation of cubes in all directions. Using the smoothness conditions, we compute $a_{93}, \ldots, a_{105}$ which completes the bottom sides of layers three and two (cf. Figure 9.2) for all cubes. Then, we determine the top side of layer three, i.e. the coefficients $a_{106}, \ldots, a_{114}$ and of layer two, i.e. $a_{115}, \ldots, a_{136}$. After that, the inner layers one and zero of all black cubes can be computed, i.e. the coefficients $a_{137}, \ldots, a_{147}$ and the spline becomes smooth at the midpoints of black cubes. We have now fixed all coefficients marked light green, where we used smoothness conditions from the C^1-spline spaces. At that point, one can see that there would be some hidden overdetermination for the overall C^1- splines (cf. [Schumaker and Sorokina 2004b], Remark 7.3) - therefore, we have to proceed differently.

Now, consider the unknown five coefficients $a_{148}, \ldots, a_{152}$ on the layer two of the black cubes (marked by light red squares). Applying the C^1-conditions on the same layer and between layers two and one, we obtain the six equations

$$\begin{aligned}
a_{152} &= 2\, a_{149} - a_{33} \\
a_{152} &= 2\, a_{148} - a_{11} \\
a_{152} &= 2\, a_{151} - a_{116} \\
a_{152} &= 2\, a_{150} - a_{118} \\
a_{148} + a_{149} &= \tfrac{1}{2}\,(a_{12} + a_{152}) + a_{17} \\
a_{150} + a_{151} &= \tfrac{1}{2}\,(a_{58} + a_{152}) + a_{141}
\end{aligned}$$

for the five unknowns. As the system is over-determined, we suggest to *average smoothness conditions* as follows. Straightforward substitution provides

$$\begin{aligned}
a_{148} &= \tfrac{1}{2}\,(a_{12} - a_{33}) + a_{17} \\
a_{149} &= \tfrac{1}{2}\,(a_{12} - a_{11}) + a_{17} \\
a_{150} &= \tfrac{1}{2}\,(a_{58} - a_{116}) + a_{141} \\
a_{151} &= \tfrac{1}{2}\,(a_{58} - a_{118}) + a_{141}
\end{aligned}$$

and hence determines $a_{148}, \ldots, a_{151}$. Back-substitution gives four conditions on a_{152} that are averaged to

$$a_{152} \;=\; \tfrac{1}{2}(a_{12} - a_{11} - a_{33} + a_{58} - a_{116} - a_{118}) + a_{17} + a_{141}.$$

We apply the same averaging of coefficients obtained from smoothness conditions symmetrically to determine $a_{153}, \ldots, a_{157}$ on the front left side of the black cubes (marked by red squares).

Next we compute the coefficients a_{158} and a_{159} in the black cubes (marked by light blue rhombs). Due to an intra-pyramid smoothness condition, we can use $a_{158} = 2\, a_{159} - a_{135}$. In order to uniquely determine both coefficients, in addition, we impose the individual C^2-super-smoothness condition

$$4\, a_{159} \;=\; a_{39} + 4\, a_{158} - a_{78}\ ,$$

which is a standard procedure to eliminate undesirable degrees of freedom for splines (see [Davydov and Zeilfelder 2003, Nürnberger et al. 2004b, Schumaker and Sorokina 2004a]). This is illustrated by using the dashed line in Figure 9.2 showing the coefficients that are involved, here.

Now, we uniquely determine $a_{160}, \ldots, a_{167}$ using the smoothness conditions involving these coefficients around the vertical edge. Note that this is possible due to the careful choice of a_{152} and a_{159}. Analogously, we determine a_{168} and a_{169} (marked by blue rhombs) using a C^2-condition (illustrated as a dashed line), and walking around the corresponding edge uniquely determines the coefficients $a_{170}, \ldots, a_{175}$ via smoothness conditions.

We now complete the outermost layer of the black cubes by applying intra-pyramid rules, and we obtain $a_{176}, \ldots, a_{193}$. For the computation of the coefficients $a_{194}, \ldots, a_{204}$ on layer two of the black cubes we consider intra-pyramid conditions only, and we *skip* seven inter-pyramid conditions. Note that this does not affect the smoothness across the common faces of black and white cubes. Finally, we determine the inner levels one and zero of the white cubes by using smoothness conditions, and we obtain $a_{205}, \ldots, a_{224}$ and $a_{225}, \ldots, a_{239}$, respectively.

Now, all the remaining coefficients of the spline (essential for the representation on Ω) are uniquely determined, and hence we have extended the local polynomial approximations from the first step to a spline defined on the whole domain Ω. The extension to the spline turns out to be a repeated averaging of coefficients using very simple and natural rules most of which representing smoothness conditions, making this step easy to implement and very efficient. Let us point out that the resulting consistent spline is C^1 between cubes sharing a common square face, as well as inside the pyramids (consisting of four tetrahedra sharing a common edge) and at the midpoints of all cubes. Moreover, many additional

smoothness conditions are automatically satisfied by applying the above method. In the diagram of Figures 9.2 and 9.3 the coefficients are indexed for illustration purposes, and we assume that all a_i (for fixed index i and variable cube index) are computed simultaneously for all the cubes. In the implementation this would require an iteration over all cubes and to compute each a_i individually. We can minimize the number of iterations to six by reordering the computation of coefficients (or appropriately permutating the 240 indices) while using the same rules to compute the coefficients.

9.5 Results

In this section we demonstrate the efficiency of the algorithm and the high quality of the spline approximations. In the following, all computation times are measured in seconds on a (single) 3 GHz Intel Xeon CPU using double precision arithmetic. Tests on an SGI Onyx3 using eight 400 MHz R12k processors concurrently show that we get nearly optimal speedup from parallelizing the algorithm. The isosurfaces of the approximating splines are visualized as very fine triangular meshes generated by applying the *Marching Cubes algorithm* [Lorensen and Cline 1987].

In order to investigate the quality of the approximation, we first consider the smooth trivariate test function f_{test} as defined in (8.10), see Section 8.6.2. Fig. 9.4 shows several isosurfaces of the spline approximation $s_{f_{test}}$ of f_{test}.

We sample f_{test} at N randomly distributed points $\mathbf{x}$ and approximate the values $f_{\text{test}}(\mathbf{x})$ at these data points with $s_{f_{\text{test}}}$. Here, the number of cubes in every dimension is chosen as described in Section 9.3 so that the degrees of freedom of the spline approximately matches the number of data points. Table 9.1 shows different measurements of approximation errors and computation times for increasing numbers N of random samples and the respective choice of n. The third column shows the average error measured at the data points, the fourth columns lists the maximum error to the data, and the fifth column contains the maximal error to f_{test} in the uniform norm. The latter error is approximately computed by choosing 20 uniformly distributed points in each tetrahedron of Δ, evaluating the error for all these points, and computing the maximal error over all these approximative errors. The computed errors are obtained by considering the essential tetrahedra, i.e. tetrahedra contained in cubes from the complete interior of $\Omega_{f_{\text{test}}}$. Note that passing from the i-th row to $(i+1)$-th row of the table (doubling N), the side length of the cubes decreases only by the factor $2^{-(1/3)} \approx 0.79$. The last column shows the time for the local least squares approximation measured in seconds. Every row in the table is an average of 50 independent scattered tests, each of which uses a different random distribution of the data points. The time for

N	n	err_{mean}	err_{data}	err_{max}	time
1 000	4	0.05612090	0.16392300	0.18640600	0.05
2 000	5	0.01758270	0.07843640	0.08240110	0.08
4 000	7	0.00247124	0.02297960	0.02867510	0.17
8 000	9	0.00076832	0.00766123	0.00922203	0.32
16 000	11	0.00030479	0.00374684	0.00429214	0.57
32 000	14	0.00010208	0.00147255	0.00168576	1.10
64 000	18	0.00003375	0.00054775	0.00062212	2.20
128 000	23	0.00001175	0.00021318	0.00023703	4.38
256 000	29	0.00000441	0.00008587	0.00009686	8.52
512 000	37	0.00000160	0.00003636	0.00004275	17.14
1 024 000	46	0.00000065	0.00001530	0.00001765	32.90
2 048 000	58	0.00000025	0.00000618	0.00000747	65.26
4 096 000	74	0.00000009	0.00000292	0.00000375	133.49

Table 9.1: Approximation errors of the splines $s_{f_{\text{test}}}$ and computation times in seconds.

determining the coefficients in the extension step (see Section 9.4.2) are not listed explicitly, since it is clearly linear in the number of cubes. In our computations, we observed that this is less than 5% of the time required for determining the local polynomial approximations (first step of algorithm). The test shows the quality of the spline approximation as well as the efficiency of its computation, particularly confirming the linear complexity of the algorithm. Moreover, we give a test that shows that our algorithm provides the potential to deal with noisy input data. It is illustrated by the results shown in Figure 9.5. In this test, the domain is decomposed into 29^3 cubes to approximate $128\ 000$ samples as before (cf. Table 9.1), and we added uniformly distributed noise to the sampled function values of f_{test}. Similar results were obtained for different smooth test functions. Obviously, for cubic trivariate polynomials, our method yields errors which are negligible. Examples for simple but non-trivial test functions are of truncated power type, e.g.

$$g(x,y,z) = (x-\tfrac{1}{2})_+^5 + (x(y-\tfrac{1}{2})(z-1))_+^5 + (x(y-1)(z-\tfrac{1}{2}))_+^5,$$

where $(x,y,z) \in \Omega_g = [0,1]^3$. Choosing $N = 8000$, for this simple test function we obtain the following errors of the approximating spline s_g: $\text{err}_{\text{mean}} = 0.00000950$, $\text{err}_{\text{data}} = 0.00010017$, and $\text{err}_{\text{max}} = 0.00011770$.

We proceed by applying our method to some real-world data sets. An example for an interesting test in computer graphics is volume-based surface reconstruction. Here, we assume that samples of the *signed distance* to a surface are given, and the goal is to (locally) find a trivariate model representing the data whose

zero-set approximates the surface. We consider this as a very intuitive test because the desired result is the shape of a known object and hence needs no extra interpretation. However, we currently do not directly compete with existing surface reconstruction methods, as the goal of our algorithm is more general, and our current setup is not optimized for the specific requirements of efficient reconstruction of surfaces. We consider an existing, high-resolution triangular mesh that was initially acquired by digitizing a real-world object as the *Max-Planck* bust. We then sample the signed distance to this surface not only in vicinity of the surface but *randomly distributed* in an extended bounding box. This way we generate a huge amount of data even for regions of the volume that are very distant from the zero-set and would be negligible for surface reconstruction. Hence, this is obviously a difficult test for any method. For the test, we explicitly want to cover the whole volume and stress the algorithm with large input. Given the *Max-Planck* data set, we randomly distribute $N = 2 * 10^6$ data points at which we measure the signed distance in a straightforward (but rough) way. The domain is decomposed into $58 \times 97 \times 73$ cubes, hence about $205\ 000$ local least squares polynomial approximations are computed to determine the approximating spline $s_{\text{planck}\cdot}$ The average number of data points used for the local approximation in this test is 66. The approximation takes about 133 seconds, so more than 1500 local approximations are performed per second. Figures 9.6 and 9.7 visualize the results of our algorithm — we observe that the reconstructed surfaces inherit a visual smooth appearance from the trivariate splines. In addition, we provide a similar test using the *mechpart* data set, which is a well-known benchmark in CAGD (cf. [Hoschek and Dankwort 1996]). The original data is a discrete height field over a two-dimensional 82×50 grid. A known difficulty for the reconstruction is the coarseness of this data in conjunction with the relatively high variation of the heights. As a test, we distributed $N = 512\ 000$ points randomly in the volume bounding box and measured signed distances w.r.t. an almost regular (two-dimensional) triangulation of the data. Fig. 9.8 shows the reference data and zero-sets of spline approximations s_{mp} of the signed distance for two different partitions Δ.

We integrated our algorithm in the AMIRA visualization system [Stalling et al. 2003], and note that all the visualizations given in this section have been created with AMIRA. Using this framework enables experiments like the interactive approximation or visualization of local pieces of the spline as for instance single polynomials on prescribed tetrahedra of Δ. Fig. 9.9 shows an example, where we illustrate the behavior of the extension step of our algorithm.

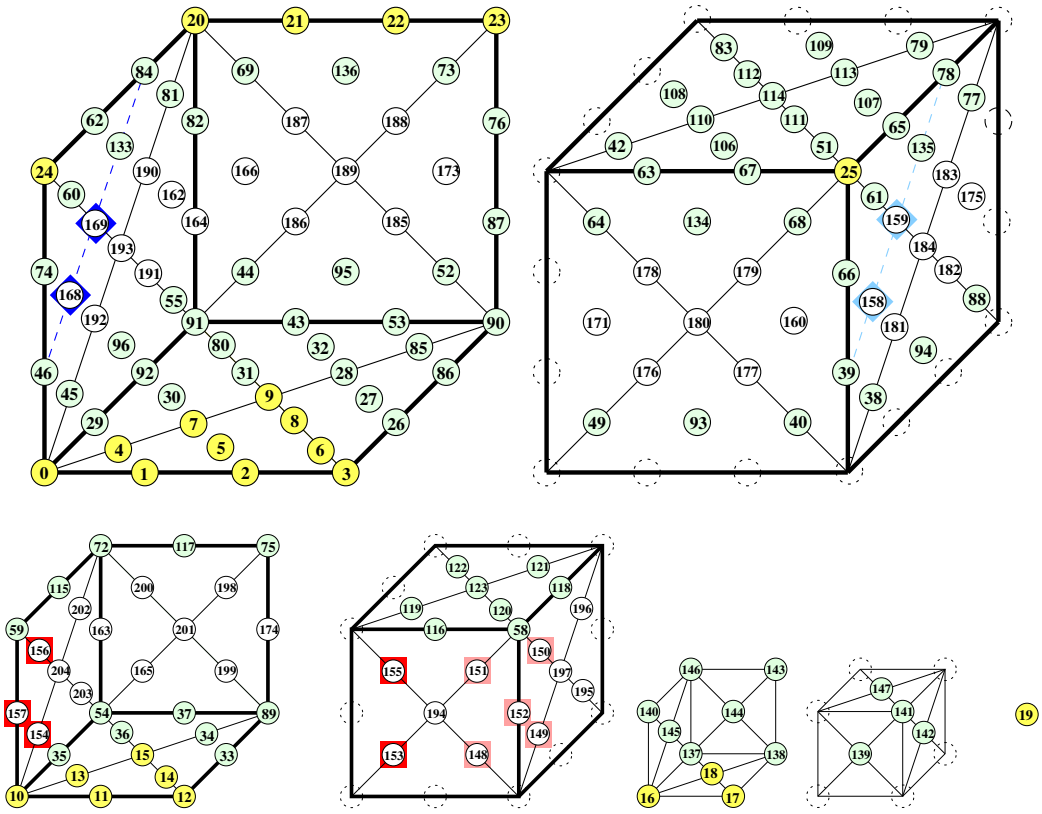


Figure 9.2: The four layers $3, 2, 1, 0$ of Bézier points for a black cube, each cube layer is cut into its back-facing and front-facing part. The dots show the 175 associated Bernstein-Bézier coefficients, and their numeric labels. The meaning of the colors are described in Section 9.4.2. The 24 polynomial pieces of the splines inside the cube are represented by the $10, 6$ and 3 domain points in the respective triangles and the midpoint over all layers, e.g. points with label 0–19 on the black tetrahedron (cf. Sections 7.2 and 9.3 and Figure 7.3). Note that coefficients on the edges of the outermost layer (top row) coincide with those of some nearby black cubes. The innermost layer (numbered with 0) consists of a single point (bottom right). Fig. 9.3 shows the remaining coefficients of the white cubes.

