Skeleton-Based Visualization of Massive Voxel Objects with Network-Like Architecture

Steffen Prohaska
geb. am 22. September 1972
in Pirmasens

Potsdam, den 16. April 2007

Dissertation
zur Erlangung des akademischen Grades
„doctor rerum naturalium“
(Dr. rer. nat.)
in der Wissenschaftsdisziplin Informatik

eingereicht an der
Mathematisch-Naturwissenschaftlichen Fakultät
der Universität Potsdam

Elektronisch veröffentlicht auf dem
Publikationsserver der Universität Potsdam:
http://opus.kobv.de/ubp/volltexte/2007/1488/
urn:nbn:de:kobv:517-opus-14888
[http://nbn-resolving.de/urn:nbn:de:kobv:517-opus-14888]

Abstract

This work introduces novel internal and external memory algorithms for computing voxel skeletons of massive voxel objects with complex network-like architecture and for converting these voxel skeletons to piecewise linear geometry, that is triangle meshes and piecewise straight lines. The presented techniques help to tackle the challenge of visualizing and analyzing 3d images of increasing size and complexity, which are becoming more and more important in, for example, biological and medical research.

Section 2.3.1 contributes to the theoretical foundations of thinning algorithms with a discussion of homotopic thinning in the grid cell model. The grid cell model explicitly represents a cell complex built of faces, edges, and vertices shared between voxels. A characterization of pairs of cells to be deleted is much simpler than characterizations of simple voxels were before. The grid cell model resolves topologically unclear voxel configurations at junctions and locked voxel configurations causing, for example, interior voxels in sets of non-simple voxels. A general conclusion is that the grid cell model is superior to indecomposable voxels for algorithms that need detailed control of topology.

Section 2.3.2 introduces a noise-insensitive measure based on the geodesic distance along the boundary to compute two-dimensional skeletons. The measure is able to retain thin object structures if they are geometrically important while ignoring noise on the object's boundary. This combination of properties is not known of other measures. The measure is also used to guide erosion in a thinning process from the boundary towards lines centered within plate-like structures. Geodesic distance based quantities seem to be well suited to robustly identify one- and two-dimensional skeletons. Chapter 6 applies the method to visualization of bone micro-architecture.

Chapter 3 describes a novel geometry generation scheme for representing voxel skeletons, which retracts voxel skeletons to piecewise linear geometry per dual cube. The generated triangle meshes and graphs provide a link to geometry processing and efficient rendering of voxel skeletons. The scheme creates non-closed surfaces with boundaries, which contain fewer triangles than a representation of voxel skeletons using closed surfaces like small cubes or iso-surfaces. A conclusion is that thinking specifically about voxel skeleton configurations instead of generic voxel configurations helps to deal with the topological implications. The geometry generation is one foundation of the applications presented in Chapter 6.

Chapter 5 presents a novel external memory algorithm for distance ordered homotopic thinning. The presented method extends known algorithms for computing chamfer distance transformations and thinning to execute I/O-efficiently when input is larger than the available main memory. The applied block-wise decomposition schemes are quite simple. Yet it was necessary to carefully analyze effects of block boundaries to devise globally correct external memory variants of known algorithms. In general, doing so is superior to naive block-wise processing ignoring boundary effects. Chapter 6 applies the algorithms in a novel method based on confocal microscopy for quantitative study of micro-vascular networks in the field of microcirculation.

Zusammenfassung

Die vorliegende Arbeit führt I/O-effiziente Algorithmen und Standard-Algorithmen zur Berechnung von Voxel-Skeletten aus großen Voxel-Objekten mit komplexer, netzwerkartiger Struktur und zur Umwandlung solcher Voxel-Skelette in stückweise-lineare Geometrie ein. Die vorgestellten Techniken werden zur Visualisierung und Analyse komplexer drei-dimensionaler Bilddaten, beispielsweise aus Biologie und Medizin, eingesetzt.

Abschnitt 2.3.1 leistet mit der Diskussion von topologischem Thinning im Grid-Cell-Modell einen Beitrag zu den theoretischen Grundlagen von Thinning-Algorithmen. Im Grid-Cell-Modell wird ein Voxel-Objekt als Zellkomplex dargestellt, der aus den Ecken, Kanten, Flächen und den eingeschlossenen Volumina der Voxel gebildet wird. Topologisch unklare Situationen an Verzweigungen und blockierte Voxel-Kombinationen werden aufgelöst. Die Charakterisierung von Zellpaaren, die im Thinning-Prozess entfernt werden dürfen, ist einfacher als bekannte Charakterisierungen von so genannten »Simple Voxels«. Eine wesentliche Schlussfolgerung ist, dass das Grid-Cell-Modell atomaren Voxeln überlegen ist, wenn Algorithmen detaillierte Kontrolle über Topologie benötigen.

Abschnitt 2.3.2 präsentiert ein rauschunempfindliches Maß, das den geodätischen Abstand entlang der Oberfläche verwendet, um zweidimensionale Skelette zu berechnen, welche dünne, aber geometrisch bedeutsame, Strukturen des Objekts rauschunempfindlich abbilden. Das Maß wird im weiteren mit Thinning kombiniert, um die Erosion von Voxeln auf Linien zuzusteuern, die zentriert in plattenförmigen Strukturen liegen. Maße, die auf dem geodätischen Abstand aufbauen, scheinen sehr geeignet zu sein, um ein- und zwei-dimensionale Skelette bei vorhandenem Rauschen zu identifizieren. Eine theoretische Begründung für diese Beobachtung steht noch aus. In Abschnitt 6 werden die diskutierten Methoden zur Visualisierung von Knochenfeinstruktur eingesetzt.

Abschnitt 3 beschreibt eine Methode, um Voxel-Skelette durch kontrollierte Retraktion in eine stückweise-lineare geometrische Darstellung umzuwandeln, die als Eingabe für Geometrieverarbeitung und effizientes Rendering von Voxel-Skeletten dient. Es zeigt sich, dass eine detaillierte Betrachtung der topologischen Eigenschaften eines Voxel-Skeletts einer Betrachtung von allgemeinen Voxel-Konfigurationen für die Umwandlung zu einer geometrischen Darstellung überlegen ist. Die diskutierte Methode bildet die Grundlage für die Anwendungen, die in Abschnitt 6 diskutiert werden.

Abschnitt 5 führt einen I/O-effizienten Algorithmus für Thinning ein. Die vorgestellte Methode erweitert bekannte Algorithmen zur Berechung von Chamfer-Distanztransformationen und Thinning so, dass diese effizient ausführbar sind, wenn die Eingabedaten den verfügbaren Hauptspeicher übersteigen. Der Einfluss der Blockgrenzen auf die Algorithmen wurde analysiert, um global korrekte Ergebnisse sicherzustellen. Eine detaillierte Analyse ist einer naiven Zerlegung, die die Einflüsse von Blockgrenzen vernachlässigt, überlegen. In Abschnitt 6 wird, aufbauend auf den I/O-effizienten Algorithmen, ein Verfahren zur quantitativen Analyse von Mikrogefäßnetzwerken diskutiert.

Contents

List of Figures

List of Tables

List of Algorithms

Acknowledgements

I thank all members of the Department of Visualization and Data Analysis at the Zuse Institute Berlin (ZIB) and people working on the development of Amira at Mercury Computer Systems, Inc. for their support and valuable discussions. I also like to thank all collaborators in the MAP-Project AO-99-030, which was funded by the European Space Agency under ESA Contract 14592, and all partners of the project »MV3D, Analysis of vascular networks«, which was funded by a »Fond de la Recherche Technologique« grant of the French government. Finally, I wish to thank the members of my committee for their support and their encouragement.

1 Introduction

Three-dimensional imaging devices are becoming a workhorse in biological and medical research as well as clinical practice. Computed tomography (CT) and magnetic resonance imaging (MRI) are used on a daily basis in diagnostics. High-resolution variants, like micro-CT (Rüegsegger et al., 1996) and high-resolution MRI (Majumdar et al., 1998), become abundant in basic research. Confocal microscopy extends traditional two-dimensional histomorphometry to the third dimension.

On an abstract level all imaging techniques can be described as sampling some three-dimensional physical signal, $O : \mathbb{R}^3 \rightarrow \mathbb{R}$, at regularly spaced points, resulting in a three-dimensional image. This thesis will only consider scalar signals, although vector or tensor data may also be acquired, for example in MRI, as well as multi-variate data joining multiple signals per sample point.

The image signal is the result of some underlying physical process and its spatial distribution typically represents objects contained in the volume under examination. Examples from medical diagnostics are bone, measured by a CT-scanner; brain tissue, measured by an MRI-scanner; examples from basic research are bone micro-architecture, measured by micro-CT; and micro-vascular networks, measured by confocal microscopy. The latter two examples are illustrated in Figure 1.3 and Figure 1.4. Both examples share the property that the object's architecture forms a complex network, built of either tube-like parts only, in the case of vessels, or rods and plates, in the case of bone.

The objective of this thesis is to visualize and analyze objects with such complex, network-like architecture using a skeleton representation as one part of the processing pipeline as illustrated in Figure 1.1. In a first step, the original image is preprocessed and segmented. The resulting voxel object is input to a skeletonization step, which computes a voxel skeleton that is converted to geometry. After geometry post-processing, rendering yields the image presented to the user. Figure 1.4 displays an example: on the very right, it depicts a rendering of the center lines extracted from the image data in the center. The complex graph formed by the vessel network is explicitly available and can be utilized for further analysis like high-lighting vein and artery trees. Chapter 6 presents more details.

Visualization's overall goal is to gain knowledge about data by a process including user feedback to control a computer which is generating a visual representation of the data (Brodlie et al. (2004), van Wijk (2005), Johnson et al. (2006)). Fig 1.2 illustrates the idea. Following Brodlie et al. (2004), the process can be split in three semantic contexts illustrated in the figure from bottom to top: »Making displayable by a computer« captures the technical process transforming input data into images. The visualization pipeline is a conceptual description of this transformation: A data source, in our case the imaging device, provides raw data. A subset of the data is selected in a filtering step and mapped to a geometric representation, which is rendered resulting in a two-dimensional