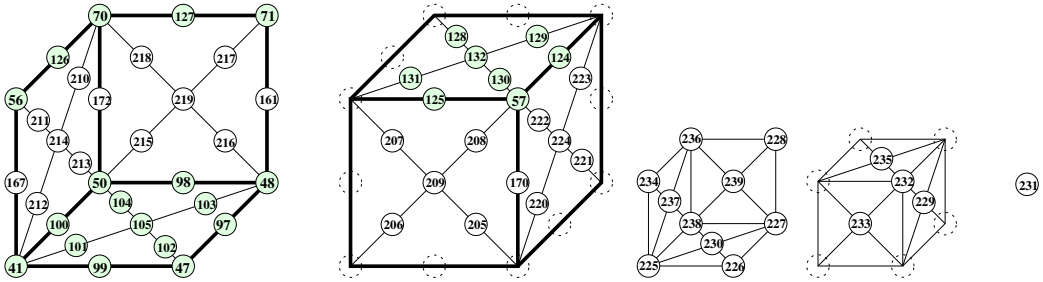


Figure 9.3: The inner layers $2, 1, 0$ of a white cube contain 65 Bézier points, see also Fig. 9.2 and Section 9.4.2. The white cube shares faces with six black cubes, and their coefficients on the outermost layer coincide by the continuity. For this reason it is sufficient to consider the three inner layers only.

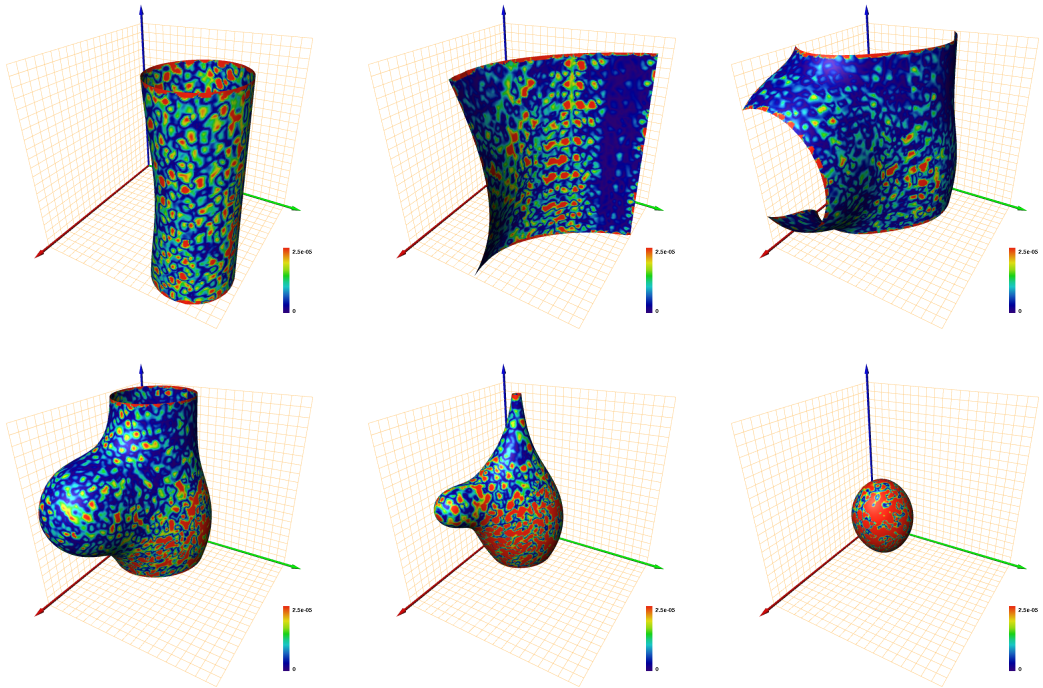


Figure 9.4: Isosurfaces of an approximation s_f to the test function f_{test} sampled at $N = 128\,000$ randomly distributed points. The color code visualizes the approximation error for the surface points. The isovalues starting from top left are -0.1, 0, 0.1, 0.3, 0.5, and 0.8.

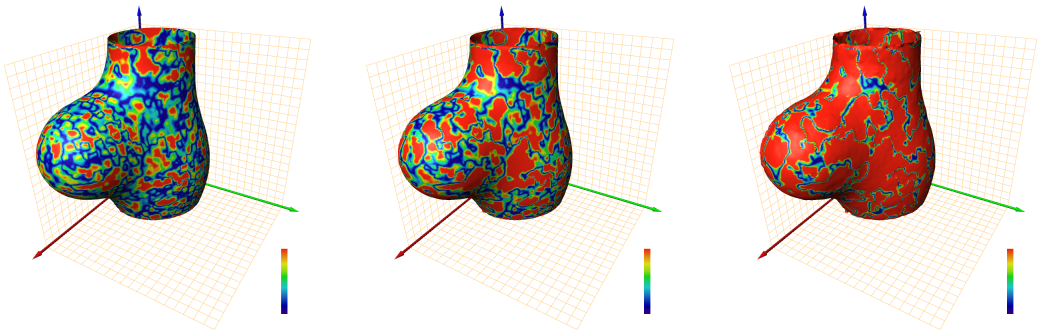


Figure 9.5: Approximation of noisy data. As for Fig. 9.4, $N = 128\,000$ randomly distributed samples are used for the spline approximation, and uniformly distributed noise is added. The maximal amplitude of noise is (from left) 0.5%, 1% and 2.5% relative to the maximum range of f, where $f(x, y, z) \in [-0.25, 1.27], (x, y, z) \in \Omega_{f_{\text{test}}}$. We observe average approximation errors (err_{mean}) of about 0.0038, 0.0075, and 0.019 respectively. The pictures show the isosurface with isovalue 0.3 extracted from the approximating splines $s_{f_{\text{test}}}$.

Figure 9.6: Visualization of the spline approximation s_{planck} for the *Max-Planck* data set. The domain is decomposed into $58 \times 97 \times 73$ cubes. The left and center image show different views of the zero-set of the spline s_{planck}. The right image shows a slice through the approximative volume model $s_{\text{planck}}(x, y, const)$ for signed distance values, where the distances are color coded.

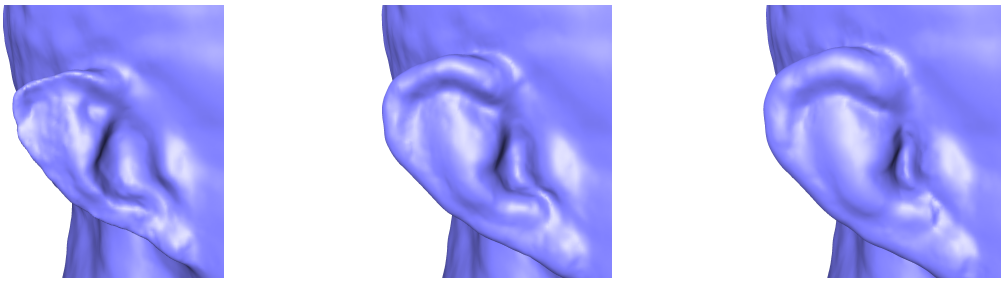


Figure 9.7: Close-ups of different isosurfaces of the spline approximation s_{planck} for the *Max-Planck* data set as used in Fig. 9.6. The isovalues are (from left) 2, -2, and -4. The corresponding isosurfaces can be considered as offset surfaces to the zero-set of the spline s_{planck} shown in Fig. 9.6.

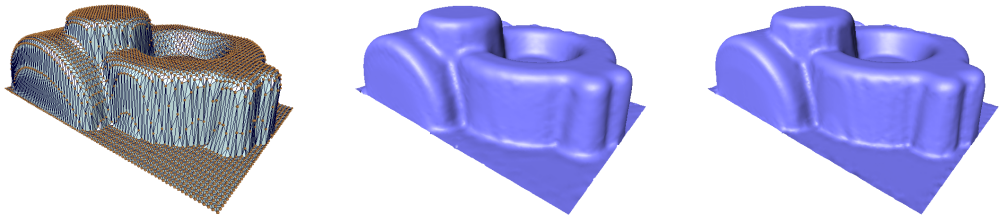


Figure 9.8: Approximation of the *mechpart* data set. Top: The original surface that was sampled at $N = 512\ 000$ randomly distributed points. Center and bottom: Zero-sets of the approximating splines s_{mp}. The domain is decomposed into $64 \times 39 \times 22$ and $42 \times 26 \times 15$ cubes, the computation times are $26s$ s and 20 s, respectively.

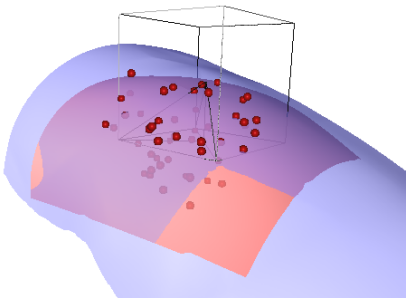


Figure 9.9: An isosurface of a least squares approximating polynomial piece on a black tetrahedron (transparent red) together with the positions of the samples (red spheres) that were used for the local fitting procedure (using a local portion $\mathcal{X}_{loc}$ of the extended *mechpart* data set). For this visualization the polynomial is extrapolated out of the respective tetrahedron as indicated. The transparent blue surface shows the same isosurface for the complete approximating spline, which shows that the local difference is small and the extending behaves in a natural and smooth way.

Chapter 10
Summary on Trivariate Splines

We presented a new model for the reconstruction of discrete volume data given on a regular grid which is a typical problem in volume visualization. In contrast to earlier approaches, our method approximates the data by quadratic trivariate super splines on a tetrahedral partition. The reconstruction is natural, completely symmetric and efficient, and reveals interesting approximation properties. The local quasi-interpolating spline model can be evaluated efficiently including precise local gradients due to appropriate smoothness properties. The new approach uses piecewise polynomials of total polynomial degree two, and it compares to existing trilinear and triquadratic approaches based on piecewise polynomials of total degree three and six, respectively. We exploit this fact for efficient and precise isosurface ray-casting. The results show that the model is effective, efficient, simple in implementation and appropriate for high-quality volume rendering.

In addition, we present a new method for the efficient approximation of huge volumetric data sets distributed over an arbitrarily shaped domain. The two-step algorithm is of linear algorithmic complexity w.r.t. the number of samples, it uses the same natural, uniform tetrahedral partition of the domain which is given implicitly, it requires only the (independent) solution of small linear systems, it automatically adapts to local variation and distribution of the data, and it automatically smoothes noisy data. The method is based on trivariate, cubic splines, and it is known that finding local constructions based on these spaces is a complex task. In this first approach to the problem of local approximation of general volumetric data, we balance computational simplicity against overall smoothness and construct consistent cubic splines which satisfy almost all smoothness conditions. The results confirm the high quality of the approximation and show visually smooth isosurfaces of the reconstructed real-world objects. Further, we note that the current implementation can be improved by either applying the average operators introduced in [Davydov and Zeilfelder 2003] and perhaps by making use of

different local approximations similar as in [Davydov et al. 2004]. Moreover, one can think of integrating our method in straightforward hierarchical constructions over nested sequences of cube partitions, tuning the approach towards surface reconstruction and providing direct visualization of the trivariate splines, e.g. by ray-casting.

We addressed two independent problems, the reconstruction from structured, gridded data and the approximation of general data. While the settings are different and lead to different algorithms, we apply a similar basic framework. Both approaches share the same tetrahedral partition. We observe that the type-6 partitions provide the flexibility for the local construction. This is not obvious but was indicated by the relation to the four-directional meshes commonly used in the bivariate case. Our experiments and considerations lead us to this choice. Also, in both approaches consistent splines are constructed, which satisfy many smoothness conditions, essential for visualization. We analyzed the smoothness and approximation properties, and the theoretical results are confirmed by numerical experiments with synthetic and real-world data. We showed that these splines provide a valuable tool for visualization.

Chapter 11
Conclusions and Future Work

In this thesis, we focus on various aspects in the visualization of digital data, representing surfaces or volumes. Of course, visualization is a broad field in computer graphics, and we identify specific fundamental problems: the quest for an appropriate model of the data, the analysis of the data, and data compression. We developed new techniques for the solution of each subproblem given a particular context. Complementing the summaries on the individual contributions, we conclude the thesis with a brief summary, reviewing the fundamental problems.

Throughout this thesis we applied piecewise polynomials as *model of the data*. Typically such spline models combine simplicity with efficiency, and they are commonly used in visualization. In the two-dimensional case, we chose the well-known triangular meshes as the simplest model, more precisely, we use piecewise linear polynomials for the representation of surfaces and vector fields. The design of suitable mathematical models of the data is much more involved in the volumetric case. In this context, we develop two new models for the reconstruction from volumetric samples. Both models are based on piecewise polynomials with respect to the same uniform tetrahedral partition of the domain. Our considerations show that the design of methods for the efficient reconstruction with smooth splines is a difficult task. We approach two related but different tasks: the reconstruction from structured samples, which are laid out on a regular grid, and the approximation of general data, consisting of distributed samples. In either case, the number of samples can be huge, so there is a demand for efficient, local methods. Consequently, we focused on keeping the polynomial degree as low as possible and presented models based on piecewise quadratic and piecewise cubic polynomials, respectively. We note that the commonly applied trilinear interpolation — the simplest tensor spline model for gridded data — already has total polynomial degree three, i.e., it consists of piecewise cubics. The presented models provide robust and efficient computation and evaluation, and Bernstein-Bézier techniques,

well-known from CAGD, can be exploited. We analyzed smoothness and approximation properties theoretically and by numerical experiments. Moreover, the spline models enable efficient and precise ray-casting, e.g., for the visualization of isosurfaces.

On the *data analysis* side, we focused on curvature estimation of discrete shapes. The tensor of curvature is only defined for sufficiently smooth surfaces. Consequently, the problem is to find an appropriate discretization of curvature, which fits the non-smooth piecewise linear surface model. Here, we develop a new method, which views every triangle individually and which is inspired by Phong shading: The linear interpolation of the estimated vertex normals imitates higher order smoothness inside the triangle. This "fake smoothness" from combining a piecewise linear surface and a piecewise linear normal field is a common tool in visualization and computer graphics. Here, it is applied for the first time to estimate surface intrinsics, resulting in a similarly simple algorithm, which provides the tensor of curvature as a smooth function over the triangle and elegant closed formulas for the Gaussian and mean curvature. We provide a theoretical analysis of the method, and our numerical results show that it competes well in approximation error and efficiency with the best existing alternatives. We show applications of the curvature analysis which result in new algorithms for the recovering structural information from data sets in a reverse engineering context and the semi-automatic generation of line-art illustrations. The latter application from the field of non-photorealistic rendering is particularly interesting here, because it gives a good impression of the shape using only a very limited number of drawing primitives or strokes. There is a similar setting in flow visualization, where huge, complex data sets are processed and only a meaningful fraction of information is displayed. We identify this most significant information as the topological skeleton.

This leads to *data compression*. Here, we considered the encoding of the triangulation, i.e. the mesh connectivity, and the topology preserving compression of piecewise linear vector fields. The two settings are very different: the first one encodes all information (up to permutation of the vertex sequence) without loss, while the second one enables efficient compression by simplifying the data under a global constraint. Each setting comes with its particular challenges, and we developed new algorithms for either one. For connectivity encoding, we apply a divide-and-conquer approach, which partitions a mesh into two submeshes by growing a triangle strip and then recursively encodes the left and right submeshes. The result is a weighted binary tree data structure with subtrees corresponding to submeshes, where each node corresponds to triangle strip and stores its length as weight. We analyze this intuitive algorithm, provide variations, and show that the resulting data structure can be applied for efficient visualization by rendering triangle strips. For vector field compression, we already emphasized the importance

of vector field topology and the topological skeleton in flow visualization applications. This leads to the design of compression algorithms, which preserve the topology of the vector field. The major difficulty here is to design efficient, practical algorithms, which is constrained by the fact that topology is a global property while efficient simplification algorithms work locally. We provide a theoretical framework and show how all decisions in the simplification can be carried out based on local information only. This results in a new algorithm,which is efficient in computation and compression. Based on this, we discuss several modifications that affect the compression rate by applying a stronger and a relaxed definition of topological equivalence, and we design a new method for topology simplification.

In conclusions, we presented several new techniques for the visualization of surfaces and volumes. We focused on particular problems of designing appropriate models of the data, analyzing and compressing the data. New solutions tend to reveal new questions, and we list the following for *future work*.

As we consider surface and volume data, it seems natural to generalize techniques to higher dimensions, i.e., from surfaces to volumes. Here, we mention the topology preserving compression of vector fields. In our implementation and experiments, we so far only considered a planar domain. However, we already remarked that there is no general restriction, and we can easily process surfaces embedded in a three-dimensional domain. The necessary extensions are rather straightforward compared to a volumetric setting: in this case, we would like to simplify trivariate, piecewise linear vector fields defined with respect to an arbitrary tetrahedral partition of the domain. We expect that the general concepts from the theoretical framework translate to the trivariate setting, however, more cases have to be considered and the overall setup gets much more complex. Considering the new trivariate spline models, we remind that we took kind of a compromise, trading smoothness against simplicity and locality. We showed that the resulting splines provide useful tools with advantageous properties for the various requirements of efficient visualization. According to our knowledge these are the first suchlike approaches in the literature, and consequently they reveal new questions. The theoretical analysis of the dimension of C^1-splines indicates the limits of any approaches using type-6 tetrahedral partitions (and low polynomial degree). According to our current knowledge, overall smooth trivariate spline models require some additional degrees of freedom, so that we might either need a higher (but as small as possible) degree of the piecewise polynomials or different partitions. Both options are subject of future research, and from a current point of view it seems to be a complex task to end up with an intuitive and computationally simple smooth reconstruction.

Appendix A

Normal Based Curvature Estimation

We derive the formulas (3.13) and (3.14) for approximation of the Gaussian and mean curvature, respectively. As intermediate results we obtain new expressions for the Weingarten matrix and the curvatures in terms of determinants.

Let $\tilde{\mathbf{x}}$, $\tilde{\mathbf{x}}_u$, $\tilde{\mathbf{x}}_v$, $\tilde{\mathbf{n}}$, and $\mathbf{n}$ be defined as in Section 3.2.2. From the definition of $\mathbf{n}$ we get

$$\mathbf{x}_u \;=\; \tilde{\mathbf{x}}_u - \frac{(\tilde{\mathbf{x}}_u\tilde{\mathbf{n}})}{(\tilde{\mathbf{n}}\tilde{\mathbf{n}})}\tilde{\mathbf{n}}\;,$$

and a similar term for $\mathbf{x}_v$. We derive the elements of the Weingarten curvature matrix, starting with the elements of the fundamental forms. We obtain

$$\begin{aligned} F \;=\; (\mathbf{x}_u\mathbf{x}_v) \;&=\; (\tilde{\mathbf{x}}_u\tilde{\mathbf{x}}_v) - 2\frac{(\tilde{\mathbf{x}}_u\tilde{\mathbf{n}})(\tilde{\mathbf{x}}_v\tilde{\mathbf{n}})}{(\tilde{\mathbf{n}}\tilde{\mathbf{n}})} + \frac{(\tilde{\mathbf{x}}_u\tilde{\mathbf{n}})(\tilde{\mathbf{x}}_v\tilde{\mathbf{n}})}{(\tilde{\mathbf{n}}\tilde{\mathbf{n}})^2}(\tilde{\mathbf{n}}\tilde{\mathbf{n}}) \\ &=\; (\tilde{\mathbf{x}}_u\tilde{\mathbf{x}}_v) - \frac{(\tilde{\mathbf{x}}_u\tilde{\mathbf{n}})(\tilde{\mathbf{x}}_v\tilde{\mathbf{n}})}{(\tilde{\mathbf{n}}\tilde{\mathbf{n}})}\;, \end{aligned}$$

and analog for the symmetric terms

$$\begin{aligned} E \;=\; (\mathbf{x}_u\mathbf{x}_u) \;&=\; (\tilde{\mathbf{x}}_u\tilde{\mathbf{x}}_u) - \frac{(\tilde{\mathbf{x}}_u\tilde{\mathbf{n}})^2}{(\tilde{\mathbf{n}}\tilde{\mathbf{n}})}\;, \\ G \;=\; (\mathbf{x}_v\mathbf{x}_v) \;&=\; (\tilde{\mathbf{x}}_v\tilde{\mathbf{x}}_v) - \frac{(\tilde{\mathbf{x}}_v\tilde{\mathbf{n}})^2}{(\tilde{\mathbf{n}}\tilde{\mathbf{n}})} \end{aligned}$$

of the first fundamental form. The partial derivatives of the (not normalized) normal $\tilde{\mathbf{n}}$ are $\tilde{\mathbf{n}}_u = \mathbf{n}_1 - \mathbf{n}_0$ and $\tilde{\mathbf{n}}_v = \mathbf{n}_2 - \mathbf{n}_0$. Hence for the (normalized) normal $\mathbf{n}$

we get

$$\begin{aligned}\mathbf{n}_u &= D_u\frac{\tilde{\mathbf{n}}}{\sqrt{(\tilde{\mathbf{n}}\tilde{\mathbf{n}})}} = -\frac{(\tilde{\mathbf{n}}_u\tilde{\mathbf{n}})}{(\tilde{\mathbf{n}}\tilde{\mathbf{n}})\sqrt{(\tilde{\mathbf{n}}\tilde{\mathbf{n}})}}\mathbf{n}+\frac{\tilde{\mathbf{n}}_u}{\sqrt{(\tilde{\mathbf{n}}\tilde{\mathbf{n}})}}\\ &= \frac{(\tilde{\mathbf{n}}\tilde{\mathbf{n}})\tilde{\mathbf{n}}_u-(\tilde{\mathbf{n}}_u\tilde{\mathbf{n}})\tilde{\mathbf{n}}}{(\tilde{\mathbf{n}}\tilde{\mathbf{n}})\sqrt{(\tilde{\mathbf{n}}\tilde{\mathbf{n}})}}, \quad \text{and}\\ \mathbf{n}_v &= \frac{(\tilde{\mathbf{n}}\tilde{\mathbf{n}})\tilde{\mathbf{n}}_v-(\tilde{\mathbf{n}}_v\tilde{\mathbf{n}})\tilde{\mathbf{n}}}{(\tilde{\mathbf{n}}\tilde{\mathbf{n}})\sqrt{(\tilde{\mathbf{n}}\tilde{\mathbf{n}})}}.\end{aligned}$$

Then the second fundamental form is defined by

$$\begin{aligned}L &= -\tilde{\mathbf{n}}_u\tilde{\mathbf{x}}_u\\ &= -\frac{1}{(\tilde{\mathbf{n}}\tilde{\mathbf{n}})\sqrt{(\tilde{\mathbf{n}}\tilde{\mathbf{n}})}}\left((\tilde{\mathbf{n}}\tilde{\mathbf{n}})\tilde{\mathbf{n}}_u-(\tilde{\mathbf{n}}_u\tilde{\mathbf{n}})\tilde{\mathbf{n}})(\tilde{\mathbf{x}}_u-\frac{(\tilde{\mathbf{x}}_u\tilde{\mathbf{n}})}{(\tilde{\mathbf{n}}\tilde{\mathbf{n}})}\tilde{\mathbf{n}})\right)\\ &= -\frac{1}{(\tilde{\mathbf{n}}\tilde{\mathbf{n}})\sqrt{(\tilde{\mathbf{n}}\tilde{\mathbf{n}})}}\left((\tilde{\mathbf{n}}\tilde{\mathbf{n}})(\tilde{\mathbf{x}}_u\tilde{\mathbf{n}}_u)-(\tilde{\mathbf{x}}_u\tilde{\mathbf{n}})(\tilde{\mathbf{n}}_u\tilde{\mathbf{n}})-(\tilde{\mathbf{n}}_u\tilde{\mathbf{n}})(\tilde{\mathbf{x}}_u\tilde{\mathbf{n}})+(\tilde{\mathbf{n}}_u\tilde{\mathbf{n}})(\tilde{\mathbf{x}}_u\tilde{\mathbf{n}})\right)\\ &= -\frac{1}{(\tilde{\mathbf{n}}\tilde{\mathbf{n}})\sqrt{(\tilde{\mathbf{n}}\tilde{\mathbf{n}})}}\left((\tilde{\mathbf{n}}\tilde{\mathbf{n}})(\tilde{\mathbf{x}}_u\tilde{\mathbf{n}}_u)-(\tilde{\mathbf{x}}_u\tilde{\mathbf{n}})(\tilde{\mathbf{n}}_u\tilde{\mathbf{n}})\right), \quad \text{and similar}\\ M_1 &= -\tilde{\mathbf{n}}_u\tilde{\mathbf{x}}_v = -\frac{1}{(\tilde{\mathbf{n}}\tilde{\mathbf{n}})\sqrt{(\tilde{\mathbf{n}}\tilde{\mathbf{n}})}}\left((\tilde{\mathbf{n}}\tilde{\mathbf{n}})(\tilde{\mathbf{x}}_v\tilde{\mathbf{n}}_u)-(\tilde{\mathbf{x}}_v\tilde{\mathbf{n}})(\tilde{\mathbf{n}}_u\tilde{\mathbf{n}})\right),\\ M_2 &= -\tilde{\mathbf{n}}_v\tilde{\mathbf{x}}_u = -\frac{1}{(\tilde{\mathbf{n}}\tilde{\mathbf{n}})\sqrt{(\tilde{\mathbf{n}}\tilde{\mathbf{n}})}}\left((\tilde{\mathbf{n}}\tilde{\mathbf{n}})(\tilde{\mathbf{x}}_u\tilde{\mathbf{n}}_v)-(\tilde{\mathbf{x}}_u\tilde{\mathbf{n}})(\tilde{\mathbf{n}}_v\tilde{\mathbf{n}})\right), \quad \text{and}\\ N &= -\tilde{\mathbf{n}}_v\tilde{\mathbf{x}}_v = -\frac{1}{(\tilde{\mathbf{n}}\tilde{\mathbf{n}})\sqrt{(\tilde{\mathbf{n}}\tilde{\mathbf{n}})}}\left((\tilde{\mathbf{n}}\tilde{\mathbf{n}})(\tilde{\mathbf{x}}_v\tilde{\mathbf{n}}_v)-(\tilde{\mathbf{x}}_v\tilde{\mathbf{n}})(\tilde{\mathbf{n}}_v\tilde{\mathbf{n}})\right).\end{aligned}$$

(Note that $M_1 \neq M_2$.) Next, we obtain

$$\begin{aligned}EG-F^2 = \frac{1}{(\tilde{\mathbf{n}}\tilde{\mathbf{n}})}\big((\tilde{\mathbf{x}}_u\tilde{\mathbf{x}}_u)(\tilde{\mathbf{x}}_v\tilde{\mathbf{x}}_v)(\tilde{\mathbf{n}}\tilde{\mathbf{n}})-(\tilde{\mathbf{x}}_u\tilde{\mathbf{x}}_u)(\tilde{\mathbf{x}}_v\tilde{\mathbf{n}})^2-(\tilde{\mathbf{x}}_v\tilde{\mathbf{x}}_v)(\tilde{\mathbf{x}}_u\tilde{\mathbf{n}})^2\\ -(\tilde{\mathbf{x}}_u\tilde{\mathbf{x}}_v)^2(\tilde{\mathbf{n}}\tilde{\mathbf{n}})+2(\tilde{\mathbf{x}}_u\tilde{\mathbf{x}}_v)(\tilde{\mathbf{x}}_u\tilde{\mathbf{n}})(\tilde{\mathbf{x}}_v\tilde{\mathbf{n}})\big),\end{aligned}$$

and define for convenience $\mu := \sqrt{(\tilde{\mathbf{n}}\tilde{\mathbf{n}})}(\tilde{\mathbf{n}}\tilde{\mathbf{n}})(EG-F^2)$. Inserting in (3.3) gives

the elements $\tilde{w}_{ij}$, $i,j \in \{1,2\}$, of the Weingarten matrix $\widetilde{\mathbf{W}}$ as