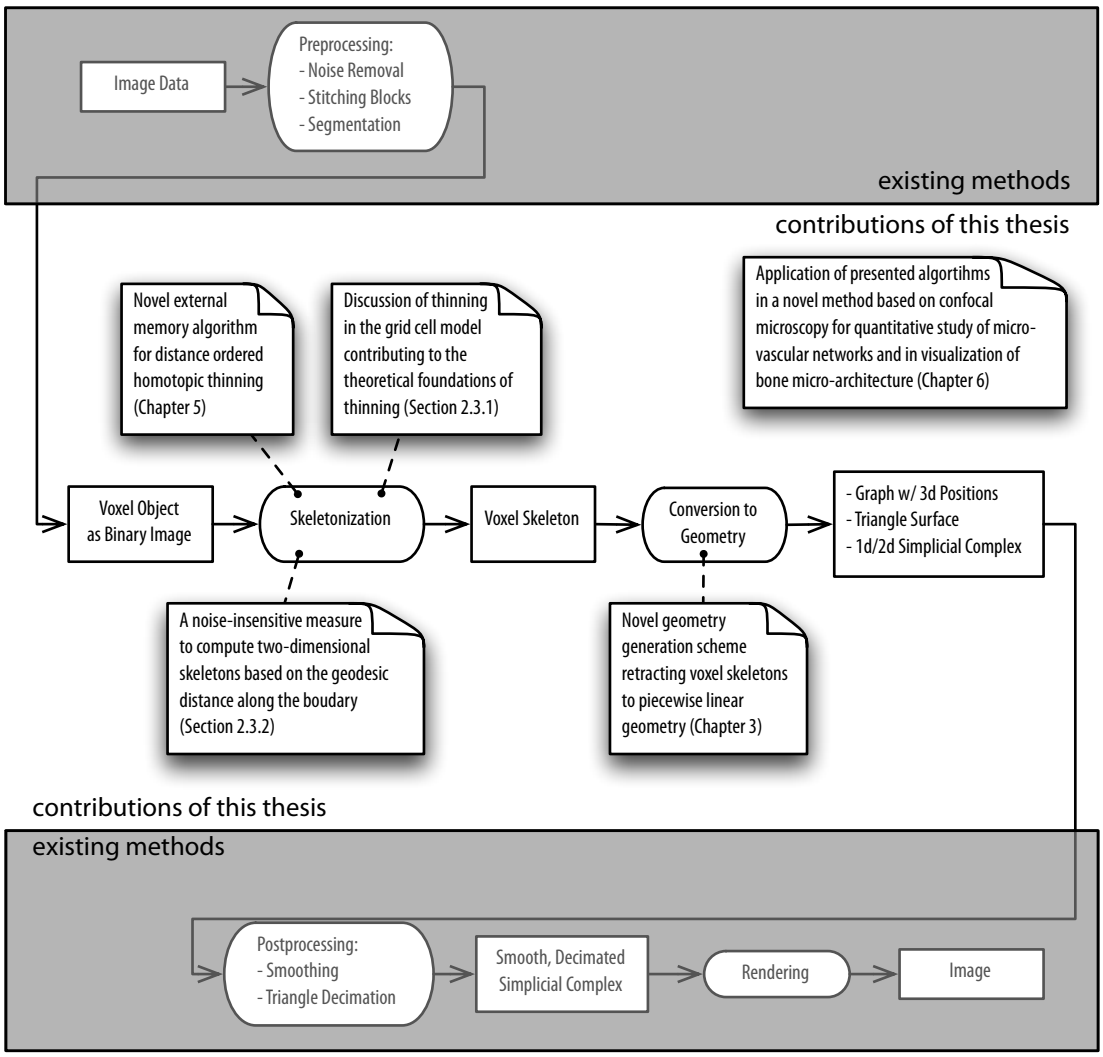


Figure 1.1: Processing steps from image data to skeletons to rendered images. Preprocessing of input data, including segmentation, yields a voxel object to which skeletonization is applied. The resulting voxel skeleton is converted to a piecewise linear geometric representation and used, after postprocessing, for rendering. Main contributions of this thesis are depicted as annotations. All other steps rely on existing methods.

image presented to the user. The stages of the pipeline are controlled by specifications (parameters). The objective is to generate a real image (»making visible to one's eye«) that effectively enables the human viewer to build a mental image of the data (»making visible to one's mind«). Based on the mental image, she would be able to understand the data, draw conclusions, and make decisions; that is to gain knowledge. Knowledge is also the basis for further exploration of the data by modifying the specification that guide the image generation process. Efficient utilization of available resources, such as main processor (CPU), main memory (RAM), and specialized graphics processors (GPUs) with specialized video memory (VRAM), helps to generate images at interactive frame rates.

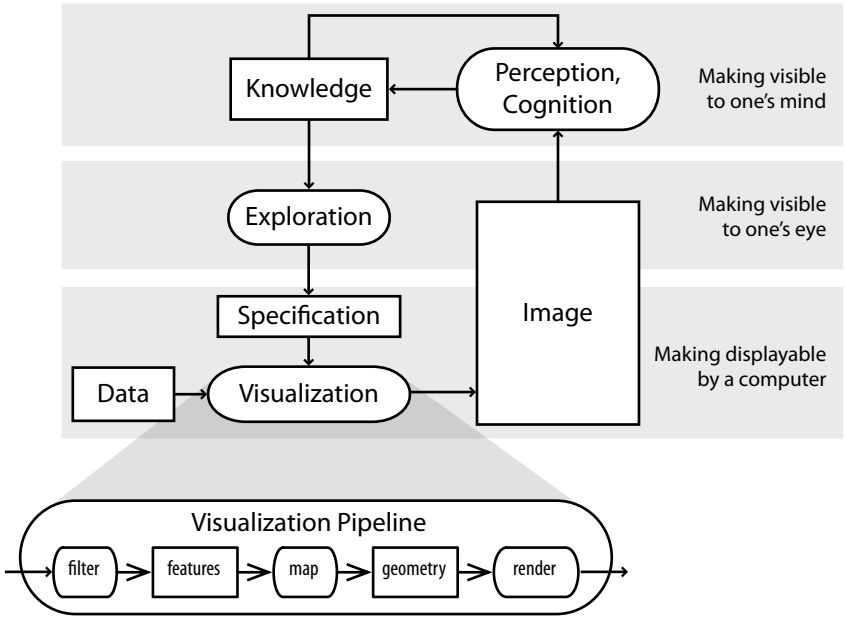


Figure 1.2: The process of visualization in three semantic contexts. Visualization includes a feedback loop of creating images from the data, looking at these images, gaining knowledge, and exploring the data further by adjusting the specifications. The visualization pipeline depicts the detailed steps required for making data displayable by a computer. The illustration is adapted from van Wijk (2005), Johnson et al. (2006), and Brodlie et al. (2004).

The skeletonization pipeline introduced in the previous paragraph and discussed in more detail below is one instance of the generic concept described in this paragraph.

Techniques from image processing, computer vision, computer graphics, and user interface studies are used to achieve the overall goal of visualization and started to merge to a scientific field of its own. Hundreds of articles have been published on visualization methods during the last twenty years, after the seminal NSF Landmark Report »Visualization in Scientific Computing« (McCormick et al., 1987) was published. A comprehensive overview of the current state can be found in Hansen and Johnson (2005). Research challenges are also discussed in Johnson et al. (2006).

Visualization algorithms can be classified by the data type of the input (Haber and McNabb, 1990) or similarly by the underlying data model used during computations, as proposed by Tory and Möller (2004). Information visualization of abstract data is often discussed separately from scientific visualization of physical data (see for example Card et al., 1999).

Image data falls in the category of scalar data over a three-dimensional domain. Thus, direct scalar field visualization techniques can be applied. Two of the most thoroughly researched techniques are direct volume rendering and iso-contouring/iso-surfacing. Figure 1.3 displays examples: on the left and top a volume-rendering of human vertebral bone and on the right an iso-surface rendering of a small sub-volume.

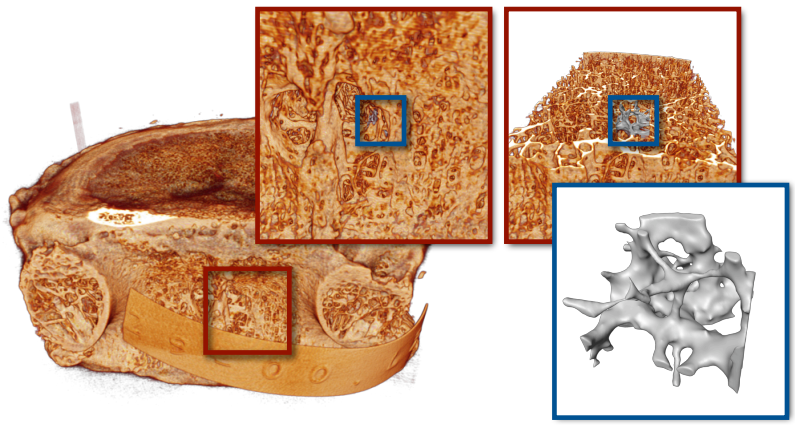


Figure 1.3: Scalar field visualizations of a micro-CT scan of a human vertebrae. Volume rendering was used to generate the overview of the full volume (left and top). Iso-surface rendering was used to create the close-up view on the trabecular bone structure at the bottom right.

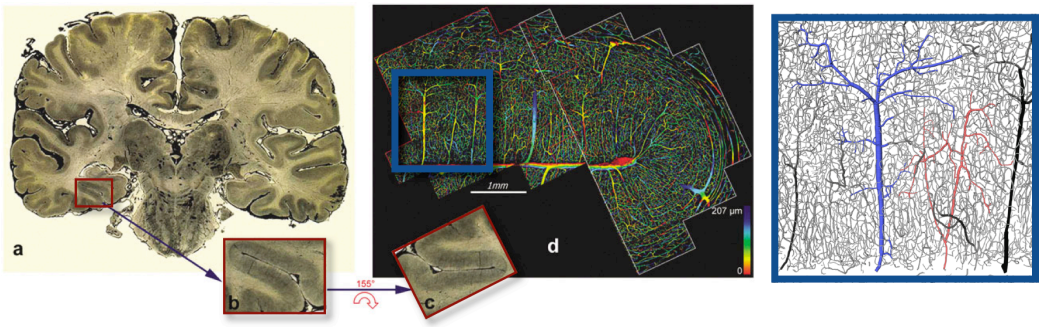


Figure 1.4: Visualizations of a vascular network using skeletonization. A photograph of a slice of human brain tissue is displayed on the left. In the center, the confocal microscopy image data acquired from a small area of brain tissue is depicted as a maximum intensity projection with color-coded depth value. On the right, center lines computed using skeletonization are rendered for a sub-volume corresponding to the area of the blue box. A vein tree is highlighted in blue, an artery tree in red.

The ever increasing size and complexity of input data poses challenges to visualization. The rapidly increasing resolution of imaging devices illustrates this fact: CT-scanners regularly acquire data at 100µ resolution in data matrix sizes of 1024^3; commercially available research scanner at $10\mu/2048^3$; and synchrotron CT-imaging reach 1µ in matrix sizes of up to 4096^3. Confocal microscopes achieve $0.5\mu/512^3$ in one scan, and motorized stages allow to collect hundreds of such images, which can be automatically stitched to a single, large volume. Figure 1.3 and Figure 1.4 depict structures contained in such acquired images.

Two main challenges of massive data are 1) to deal with the sheer size of the data and 2) to clearly display the complex structure hidden in the data. The challenges of dealing with

the data size may be tackled by hierarchical methods, compression, and external memory algorithms and data structures (Silva et al., 2002).

This thesis explores the approach to deal with massive data by extracting higher level information in a pre-processing step and subsequently display data on a more abstract level. In addition to dealing with the size of the data, this approach bears the opportunity of generating expressive and effective visualizations of complex structures. Data is analyzed in a feature extraction step that computes such high level structural information. It is typically run as an offline batch computation prior to interactive visualization. Extending algorithms to massive data may be easier achievable in a batch computation than in an interactive scenario with on-line computations. The results of the batch computations contain the essential information about the input data in compressed form.

We will focus on voxel objects with a network-like architecture. The network may be built of tubular structures, as illustrated in Figure 1.4, or it may consist of plate-like and tube-like parts, as illustrated in Figure 1.3. We will limit our investigations to voxel objects resulting from image segmentation, that is classifying an image into object and background.

The central question discussed throughout the thesis is »How to visualize (and analyze) massive voxel objects with complex, network-like architecture?«. A supported claim is that skeletonization allows to generate a high level representation, the skeleton, requiring only a reduced amount of data, which can be used to create effective visualizations.

Following the idea of skeletonization, the central question splits into »How to efficiently compute skeletons of massive voxel objects?« and »How to render those skeletons?«. These questions will be discussed for specific data types—voxels and piecewise linear geometry—and specific architectures—rods or plates—of the input object. Restricting the discussion to specific cases is justified by the claims that 1) knowing the architecture of the object simplifies skeletonization methods; 2) voxels are an efficient representation for computing and storing skeletons; and 3) a piecewise linear representation, that is straight line segments and triangles, is efficient for rendering.

The following list presents the specific questions that will be answered throughout the text:

- How to compute one-dimensional (curve) voxel skeletons of massive voxel objects?
- How to represent one-dimensional and two-dimensional structures and junction by voxels?
- How to compute two-dimensional voxel skeletons (›central surfaces‹) that are robust to noise in the input?
- How to compute line skeletons of plate-like structures?
- How to compute mixed one-/two-dimensional skeletons that guarantee certain topological properties starting from mixed one-/two-dimensional structures?
- How to render voxel skeletons using piecewise linear geometry?

- How to visualize huge and complex micro-vascular networks?
- How to visualize huge and complex networks of interconnected bone rods and plates?

Figure 1.1 illustrates the processing steps and the contributions made by this thesis to answer these questions. This text presents algorithms to extract skeletons, that is center lines and central surfaces, of objects represented in the input data. Domain specific knowledge is incorporated to decide if one-dimensional structures, like for example blood vessels, or two-dimensional structures, like for example trabecular bone plates, are searched for. The results are skeletons of the original object, concisely representing its structure. Algorithms are extended to external memory allowing to process data of virtually any size. The remainder of the introduction briefly discusses the data types and processing steps; and gives and overview over the following chapters.

Original input data is given as samples comprising a three-dimensional image, which is formally described as a function $I : \mathbb{Z}^3 \to \mathbb{R}$ assigning a real value to each sample point. In real world applications the integer positions would need to be scaled and translated to match physical units. Samples may be interpreted as weights for some interpolation scheme, which would reconstruct a scalar function on whole $\mathbb{R}^3$. We will not follow this interpretation—except for some preprocessing—but instead assume that samples can be segmented in either belonging to the foreground voxel object $P \subset \mathbb{Z}^3$ or belonging to the background $\bar{P} = \mathbb{Z}^3 \setminus P$ by some classification procedure, for example filtering and thresholding.

A classification of each voxel as either foreground or background can also be expressed as a characteristic function $I : \mathbb{Z}^3 \to \{0, 1\}$, which is called a binary image. The representation of objects as binary images not only naturally emerges from the data acquired by imaging devices, but is also efficient in regards of computation time and space. Binary images can be stored as bits in a regular array. All the neighborhood information is implicitly given by the structure of $\mathbb{Z}^3$. Voxel objects resulting from the segmentation and voxel skeletons resulting from skeletonization are two examples of objects stored as binary images.

Later in the processing pipeline, voxel skeletons are converted to piecewise linear geometry represented as a simplicial complex $C = (V, E, T)$. $V = \{v_i \in \mathbb{R}^3 \mid i = 1 \dots n\}$ contains the vertices of the geometry. $E = \{e_{ij} = (v_i, v_j)\} \subset V^2$ describes edges connecting two vertices each. $T = \{t_{ijk} = (v_i, v_j, v_k)\} \subset V^3$ holds the triangles formed by three vertices each. The edges of each triangle must also be contained in E, while E may contain edges that are not incident with a triangle. Examples of simplicial complexes are triangle meshes and line sets with piecewise straight line segments.

In a first step, Chapter 2 investigates topological properties of binary images and presents algorithms to compute voxel skeletons. One well known technique is thinning, which is based on the notion of a simple voxel, that is a voxel that can be removed from a voxel object without changing its topology. Various characterizations of a simple voxel have been given in the past. They all demand some non-trivial computations in a voxel's neighborhood.

Section 2.3 contributes to the theoretical foundations of thinning with a discussion of thinning in the grid cell model. The grid cell model explicitly represents a cell complex built of faces, edges, and vertices shared between voxels. A characterization of deletable pairs of cells is much simpler than characterizations of simple voxels were before. A general conclusion is that the grid cell model is superior to the grid point model for algorithms that need detailed control of topology. Many other known algorithms, but not all, may benefit from the grid point model, too.

Skeletonization algorithms that retain geometric features of the input object are often susceptible to noise. They typically compute a subset of the medial axis. It is well known that the medial axis is unstable under perturbations of the object's boundary. For example small changes on the object surface may cause large side branches in the medial axis.

Section 2.3.2 introduces a measure for computing skeletons based on the geodesic distance along the boundary of a voxel object, which is robust to noise. Two-dimensional voxel skeletons representing plate-like structures are extracted by a simple threshold on this measure. The measure is also used to guide erosion in a thinning process from the boundary towards lines centered within plate-like structures. This enables us to compute one-dimensional skeletons of non-tubular structures. The proposed measure was experimentally observed to be well suited to define one-dimensional skeletons directly from objects containing plate-like structures. Theoretical investigations of the observed properties should be conducted in the future.

Chapter 3 discusses a novel method for converting voxel skeletons to piecewise linear geometry, that is triangle meshes and line sets. Such representations can be rendered efficiently by today's rendering pipelines, which are based upon triangles as rendering primitives and typically implemented in the hardware of a graphics processing unit (GPU). The construction of geometry is defined locally in the neighborhood of a voxel and retracts sub-voxels to a piecewise linear geometric representation. The method creates non-closed surfaces with boundaries, which contain less triangles than a representation of voxel skeletons by closed surfaces like small cubes or iso-surfaces. A look-up table based on a $2 \times 2 \times 2$ voxel configuration suffices to provide pre-computed geometry, which is stitched to form a global geometry. A conclusion drawn from the presentation is that thinking specifically about voxel skeleton configurations instead of generic voxel configurations helps to deal with the topological implications.

Chapter 4 presents synthetic examples processed by the algorithms discussed in Chapter 2 and Chapter 3.

Applying the presented algorithms to data exceeding the size of main memory is tackled in Chapter 5, which presents newly developed external memory variants for the chamfer distance transformation, thinning, the computation of the geodesic distance based measure, and geometry generation. Data of any size may be processed as long as input and intermediate results can be stored on disk and the resulting geometric representation fits into main memory, whereas the original algorithms required all data to reside in main memory for efficient processing. The applied block-wise decomposition schemes are quite simple. Yet it was necessary to carefully analyze effects of block boundaries to devise globally correct external memory variants of known algorithms. In general, doing so is superior to naive block-wise processing ignoring boundary effects.

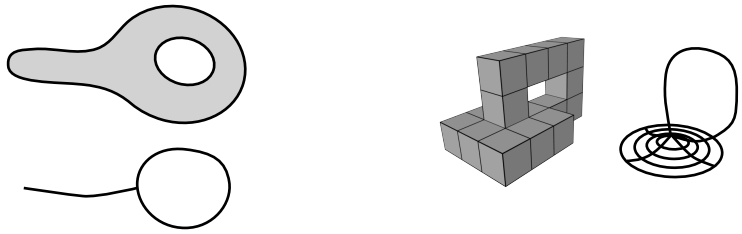
Chapter 6 presents two applications of the methods to bio-medical image data. The first example introduces a novel method based on confocal microscopy for quantitative study of micro-vascular networks. One-dimensional skeletons are computed and visualized as line sets or serve as a basis for further quantitative analysis. The second example analyzes and visualizes bone micro-architecture containing rod- and plate-like parts. Skeletons provide a way to highlight various aspects of the bone structure. They also proved a useful tool in detecting alterations in the bone micro-architecture, for example after compression testing.

Finally, Chapter 7 gives a summary of results, conclusions, and ideas for future work.

2 Voxel Skeletons

The intuitive idea of skeletonization is to shrink an object to a lower dimensional representation which still captures major features of the original object. Skeletons can be used to visualize important aspects of an object. Compared to the original input data, skeletons require less storage and thus may be more suitable for interactive rendering:

Figure 2.1: Schematic objects and their skeletons.



The following properties should be fulfilled by a skeleton:

- homotopy: the skeleton should be homotopic to the original object;
- thinness: the skeleton should have a lower dimension than the original object. A skeleton of a three-dimensional object would comprise one and two-dimensional parts;
- medialness: the skeleton should be located at the center of the object;
- reconstructability: the original object should be reconstructable from the skeleton. To do so, additional information, for example local thickness, is required at every point of the skeleton.

Not all of the above properties are fulfilled simultaneously by every skeleton. For instance, strict homotopy makes a skeleton highly sensitive to noise on the object's boundary. Loosening this requirement can make a skeleton more robust to noise.

Other type of shape representations exist besides skeletons. The introduction already mentioned voxel objects as another example for a shape representation. Before discussing how to compute skeletons of binary images, the next section will briefly review more ways of representing shapes and computing skeletons.

2.1 Shape Representations and Shape Analysis

Important classes of shape representations are volumetric representations, boundary representations, and hierarchical representations describing a shape built of its parts.

In this thesis, initially, shapes are always provided in a volumetric representation, as voxel objects. Level sets, which define the boundary of an object implicitly by a level set of a scalar function, are another variant of volumetric representations. Objects can be converted between volumetric representations and boundary representations. For example, the Marching Cubes algorithm (Lorensen and Cline, 1987) allows to convert a volumetric representation to a triangulated surface; voxelization (Kaufman et al., 1993) to reverse the process. Boundaries can also be directly modeled using splines, for example NURBS; constructive solid geometry (CSG); or specified either implicitly or explicitly by various analytic functions. Scene graphs are a basic data structure of rendering systems. They represent shapes in a hierarchical data structure, which builds the shape of its parts. For example the shape of a robot can be built of the body with a head, two arms and two legs attached to.

Skeletons can be considered a type of shape representation or they may be understood as the result of a shape analysis. Loncaric (1998) classified shape analysis methods in space domain techniques vs. scalar transformations. Space domain techniques transform the shape to another spatial representation while scalar transformations compute a feature vector of numbers from the shape. These numbers need not necessarily have any direct geometric meaning. They may, for example, be used as a key to a database system.

The medial axis (Blum, 1967) is probably the most prominent space domain technique for shape analysis. In a first description, Blum defined the medial axis as the location of maximal spheres inscribed to the object touching the boundary at two or more points. His second definition is given in analogy with a fire front. If a fire was lit at the boundary of the object and propagated towards the center the fire fronts would finally meet at the medial axis. The shocks of the propagating waves form the medial axis.

The medial axis is a well defined mathematical starting point for defining skeletons, but is known to be unstable under perturbations of the object's boundary. This instability is one reason why a lot of heuristics are involved in computing skeletons. In practice, the medial axis needs to be filtered to select important parts and suppress parts induced by noise on the object's boundary. Recent developments try to understand and control this instability in detail (Attali et al., 2004; Chazal and Lieutier, 2005).

2.2 Review of Methods for Computing Skeletons

The objective of the more detailed review of skeletonization methods below is to identify suitable methods for computing skeletons of massive voxel objects with network-like architecture. Hence methods must be able to scale to large input data and they must be able to deal with the complex topology of the input.

A common way of classifying skeletonization methods is to distinguish between *thinning (grassfire propagation)*, *ridge detection in distance transformations*, and *Voronoi based methods*. Cornea et al. (2005b) differentiate four classes: *thinning and boundary propagation*, *methods using a distance field*, *geometric methods*, and *general field function based methods*. Without further justification they introduce general field function based methods as a distinct class. Field functions and distance transformations are similar in

spirit: both indicate medialness in the object. Therefore, we follow Pizer et al. (2003b) an subsume field functions and distance transformations under *medialness function based methods*. Fol-Leymarie (2003) proposes six categories (see also Giblin and Kimia, 2004): *thinning*, *ridge following in distance transformations*, *wavefront propagation*, *boundary models*, *Voronoi diagram based methods*, and *analytical bisector computation*. The latter three classes can be summarized as *geometric methods*. Considering the dynamics of wavefront propagations adds to the static analysis of a medialness function and will therefore be listed as a distinct class.

In conclusion, we group skeletonization methods into four classes:

- *thinning and boundary propagation*;
- *medialness function based methods*;
- *wave front propagation and shock detection*;
- *geometric methods*, including Voronoi diagram based methods.

2.2.1 Thinning and Boundary Propagation

Thinning is an image processing method taking a binary image as its input and eroding it to a skeleton. Thinning erodes boundary pixels according to certain rules guaranteeing homotopy equivalence of the result with the input object. The process stops if no more pixels can be removed without violating the rules. The result is a skeleton represented as a binary image. Thinning of two-dimensional images has a long history starting in the 1950s. Lam et al. (1992) give an overview over the state of the art in 1992. They classify thinning algorithms into *sequential* removal or removal in *parallel*, which is often further subdivided by the number of sub-iterations of the algorithm. Correctness of sequential algorithms is easier to prove. Parallel algorithms provide the potential of speed-up by parallel execution.

Three-dimensional thinning algorithms emerged in the 90s and continue to be an active field of research. Various authors propose different heuristics to achieve medialness of the result within the object, for retaining end-points of lines, and for retaining boundary points of faces (Bernard and Manzanera, 1999; Borgefors et al., 1999; Jonker and Vossepoel, 95; Lee et al., 1994; Ma, 1995; Ma and Sonka, 1996; Ma and Wan, 2000, 2001; Ma et al., 2002; Manzanera et al., 1999c,a; Palági and Kuba, 1999; Saha and Chaudhuri, 1996; Saha et al., 1997; Marion-Poty and Miguet, 1994). Thinning in higher dimensional spaces is also investigated (Manzanera et al., 1999b; Jonker, 2000, 2002, 2005).