$$\begin{aligned}
\mu\,\tilde{w}_{11} &= -(\tilde{\mathbf{x}}_u\tilde{\mathbf{x}}_v)(\tilde{\mathbf{x}}_v\tilde{\mathbf{n}}_u)(\tilde{\mathbf{n}}\tilde{\mathbf{n}}) + (\tilde{\mathbf{x}}_v\tilde{\mathbf{x}}_v)(\tilde{\mathbf{x}}_u\tilde{\mathbf{n}}_u)(\tilde{\mathbf{n}}\tilde{\mathbf{n}}) - (\tilde{\mathbf{x}}_u\tilde{\mathbf{n}}_u)(\tilde{\mathbf{x}}_v\tilde{\mathbf{n}})^2 \\
&\quad +(\tilde{\mathbf{x}}_v\tilde{\mathbf{n}}_u)(\tilde{\mathbf{x}}_u\tilde{\mathbf{n}})(\tilde{\mathbf{x}}_v\tilde{\mathbf{n}}) + (\tilde{\mathbf{n}}_u\tilde{\mathbf{n}})(\tilde{\mathbf{x}}_u\tilde{\mathbf{x}}_v)(\tilde{\mathbf{x}}_v\tilde{\mathbf{n}}) - (\tilde{\mathbf{n}}_u\tilde{\mathbf{n}})(\tilde{\mathbf{x}}_v\tilde{\mathbf{x}}_v)(\tilde{\mathbf{x}}_u\tilde{\mathbf{n}})\,, \\
\mu\,\tilde{w}_{12} &= -(\tilde{\mathbf{x}}_u\tilde{\mathbf{x}}_v)(\tilde{\mathbf{x}}_v\tilde{\mathbf{n}}_v)(\tilde{\mathbf{n}}\tilde{\mathbf{n}}) + (\tilde{\mathbf{x}}_v\tilde{\mathbf{x}}_v)(\tilde{\mathbf{x}}_u\tilde{\mathbf{n}}_v)(\tilde{\mathbf{n}}\tilde{\mathbf{n}}) - (\tilde{\mathbf{x}}_u\tilde{\mathbf{n}}_v)(\tilde{\mathbf{x}}_v\tilde{\mathbf{n}})^2 \\
&\quad +(\tilde{\mathbf{x}}_v\tilde{\mathbf{n}}_v)(\tilde{\mathbf{x}}_u\tilde{\mathbf{n}})(\tilde{\mathbf{x}}_v\tilde{\mathbf{n}}) + (\tilde{\mathbf{n}}_v\tilde{\mathbf{n}})(\tilde{\mathbf{x}}_u\tilde{\mathbf{x}}_v)(\tilde{\mathbf{x}}_v\tilde{\mathbf{n}}) - (\tilde{\mathbf{n}}_v\tilde{\mathbf{n}})(\tilde{\mathbf{x}}_v\tilde{\mathbf{x}}_v)(\tilde{\mathbf{x}}_u\tilde{\mathbf{n}})\,, \\
\mu\,\tilde{w}_{21} &= (\tilde{\mathbf{x}}_u\tilde{\mathbf{x}}_u)(\tilde{\mathbf{x}}_v\tilde{\mathbf{n}}_u)(\tilde{\mathbf{n}}\tilde{\mathbf{n}}) - (\tilde{\mathbf{x}}_u\tilde{\mathbf{x}}_v)(\tilde{\mathbf{x}}_u\tilde{\mathbf{n}}_u)(\tilde{\mathbf{n}}\tilde{\mathbf{n}}) + (\tilde{\mathbf{x}}_u\tilde{\mathbf{n}}_u)(\tilde{\mathbf{x}}_u\tilde{\mathbf{n}})(\tilde{\mathbf{x}}_v\tilde{\mathbf{n}}) \\
&\quad -(\tilde{\mathbf{x}}_v\tilde{\mathbf{n}}_u)(\tilde{\mathbf{x}}_u\tilde{\mathbf{n}})^2 - (\tilde{\mathbf{n}}_u\tilde{\mathbf{n}})(\tilde{\mathbf{x}}_u\tilde{\mathbf{x}}_u)(\tilde{\mathbf{x}}_v\tilde{\mathbf{n}}) + (\tilde{\mathbf{n}}_u\tilde{\mathbf{n}})(\tilde{\mathbf{x}}_u\tilde{\mathbf{x}}_v)(\tilde{\mathbf{x}}_u\tilde{\mathbf{n}})\,, \qquad \text{and} \\
\mu\,\tilde{w}_{22} &= (\tilde{\mathbf{x}}_u\tilde{\mathbf{x}}_u)(\tilde{\mathbf{x}}_v\tilde{\mathbf{n}}_v)(\tilde{\mathbf{n}}\tilde{\mathbf{n}}) - (\tilde{\mathbf{x}}_u\tilde{\mathbf{x}}_v)(\tilde{\mathbf{x}}_u\tilde{\mathbf{n}}_v)(\tilde{\mathbf{n}}\tilde{\mathbf{n}}) + (\tilde{\mathbf{x}}_u\tilde{\mathbf{n}}_v)(\tilde{\mathbf{x}}_u\tilde{\mathbf{n}})(\tilde{\mathbf{x}}_v\tilde{\mathbf{n}}) \\
&\quad -(\tilde{\mathbf{x}}_v\tilde{\mathbf{n}}_v)(\tilde{\mathbf{x}}_u\tilde{\mathbf{n}})^2 - (\tilde{\mathbf{n}}_v\tilde{\mathbf{n}})(\tilde{\mathbf{x}}_u\tilde{\mathbf{x}}_u)(\tilde{\mathbf{x}}_v\tilde{\mathbf{n}}) + (\tilde{\mathbf{n}}_v\tilde{\mathbf{n}})(\tilde{\mathbf{x}}_u\tilde{\mathbf{x}}_v)(\tilde{\mathbf{x}}_u\tilde{\mathbf{n}})\,.
\end{aligned}$$

Now we insert vertex positions $\mathbf{x}_i = (\mathbf{x}_{i0}, \mathbf{x}_{i1}, \mathbf{x}_{i2})^\top$ and normals $\mathbf{n}_i = (\mathbf{n}_{i0}, \mathbf{n}_{i1}, \mathbf{n}_{i2})^\top$, $i = 0,1,2$. Here, without loss of generality, we assume that the triangle is translated and rotated so that $\mathbf{x}_0 = \mathbf{0}$ and $\mathbf{x}_1, \mathbf{x}_2$ are located in the plane $z = 0$. Then it is straightforward to show the identity

$$\begin{aligned}
(EF - F^2)(\tilde{\mathbf{n}}\tilde{\mathbf{n}}) &= (\mathbf{x}_{10}\mathbf{x}_{21} - \mathbf{x}_{20}\mathbf{x}_{11})^2(\lambda_0\mathbf{n}_{02} + \lambda_1\mathbf{n}_{12} + \lambda_2\mathbf{n}_{22})^2 \\
&= \det(\tilde{\mathbf{x}}_u, \tilde{\mathbf{x}}_v, \tilde{\mathbf{n}})^2\,.
\end{aligned}$$

And for $\tilde{w}_{11}$ we get

$$\begin{aligned}
\mu\,\tilde{w}_{11} &= (\,\lambda_0(\mathbf{x}_{20}\mathbf{n}_{01}\mathbf{n}_{12} - \mathbf{x}_{20}\mathbf{n}_{02}\mathbf{n}_{11} - \mathbf{x}_{21}\mathbf{n}_{00}\mathbf{n}_{12} + \mathbf{x}_{21}\mathbf{n}_{02}\mathbf{n}_{10}) \\
&\quad +\lambda_1(\mathbf{x}_{20}\mathbf{n}_{01}\mathbf{n}_{12} - \mathbf{x}_{20}\mathbf{n}_{02}\mathbf{n}_{11} - \mathbf{x}_{21}\mathbf{n}_{00}\mathbf{n}_{12} + \mathbf{x}_{21}\mathbf{n}_{02}\mathbf{n}_{10}) \\
&\quad +\lambda_2(\mathbf{x}_{20}\mathbf{n}_{01}\mathbf{n}_{22} - \mathbf{x}_{20}\mathbf{n}_{02}\mathbf{n}_{21} - \mathbf{x}_{20}\mathbf{n}_{11}\mathbf{n}_{22} + \mathbf{x}_{20}\mathbf{n}_{12}\mathbf{n}_{21} \\
&\qquad -\mathbf{x}_{21}\mathbf{n}_{00}\mathbf{n}_{22} + \mathbf{x}_{21}\mathbf{n}_{02}\mathbf{n}_{20} + \mathbf{x}_{21}\mathbf{n}_{10}\mathbf{n}_{22} - \mathbf{x}_{21}\mathbf{n}_{12}\mathbf{n}_{20})\,) \\
&\quad (\mathbf{x}_{10}\mathbf{x}_{21} - \mathbf{x}_{11}\mathbf{x}_{20})\,(\lambda_0\mathbf{n}_{02} + \lambda_1\mathbf{n}_{12} + \lambda_2\mathbf{n}_{22}) \\
&= -\det(\tilde{\mathbf{x}}_v, \tilde{\mathbf{n}}_u, \tilde{\mathbf{n}})\det(\tilde{\mathbf{x}}_u, \tilde{\mathbf{x}}_v, \mathbf{n})\,.
\end{aligned}$$

Similar relations hold for $\tilde{w}_{12}$, $\tilde{w}_{21}$, and $\tilde{w}_{22}$, such that the Weingarten matrix can be expressed in terms of determinants:

$$\widetilde{\mathbf{W}} = \frac{1}{\sqrt{(\tilde{\mathbf{n}}\tilde{\mathbf{n}})}\det(\tilde{\mathbf{x}}_u, \tilde{\mathbf{x}}_v, \tilde{\mathbf{n}})} \begin{bmatrix} -\det(\tilde{\mathbf{x}}_v, \tilde{\mathbf{n}}_u, \tilde{\mathbf{n}}) & -\det(\tilde{\mathbf{x}}_v, \tilde{\mathbf{n}}_v, \tilde{\mathbf{n}}) \\ \det(\tilde{\mathbf{x}}_u, \tilde{\mathbf{n}}_u, \tilde{\mathbf{n}}) & \det(\tilde{\mathbf{x}}_u, \tilde{\mathbf{n}}_v, \tilde{\mathbf{n}}) \end{bmatrix}.$$

Hence, the mean curvature is expressed as

$$H = \frac{1}{2}(\tilde{w}_{11} + \tilde{w}_{22}) = \frac{\det(\tilde{\mathbf{x}}_u, \tilde{\mathbf{n}}_v, \tilde{\mathbf{n}}) - \det(\tilde{\mathbf{x}}_v, \tilde{\mathbf{n}}_u, \tilde{\mathbf{n}})}{2\|\tilde{\mathbf{n}}\|\det(\tilde{\mathbf{x}}_u, \tilde{\mathbf{x}}_v, \tilde{\mathbf{n}})}\,.$$

We consider the identities

$$\begin{aligned}
&\det(\tilde{\mathbf{x}}_u, \tilde{\mathbf{n}}_v, \tilde{\mathbf{n}}) - \det(\tilde{\mathbf{x}}_v, \tilde{\mathbf{n}}_u, \tilde{\mathbf{n}}) = \det(\tilde{\mathbf{n}}_u, \tilde{\mathbf{x}}_v, \tilde{\mathbf{n}}) - \det(\tilde{\mathbf{n}}_v, \tilde{\mathbf{x}}_u, \tilde{\mathbf{n}}) \\
&= (\tilde{\mathbf{n}}_v \times \tilde{\mathbf{x}}_u - \tilde{\mathbf{n}}_u \times \tilde{\mathbf{x}}_v)\,\tilde{\mathbf{n}} = ((\mathbf{n}_2 - \mathbf{n}_0) \times \tilde{\mathbf{x}}_u - (\tilde{\mathbf{n}}_1 - \tilde{\mathbf{n}}_0) \times \tilde{\mathbf{x}}_v)\,\tilde{\mathbf{n}} \\
&= (\mathbf{n}_0 \times \tilde{\mathbf{x}}_v - \tilde{\mathbf{n}}_0 \times \tilde{\mathbf{x}}_u + \mathbf{n}_1 \times (-\tilde{\mathbf{x}}_v) + \mathbf{n}_2 \times \tilde{\mathbf{x}}_u)\,\tilde{\mathbf{n}} \\
&= (\mathbf{h}\tilde{\mathbf{n}})\,,
\end{aligned}$$

where $\mathbf{h}$ is defined as in Section 3.2.2 (with $\mathbf{r}_0 = \tilde{\mathbf{x}}_v - \tilde{\mathbf{x}}_u$, $\mathbf{r}_1 = -\tilde{\mathbf{x}}_v$, $\mathbf{r}_2 = \tilde{\mathbf{x}}_u$), and

$$\det(\tilde{\mathbf{x}}_u, \tilde{\mathbf{x}}_v, \tilde{\mathbf{n}}) \;=\; (\tilde{\mathbf{x}}_u \times \tilde{\mathbf{x}}_v)\tilde{\mathbf{n}} \;=\; (\tilde{\mathbf{n}}\tilde{\mathbf{m}}) \;,$$

with the (not normalized) triangle normal $\tilde{\mathbf{m}} = \mathbf{r}_1 \times \mathbf{r}_2 = \tilde{\mathbf{x}}_u \times \tilde{\mathbf{x}}_v$ (cf. Section 3.2.2). This yields the formulation of the mean curvature (3.14).

For the Gaussian curvature we obtain

$$\begin{aligned} K &= \tilde{w}_{11}\tilde{w}_{22} - \tilde{w}_{12}\tilde{w}_{21} \\ &= \frac{\det(\tilde{\mathbf{x}}_v, \tilde{\mathbf{n}}_v, \tilde{\mathbf{n}}) \det(\tilde{\mathbf{x}}_u, \tilde{\mathbf{n}}_u, \tilde{\mathbf{n}}) - \det(\tilde{\mathbf{x}}_v, \tilde{\mathbf{n}}_u, \tilde{\mathbf{n}}) \det(\tilde{\mathbf{x}}_u, \tilde{\mathbf{n}}_v, \tilde{\mathbf{n}})}{\|\tilde{\mathbf{n}}\|^2 \det(\tilde{\mathbf{x}}_u, \tilde{\mathbf{x}}_v, \tilde{\mathbf{n}})^2} \;. \end{aligned}$$

The following identity proves the equivalence to (3.13)

$$\begin{aligned} &\det(\mathbf{n}_0, \mathbf{n}_1, \mathbf{n}_2) \det(\tilde{\mathbf{x}}_u, \tilde{\mathbf{x}}_v, \tilde{\mathbf{n}}) \\ &= \det(\mathbf{n}_0, \mathbf{n}_1, \mathbf{n}_2)(\mathbf{x}_{10}\mathbf{x}_{21} - \mathbf{x}_{11}\mathbf{x}_{20})(\mathbf{n}_{02}\lambda_0 + \mathbf{n}_{12}\lambda_1 + \mathbf{n}_{22}\lambda_2)(\lambda_0 + \lambda_1 + \lambda_2) \\ &= \det(\tilde{\mathbf{x}}_v, \tilde{\mathbf{n}}_v, \tilde{\mathbf{n}}) \det(\tilde{\mathbf{x}}_u, \tilde{\mathbf{n}}_u, \tilde{\mathbf{n}}) - \det(\tilde{\mathbf{x}}_v, \tilde{\mathbf{n}}_u, \tilde{\mathbf{n}}) \det(\tilde{\mathbf{x}}_u, \tilde{\mathbf{n}}_v, \tilde{\mathbf{n}}) \;, \end{aligned}$$

after factoring and canceling the term $\det(\tilde{\mathbf{x}}_u, \tilde{\mathbf{x}}_v, \tilde{\mathbf{n}})$, where $(\tilde{\mathbf{n}}\tilde{\mathbf{m}})$ is expressed as above for the mean curvature. (Note that for the barycentric coordinates $\lambda_0 + \lambda_1 + \lambda_2 = 1$. Here and in the following, we include the term to indicate the factorization.) Hence, we showed the equivalence, and we remark that the Gaussian curvature can be alternatively expressed in terms of determinants as

$$K \;=\; \frac{\det(\mathbf{n}_0, \mathbf{n}_1, \mathbf{n}_2)}{\|\tilde{\mathbf{n}}\|^2 \det(\tilde{\mathbf{x}}_u, \tilde{\mathbf{x}}_v, \tilde{\mathbf{n}})} \;.$$

This proves the relations (3.13) and (3.14).

Finally, we apply the limit (3.16) to prove the convergence of $\mathbf{T}$ when refining the triangulation (Theorem 3.1 (i)). Following the sketch in Section 3.2.3, we now assume that $\mathbf{x}_i$ and $\mathbf{n}_i$, are defined in terms of a scaling parameter $t > 0$, the cubic height function $z(x, y)$ and certain scalar constants x_i, y_i, $0 \le i \le 2$.