Some non-standard approaches to thinning are highlighted in the following. Deutsch (1972) investigates thinning on hexagonal grids. He claims that »the hexagonal grid has advantages over the regular grid as far as processing time and data reduction are concerned«. Amini et al. (1996) present thinning of octree encoded objects to speed up computation. Thinning of two-dimensional surfaces embedded in a three-dimensional binary image is presented by Svensson and Borgefors (1999) and Nyström et al. (2001).

Ordering voxel processing by the distance to the boundary (Pudney, 1998; Vincent, 1991) provides an elegant way to ensure medialness of the resulting skeleton. Distance ordered thinning will be discussed in more detail in Section 2.3.1. Its runtime complexity is $O(n+m)$, with the number of voxel n and the number m of voxels processed multiple times, which depends on the topology of the object (and is usually much smaller than n). Distance ordered thinning combines the benefit of thinning, being based on a local investigation of a small neighborhood around a voxel, with the benefit of distance transformations (see below), which provide a global view of the location inside the object.

Thinning methods are well suited for handling voxel objects. Our input data is large and given on regular grids as binary images. Low runtime complexity and efficient memory utilization are key to achieving practical solutions. The linear runtime of thinning is a promising starting point for scaling algorithms to massive data. Thinning naturally operates on binary images and can thus be directly applied to voxel objects.

2.2.2 Medialness Function Based Methods

Another prominent class of skeletonization algorithms has its foundation in the distance transformation of the object, which encodes the distance to the nearest background point for all object points. Distance transformations can be computed in linear time (Cuisenaire, 1999). Critical points in the distance transformation indicate medialness. They can be searched and linked by following ridges (Arcelli and di Baja, 1989; Ge and Fitzpatrick, 1996; Remy and Thiel, 2000, 2002; Zhou et al., 1998) or in a hybrid approach with thinning (Gagvani, 1999; Svensson et al., 2002). Ma et al. (2003) propose to analyze a level set representation of the object based on radial basis functions and claim »skeletons generated with this approach conform more to the human perception«. The main reason is that radial basis functions provide a controlled way for creating a smooth object boundary. Methods based on tracing shortest paths in a weighted sum of the distance to the background and the distance to a seed point are a robust means to extract center lines (Bitter et al., 2000) and tree-like skeletons (Bitter et al., 2001; Sato et al., 2000); but they are not yet able to detect loops.

Several authors propose to analyze a (generalized) potential function induced by the object's boundary instead of the distance transformation. Abdel-Hamid and Yang (1994) extract a one-dimensional skeleton by tracing field lines of a potential function. The idea is further developed to generalized potential functions by Ahuja and Chuang (1997); Chuang et al. (2000); and Cornea et al. (2005a). These authors claim that potential function based methods are less susceptible to noise because of averaging over a larger region of the object boundary; but thin regions may lead to numerical instabilities. All potential function based algorithms only compute one-dimensional skeletons. They are more expensive compared to distance transformation based methods. The time complexity is $O(n \cdot m)$ with n boundary elements and m traced skeleton points.

Measures that directly indicate medialness by a scalar value and select skeletons by a threshold are another major class of skeletonization algorithms. Talbot and Vincent (1992) propose to derive a measure based on the angle formed by an object point and its nearest background points. Thinning may be used to establish strong topological

guarantees in a subsequent step (Malandain and Fernández-Vidal, 1998; Couprie and Zrour, 2005). Costa (1999) bases a measure on the geodesic distance along the object's boundary. A similar approach will be presented later in this thesis. All these approaches have in common that they are robust to noise but do not in general guarantee homotopy equivalence of the skeleton with the original object.

Burbeck and Pizer (1994) fit medial atoms to the grey value image of an object. Medial atoms have a location, orientation, and a thickness, which match the boundary of the object. This idea was further developed to m-reps, which form a network of medial atoms (Pizer et al., 2003a). If the topology of the object is known in advance, a model of the object built from medial atoms can be fit to the image data. If the topology is not know such a model of medial atoms can not be built in advance. Hence applying the method to object with unknown topology is not straight forward.

Topological analysis of scalar functions is an interesting recent approach to capture the information contained in a distance transformation; see for example (Hilaga et al., 2001; Takahashi et al., 2004). A hope is that a well defined way of pruning »unimportant« topological features, for example small loops, can be based on such methods. Zomorodian (2001) proposes in his outlook section to create a hierarchical representation of the medial axis based on morse complexes of a distance transformation.

Some medialness function based methods match our type of input. Methods for efficiently computing distance transformations of voxel objects with low runtime complexity, path tracing, and combinations with thinning will be used below as a foundation for skeletonization algorithms scaling to massive data and dealing with the complex architecture of the input object.

2.2.3 Wavefront Propagation

Blum's analogy with fire propagation is the foundation of methods simulating wave front propagation starting at the object's boundary. The shocks of the meeting fire fronts can be classified (Siddiqi et al., 1998; Giblin and Kimia, 2002, 2004) and organized as a hypergraph called the *shock scaffold* (Fol-Leymarie, 2003). The hypergraph structure reveals the organization of the medial axis in a compact representation, which is useful in shape recognition and other applications. Siddiqi et al. (1999) investigate simulated wave propagation and detection of shocks based on the flux of a vector field. Bouix (2003) proposes a combination with thinning, see also Bouix and Siddiqi (2000), to establish topological guarantees.

We will not further explore wavefront propagation because most of the shock classification methods do not match our input type but require a geometric boundary representation.

2.2.4 Geometric Methods

The last class of methods is based on computational geometry; mainly on the Voronoi diagram. Several authors (Attali et al., 2004; Reddy and Turkiyyah, 1995; Ogniewicz and Kübler, 1995) propose to compute and analyze variants of the Voronoi diagram.

The guiding idea is to compute the full Voronoi diagram of a point sampled boundary representation and prune unstable side branches in a second step (Shaked and Bruckstein, 1998) to achieve an approximation of the medial axis. In two dimensions a subset of the Voronoi diagram can be chosen, which converges (under some technical conditions) towards the medial axis with increasingly dense sampling of the boundary. In three dimensions the situation is more complex. The Voronoi diagram may contain vertices near to the object boundary and far from the medial axis, even if sampling of a smooth boundary is noise free and arbitrarily dense. To solve this problem, Amenta et al. (2001) replace the euclidean metric by the power distance and define the power crust, which »produces a surface mesh and an approximated medial axis«. The worst-case complexity of computing three-dimensional Voronoi diagrams is quadratic in the number of input points. In real-world examples runtime is often measured to be linear. Amenta et al. (2001) cite implementations capable of handling hundred thousands of input points in minutes.

Other types of geometric methods are based on analytical and piecewise linear representations of skeletons. Peternell (2000) discusses bisectors of »low-degree rational curves and surfaces, since they are of particular interest in surface modeling«. Kégl and Krzyżak (2002) propose to fit piecewise linear curves in two dimensions to find a skeleton. The authors claim their algorithm »finds a smooth medial axis of the great majority of a wide variety of character templates and substantially improves the pixelwise skeleton obtained by traditional thinning methods«.

We will not further explore geometric methods. Geometric methods do not utilize the specific organization of our input on a regular grid but instead start from point samples or an analytic boundary representation. The size of input that can be handled by geometric methods today is far from the size of the massive voxel objects targeted in this thesis.

2.3 Voxel and Grid Cell Skeletons

Thinning and medial function based algorithms are promising foundations for devising skeletonization methods for massive voxel objects with complex architecture. Efficient algorithms operating on regular grids exist for both type of methods, as discussed in the previous section.

We will now transfer topological notions from the continuous space $\mathbb{R}^3$ to binary images and develop a clear understanding of how to shrink a binary image to its skeleton. This process is commonly denoted as thinning; see for example Lam et al. (1992). Thinning successively removes voxels starting at the boundary of the object. Voxels are eligible for removal if homotopy equivalence is preserved. Theses voxels are called simple. A detailed characterization will be given shortly but first a more formalized view of binary images is required for an in-depth discussion of voxel skeletons.

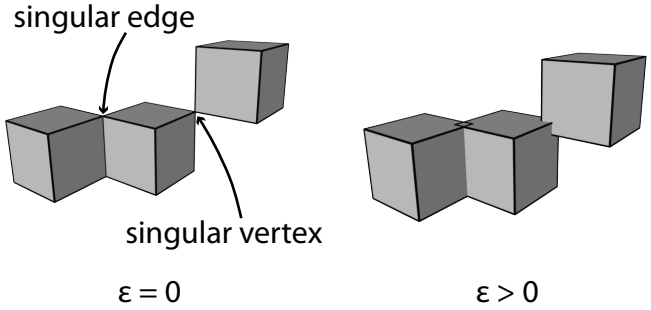
We start with the definition of a binary image (which was already sketched in the introduction). A **binary image** is a function $I : \mathbb{Z}^3 \to \{0, 1\}$. Elements of the finite set $P \subset \mathbb{Z}^3$ of points that map to 1 are called **object voxels** or **foreground voxels**. P is called a **voxel object**. Elements of $\bar{P} = \mathbb{Z}^3 \setminus P$ are called **background voxels**.

An intuitive starting point for understanding topological properties of binary images is thinking of them as geometrically represented in continuous space, which is achieved by the function

$$\begin{aligned} R_\epsilon : \mathbb{Z}^3 &\to \mathcal{P}(\mathbb{R}^3) \\ p &\mapsto R_\epsilon(p) = \left\{ x \mid \|x - p\|_{\max} \leq \frac{1}{2} + \epsilon \right\} \end{aligned} \tag{2.1}$$

which maps object voxels to unit cubes in $\mathbb{R}^3$ inflated by ϵ and centered at the voxel position p. The object is mapped to the union of the images of its voxels. The background of the binary image is mapped to $\mathbb{R}^3 \setminus R_\epsilon(P)$. Note the asymmetry in the mapping of foreground and background. The asymmetry is present even for $\epsilon = 0$. But it is easier to grasp for a small positive ϵ, which inflates the unit cubes to avoid singular edges and vertices:

Figure 2.2: Representation of voxels as unit cubes and inflated cubes.



The geometric representation $R_\epsilon(P)$ has two properties that help intuition:

1. Its boundary $\partial R_\epsilon(P)$ is a two-dimensional manifold separating interior from exterior.

2. If two object voxels $p, q \in \mathbb{Z}^3$ are connected by a path in $R_\epsilon(P)$, this path can be chosen to be located completely in the interior of $R_\epsilon(P)$.

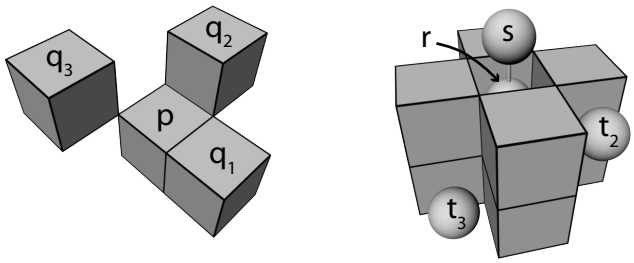
The following discussion presupposes some basic knowledge of topology. Here, notions will be formally defined only if essential for the further discussion. An introduction to basic concepts of topology is found in, for example, Jänich (1994). Combinatorial aspects are discussed in Aleksandrov (1956) or Boltianskij and Efremovich (2000); Hatcher (2002) focuses on cell complexes and homotopy theory (and more) in a modern language.

The Grid Point Model

The connectivity in a local neighborhood behaves differently for background and for foreground voxels. Two object voxels p and $q_{1/2/3}$ (see figure below) can be connected by a straight path in their geometric representation $R_\epsilon(\{p, q_{1/2/3}\})$ if each of their integer coordinates differs by not more than one. The situation is slightly different for two background voxels r and s. Independently of the surrounding configuration they can only then always be connected by a straight path in their geometric representation if exactly

one coordinate differs by not more than one, and all other coordinates are equal (r and $t_{2/3}$ in the figure below can not be connected by a straight path):

Figure 2.3: Adjacency of neighboring object voxels and background voxels.

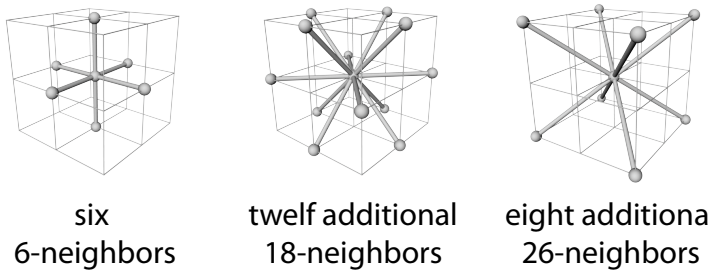


This observation is formalized in the **grid point model** as follows (see Klette and Rosenfeld (2004) for a comprehensive introduction to all parts of digital geometry and topology). Voxels are interpreted as grid points together with an adjacency relation. Two **grid points** (also called **voxels**) $p, q \in \mathbb{Z}^3$ are α-**adjacent**, in the following way: they are

$$\begin{aligned} \text{6-adjacent, if } & 0 \leq |p-q| \leq 1 \,; \\ \text{18-adjacent, if } & 0 \leq |p-q| \leq \sqrt{2} \,; \\ \text{26-adjacent, if } & 0 \leq |p-q| \leq \sqrt{3} \,. \end{aligned}$$

A voxel q is said to be an α-**neighbor** of p, if q is α-adjacent to p. All α-neighbors of a voxel p form its α-**adjacency set** $A_\alpha(p)$. $N_\alpha(p) = A_\alpha(p) \cup \{p\}$ is called the α-**neighborhood** of p. The prefixed numbers indicate the number of α-neighbors:

Figure 2.4: The three grid point adjacency relations.



We also need the notion of a path and a (connected) component. An α-**path** from voxel q to voxel r is a sequence of points $\rho = (p_0, \ldots, p_n)$, with $p_0 = q, p_n = r$, such that p_{i+1} is α-adjacent to p_i for $i = 0, \ldots, n-1$. The **length** of the path is n. q and r are said to be α-**connected** (by the path ρ). A maximal set of α-connected points is called an α-**component**.

Consistency with the geometric representation in $\mathbb{R}^3$ requires that object voxels are chosen to be 26-adjacent while background voxels are 6-connected reflecting the asymmetry in their geometric representation. This text will only use (26,6)-adjacency, although other valid pairings may be investigated as well; for example 6-connected foreground and 26-connected background. See Klette and Rosenfeld (2004) for a comprehensive discussion.

The grid point model can be used to efficiently analyze questions such as computing connected components, computing distance transformations, and thinning, as presented below. But its reduction of voxels to points related by adjacency may restrict intuitive understanding of the topological properties of the represented object. Transferring notions from continuous topology may also be difficult. A second interpretation of binary images, the grid cell model, is more successful in representing details of topological configurations of lower dimension.

The Grid Cell Model

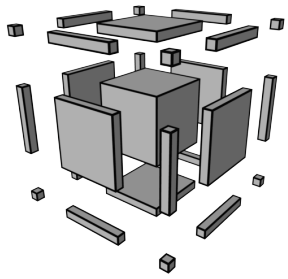
In the **grid cell model** voxel objects are understood as cell complexes. The presentation loosely follows Klette and Rosenfeld (2004); see also Kovalevsky (1989). A unit cube may be regarded as decomposed into cells of dimensions 0 up to 3, so called d-cells c^d, which form a cell complex. A (finite dimensional) **cell complex** X is built in the following way:

- The discrete set of 0-cells forms the 0-skeleton X^0.
- Inductively build the **d-skeleton** X^d from X^{d-1} by attaching d-dimensional disks, the **d-cells**, such that the boundary of each disk is continuously mapped to X^{d-1}.
- Stop the process at a finite stage $d < \infty$ and set $X = X^d$.

We will only regard cell complexes of up to $d = 3$ in the following.

In the grid cell model, a cube is decomposed into a cell complex with 8 vertices (0-cells), 12 edges (1-cells), 6 faces (2-cells) and the enclosed volume (3-cell). The different cells will be depicted as cuboids, scaled and oriented according to the cell dimension, position, and orientation:

Figure 2.5: A voxel decomposed into grid cells.



The grid cell complex $X(\mathbb{Z}^3)$ of all voxels $p \in \mathbb{Z}^3$ is built from cubic geometric representations $R(p) = R_{\epsilon=0}(p)$ in the following way (see also Figure 2.6):

- The intersection of the geometric representation of eight pairwise 26-adjacent voxels $p_1, \ldots, p_8$ forms a 0-cell $c^0_{\{p_1,\ldots,p_8\}} = \bigcap_{i=1\ldots8} R(p_i)$. These cells are the vertices shared by cubic voxels.
- The intersection of the geometric representation of four pairwise 18-adjacent voxels $p_1, \ldots, p_4$ forms a 1-cell $c^1_{\{p_1,\ldots,p_4\}} = \bigcap_{i=1\ldots4} R(p_i)$. These cells are the edges shared by cubic voxels.

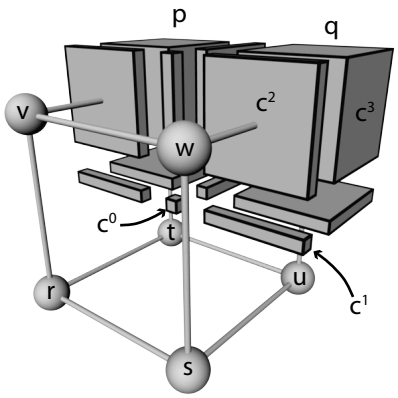


Figure 2.6: A voxel configuration and its equivalent grid cell configuration. Two object voxels, p, q, six background voxels, r, s, t, u, v, w, and some of the associated grid cells are depicted. The labeled grid cells are a 0-cell $c^0 = c^0_{\{p,q,r,s,t,u,v,w\}}$, a 1-cell $c^1 = c^1_{\{q,s,u,w\}}$, a 2-cell $c^2 = c^2_{\{q,w\}}$, and a 3-cell $c^3 = c^3_{\{q\}}$.

- The intersection of the geometric representation of two 6-adjacent voxels p, q forms a 2-cell $c^2_{\{p,q\}} = R_0(p) \cap R(q)$. These cells are the faces shared by cubic voxels.

- The geometric representation of a voxel forms a 3-cell $c^3_{\{p\}} = R(p)$. These cells are the voxels itself.

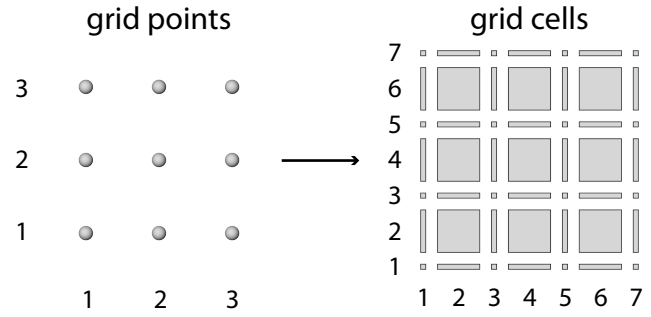
A voxel object P is represented as a sub-complex $X(P) \subset X(\mathbb{Z}^3)$. Cells c^d_S are included in the **grid cell complex** $X(P)$ if any of the voxels in the set S, labeling the cell, is included in the object P, that is if $S \cap P \neq \emptyset$; in other words if any of the voxels sharing the cell is an object voxel. An example is depicted in Figure 2.6. $X(P)$ itself is a cell complex while its complement in $X(\mathbb{Z}^3)$, representing the background, is not a cell complex.

Two cells are said to be **incident** if one is a subset of the other. Based on their dimension, a partial order is defined on the cells of a complex by $c^d \prec c^e$ if c^d incident with c^e and $d < e$. This relation will be denoted as the **partial incidence order**.

The overall number of grid cells is eight times the number of corresponding voxels. Each cube has six faces and each face is shared by two cubes. Hence the number of 2-cells is three times the number of 3-cells. Similarly, the number of 1-cells is three times the number of 3-cells (12 edges shared by 4 cubes each). Finally, the number of 0-cells is the same as the number of 3-cells (8 vertices shared by 8 cubes). The overall number of cells computes to 1 + 3 + 3 + 1 = 8 times the number of voxels.

This suggests that cells can be enumerated by $\mathbb{Z}^3$ using $2 \times 2 \times 2$ elements per voxel. Indeed, we succeed by associating every cell c^d_S with $x = 2 \cdot \langle S \rangle + (1, 1, 1)$, with $\langle S \rangle$ the mean value of S's elements computed as the mean of the coordinates of S's elements in each dimension as illustrated below:

Figure 2.7: Enumerating grid points and grid cells on a regular grid in 2d.



The discussion of the grid cell model could also start from a discrete topology defined on $\mathbb{Z}$ with alternating open and closed points (0, 2,… closed; 1, 3,… open). The cartesian product of three such topological spaces defines the so-called *Khalimsky digital space* $\mathbb{K}^3$ (Khalimsky et al., 1990). Kong et al. (1991) showed that this is equivalent with the above discussion of the grid cell model.

The grid cell model and the topological properties of its cells can be efficiently represented by an array requiring eight times the storage space of a grid point model. The integer coordinates' parities completely determine the topological properties of a cell in the following way. For a cell c_x^d, with $x = (x_1, x_2, x_3) \in \mathbb{Z}^3$,

- the dimension d equals the number of odd coordinate entries in x;
- the cell is incident with two higher dimensional cells in the coordinate direction i if entry x_i is even;
- the cell is incident with two lower dimensional cells in the coordinate direction i if entry x_i is odd.

Simple integer operations on the coordinates allow to compute topological properties. Encoding cell coordinates as single packed integers may reduce storage space and allows efficient topological operations as discussed by Lachaud (2003).

A major advantage of the grid cell model compared to the grid point model will become apparent below: lower dimensional cells are explicitly represented. This gives the opportunity of encoding digital surfaces and lines explicitly, in contrast to approximating them by voxels.

2.3.1 One-Dimensional Skeletons: Thinning

We will continue the discussion with how to shrink a voxel object to an one-dimensional voxel skeleton. One requirement on a skeleton is to preserve the main topological structure, that is the skeleton must be homotopy equivalent to the object. Intuitively this means the object can be transformed to the skeleton by elastic deformations such as bending and shrinking (see a textbook on topology for exact definitions).

However, homotopy equivalence is not strong enough to characterize skeletons as illustrated in Figure 2.8. The figure illustrates how a homotopy can be established between the original object at the top and the object at the bottom, from which one voxel was

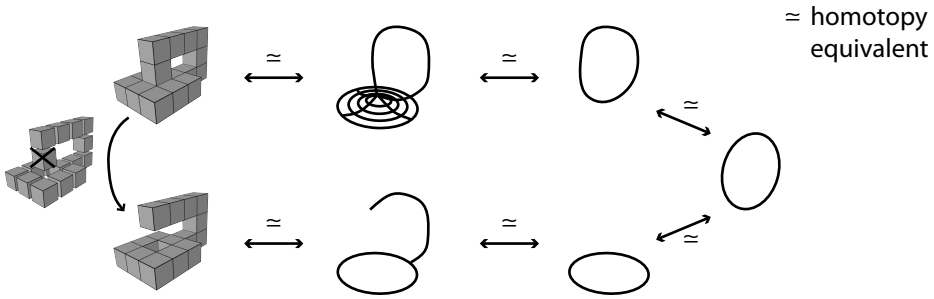


Figure 2.8: A homotopy equivalent object that is not a skeleton. The object at the bottom is created from the object at the top by removing the indicated voxel. Towards the right both objects are retracted to a ring. The ring can be expanded to any of the original objects to establish a homotopy. But there is no deformation retraction from the object before and after removing the indicated voxel. Hence the object at the bottom should not be a skeleton of the original object at the top.

removed. If homotopy equivalence with the original object was a sufficient requirement for a skeleton the object at the bottom would be a skeleton of the object at the top. But this contradicts the intuition of contracting an object to its skeleton.