We examine the limit of the Weingarten map $\widetilde{\mathbf{W}}$ for $t \to 0$. Let $p_Q^{(j)}(t) := \sum_{i \in Q} c_{ij} t^i$ denote certain polynomials with coefficients c_{ij}. Considering the indi-

vidual terms of $\widetilde{\mathbf{W}}$, we obtain polynomials of the following types

$$\begin{aligned}
(\tilde{\mathbf{n}}\tilde{\mathbf{n}}) &= (\lambda_0+\lambda_1+\lambda_2)^2 \; + p^{(0)}_{\{2,3,4\}}(t)\,, \\
\det(\tilde{\mathbf{x}}_u, \tilde{\mathbf{x}}_v, \tilde{\mathbf{n}}) &= (x_0y_1 - x_0y_2 - x_1y_0 + x_1y_2 + x_2y_0 - x_2y_1)\, t^2 \; + p^{(1)}_{\{4,5,6\}}(t)\,, \\
\det(\tilde{\mathbf{x}}_v, \tilde{\mathbf{n}}_u, \tilde{\mathbf{n}}) &= ((x_0y_0 - x_0y_2 - x_1y_0 + x_1y_2)\,\kappa_1 - (x_0y_0 - x_0y_1 - x_2y_0 + x_2y_1)\,\kappa_2) \\
&\quad (\lambda_0+\lambda_1+\lambda_2)\, t^2 \; + p^{(2)}_{\{3,4,5,6,7\}}(t)\,, \\
\det(\tilde{\mathbf{x}}_v, \tilde{\mathbf{n}}_v, \tilde{\mathbf{n}}) &= (y_0-y_2)(x_0-x_2)(\kappa_1-\kappa_2)(\lambda_0+\lambda_1+\lambda_2)\, t^2 \; + p^{(3)}_{\{3,4,5,6,7\}}(t)\,, \\
\det(\tilde{\mathbf{x}}_u, \tilde{\mathbf{n}}_u, \tilde{\mathbf{n}}) &= (y_0-y_1)(x_0+x_1)(\kappa_1-\kappa_2)(\lambda_0+\lambda_1+\lambda_2)\, t^2 \; + p^{(4)}_{\{3,4,5,6,7\}}(t)\,, \\
\det(\tilde{\mathbf{x}}_u, \tilde{\mathbf{n}}_v, \tilde{\mathbf{n}}) &= (y_2-y_0)(x_0-x_2)(\kappa_1-\kappa_2)(\lambda_0+\lambda_1+\lambda_2)\, t^2 \; + p^{(5)}_{\{3,4,5,6,7\}}(t)\,.
\end{aligned}$$

We realize that for $\lim_{t\to 0}(\tilde{\mathbf{n}}\tilde{\mathbf{n}}) = 1$. In order to compute the limit of $\widetilde{\mathbf{W}}$, we factor t^2 in the remaining expressions. Then $\lim_{t\to 0}\tilde{w}_{ij}$, $i,j \in \{1,2\}$, are given as the respective quotients. After canceling t^2 all remaining non-constant terms $p^{(j)}_Q(t)/t^2$ vanish as $t \to 0$. From this, we compute the limits of the Gaussian and mean curvature, which yields

$$\begin{aligned}
\lim_{t\to 0} K &= \lim_{t\to 0}(\tilde{w}_{11}\tilde{w}_{22} - \tilde{w}_{12}\tilde{w}_{21}) \;=\; \kappa_1\,\kappa_2\,, \quad \text{and} \\
\lim_{t\to 0} H &= \lim_{t\to 0}\left(\frac{1}{2}(\tilde{w}_{11}+\tilde{w}_{22})\right) \;=\; \frac{1}{2}(\kappa_1+\kappa_2)\,,
\end{aligned}$$

and hence shows the equivalence of the principal curvatures κ_1, κ_2 of $z(x,y)$ to the eigenvalues of $\widetilde{\mathbf{W}}$. (Here, we commemorate the definition of the cubic height surface $z(x,y)$.) From this result we obtain the principal directions to construct the tensor of curvature, which proves (3.16).

Appendix B

Approximation Properties of Quadratic Super Splines

We give a short sketch of the proof of Theorem 8.2 (Section 8.4), including the main ideas. The complete proof is given in [Nürnberger et al. 2004c].

We associate the operator $\mathcal{Q}(f) :\equiv s_f$, $f \in C(\Omega^*)$ with the reconstruction method described in Section 8.3. Furthermore, $\|.\|_B$ is the standard infinity norm

$$\|f\|_B := \sup\{|f(x,y,z)| : (x,y,z) \in B\}$$

for continuous functions $f \in C(\Omega)$ on a compact subset $B \subseteq \Omega$, and $W_\infty^m(\Omega^*)$ and $|f|_{m,\infty,\Omega^*}$ are defined as before. We use the following Lemmata from [Nürnberger et al. 2004c]:

Lemma 1 *(boundedness of derivatives) Let $T \in \Delta$ be a tetrahedron and $p \in \mathcal{P}_2$ a quadratic polynomial on T. Then the following statements hold.*
(i) For the first derivatives of p in $x, y,$ and z direction denoted by $D_x\, p, D_y\, p, D_z\, p$, respectively, we have

$$\max\{\|D_x^\alpha D_y^\beta D_z^\gamma\, p\|_T : \alpha+\beta+\gamma = 1\} \le (29/h)\, \|p\|_T.$$

(ii) For the second derivatives of p in $x, y,$ and z direction denoted by $D_x^2\, p$, $D_y^2\, p$, $D_z^2\, p$, $D_xD_y\, p$, $D_xD_z\, p, D_yD_z\, p$, respectively, we have

$$\max\{\|D_x^\alpha D_y^\beta D_z^\gamma\, p\|_T : \alpha+\beta+\gamma = 2\} \le (128/h^2)\, \|p\|_T.$$

Lemma 2 *(uniform boundedness)) Let $f \in C(\Omega^*)$, $T \in \Delta$ and $Q_{i,j,k} \in \diamondsuit$ the cube with $T \subseteq Q_{i,j,k}$. Then, we have*

$$\|\mathcal{Q}(f)\|_T \le (9/8)\, \|f\|_{\Omega_T},$$

where $\Omega_T \subseteq \Omega^*$ is an appropriate cube with edge length $3h$ containing $Q_{i,j,k}$.

Lemma 1 follows from the definition of the tetrahedral partition Δ and the expression of the partial derivatives in Bernstein-Bézier form. Lemma 2 can be shown by considering the reconstruction from the 27 samples of an arbitrary function. Similarly, to show Theorem 8.3 (reproduction of polynomials) the respective polynomials are considered. We omit the details here and refer to [Nürnberger et al. 2004c]. Now the proof of Theorem 8.2 reads as follows:
Let $T \in \Delta$ be an arbitrary tetrahedron with $T \subseteq \Omega$, and $Q_{i,j,k} \in \diamondsuit$ the cube with $T \subseteq Q_{i,j,k}$. Moreover, let Ω_T be chosen as in Lemma 2.

In order to show (i), we consider the linear Taylor polynomial p_f of $f \in W_\infty^2(\Omega^*)$ at the center point $v_{Q_{i,j,k}} = (v_x, v_y, v_z)$ of $Q_{i,j,k}$, i.e.

$$p_f(v) = \sum_{0 \le \alpha+\beta+\gamma \le 1} D_x^\alpha D_y^\beta D_z^\gamma \, f(v_{Q_{i,j,k}}) \, (x - v_x)^\alpha (y - v_y)^\beta (z - v_z)^\gamma,$$

for all $v = (x, y, z) \in \Omega_T$. Since the diameter of the smallest sphere containing Ω_T is $3\sqrt{3}\, h$, it follows from a well known property of p_f that

$$\|f - p_f\|_{\Omega_T} \le \ (27/8) \, |f|_{2,\infty,\Omega^*} \, h^2. \tag{B.1}$$

Similarly,

$$\|f - p_f\|_T \le \ (3/8) \, |f|_{2,\infty,\Omega^*} \, h^2. \tag{B.2}$$

Moreover, applying the mean value Theorem, we have for all $v \in T$,

$$\begin{aligned} |D_x^\alpha D_y^\beta D_z^\gamma \, (f - p_f)(v)| &= |D_x^\alpha D_y^\beta D_z^\gamma \, f(v) - D_x^\alpha D_y^\beta D_z^\gamma \, f(v_{Q_{i,j,k}})| \\ &\le \ (\sqrt{3}/2) \, |f|_{2,\infty,\Omega^*} \, h, \qquad \alpha + \beta + \gamma = 1. \end{aligned} \tag{B.3}$$

It follows from Theorem 8.3, (i) that $\mathcal{Q}(p_f) = p_f$, and therefore the triangle inequality implies that for all $\alpha + \beta + \gamma \in \{0, 1\}$,

$$\begin{aligned} &\|D_x^\alpha D_y^\beta D_z^\gamma (f - \mathcal{Q}(f))\|_T \le \\ &\qquad \|D_x^\alpha D_y^\beta D_z^\gamma (f - p_f)\|_T + \|D_x^\alpha D_y^\beta D_z^\gamma \mathcal{Q}(p_f - f)\|_T. \end{aligned} \tag{B.4}$$

In view of (B.2) and (B.3), it remains to consider the second term on the right of this inequality. If $\alpha + \beta + \gamma = 0$, then it follows from Lemma 2 and (B.1) that

$$\|\mathcal{Q}(p_f - f)\|_T \le \ (9/8) \, \|p_f - f\|_{\Omega_T} \le \ (243/64) \, |f|_{2,\infty,\Omega^*} \, h^2.$$

Moreover, since obviously $\mathcal{Q}(p_f - f)|_T \in \mathcal{P}_2$, we can apply Lemma 1 to obtain for all $\alpha + \beta + \gamma = 1$,

$$\|D_x^\alpha D_y^\beta D_z^\gamma \mathcal{Q}(p_f - f)\|_T \le \ (29/h) \, \|\mathcal{Q}(p_f - f)\|_T \le \ (881/8) \, |f|_{2,\infty,\Omega^*} \, h.$$

The assertions in (i) now follow from (B.2), (B.3), and (B.4).

For proving (ii), we argue differently. We already know from Theorem 8.3 that if $p \in \mathcal{P}_2$ and $\alpha+\beta+\gamma \in \{1,2\}$ then $D_x^\alpha D_y^\beta D_z^\gamma\ \mathcal{Q}(p) = \mathcal{Q}(D_x^\alpha D_y^\beta D_z^\gamma\ p)$. In the following we will use this fact.

Consider the Taylor polynomial $p_f \in \mathcal{P}_2$ of $f \in W_\infty^3(\Omega^*)$ at the center point $v_{Q_{i,j,k}} = (v_x, v_y, v_z)$ of $Q_{i,j,k}$, i.e.

$$p_f(v) = \sum_{0\le\alpha+\beta+\gamma\le 2} D_x^\alpha D_y^\beta D_z^\gamma\ f(v_{Q_{i,j,k}})/(\alpha!\beta!\gamma!)\ (x-v_x)^\alpha (y-v_y)^\beta (z-v_z)^\gamma,$$

for all $v=(x,y,z) \in \Omega_T$. It is known that

$$\|f-p_f\|_{\Omega_T} \le\ (27\sqrt{3}/16)\ |f|_{3,\infty,\Omega^*}\ h^3, \tag{B.5}$$

and similar as above, we have

$$\|D_x^\alpha D_y^\beta D_z^\gamma\ (f-p_f)\|_T \le\ (\sqrt{3}/2)\ |f|_{3,\infty,\Omega^*}\ h, \qquad \alpha+\beta+\gamma = 2. \tag{B.6}$$

By applying the mean value Theorem to the first derivative of the error and a standard argument, we get

$$\|D_x^\alpha D_y^\beta D_z^\gamma\ (f-p_f)\|_T \le\ (3/4)\ |f|_{3,\infty,\Omega^*}\ h^2, \qquad \alpha+\beta+\gamma = 1. \tag{B.7}$$

The triangle inequality implies that for all $\alpha+\beta+\gamma \in \{1,2\}$,

$$\begin{aligned}\|D_x^\alpha D_y^\beta D_z^\gamma (f-\mathcal{Q}(f))\|_T \le \\ \|D_x^\alpha D_y^\beta D_z^\gamma (f-p_f)\|_T + \|D_x^\alpha D_y^\beta D_z^\gamma (p_f-\mathcal{Q}(f))\|_T.\end{aligned} \tag{B.8}$$

In view of (B.6) and (B.7), it remains to consider the second term on the right of this inequality. For $\alpha+\beta+\gamma \in \{1,2\}$, the polynomial $D_x^\alpha D_y^\beta D_z^\gamma\ p_f$ is obviously linear or constant, and therefore Theorem 8.3 (i) implies that

$$D_x^\alpha D_y^\beta D_z^\gamma\ p_f \equiv \mathcal{Q}(D_x^\alpha D_y^\beta D_z^\gamma\ p_f).$$

On the other hand, we have $p_f \in \mathcal{P}_2$, and therefore it follows from Theorem 8.3 as pointed out above that for all $\alpha+\beta+\gamma \in \{1,2\}$,

$$D_x^\alpha D_y^\beta D_z^\gamma\ (p_f-\mathcal{Q}(f)) \equiv D_x^\alpha D_y^\beta D_z^\gamma\ \mathcal{Q}(p_f-f).$$

If $\alpha+\beta+\gamma = 1$, then it follows from Lemma 1, Lemma 2 and (B.5) that

$$\|D_x^\alpha D_y^\beta D_z^\gamma (p_f-\mathcal{Q}(f))\|_T \ \ \le\ \ (111\sqrt{3})/2\ |f|_{3,\infty,\Omega^*}\ h^2.$$

Analogously, we have for $\alpha+\beta+\gamma = 2$,

$$\|D_x^\alpha D_y^\beta D_z^\gamma (p_f-\mathcal{Q}(f))\|_T \ \ \le\ \ 243\sqrt{3}\ |f|_{3,\infty,\Omega^*}\ h.$$

The assertions in (ii) now follow from (B.6), (B.7), and (B.8). This completes the proof of the theorem.

Bibliography

[Alfeld et al. 1987] ALFELD, P., PIPER, B., AND SCHUMAKER, L. L. 1987. An explicit basis for C^1 quartic bivariate splines. *SIAM J. Numer. Anal. 24*, 891–911.

[Alfeld et al. 1992] ALFELD, P., SCHUMAKER, L., AND SIRVENT, M. 1992. The dimension and existence of local bases for multivariate spline spaces. *Journal of Approximation Theory 70*, 243–264.

[Alfeld 1984] ALFELD, P. 1984. A trivariate C^1 Clough-Tocher interpolating scheme. *CAGD 1*, 169–181.

[Alfeld 1990] ALFELD, P. 1990. Scattered data fitting in two and three variables. In *Curves and Surfaces: Olso 1989*, Academic Press.

[Alliez and Desbrun 2001a] ALLIEZ, P., AND DESBRUN, M. 2001. Valence-Driven connectivity encoding for 3D meshes. In *Proc. Eurographics*, 480–489.

[Alliez and Desbrun 2001b] ALLIEZ, P., AND DESBRUN, M. 2001. Progressive compression for lossless transmission of triangle meshes. In *Proc. SIGGRAPH*, 195–202.

[Alliez and Gotsman 2005] ALLIEZ, P., AND GOTSMAN, C. 2005. Recent advances in compression of 3d meshes. In *Advances in Multiresolution for Geometric Modelling*, Springer, N. Dodgson, M. Floater, and M. Sabin, Eds., 3–26.

[Alliez et al. 2003] ALLIEZ, P., COHEN-STEINER, D., DEVILLERS, O., LÉVY, B., AND DESBRUN, M. 2003. Anisotropic polygonal remeshing. *ACM Transactions on Graphics 22*, 3 (July), 485–493.

[Appel et al. 1979] APPEL, A., ROHLF, F., AND STEIN, A. 1979. The haloed line effekt for hidden line elimination. In *Proc. SIGGRAPH*, 151–157.

[Arqus and Braud 2000] ARQUS, A., AND BRAUD, J.-F. 2000. Rooted maps on orientable surfaces, Riccati's equation and continued fractions. *Discrete Mathematics 215*, 1-3, 1–12.

[Bajaj and Schikore 1998] BAJAJ, C., AND SCHIKORE, D. 1998. Topology-preserving data simplification with error bounds. *Comput. & Graphics 22*, 1, 3–12.

[Bajaj et al. 1995] BAJAJ, C., BERNARDINI, F., AND XU, G. 1995. Automatic reconstruction of surfaces and scalar fields from 3D scans. In *SIGGRAPH'95*, 109–118.

[Bajaj et al. 1998] BAJAJ, C., PASCUCCI, V., AND SCHIKORE, D. 1998. Visualization of scalar topology for structural enhancement. In *Proc. IEEE Visualization '98*, 51–58.

[Bajaj 1997] BAJAJ, C. 1997. Implicit Surface Patches. In *Introduction to Implicit Surfaces*, Morgan Kaufmann, J. Bloomenthal, Ed., 99–125.