Restricting elastic deformations to continuous shrinking of the object (excluding bending and inflation) establishes a stronger condition than homotopy equivalence. In topological terms this is expressed as a deformation retraction. Assume a subset $Y \subset X$. A (strong) **deformation retraction** is a mapping $\rho : X \times [0,1] \to X$ with $(x,t) \mapsto \rho(x,t) = \rho_t(x)$, such that $\rho_0 \equiv \mathrm{id}_X, \rho_1(X) = Y, \rho_t|_Y \equiv \mathrm{id}_Y$, and ρ is continuous in x and t. The subset Y is said to be a **deformation retract** of X. You may think of deflating the volume X continuously until Y remains. Every deformation retraction is a homotopy and the deformation retract Y is homotopy equivalent to the starting set X.

One necessary condition for a **homotopic voxel skeleton** S is to be a deformation retract of the original voxel object P, in the sense that this holds for the geometric representations, that is $R_\epsilon(S)$ is a deformation retract of $R_\epsilon(P)$, and thus also for their grid cell complexes $X(S)$ and $X(P)$. We will use »deformation retract« in this sense on voxel objects directly. As explained shortly, being a deformation retract is not always sufficient for a definition of a voxel skeleton as other properties, such as medialness or reconstructability, may also play a role.

Simple Voxels

In practice, a deformation retraction is not given explicitly but instead so called simple voxels are removed sequentially such that each removal preserves homotopy equivalence. A voxel p of a voxel object P is said to be a **simple voxel** if $P \setminus \{p\}$ is a deformation retract of P. Various characterizations have been given for simple voxels. Klette (2003) compares some of them. All are equivalent to the definition in terms of a deformation retraction. But they may have the advantage of being computationally more tractable or

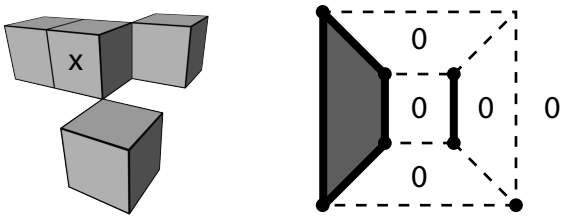


Figure 2.9: A voxel configuration and its Schlegel diagram. The Schlegel diagram of voxel x is depicted at the right. Faces in the Schlegel diagram are associated with 6-neighbors, edges with 18-neighbors, and vertices with 26-neighbors. An element of the diagram is colored black if a neighboring voxel is present.

being defined directly on the grid point model, without recurring to homotopy theory. Two characterization will be presented in more depth now.

Kong (1995) proved that the definition based on deformation retraction is equivalent to a characterization given in terms of the P-attachment set. The **P-attachment set** of a voxel p of a voxel object P contains all grid cells of $P \setminus \{p\}$ that are incident with p. These are all cells in the boundary of p that are also contained in a neighboring object voxel. The P-attachment set can be visualized in a **Schlegel diagram**, see Figure 2.9. Faces in the Schlegel diagram are associated with 6-neighbors, edges with 18-neighbors, and vertices with 26-neighbors. Elements are labeled with 1 (indicated in black in Figure 2.9) if the associated neighbor is an object voxel; they are labeled with 0 if the neighbor is a background voxel. The outside of the Schlegel diagram represents the front face of the voxel. Thus the diagram represents the configuration in the 26-neighborhood of a voxel.

Kong proved the following (in a slightly different formulation):

Characterization 1 *A voxel p is simple in P if and only if both the P-attachment set, and the complement of that set in the boundary of p (that is in the Schlegel diagram of p) are non-empty and connected.*

The main idea of his proof is to subdivide the voxel p into 12 tetrahedra by splitting each face into two triangles and choosing the centroid of the voxel as the fourth vertex. Starting from the non-empty background component, all these tetrahedra can be collapsed. Next, the induced triangles in the boundary of p are collapsed until the P-attachment set remains. This sequence constructs a strong deformation retraction from P to $P \setminus \{p\}$. For the remainder of the proof—that a strong deformation retract implies two components—the reader is referred to Kong's article.

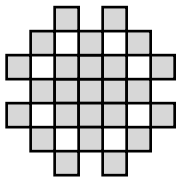
Kong also proved that his characterization is equivalent to characterizations based on the investigation of connected components in the neighborhood of a voxel. Such characterizations are discussed by Saha et al. (1991); Malandain and Bertrand (1992); and Bertrand and Malandain (1994). The characterization given in the last paper is stated below:

Characterization 2 *A voxel p of an object P is simple if it is 26-adjacent to exactly one 26-component of voxels in its 26-adjacency set $A_{26}(p)$ and it is 6-adjacent to exactly one 6-component of background voxels in its 18-adjacency set $A_{18}(p)$.*

The presented characterizations are used in **thinning** algorithms to construct a voxel skeleton by sequentially removing simple object voxels $p_1, \ldots, p_n$ until no more voxels can be removed. Each of these voxels p_i must be simple in the object $P_{i-1} = P \backslash \{p_1, \ldots, p_{i-1}\}$ remaining after removal of the first $i-1$ voxels of the sequence. The construction induces a deformation retraction of the voxel object P_0 to the object P_n. Hence, homotopy equivalence is preserved during the process. We also require that all voxels of P_n are non-simple. Otherwise removal could continue.

The goal is to create a **thin voxel object**, that is a voxel object in which every voxel has a background neighbor. In some cases, this is not achievable and thinning stops at a point where interior voxels remain in the result. In the following two-dimensional example

Figure 2.10: Set of non-simple voxels containing interior voxels.



all voxels are non-simple but the voxels in the center do not have any background neighbors. Similar configurations can be constructed in three dimensions. Thinning in the grid point model fails to resolve such configurations. As discussed below, they can be resolved in the grid cell model or by enforcing an interpretation of voxel configurations as geometry of a specific dimension, for example by collapsing a connected component of interior voxels to a single point.

Thinning, discussed so far, achieves homotopy and thinness (in most cases) of the constructed skeleton but medialness and reconstructability may also be required, as already mentioned in the introduction to this chapter. **Medialness** means each point on the skeleton has at least two nearest neighbors on the boundary of the object. In the figure below, the dashed line on the left illustrates a homotopic set, which is *not* medial:

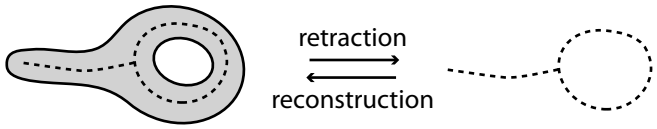
Figure 2.11: A homotopic skeleton that is not medial and a medial skeleton.

If this set were chosen as a skeleton, it would give misleading information about the position of the original object. The dashed line on the right illustrates a medial set which is a suitable skeleton.

Reconstructability means the original object can be reconstructed from the skeleton. This can be achieved by storing the local thickness at each skeleton point, that is the distance to the nearest background point, and reconstructing the object as the union of balls centered at all skeleton points with a radius given by the local thickness. Depending

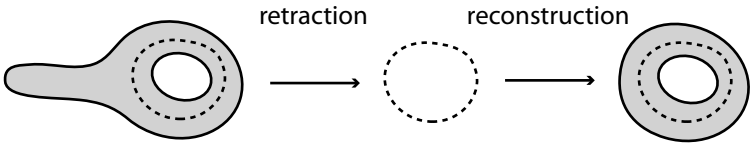
on the details of the process, the reconstructed object will either match the original object exactly or only approximately. The following skeleton on the right is able to reconstruct the original object on the left:

Figure 2.12: An object and a skeleton that is able to reconstruct the object.



Whereas the skeleton in the following counter example fails to reconstruct the original object:

Figure 2.13: An object and a skeleton that fails to reconstruct the object.



Medialness will be discussed in more detail next; reconstructability shortly after.

Distance Ordered Homotopic Thinning

Pudney (1998) proposes distance ordered homotopic thinning (DOHT), a simple and efficient method to compute a medial voxel skeleton. Algorithm 1 presents the basic version of the algorithm. Distance ordered homotopic thinning stores voxels waiting for processing in a priority queue. Voxels with a low distance to the nearest boundary are processed first. Processing starts by queuing all boundary voxels. Other voxels are added to the priority queue only after their neighborhood changed and tests for their deletion might succeed. Medial voxels have a high distance value and therefore are deleted last or are retained if they are non-simple. Some voxels may be queued several times if their neighborhood changed more than once. The overall number of tests for deletion is $O(n + m)$, with the number of voxels n and the number m of voxels queued multiple times, which depends on the topology of the object.

In addition to the binary image itself, DOHT requires a distance transformation. The **distance transformation** of a voxel object P is a mapping $d : P \to \mathbb{R}$ that assigns every foreground voxel p the distance to the nearest background voxel

$$d(p) = \min_{q \in \bar{P}} d(p, q)$$

where $d(p, q)$ is a metric. If exact medialness is desired, the euclidean metric should be used. Otherwise an approximation of the euclidean metric may be sufficient. A comprehensive discussion of algorithms for computing distance transformations can be found in Cuisenaire (1999).

The chamfer distance transformation is an efficient approximation of the exact distance map. Its computation is based on integer weights along voxel paths to the background. The **weighted length** of a path $\rho = (p_0, \ldots, p_n)$ is computed as $\text{len}(\rho) = \sum_{i=1}^{n} d(p_{i-1}, p_i)$.

Algorithm 1 Basic Distance Ordered Homotopic Thinning takes a voxel object P and its distance transformation d; and shrinks P to its homotopic skeleton. Adapted from Pudney (1998).

```
1:  procedure BasicDOHT(P, d)
2:      for all p ∈ P do
3:          if A_6(p) ∩ P̄ ≠ ∅ then                ▷ If p has 6-neighbor in background,
4:              Enqueue(Q, p, d(p))                ▷ queue with priority for low distances.
5:              l(p) ← QUEUED
6:          else
7:              l(p) ← UNQUEUED
8:          end if
9:      end for
10:
11:     while Q not empty do
12:         r ← Dequeue(Q)                         ▷ Take head of queue.
13:         l(r) ← UNQUEUED
14:         if r deletable in P then
15:             P ← P \ {r}
16:             for all s ∈ A_26(r) ∩ P do         ▷ Place 26-object neighbors,
17:                 if l(s) = UNQUEUED then        ▷ if not already queued,
18:                     Enqueue(Q, s, d(s))        ▷ into queue.
19:                     l(s) ← QUEUED
20:                 end if
21:             end for
22:         end if
23:     end while
24: end procedure
```

The **chamfer distance transformation** is defined as assigning to a voxel $p \in P$ the lowest weighted length along all paths from p to the background

$$d_C(p) = \min_{\rho(p,q) \text{ with } q \in \bar{P}} \text{len}(\rho(p,q))$$

The euclidean distance between two neighboring points is approximated by integer values. This allows to save storage space and computation time. The following integer values are used in the original paper:

$$d(p,q) = \begin{cases} 3 & \text{if} \quad |p-q|^2 = 1 \\ 4 & \text{if} \quad |p-q|^2 = 2 \\ 5 & \text{if} \quad |p-q|^2 = 3 \end{cases}$$

These values minimize the upper bound on the difference of the chamfer distance transformation and the euclidean distance transformation, as proved by Borgefors (1984).

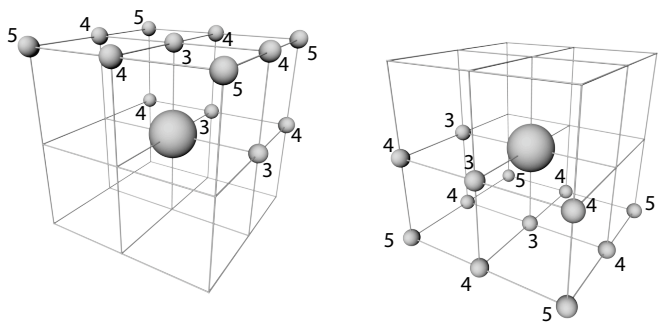
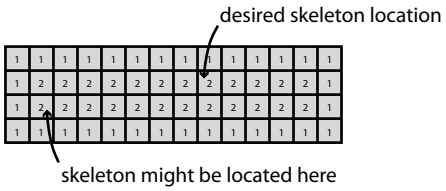


Figure 2.14: Propagation masks used in 3d chamfer raster scans. The value stored at the center voxel is increased by the integer weights 3, 4, 5 noted at the neighboring voxels. Two masks are used. The first mask propagates to the right, back and top. The second mask propagates to the left, front, and bottom.

Chamfer distance transformations can be efficiently computed by shortest path propagation with integer weights (Verwer et al., 1989) or two raster scans (Borgefors, 1986). Path propagation uses a bucket sorted queue to store a front of propagating voxels sorted by their current integer distance and propagates the lowest value to neighboring voxels. The raster scan algorithm uses the two masks illustrated in Figure 2.14 in a forward and a backward raster scan. Values at the central voxel are increased by the value indicated in the mask and propagated if the sum is smaller than the value at the target voxel. Equipped with a distance transformation, basic DOHT is straight forward as detailed in Algorithm 1.

A potential problem is that BasicDOHT's results may depend on the input order. BasicDOHT removes voxels sequentially. Because deletion of a voxel may influence tests for deletion of neighboring voxels, the result can strongly depend on the input order, in which the voxel object is processed. The problem is especially obvious for large two-dimensional planes of voxels. Such planes may contain two-voxel-thick layers of voxels with the same distance value. An example is illustrated by a two-dimensional slice through a plane in three dimensions:

Figure 2.15: Distance map containing two-voxel-thick layer with same values.



The order of tests depends on the input order of the voxels. Most resulting skeletons would not be centered within the two-dimensional plane. Imposing restrictions on the order of processing may improve the »smoothness« of the shrinking process and alleviate these deficiencies.

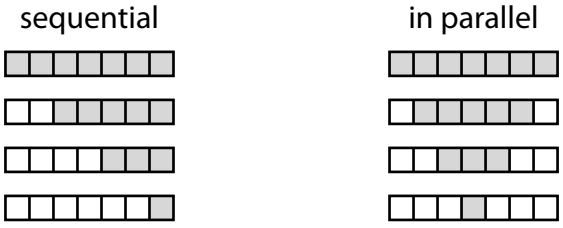
Organizing the algorithm in iterations of voxels deleted simultaneously tackles the problem of inter-dependency of tests for deletion. The skeleton is constructed as a

sequence $S_0, S_1, \ldots$ of iterations. Tests for deletion in iteration S_i are based only on S_{i-1}. This idea is often denoted as testing voxels in parallel (Lam et al., 1992). As noted above, tests for deletion can not be performed independently of the removal of neighboring voxels. Care must be taken to guarantee that S_i is in fact a deformation retract of S_{i-1}. But obviously subsets of voxels exist that can be removed simultaneously. A set $Q \subset P$ is said to be a **simple set** in P if there exists an ordering $q_1, \ldots, q_n$ of its elements, such that each q_i is simple in the object $P_{i-1} = P \setminus \{q_1, \ldots, q_{i-1}\}$ remaining after removal of the first $i-1$ voxels. Each $\Delta S = S_{i-1} \setminus S_i$ must be a simple set in S_{i-1}.

One way to guarantee simple sets are restrictions in each iteration on the candidate voxels tested for deletion. Note, the 26-neighborhood of a voxel fully determines whether it is simple. Both characterizations of simple voxels given earlier only rely on the direct 26-neighbors of a voxel. Thus, every second voxel in each dimension can be tested and removed independently of each other. This partitions the image into eight so called sub-fields. Weaker restrictions may be sufficient, as discussed, besides others, by Kong (1995) or Ma et al. (2002).

Homotopy can also be preserved by a strategy denoted by Lee et al. (1994) as sequential rechecking. A first processing step searches candidate voxels for deletion by testing all voxels in S_{i-1} and a second step removes voxels sequentially, but only if they are still deletable in the remaining object. Cases like the one illustrated below are now handled as expected. The skeleton is located in the center of the voxel configuration because voxels are deleted »in parallel« from both sides:

Figure 2.16: Result of removing sequentially and of removing »in parallel«.



Difficulties caused by two-voxel-thick planes with the same distance value can be mitigated by introducing sub-iterations. Each iteration is split in two sub-iterations. In each sub-iteration only selected spatial directions are tested. For example in the first sub-iteration only voxels at the top boundary would be tested, in the second sub-iteration only voxels at the bottom. The 26-neighbors define 26 spatial directions that can be grouped into two sub-iterations as illustrated in Figure 2.17. The depicted masks will be used in SmoothDOHT (see Algorithm 2) to compute subiteration(t, P) for a voxel $t \in P$ depending on the configuration in its neighborhood. Only a single sub-iteration is processed »in parallel«. If the object contains no plate-like structures sub-iterations may be omitted.

Algorithm 2 presents the discussed modifications to DOHT. SmoothDOHT computes a centered homotopic skeleton. The process is split into iterations and sub-iterations. Sub-iterations alternate based on an integer counter. In each sub-iteration sequential rechecking is applied to ensure homotopy equivalence when removing voxels »in parallel«.

Algorithm 2 Smooth Distance Ordered Homotopic Thinning takes a voxel object P and its distance transformation d; and shrinks P to its homotopic skeleton.

```
1:  procedure SmoothDOHT(P, d)
2:      for all p ∈ P do
3:          if A_6(p) ∩ P̄ ≠ ∅ then                ▷ If p has 6-neighbor in background,
4:              Q(d(p)) ← Q(d(p)) ∪ {p}            ▷ queue for processing at distance d(p).
5:              l(p) ← QUEUED
6:          else
7:              l(p) ← UNQUEUED
8:          end if
9:      end for
10:     i ← 0  ;  d_max ← max(d(P))
11:     repeat
12:         i ← i + 1  ;  d ← 0
13:         X ← ∅                                  ▷ Set of matched sub-iterations.
14:         while d ≤ d_max ∧ X = ∅ do             ▷ Break if a voxel was deleted.
15:             T ← Q(d)  ;  Q(d) ← ∅              ▷ Take all voxels with distance d.
16:             for all t ∈ T do                   ▷ First, parallel check.
17:                 l(t) ← UNQUEUED
18:                 if t deletable in P then
19:                     l(t) ← (CANDIDATE, subiteration(t, P))
20:                     X ← X ∪ {subiteration(t, P)}
21:                 end if
22:             end for
23:             if X ≠ ∅ then
24:                 x ← choosesubiteration(X, i)   ▷ select sub-iteration based on counter i
25:                 for all t ∈ T do               ▷ Sequential rechecking
26:                     if l(t) = (CANDIDATE, x) then   ▷ of selected sub-iteration.
27:                         l(t) ← UNQUEUED
28:                         if t deletable in P then
29:                             P ← P \ {t}
30:                             for all s ∈ A_26(t) ∩ P do   ▷ Queue 26-object neighbors,
31:                                 if l(s) = UNQUEUED then   ▷ if not queued,
32:                                     Q(d(s)) ← Q(d(s)) ∪ {s}   ▷ for processing.
33:                                     l(s) ← QUEUED
34:                                 end if
35:                             end for
36:                         end if
37:                     else if l(t) = (CANDIDATE, ·) then   ▷ Re-queue other sub-iterations.
38:                         Q(d(t)) ← Q(d(t)) ∪ {s}
39:                         l(t) ← QUEUED
40:                     end if
41:                 end for
42:             end if
43:             d ← d + 1
44:         end while
45:     until X = ∅                                ▷ Continue as long as deletable voxels were found.
46: end procedure
```

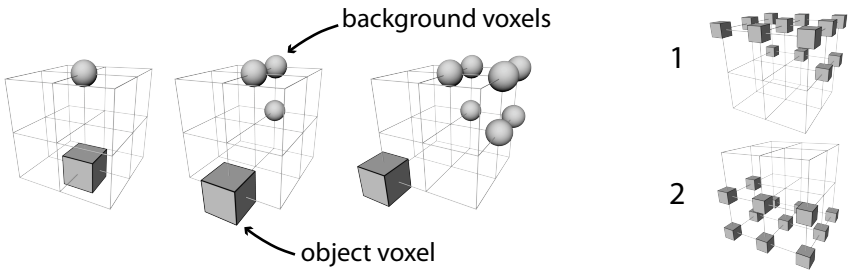
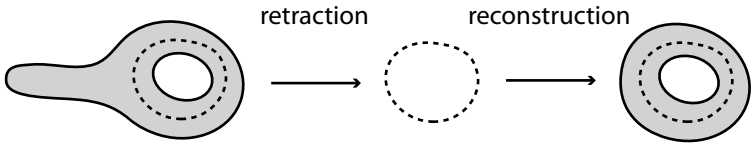


Figure 2.17: Masks selecting spatial sub-iterations. The indicated object and background voxels must match a mask to select the central voxel, which is omitted in the figure, for processing. Neighbors left free in the figure are ignored during matching. Representative masks to detect face, edge, and corner configurations are depicted from left to right. The masks to select a sub-iteration are illustrated at the very right. The object voxel of a face, edge, or corner mask must be located at any of the indicated positions to be included in sub-iteration 1 respectively 2.

The skeleton computed by SmoothDOHT is not yet necessarily able to reproduce the original shape of the object if all simple voxels are removed as illustrated below:

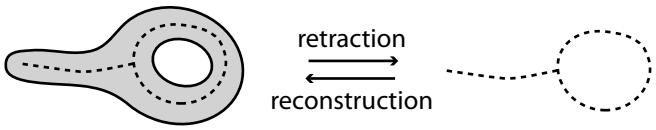
Figure 2.18: Skeleton that fails to reconstruct an object.



The left side branch is not represented in the skeleton, although it is homotopic to the object. We did not yet require reconstructability of the original object from its skeleton.

Changing the rule for deletion of a voxel provides a way to integrate reconstructability with thinning. Being simple alone is no longer sufficient, but geometric properties, for example not being an end-point, are tested in addition:

Figure 2.19: Skeleton, with local end-points, capable of reconstructing the object.



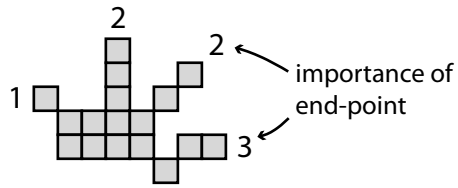
Other geometric properties, for examples not being a boundary voxel of a surface, may be regarded, too. See for example Borgefors et al. (1999); Jonker (2002, 2005).

Geometric properties that are specified in a local neighborhood of a voxel are susceptible to noise. Spurious side branches are a typical problem. Several authors propose to remove them in a post-processing pruning step; among others, see Shaked and Bruckstein (1998). Modifying the geometric conditions included in the test for deletion allows to avoid them early in the process. The following discussion covers only one-dimensional voxel skeletons. The idea might be extended to two dimension. A measure of the local importance of a surface boundary voxel would than be needed. But this is beyond the

scope of this thesis.

The test for end-points based on an investigation of the local A_{26} neighborhood can be improved by considering a larger neighborhood. An end-point is a voxel with only one neighbor. Requiring a certain »length« of the branch ending there makes the characterization more robust to noise. The idea is formalized in the following. An object voxel $p \in P$ is said to be a **local end-point** of P, if p has exactly one 26-neighbor. A path following a side branch is used to compute the **importance of an end-point** as the length of a path $\rho = (p_0, \ldots, p_n)$ with $p_0 = p$, such that the first $n - 1$ voxels p_i are local end-points of the objects $P_i = P \setminus \{p_0, \ldots, p_{i-1}\}$, with $P_0 = P$. The last voxel p_n is not an end-point of P_n. It »roots« the path in the »body« of the object. Some cases are illustrated below:

Figure 2.20: End-points of branches of various lengths.



The test for deletion in the thinning algorithm is modified by integrating a parameter dependent end-point detection in the following way. A voxel p is deletable if

- p is simple;
- p is not a local end-point of a side branch longer than a user specified threshold l.

The parameter l controls sensitivity of detecting side branches and needs to be balanced against susceptibility to noise.