[Bajaj 1999] BAJAJ, C. 1999. *Data Visualization Techniques*. John Wiley & Sons.

[Barthe et al. 2002] BARTHE, L., MORA, B., DODGSON, N., AND SABIN, M. 2002. Triquadratic reconstruction for interactive modelling of potential fields. In *Shape Modeling International 2002*, 145–153.

[Batra et al. 1999] BATRA, R., KLING, K., AND HESSELINK, L. 1999. Topology based vector field comparison

using graph methods. In *Proc. IEEE Visualization '99, Late Breaking Hot Topics*, 25–28.

[Beatson and Ziegler 1985] BEATSON, R. K., AND ZIEGLER, Z. 1985. Monotonicity preserving surface interpolation. *SIAM J. Numer. Anal. 22*, 2, 401–411.

[Berztiss 1986] BERZTISS, A. 1986. A taxonomy of binary tree traversals. *BIT 26*, 266–276.

[Bischoff and Kobbelt 2004] BISCHOFF, S., AND KOBBELT, L. 2004. Teaching meshes, subdivision and multiresolution techniques. *Computer Aided Design 36*, 14, 1483–1500.

[Bogomjakov and Gotsman 2002] BOGOMJAKOV, A., AND GOTSMAN, C. 2002. Universal rendering sequences for transparent vertex caching of progressive meshes. *Computer Graphics Forum 21*, 2, 137–148.

[Bonneau et al. 1996] BONNEAU, G.-P., HAHMANN, S., AND NIELSON, G. 1996. BLaC-Wavelets: A multiresolution analysis with non-nested spaces. In *IEEE Visualization 1996*, 43–48.

[Botsch et al. 2000] BOTSCH, M., RÖSSL, C., AND KOBBELT, L. 2000. Feature sensitive sampling for interactive remeshing. In *Proc. Vision, Modeling, and Visualization*, 129–136.

[Brodlie and Wood 2001] BRODLIE, K., AND WOOD, J. 2001. Recent Advances in Volume Visualization. *Computer Graphics Forum 20*, 2, 125–148.

[Brown 1964] BROWN, W. 1964. Enumeration of triangulations of the disk. *Proc. Lond. Math. Soc., III. Ser. 14*, 746–768.

[Buhmann 2000] BUHMANN, M. D. 2000. Radial Basis Functions. *Acta Numerica*, 1–38.

[Campagna et al. 1998] CAMPAGNA, S., KOBBELT, L., AND SEIDEL, H.-P. 1998. Directed edges – a scalable representation for triangle meshes. *Journal of Graphics Tools 3*, 4, 1–12.

[Carbal and Leedom 1993] CARBAL, B., AND LEEDOM, L. 1993. Imaging vector fields using line integral convolution. In *Proc. SIGGRAPH*, 263–272.

[Carr et al. 2001a] CARR, H., MÖLLER, T., AND SNOEYINK, J. 2001. Simplicial subdivisions and sampling artifacts. In *IEEE Visualization 2001*, 99–106.

[Carr et al. 2001b] CARR, J., BEATSON, R., CHERRIE, J., MITCHELL, T., FRIGHT, W., MCCALLUM, B., AND EVANS, T. 2001. Reconstruction and representation of 3d objects with radial basis functions. In *SIGGRAPH'01*, 67–76.

[Cazals and Pouget 2003] CAZALS, F., AND POUGET, M. 2003. Estimating differential quantities using polynomial fitting of osculating jets. In *Symposium on Geometry Processing*, 177–187.

[Chen et al. 2000] CHEN, M., KAUFMAN, A., AND YAGEL, R. 2000. *Volume Graphics*. Springer.

[Chui and He 1986] CHUI, C. K., AND HE, X. 1986. On the location of sample points for interpolation by c^1 quadratic splines. In *Numerical Methods in Approximation Theory, Vol. 8.*, L. Collatz, G. Meinardus, and G. Nürnberger, Eds., 30–43.

[Chui 1988] CHUI, C. 1988. *Multivariate Splines*. CBMS 54, SIAM.

[Cohen-Or et al. 1999] COHEN-OR, D., LEVIN, D., AND REMEZ, O. 1999. Progressive compression of arbitrary triangular meshes. In *Proc. IEEE Visualization*, 67–72.

[Cohen-Steiner and Morvan 2003] COHEN-STEINER, D., AND MORVAN, J.-M. 2003. Restricted delaunay triangulations and normal cycle. In *Proceedings of the nineteenth Conference on Computational Geometry (SCG-03)*, 312–321.

[Dagnino and Lamberti 2004] DAGNINO, C., AND LAMBERTI, P. 2004. Some performances of local bivariate quadratic

c^1 quasi-interpolating splines on non-uniform type-2 triangulations. In *University of Turin, Department of Mathematics (preprint)*.

[Dahmen and Thamm-Schaar 1993] DAHMEN, W., AND THAMM-SCHAAR, T.-M. 1993. Cubicoids: modeling and visualization. *Computer Aided Geometric Design 10*, 2, 89–108.

[Dahmen 1989] DAHMEN, W. 1989. Smooth piecewise quadric surfaces. In *Math. Methods in CAGD*, Academic Press, T. Lyche and L. Schumaker, Eds., 181–194.

[Davydov and Zeilfelder 2003] DAVYDOV, O., AND ZEILFELDER, F. 2003. Scattered data fitting by direct extension of local polynomials with bivariate splines. *Adv. Comp. Math. (to appear)*.

[Davydov et al. 2004] DAVYDOV, O., MORANDI, R., AND SESTINI, M. 2004. Local hybrid approximation for scattered data fitting with bivariate splines. *(preprint)*.

[Davydov 2002] DAVYDOV, O. 2002. On the approximation power of local least squares polynomials. In *Algorithms for Approximation IV*, J. Levesley, I. Anderson, and J. Mason, Eds., 346–353.

[de Boor 1987] DE BOOR, C. 1987. B-form basics. In *Geometric Modelling*, SIAM, G. Farin, Ed., 131–148.

[de Casteljau 1963] DE CASTELJAU, P. 1963. Courbes et surfaces à poles. *André Citroën, Automobiles SA, Paris*.

[de Leeuw and van Liere 1999a] DE LEEUW, W., AND VAN LIERE, R. 1999. Collapsing flow topology using area metrics. In *Proc. IEEE Visualization '99*, D. Ebert, M. Gross, and B. Hamann, Eds., 149–354.

[de Leeuw and van Liere 1999b] DE LEEUW, W., AND VAN LIERE, R. 1999. Visualization of global flow structures using multiple levels of topology. In *Data Visualization 1999. Proc. VisSym 99*, 45–52.

[Denny and Sohler 1997] DENNY, M., AND SOHLER, C. 1997. Encoding a triangulation as a permutation of its point set. In *Proceedings of the 9th Canadian Conference on Computational Geometry*, 39–43.

[Deo and Litow 1998] DEO, N., AND LITOW, B. 1998. A structural approach to graph compression. In *MFCS Workshop on Communications*, 91–101.

[Deussen et al. 1999] DEUSSEN, O., HAMEL, J., RAAB, A., SCHLECHTWEG, S., AND STROTHOTTE, T. 1999. An illustration technique using hardware-based intersections and skeletons. In *Proc. Graphics Interface*, 175–182.

[Dinh et al. 2001] DINH, H., TURK, G., AND SLABAUGH, G. 2001. Reconstructing surfaces using anisotropic basis functions. In *International Conference on Computer Vision (ICCV)*, 606–613.

[Do Carmo 1976] DO CARMO, P. 1976. *Differential Geometry of curves and surfaces*. Prentice-Hall, Englewood Cliffs.

[Dooley and Cohen 1990] DOOLEY, D., AND COHEN, M. 1990. Automatic illustration of 3d geometric models: Lines. *Computer Graphics 23*, 77–82.

[Edelsbrunner et al. 2001] EDELSBRUNNER, H., HARER, J., AND ZOMORODIAN, A. 2001. Hierarchical morse complexes for piecewise linear 2-manifolds. In *Proc. 17th Sympos. Comput. Geom. 2001*.

[Elber 1995] ELBER, G. 1995. Line art rendering via a coverage of isoparametric curves. *IEEE Transactions on Visualization and Computer Graphics 1*, 3, 231–239.

[Elber 1998] ELBER, G. 1998. Line art illustrations of parametric and implicit forms. *IEEE Transactions on Visualization and Computer Graphics 4*, 1, 1–11.

[Elber 1999] ELBER, G. 1999. Interactive line art rendering of freeform-surfaces. In *Proc. Eurographics*, 1–12.

[Evans et al. 1996] EVANS, F., SKIENA, S., AND VARSHNEY, A. 1996. Optimizing triangle strips for fast rendering. In *Proce. IEEE Visualization*, 319–326.

[Farin 1986] FARIN, G. 1986. Triangular Bernstein-Bézier patches. *CAGD 3*, 2, 83–127.

[Firby and Gardiner 1982] FIRBY, P., AND GARDINER, C. 1982. *Surface Topology*. Ellis Horwood Ltd., ch. 7, 115–135. Vector Fields on Surfaces.

[Foley et al. 1996] FOLEY, J., VAN DAM, A., FEINER, S., AND HUGHES, J. 1996. *Computer Graphics, Principles and Practice*. Addison-Wesley Publishing Company, Reading Massachusetts.

[Foley 1986] FOLEY, A. 1986. Scattered Data Interpolation and Approximation with Error Bounds. *CAGD 3*, 163–177.

[Franke and Hagen 1999] FRANKE, R., AND HAGEN, H. 1999. Least Squares Surface Approximation using Multiquadrics and Parameter Domain Distortion. *CAGD 16*, 3, 177–196.

[Garcke et al. 2000] GARCKE, H., PREUSSER, T., RUMPF, M., TELEA, A., WEIKARDT, U., AND VAN WIJK, J. 2000. A continuous clustering method for vector fields. In *Proc. IEEE Visualization 2000*, T. Ertl, B. Hamann, and A. Varshney, Eds., 351–358.

[Gerstner and Rumpf 2000] GERSTNER, T., AND RUMPF, M. 2000. Multiresolutional Parallel Isosurface Extraction based on Tetrahedral Bisection. In *Volume Graphics*, Springer, M. Chen, A. Kaufman, and R. Yagel, Eds., 267–278.

[Globus and Levit 1991] GLOBUS, A., AND LEVIT, C. 1991. A tool for visualizing of three-dimensional vector fields.

In *Proc. IEEE Visualization '91*, IEEE Computer Society Press, 33–40.

[Goldfeather and Interrante 2004] GOLDFEATHER, J., AND INTERRANTE, V. 2004. A novel cubic-order algorithm for approximating principal directions vectors. *ACM Transactions on Graphics 23*, 1, 45–63.

[Gonzales and Woods 1993] GONZALES, R., AND WOODS, R. 1993. *Digital Image Processing*. Addison-Wesley.

[Gooch and Gooch 2001] GOOCH, B., AND GOOCH, A. 2001. *Non-Photorealistic Rendering*. A K Peters.

[Gotsman et al. 2002] GOTSMAN, C., GUMHOLD, S., AND KOBBELT, L. 2002. Simplification and compression of 3D meshes. In *Tutorials on Multiresolution in Geometric Modelling*, A. Iske, E. Quak, and M. S. Floater, Eds., Mathematics and Visualization. Springer, 319–361.

[Graham et al. 1994] GRAHAM, R. L., KNUTH, D. E., AND PATASHNIK, O. 1994. *Concrete Mathematics: A Foundation for Computer Science*. Addison-Wesley.

[Grosso et al. 1997] GROSSO, R., LÜRIG, C., AND ERTL, T. 1997. The multilevel finite element method for adaptive mesh optimization and visualization of volume data. In *IEEE Visualization 1997*, 387–394.

[Gumhold and Straßer 1998] GUMHOLD, S., AND STRASSER, W. 1998. Real time compression of triangle mesh connectivity. In *Proc. SIGGRAPH*, 133–140.

[Gumhold 1999] GUMHOLD, S. 1999. Improved cut-border machine for triangle mesh compression. In *Proc. Vision, Modeling and Visualization*, 261–267.

[Gumhold 2000] GUMHOLD, S. 2000. New bounds on the encoding of planar triangulations. Tech. Rep.

WSI-2000-1, Wilhelm-Schikard-Institut für Informatik, Tübingen, Mar.

[Haber et al. 2001] HABER, J., ZEILFELDER, F., DAVYDOV, O., AND SEIDEL, H.-P. 2001. Smooth approximation and rendering of large scattered data sets. In *IEEE Visualization 2001*, 341–347.

[Hagen et al. 1992] HAGEN, H., ET AL. 1992. Surface interrogation algorithms. *IEEE Computer Graphics and Applications 12*, 5, 53–60.

[Hangelbroek et al. 2004] HANGELBROEK, T., NÜRNBERGER, G., RÖSSL, C., SEIDEL, H.-P., AND ZEILFELDER, F. 2004. Dimension of C^1 splines on type-6 tetrahedral partitions. *Journal of Approximation Theory (to appear)*.

[Haralick et al. 1987] HARALICK, R., STERNBERG, S., AND ZHUANG, X. 1987. Image analysis using mathematical morphology. *IEEE Transactions on Pattern Analysis and Machine Intelligence 9*, 4, 532–560.

[Heckel et al. 1999] HECKEL, B., WEBER, G., HAMANN, B., AND K.I.JOY. 1999. Construction of vector field hierarchies. In *Proc. IEEE Visualization '99*, D. Ebert, M. Gross, and B. Hamann, Eds., 19–26.

[Helman and Hesselink 1989] HELMAN, J., AND HESSELINK, L. 1989. Representation and display of vector field topology in fluid flow data sets. *IEEE Computer 22*, 8 (August), 27–36.

[Helman and Hesselink 1991] HELMAN, J., AND HESSELINK, L. 1991. Visualizing vector field topology in fluid flows. *IEEE Computer Graphics and Applications 11* (May), 36–46.

[Hildebrandt and Polthier 2004] HILDEBRANDT, K., AND POLTHIER, K. 2004. Anisotropic filtering of non-linear surface features. In *Proc. Eurographics*, 391–400.

[Holliday and Nielson 2000] HOLLIDAY, D., AND NIELSON, G. 2000. Progressive volume models for rectilinear data using tetrahedral Coons volumes. In *Data Visualization 2000*, Springer, 83–92.

[Hoppe 1996] HOPPE, H. 1996. Progressive meshes. In *Proc. SIGGRAPH*, 99–108.

[Hoschek and Dankwort 1996] HOSCHEK, J., AND DANKWORT, W. 1996. *Reverse Engineering*. Teubner.

[Hoschek and Dietz 1998] HOSCHEK, J., AND DIETZ, U. 1998. A geometric concept of reverse engineering of shape: Approximation and feature lines. In *Mathematical Methods for Curves and Surfaces II*, 253–262.

[Hoschek and Lasser 1993] HOSCHEK, J., AND LASSER, D. 1993. *Fundamentals of Computer Aided Geometric Design*. A.K. Peters.

[Interrante 1997] INTERRANTE, V. 1997. Illustrating surface shape in volume data via principal direction driven 3d line integral convolution. In *Proc. SIGGRAPH*, 109–116.

[Isenburg and Snoeyink 1999] ISENBURG, M., AND SNOEYINK, J. 1999. Mesh collapse compression. In *Proceedings of the Conference on Computational Geometry*, 419–420.

[Isenburg and Snoeyink 2000] ISENBURG, M., AND SNOEYINK, J. 2000. Face fixer: Compressing polygon meshes with properties. In *Proc. SIGGRAPH*, 263–270.

[Isenburg and Snoeyink 2001] ISENBURG, M., AND SNOEYINK, J. 2001. Spirale reversi: Reverse decoding of the edgebreaker encoding. *CGTA: Computational Geometry: Theory and Applications 20*.

[Isenburg 2000] ISENBURG, M. 2000. Triangle strip compression. In *Proc. Graphics Interface*, 197–204.

[Isselhard et al. 1998] ISSELHARD, F., BRUNNET, G., AND SCHREIBER, T. 1998. Extraction of first-order feature lines from a discretized surface. In *Mathematics of surfaces*, R. Cripps, Ed., 125–137.

[Ivrissimtzis et al. 2002] IVRISSIMTZIS, I., RÖSSL, C., AND SEIDEL, H.-P. 2002. A divide and conquer algorithm for triangle mesh connectivity encoding. In *Proc. Pacific Graphics*, 294–303.

[Ivrissimtzis et al. 2003] IVRISSIMTZIS, I., RÖSSL, C., AND SEIDEL, H.-P. 2003. Tree-based data structures for triangle mesh connectivity encoding. In *Geometric Modeling for Scientific Visualization*, G. Brunnett, B. Hamann, and H. Müller, Eds. Springer, Heidelberg, Germany, 171–187.

[Jeeawock-Zedek 1994] JEEAWOCK-ZEDEK, F. 1994. Operator norm and error bounds for interpolating quadratic splines on a non-uniform type-2 triangulation of a rectangular domain. *Approx. Theory and Appl. 10*, 2, 1–16.

[Kähler et al. 2001] KÄHLER, K., RÖSSL, C., SCHNEIDER, R., VORSATZ, J., AND SEIDEL, H.-P. 2001. Efficient processing of large 3d meshes. In *Proc. Shape Modeling International*, 228–239.

[Katajainen and Mäkinen 1990] KATAJAINEN, J., AND MÄKINEN, E. 1990. Tree compression and optimization with applications. *Int. J. Found. Comput. Sci. 1*, 4, 425–447.

[Kaufman 2000] KAUFMAN, A. 2000. State-of-the-art in volume graphics. In *Volume Graphics*, Springer, M. Chen, A. Kaufman, and R.Yagel, Eds., 3–28.

[Kenwright et al. 1999] KENWRIGHT, D., HENZE, C., AND LEVIT, C. 1999. Feature extraction of separation and attachment lines. *IEEE Transactions on Visualization and Computer Graphics 5*, 2, 135–144.

[Kettner 1998] KETTNER, L. 1998. Designiung a data structure for polyhedral surafces. In *Proc. ACM Symposium on Computational Geometry*, 146–154.

[King and Rossignac 1999] KING, D., AND ROSSIGNAC, J. 1999. Guaranteed 3.67V bit encoding of planar triangle graphs. In *Proceedings of the 11th Canadian Conference on Computational Geometry*, 146–149.

[Kobbelt et al. 2000] KOBBELT, L., BISCHOFF, S., BOTSCH, M., KÄHLER, K., RÖSSL, C., SCHNEIDER, R., AND VORSATZ, J. 2000. Geometric modeling based on polygonal meshes. In *Tutorials Eurographics*, 1–47.

[Kohlmüller et al. 2003a] KOHLMÜLLER, N., G.NÜRNBERGER, AND ZEILFELDER, F. 2003. Construction of cubic 3D spline surfaces by Lagrange interpolation at selected points. In *Curve and Surface Fitting, Saint-Malo 2002*, 245–254.

[Kohlmüller et al. 2003b] KOHLMÜLLER, N., NÜRNBERGER, G., AND ZEILFELDER, F. 2003. Optimal approximation order of interpolation by cubic spline surfaces. In *Curve and Surface Fitting, Saint-Malo 2002*, Vanderbilt University Press Nashville, 235–245.

[Lai and Méhauté 2003] LAI, M.-J., AND MÉHAUTÉ, A. L. 2003. A new kind of trivariate C^1 spline. *Adv. Comp. Math. (to appear)*.

[LaMar et al. 1999] LAMAR, E., HAMANN, B., AND JOY, K. 1999. High-quality rendering of smooth isosurfaces. *Journal of Visualization and Computer Animation 10(2)*, 79–90.

[Lancaster and Šalkauskas 1986] LANCASTER, P., AND ŠALKAUSKAS, K. 1986. *Curve and Surface Fitting*. Academic Press.

[Lavin et al. 1998] LAVIN, Y., BATRA, R., AND HESSELINK, L. 1998. Feature comparisons of vector fields using earth mover's distance. In *Proc. IEEE Visualization '98*, 103–109.

[Lee and Lee 2001] LEE, Y., AND LEE, S. 2001. Geometric snakes for triangular meshes. In *Proc. Eurographics*, 229–238.

[Lee et al. 2002] LEE, H., ALLIEZ, P., AND DESBRUN, M. 2002. Angle-analyzer: A triangle-quad mesh codec. In *Proc. Eurographics*, 383–392.

[Leister 1994] LEISTER, W. 1994. Computer generated copper plates. *Computer Graphics forum 13*, 1, 69–77.

[Lodha and Franke 1999] LODHA, S. K., AND FRANKE, R. 1999. Scattered Data Techniques for Surfaces. In *Proc. Dagstuhl Conf. Scientific Visualization*, H. Hagen, G. Nielson, and F. Post, Eds., 182–222.

[Lodha et al. 2000] LODHA, S., RENTERIA, J., AND ROSKIN, K. 2000. Topology preserving compression of 2D vector fields. In *Proc. IEEE Visualization 2000*, 343–350.

[Lorensen and Cline 1987] LORENSEN, W., AND CLINE, H. 1987. Marching cubes: A high resolution 3D surface construction algorithm. *SIGGRAPH'87 21*, 5, 79–86.

[Lukács and Andor 1998] LUKÁCS, G., AND ANDOR, L. 1998. Computing natural division lines on free-form surfaces based on measured data. In *Mathematical Methods for Curves and Surfaces II*, 319–326.

[Mäkinen 1991] MÄKINEN, E. 1991. A survey in binary tree codings. *The Computurer Journal 34*, 5, 438–443.

[Marschner and Lobb 1994] MARSCHNER, S., AND LOBB, R. 1994. An evaluation of reconstruction filters for volume

rendering. In *IEEE Visualization 1994*, 100–107.

[Martin and Cohen 2001] MARTIN, W., AND COHEN, E. 2001. Representation and extraction of volumetric attributes using trivariate splines: a mathematical framework. In *Solid Modelling and Applications 2001*, 234–240.

[Max 1999] MAX, N. 1999. Weights for computing vertex normals from facet normals. *Jounal of Graphics Tools 4*, 2, 1–6.

[Meissner et al. 2000] MEISSNER, M., HUANG, J., BARTZ, D., MUELLER, K., AND CRAWFIS, R. 2000. A practical comparison of popular volume rendering algorithms. In *Symposium on Volume Visualization and Graphics 2000*, 81–90.

[Meyer et al. 2002] MEYER, M., DESBRUN, M., SCHRÖDER, M., AND BARR, A. H. 2002. Discrete differential-geometry operators for triangulated 2-manifolds. In *Proceedings of VisMath*.

[Milroy et al. 1997] MILROY, M. J., BRADLEY, C., AND VICKERS, G. 1997. Segmentation of a wrap-around model using an active contour. *Computer Aided Design 29*, 299–320.

[Mitchell and Netravali 1988] MITCHELL, D., AND NETRAVALI, A. 1988. Reconstruction filters in computer graphics. *SIGGRAPH'88*, 221–228.

[Moffat et al. 1998] MOFFAT, A., NEAL, R., AND WITTEN, I. 1998. Arithmetic coding revisited. *ACMTOIS: ACM Transactions on (Office) Information Systems 16*.

[Möller et al. 1998] MÖLLER, T., MUELLER, K., KURZION, Y., MACHIRAJU, R., AND YAGEL, R. 1998. Design of accurate and smooth filters for function and derivative reconstruction. In *Symposium on Volume Visualization 1998*, 143–151.

[Mora et al. 2001] MORA, B., JESSEL, J.-P., AND CAUBET, R. 2001. Visualization of isosurfaces with parametric cubes. In *Eurographics 2001*, 377–384.

[Moreton and Séquin 1992] MORETON, H., AND SÉQUIN, C. 1992. Functional optimization for fair surface design. In *Proceedings of the 19th Annual ACM Conference on Computer Graphics and Interactive Techniques*, 167–176.

[Morgan and Scott] MORGAN, J., AND SCOTT, R. The dimension of the space of c^1 piecewise polynomials. In *Unpublished manuscript*.

[Nielson 2000] NIELSON, G. 2000. Volume modelling. In *Volume Graphics*, Springer, M. Chen, A. Kaufman, and R.Yagel, Eds., 29–50.

[Nürnberger and Zeilfelder 2000] NÜRNBERGER, G., AND ZEILFELDER, F. 2000. Developments in bivariate spline interpolation. *J. Comput. Appl. Math. 121*, 125–152.

[Nürnberger and Zeilfelder 2001] NÜRNBERGER, G., AND ZEILFELDER, F. 2001. Local lagrange interpolation on powell-sabin triangulations and terrain modelling. In *Recent Progress in Multivariate Approximation*, W. Haussmann, K. Jetter, and M. Reimer, Eds., vol. 137, 227–244.

[Nürnberger and Zeilfelder 2003] NÜRNBERGER, G., AND ZEILFELDER, F. 2003. Lagrange interpolation by bivariate C^1-splines with optimal approximation order. *Adv. Comp. Math. (to appear)*.

[Nürnberger and Zeilfelder 2004] NÜRNBERGER, G., AND ZEILFELDER, F. 2004. Lagrange interpolation by bivariate C^1-splines with optimal approximation order. *Adv. Comp. Math. (to appear)*.

[Nürnberger et al. 2001] NÜRNBERGER, G., SCHUMAKER, L., AND ZEILFELDER, F. 2001. Local Lagrange Interpolation by Bivariate C^1 Cubic Splines. In *Mathematical Methods In CAGD*, Vanderbilt University Press, 393–404.

[Nürnberger et al. 2003a] NÜRNBERGER, G., SCHUMAKER, L., AND ZEILFELDER, F. 2003. Lagrange interpolation by C^1 cubic splines on triangulated quadrangulations. *Adv. Comp. Math. (to appear)*.

[Nürnberger et al. 2003b] NÜRNBERGER, G., STEIDL, G., AND ZEILFELDER, F. 2003. Explicit estimates for bivariate hierarchical bases. *Comm. Appl. Anal. 7*, 1, 133–151.

[Nürnberger et al. 2004a] NÜRNBERGER, G., RAVESKAYA, V., SCHUMAKER, L., AND ZEILFELDER, F. 2004. Lagrange Interpolation by C^2-Splines of degree 7 on triangulations. In *Proc. Adv. Constr. Approx. (to appear)*.

[Nürnberger et al. 2004b] NÜRNBERGER, G., RAVESKAYA, V., SCHUMAKER, L., AND ZEILFELDER, F. 2004. Local Lagrange interpolation by bivariate splines of arbitrary smoothness. *(submitted)*.

[Nürnberger et al. 2004c] NÜRNBERGER, G., RÖSSL, C., SEIDEL, H.-P., AND ZEILFELDER, F. 2004. Quasi-interpolation by quadratic piecewise polynomials in three variables. *Computer Aided Geometric Design (accepted for publication)*.

[Nürnberger 1989] NÜRNBERGER, G. 1989. *Approximation by Spline Functions*. Springer.

[Ohtake and Belyaev 2004] OHTAKE, Y., AND BELYAEV, A. 2004. Ridge-valley lines on meshes via implicit surface fitting. In *Proc. SIGGRAPH*, 609–612.

[Ohtake et al. 2003] OHTAKE, Y., BELYAEV, A., ALEXA, M., TURK, G., AND SEIDEL, H.-P. 2003. Multi-level partition of unity implicits. In *SIGGRAPH'03*, 463–470.

[Ostromoukhov 1999] OSTROMOUKHOV, V. 1999. Digital facial engraving. In *Proc. SIGGRAPH*, 417–424.

[Page et al. 2001] PAGE, D. L., KOSCHAN, A., SUN, Y., PAIK, J., AND ABIDI, A. 2001. Robust crease detection and curvature estimation of piecewise smooth surfaces from triangle mesh approximations using normal voting. In *Proceedings on Computer Vision and Pattern Recongition.*

[Pajarola and Rossignac 2000] PAJAROLA, R., AND ROSSIGNAC, J. 2000. Compressed progressive meshes. In *IEEE Transactions on Visualization and Computer Graphics*, vol. 6 (1). 79–93.

[Park and Lee 1997] PARK, S., AND LEE, K. 1997. High-dimensional trivariate NURBS representation for analyzing and visualizing fluid flow data. *Computers & Graphics 21*, 4, 473–482.

[Parker et al. 1998] PARKER, S., SHIRLEY, P., LIVNAT, Y., HANSEN, C., AND SLOAN, P.-P. 1998. Interactive ray tracing for isosurface rendering. In *IEEE Visualization 1998*, 233–238.

[Petitjean 2001] PETITJEAN, S. 2001. A survey of methods for recovering quadrics in triangle meshes. *ACM Computing Surveys*, 2.

[Pfeifle and Seidel 1995] PFEIFLE, R., AND SEIDEL, H.-P. 1995. Fitting triangular B-splines to functional scattered data. *Eurographics 1995 14*, 3, 15–23.

[Pfeifle and Seidel 1996] PFEIFLE, R., AND SEIDEL, H.-P. 1996. Scattered data approximation with triangular B-splines. *Advance Course on Fairshape*, 253–263.

[Phong 1975] PHONG, B. T. 1975. Illumination for computer generated pictures. *Communications of ACM 18*, 6, 311–317.

[Pnueli and Bruckstein 1994] PNUELI, AND BRUCKSTEIN, A. M. 1994. **dig**$^{\mathbf{i}}_{\mathrm{d}}$ürer – a digital engraving system. *The Visual Computer 10*, 277–292.

[Post et al. 2002] POST, F., VROLIJK, B., HAUSER, H., LARAMEE, R., AND DOLEISCH, H. 2002. Feature extraction and visualisation of flow fields. In *Proc. Eurographics 2002, State of the Art Reports*, 69–100.

[Powell and Sabin 1977] POWELL, M. J., AND SABIN, M. A. 1977. Piecewise quadratic approximation on triangles. *ACM Trans. Math. Software 4*, 316–325.

[Powell 1974] POWELL, M. J. 1974. Piecewise quadratic surface fitting for contour plotting. In *Software for Numerical Analysis*, Academic Press, D. Evans, Ed., 253–277.

[Praun et al. 2001] PRAUN, E., HOPPE, H., WEBB, M., AND FINKELSTEIN, A. 2001. Real-time hatching. In *Proc. SIGGRAPH*, 579–584.

[Prautzsch et al. 2002] PRAUTZSCH, H., BOEHM, W., AND PALUSZNY, M. 2002. *Bézier and B-Spline Techniques*. Springer.

[Rossignac and Szymczak 1999] ROSSIGNAC, J., AND SZYMCZAK, A. 1999. Wrap&zip decompression of the connectivity of triangle meshes compressed with edgebreaker. *CGTA: Computational Geometry: Theory and Applications 14*.

[Rossignac 1999] ROSSIGNAC, J. 1999. Edgebreaker: Connectivity compression for triangle meshes. *IEEE Transactions on Visualization and Computer Graphics 5*, 2, 47–61.

[Rössl and Kobbelt 1999] RÖSSL, C., AND KOBBELT, L. 1999. Approximation and visualization of discrete curvature on triangulated surfaces. In *Proc. Conference on Vision, Modeling, and Visualization*, 339–346.

[Rössl and Kobbelt 2000] RÖSSL, C., AND KOBBELT, L. 2000. Line-art rendering of 3D-models. In *Proc. Pacific Graphics*, 87–96.

[Rössl et al. 2000a] RÖSSL, C., KOBBELT, L., AND SEIDEL, H.-P. 2000. Extraction of feature lines on triangulated surfaces using morphological operators. In *Smart Graphics (AAAI Spring Symposium-00)*, Technical Report / SS / American Association for Artificial Intelligence, 71–75.

[Rössl et al. 2000b] RÖSSL, C., KOBBELT, L., AND SEIDEL, H.-P. 2000. Line art rendering of triangulated surfaces using discrete lines of curvature. In *Proc. WSCG*, 168–175.

[Rössl et al. 2001] RÖSSL, C., KOBBELT, L., AND SEIDEL, H.-P. 2001. Recovering structural information from triangulated surfaces. In *Mathematical Methods for Curves and Surfaces: Oslo 2000*, Vanderbilt University, Oslo, Norway, T. Lyche and L. L. Schumaker, Eds., 423–432.

[Rössl et al. 2003a] RÖSSL, C., ZEILFELDER, F., NÜRNBERGER, G., AND SEIDEL, H.-P. 2003. Visualization of volume data with quadratic super splines. In *IEEE Visualization 2003*, 393–400.

[Rössl et al. 2003b] RÖSSL, C., IVRISSIMTZIS, I., AND SEIDEL, H.-P. 2003. Tree-based triangle mesh connectivity encoding. In *Curve and Surface Fitting: Saint-Malo 2002*, Nashboro Press, Saint Malo, France, A. Cohen, J.-L. Merrien, and L. L. Schumaker, Eds., 345–354.

[Rössl et al. 2004a] RÖSSL, C., ZEILFELDER, F., NÜRNBERGER, G., AND SEIDEL, H.-P. 2004. Reconstruction of volume data with quadratic super splines. *IEEE Transactions on Visualization and Computer Graphics 10*, 4, 397–409.

[Rössl et al. 2004b] RÖSSL, C., ZEILFELDER, F., NÜRNBERGER, G., AND SEIDEL, H.-P. 2004. Spline approximation of general volumetric data. In *Proc. ACM Symposium on Solid Modeling and Applications*, 71–82.

[Rusinkiewicz 2004] RUSINKIEWICZ, S. 2004. Estimating curvatures and their derivatives on triangle meshes. In *Proc. of Second International Symposium on 3D Data Processing, Visualization, and Transmission (3DPVT)*.

[Sablonnière 1987] SABLONNIÈRE, P. 1987. Error bounds for hermite interpolation by quadratic splines on an α-triangulation. *IMA J. Numer. Anal. 7*, 495–508.

[Sablonnière 2003a] SABLONNIÈRE, P. 2003. Quadratic b-splines on non uniform criss-cross triangulations of bounded rectangular domains in the plane. *IRMAR (Preprint)*.

[Sablonnière 2003b] SABLONNIÈRE, P. 2003. Quadratic spline quasi interpolants on bounded domain of $\mathbb{R}^d$, $d = 1, 2, 3$. *Spline and radial functions (preprint)*.

[Saito and Takahashi 1990] SAITO, T., AND TAKAHASHI, T. 1990. Comprehensible rendering of 3-d shapes. In *Proc.SIGGRAPH*, 197–206.

[Salisbury et al. 1994] SALISBURY, M. P., ANDERSON, S. E., BARZEL, R., AND SALESIN, D. H. 1994. Interactive pen-and-ink illustrations. In *Proc. SIGGRAPH*, 101–108.

[Salisbury et al. 1997] SALISBURY, M. P., WONG, M. T., HUGES, J. F., AND SALESIN, D. H. 1997. Orientable textures for image-based pen-and-ink illustration. In *Proc. SIGGRAPH*, 401–406.

[Sapidis and Besl 1995] SAPIDIS, N. S., AND BESL, P. 1995. Direct construction of polynomial surfaces from dense range images through region growing. *ACM Transactions of Graphics 14*, 2, 171–200.

[Schaback 2000] SCHABACK, R. 2000. Remarks on meshless local construction of surfaces. In *The Mathematics of Surfaces IV*, Springer, 34–58.

[Scheuermann et al. 1998] SCHEUERMANN, G., KRÜGER, H., MENZEL, M., AND ROCKWOOD, A. 1998. Visualizing non-linear vector field topology. *IEEE Transactions on Visualization and Computer Grapics 4*, 2, 109–116.

[Schumaker and Sorokina 2004a] SCHUMAKER, L., AND SOROKINA, T. 2004. Quintic spline interpolation on type-4 tetrahedral partitions. *Adv. Comput. Math. (preprint)*.

[Schumaker and Sorokina 2004b] SCHUMAKER, L., AND SOROKINA, T. 2004. A trivariate box macro element. *(preprint)*.

[Schumaker 1976] SCHUMAKER, L. 1976. Fitting surfaces to scattered data. *Approximation Theory II*, 203–268.

[Schumaker 1984] SCHUMAKER, L. L. 1984. Bounds on the dimension of spaces of multivariate piecewise polynomials. *Rocky Mountain Journal of Mathematics 14*, 251–264.

[Schwarze 1990] SCHWARZE, J. 1990. Cubic and quartic roots. In *Graphics Gems*, A. Glassner, Ed. Academic Press, 404–407.

[Stalling et al. 2003] STALLING, D., WESTERHOFF, M., AND HEGE, H.-C. 2003. Amira — an Object Oriented System for Visual Data Analysis. In *Visualization Handbook*, Academic Press (preprint).

[Storer 1992] STORER, J. A., Ed. 1992. *Image and text compression*, 2nd printing 1995 ed. Kluwer, Dordrecht.

[Strothotte and Schlechtweg 2002] STROTHOTTE, T., AND SCHLECHTWEG, S. 2002. *Non-Photorealistic Computer Graphics: Modeling, Rendering, and Animation*. Morgan Kaufmann.

[Szymczak et al. 2001] SZYMCZAK, A., KING, D., AND ROSSIGNAC, J. 2001. An edgebreaker-based efficient compression scheme for regular meshes. *CGTA:*

Computational Geometry: Theory and Applications 20.

[Taubin and Rossignac 1998] TAUBIN, G., AND ROSSIGNAC, J. 1998. Geometric compression through topological surgery. *ACM Transactions on Graphics 17*, 2, 84–115.

[Taubin et al. 1998] TAUBIN, G., GUEZIEC, A., HORN, W., AND LAZARUS, F. 1998. Progressive forest split compression. In *Proc. SIGGRAPH*, 123–132.

[Taubin 1995] TAUBIN, G. 1995. Estimating the tensor of curvature of a surface from a polyhedral approximation. In *Proceedings International Conference on Computer Vision*, 902–907.

[Telea and van Wijk 1999] TELEA, A., AND VAN WIJK, J. 1999. Simplified representation of vector fields. In *Proc. IEEE Visualization '99*, D. Ebert, M. Gross, and B. Hamann, Eds., 35–42.

[Theisel and Seidel 2003] THEISEL, H., AND SEIDEL, H.-P. 2003. Feature flow fields. In *Data Visualization 2003. Proc. VisSym 03*, 141–148.

[Theisel and Weinkauf 2002] THEISEL, H., AND WEINKAUF, T. 2002. Vector field metrics based on distance measures of first order critical points. In *Journal of WSCG, Short Communication*, vol. 10, 121–128.

[Theisel et al. 2003a] THEISEL, H., RÖSSL, C., AND SEIDEL, H. 2003. Compression of 2D 'vector fields under guaranteed topology preservation. In *Proc. Eurographics*, 333–342.

[Theisel et al. 2003b] THEISEL, H., RÖSSL, C., AND SEIDEL, H.-P. 2003. Combining topological simplification and topology preserving compression for 2d vector fields. In *Proc. Pacific Graphics*, 419–423.

[Theisel et al. 2003c] THEISEL, H., RÖSSL, C., AND SEIDEL, H.-P. 2003. Using feature flow fields for topological

comparison of vector fields. In *Proc. Vision, Modeling and Visualization*, 521–528.

[Theisel et al. 2004a] THEISEL, H., RÖSSL, C., AND SEIDEL, H.-P. 2004. Topology preserving thinning of vector fields on triangular meshes. In *Advances in Multiresolution for Geometric Modelling*, N. A. Dodgson, M. S. Floater, and M. A. Sabin, Eds. Springer, Heidelberg, 353–366.

[Theisel et al. 2004b] THEISEL, H., RÖSSL, C., ZAYER, R., AND SEIDEL, H.-P. 2004. Normal based estimation of the curvature tensor for triangular meshes. In *Proc. Pacific Graphics*.

[Theisel 2002] THEISEL, H. 2002. Designing 2D vector fields of arbitrary topology. *Computer Graphics Forum (Eurographics 2002) 21*, 3, 595–604.

[Theußl et al. 2003] THEUSSL, T., MÖLLER, T., HLADUVKA, J., AND GRÖLLER, M. 2003. Reconstruction issues in volume visualization. In *Data Visualization: The State of the Art*, F. Post, B. Hamann, and G.-P. Bonneau, Eds., 109–126.

[Thévenaz and Unser 2001] THÉVENAZ, P., AND UNSER, M. 2001. High-quality isosurface rendering with exact gradients. In *ICIP'01*, 854–857.

[Touma and Gotsman 1998] TOUMA, C., AND GOTSMAN, C. 1998. Triangle mesh compression. In *Proce. Graphics Interface*, 26–34.

[Tricoche et al. 2000] TRICOCHE, X., SCHEUERMANN, G., AND HAGEN, H. 2000. A topology simplification method for 2D vector fields. In *Proc. IEEE Visualization 2000*, 359–366.

[Tricoche et al. 2001] TRICOCHE, X., SCHEUERMANN, G., AND HAGEN, H. 2001. Continuous topology simplification of planar vector fields. In *Proc. Visualization 01*, 159 – 166.

[Trotts et al. 2000] TROTTS, I., KENWRIGHT, D., AND HAIMES, R. 2000. Critical points at infinity: a missing link in vector field topology. In *Proc. NSF/DoE Lake Tahoe Workshop on Hierarchical Approximation and Geometrical Methods for Scientific Visualization*.

[Turk and O'Brien 2002] TURK, G., AND O'BRIEN, J. 2002. Modeling with implicit surfaces that interpolate. *ACM Transactions on Graphics 21*, 4, 855–873.

[Tutte 1962] TUTTE, W. 1962. A census of planar triangulations. *Can. J. Math. 14*, 21–38.

[Várady and Benkő 2000] VÁRADY, T., AND BENKŐ, P. 2000. Reverse engineering b-rep models from multiple point clouds. In *Geometric Modeling and Processing*, 3–12.

[Várady et al. 1997] VÁRADY, T., MARTIN, R., AND COX, J. 1997. Reverse engineering of geometric models — an introduction. *Computer Aided Design 29*, 255–268.

[Vlachos et al. 2001] VLACHOS, A., PETERS, J., BOYD, C., AND MITCHELL, J. L. 2001. Curved PN triangles. In *Proceedings of the Symposium on Interactive 3D graphics*, 159–166.

[Vorsatz et al. 2001] VORSATZ, J., RÖSSL, C., KOBBELT, L., AND SEIDEL, H.-P. 2001. Feature sensitive remeshing. In *Proc. Eurographics*, 393–401.

[Vorsatz et al. 2003a] VORSATZ, J., RÖSSL, C., AND SEIDEL, H.-P. 2003. Dynamic remeshing and applications. In *Proc. ACM Symposium on Solid Modeling and Applications*, 167–175.

[Vorsatz et al. 2003b] VORSATZ, J., RÖSSL, C., AND SEIDEL, H.-P. 2003. Dynamic remeshing and applications. *Journal of Computing and Information Science in Engineering 3*, 4, 338–344.

[Wald and Slusallek 2001] WALD, I., AND SLUSALLEK, P. 2001. State of the art in interactive ray tracing. In *STAR, EUROGRAPHICS 2001*, 21–42.

[Wald 2004] WALD, I. 2004. *Realtime Ray Tracing and Interactive Global Illumination*. PhD thesis, Computer Graphics Group, Saarland University. Available at http://www.mpi-sb.mpg.de/~wald/PhD/.

[Watanabe and Belyaev 2001] WATANABE, K., AND BELYAEV, A. 2001. Detection of salient curvature features on polygonal surfaces. In *Proc. Eurographics*, 385–392.

[Welch and Witkin 1994] WELCH, W., AND WITKIN, A. 1994. Free–Form shape design using triangulated surfaces. In *Proceedings of SIGGRAPH '94*, A. Glassner, Ed., 247–256.

[Westermann et al. 2001] WESTERMANN, R., JOHNSON, C., AND ERTL, T. 2001. Topology-preserving smoothing of vector fields. *IEEE Transactions on Visualization and Computer Graphics 7*, 3, 222–229.

[Winkenbach and Salesin 1994] WINKENBACH, G., AND SALESIN, D. H. 1994. Computer-generated pen-and-ink illustration. In *Proc. SIGGRAPH*, 91–98.

[Winkenbach and Salesin 1996] WINKENBACH, G., AND SALESIN, D. H. 1996. Rendering parametric surfaces in pen-and-ink. In *Proc. SIGGRAPH*, 469–476.

[Wischgoll and Scheuermann 2001] WISCHGOLL, T., AND SCHEUERMANN, G. 2001. Detection and visualization of closed streamlines in planar flows. *IEEE Transactions on Visualization and Computer Graphics 7*, 2, 165–172.

[Woo et al. 1997] WOO, M., NEIDER, J., DAVIS, T., AND SHREINER, D. 1997. *OpenGL Programming Guide. Third Edition.* Addison-Wesley.

[Worsey and Farin 1987] WORSEY, A., AND FARIN, G. 1987. An n-dimensional clough-tocher interpolant. *Const. Approx. 3 2*, 99–110.

[Zeilfelder 2002] ZEILFELDER, F. 2002. Scattered data fitting with bivariate splines. In *Tutorials on Multiresolution and Geometric Modelling*, Springer, 243–286.

[Zorin and Hertzmann 2000] ZORIN, D., AND HERTZMANN, A. 2000. Illustrating smooth surfaces. In *Proc. SIGGRAPH*, 517–526.

Curriculum Vitae – Lebenslauf

Curriculum Vitae

1974	born in Auerbach in der Oberpfalz, Germany
1980-1984	Grundschule Auerbach
1984-1993	Gymnasium Pegnitz
1993-1994	Military service in Oberviechtach and Amberg
1994-1999	Study of Computer Science, University of Erlangen-Nuremberg, Germany
1999	Diploma (Diplom-Informatiker, Dipl.-Inf.)
1999-2005	Ph.D. Student at the Max-Planck-Institut für Informatik, Saarbrücken, Germany

Lebenslauf

1974	geboren in Auerbach in der Oberpfalz
1980-1984	Grundschule Auerbach
1984-1993	Gymnasium Pegnitz
1993-1994	Grundwehrdienst in Oberviechtach and Amberg
1994-1999	Informatikstudium an der Universität Erlangen-Nürnberg
1999	Abschluss als Diplom-Informatiker (Dipl.-Inf.)
1999-2005	Promotion am Max-Planck-Institut für Informatik, Saarbrücken