The utility of skeletons computed by the presented thinning algorithm depends on the architecture of the input object. Figure 4.1 (all examples are collected in Chapter 4) depicts the results of thinning of a rod-like architecture without cavities. If no end-points are preserved, thinning will compute a skeleton representing only the topologically stable features, that is the loops. A low threshold on the importance of end-points yields a skeleton with many side branches, which are supressed at a higher threshold. The threshold allows to trade reconstructability versus robustness to noise. Figure 4.3 illustrates objects with plate-like and sphere-like elements. Plate-like structures are represented by the computed skeleton to some extent. The topological features are reflected, while details of the geometry, like for example the main direction of the plates, are not. The situation is similar for a sphere-like part. Cavities however change the situation. Cavities are a topological feature and thus always preserved by thinning. The result is no longer one-dimensional but contains a surface enclosing the cavity.

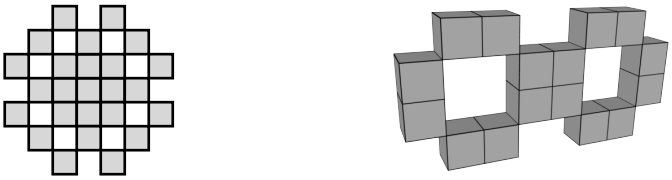
Noise may cause topological changes that lead to substantially different skeletons because thinning guarantees strict homotopy equivalence. Figure 4.4 depicts an example with a high level of noise. The skeleton includes loops and closed surfaces, which are not present in the noise-free object. The skeleton captures every topological change, which limits its utility in the presence of high noise levels.

Strict homotopy equivalence is not always a desired feature. For example it could be more appropriate to consider small loops as noise. But thinning preserves homotopy equivalence. If homotopy equivalence is not a primary objective, other algorithms might be a better choice. The TEASAR algorithm (Sato et al., 2000), as one example, is designed to detect tree like branching structures only. *All* loops will be ignored, which makes the TEASAR algorithm highly robust to noise.

Thinning grid cell complexes

We noted earlier that voxels sometimes fail to exactly represent lower dimensional structures, especially at junctions:

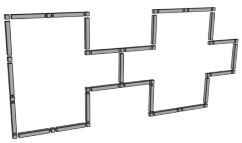
Figure 2.21: Junctions with unclear interpretation.



The voxel configuration on the left can not be further eroded. Voxels lock each other in a way that no voxel is simple. At the right, the interpretation of a voxel configurations representing a junctions poses similar problems. While it is obvious how to interpret voxel configurations as lines if each voxel has exactly two neighbors, it is not obvious for the junction in the center. A human viewer would probably connect the two loops at a single point located in the center, between the four voxels forming the junction. But a formalization of this human interpretation is challenging.

The grid cell model can explicitly capture lower dimensional structures and is therefore superior to the grid point model for representing skeletons:

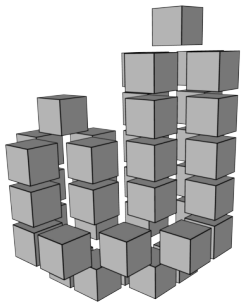
Figure 2.22: Junction in the grid cell model.



The two rings are now represented as a one-dimensional cell complex. Ambiguities in the interpretation have vanished. Note, this cell complex has no interpretation as a voxel object, as voxels are represented by 3-cells but the complex only contains 0- and 1-cells. The representation of all dimensions in the grid cell model explicitly gives this freedom. Skeletons are a lower dimensional representation. Hence, the grid cell model's capability of capturing lower dimensional structures faithfully may provide advantages compared to the grid point model.

Another example of failure of the grid point model to represent topology are voxel configurations that are homotopy equivalent to a point but can not be eroded by thinning. In the following example, each of the two »towers« encloses a tunnel:

Figure 2.23: Locked voxel configuration that can not be further eroded.



The two tunnels join at the base and are connected to the outside in the center. The object can be retracted to a point but thinning fails to do so. No simple voxel exists to start the retraction. Obviously, such configurations can be infinitely large.

The following paragraphs establish a process that is similar to thinning of voxels but operates on grid cells and solves the described problems. The input voxel object P is converted to its grid cell complex $X(P)$. The **grid cell skeleton** is a sub-complex $Y \subset X(P)$, which is a deformation retract of $X(P)$. Note, Y in general is not a grid cell complex of a voxel object because Y will not contain 3-cells. Grid cell skeletons share the same principal properties with voxel skeletons. They should also be medial, thin, and be able to reconstruct the original object.

The only reference sketching a similar idea as the one presented shortly is Kovalevsky (2001). But Kovalevsky computes skeletons in the grid cell model that are not necessarily cell complexes. As a consequence, more complex rules for deletion of cells are needed than in the solution presented below. Defining grid cell skeletons as sub-complexes links them more closely to the well known theory of cell complexes than Kovalevsky's discussion.

In a first step, we establish a simple rule for deletion of cells based on the Euler characteristic. After each deletion the remaining object shall be a cell complex and a deformation retract of the object before deletion. The Euler characteristic is a well known homotopy invariant, which remains unchanged under a homotopy. Deformation retractions are homotopy maps. Thus deformation retracts must have the same Euler characteristic as the original object. The **Euler characteristic** χ is computed from the number α_i of i-cells in a cell complex X as

$$\chi(X) = \sum_i (-1)^i \alpha_i$$

In the three-dimensional case this reduces to

$$\chi = \alpha_0 - \alpha_1 + \alpha_2 - \alpha_3$$

Deleting one cell alone will never keep χ invariant. Thus at least two cells need be deleted simultaneously. Pairs of cells contributing with different signs to χ must be chosen. The four combinations c^0/c^1, c^0/c^3, c^1/c^2, c^2/c^3 are the only candidates for deletion. We also required that a cell complex remains after deletion. Therefore, lower dimensional cells

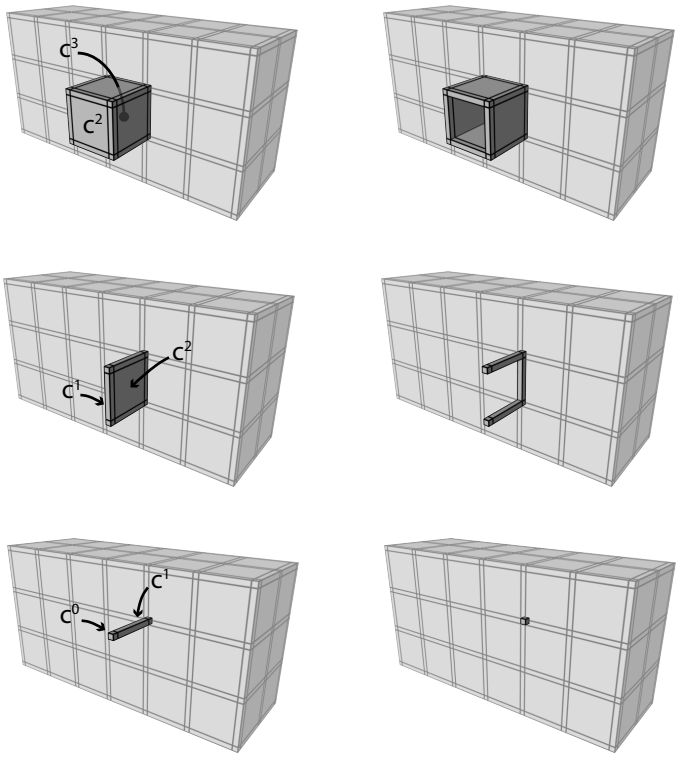
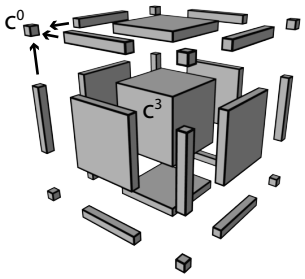


Figure 2.25: Deletable pairs of grid cells. The pairs of cells—c^0/c^1, c^1/c^2, c^2/c^3—indicated on the left are removed to form the deformation retract depicted at the right.

can only be removed together with all incident higher dimensional cells. Otherwise, the higher dimensional cells' boundary would become open. The removal of a c^0/c^3 pair is thus ruled out because the faces and edges of c^3 are also incident with c^0:

Figure 2.24: A 3-cell, a 0-cell and incident 1-, and 2-cells.

Definition of cell complex would be violated if c^0 was removed.



The rules for deletion of the pairs of cells can be concisely summarized in a single rule:

Proposition 1 *A pair of cells $\{c^d, c^{d+1}\}$ can be removed from a cell complex X such that the remaining sub-complex is a deformation retract of X if and only if c^d is incident with only c^{d+1} but no other cell.*

The deformation retraction is obvious for all three cases as illustrated in Figure 2.25.

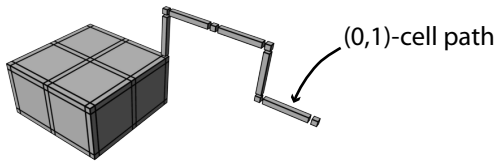
In analogy with the grid point model, a pair of cells $C = \{c^d, c^{d+1}\} \subset X$ is said to be a **simple pair of cells** in the grid cell complex X if its deletion results in a sub-complex

$Y = X \setminus C$, which is a deformation retract of X. A characterization of simple pairs is given by the proposition right above.

Grid cell thinning is sequentially removing simple pairs of grid cells until no further simple pair is found. Distance ordered homotopic thinning can be modified to compute grid cell skeletons in the following way. (the details are presented in Algorithm 3). Simple pairs near to the boundary with low dimensional cells will be removed with priority. The process starts at the boundary of the object with 2-cells incident to a background 3-cell. Tests are performed in parallel. Sub-iterations are not needed because the organization of the process by the dimension of the tested cells already cares for handling two-voxel-thick configurations. Deletion of a cell c^d influences tests for deletion at incident lower dimensional cells c^{d-1}. Thus all these cells are queued for each of the two cells of a deleted pair.

End-point preservation can also be integrated in the tests for deletion. The importance of an end-point is computed as the length of a (0,1)-cell path. A **(0,1)-cell path** is a sequence $\rho = (c^0_0, c^1_0, \ldots, c^0_{n-1}, c^1_{n-1}, c^0_n)$ such that every 0-cell is incident with the two neighboring 1-cells: $c^0_i \prec c^1_{i-1}$ and $c^0_i \prec c^1_i$; for example:

Figure 2.26: A (0,1)-cell path.



The length of the path is n, the number of 1-cells. A 0-cell c^0 is a **local cell end-point** in the grid cell complex X if c^0 is incident with exactly one $c^1 \in X$. The **importance of a cell end-point** is computed as the length of a (0,1)-cell path with the property that each c^0_i is a local cell end-point in the cell complex $X \setminus \{c^0_0, c^1_0, \ldots, c^0_{i-1}, c^1_{i-1}\}$ remaining after removal of the first $i - 1$ cell pairs of the path. Similar to the grid point model, a user specified length selects the cell end-points kept in the final skeleton. Note, short end-points naturally emerge after the removal of c^1/c^2-pairs (see Figure 2.25, middle right). Therefore, a sufficiently large threshold must be chosen.

Grid cell thinning computes skeletons similar to the results of thinning in the grid point model. Figure 4.2 illustrates results of grid cell thinning of an object built of rod-like elements. The skeletons are virtually undistinguishable from the grid point skeletons displayed in Figure 4.1—except for skeletons computed with a low threshold on the importance of cell end-points. End-point naturally emerge during grid cell thinning. As a consequence a larger threshold is needed to suppress side-branches. The behaviour for all types of architecture is comparable to thinning in the grid point model.

At the price of a higher storage cost, grid cell thinning resolves all configurations that the grid point models fails to resolve. It unequivocally represents junction configurations and is able to retract complex voxel configurations that would be locked in the grid point model.

Algorithm 3 Distance ordered grid cell thinning takes a grid cell complex X and a distance transformation d; and shrinks X to its homotopic grid cell skeleton.

```
1:  procedure SmoothDOGTT(X, d)
2:      for all c_x ∈ X do
3:          if dim(c_x) = 2 ∧ c_x ≺ c_y ∈ X̄ then                ▷ If c_x is boundary cell
4:              Q(d(c_x), dim(c_x)) ← Q(d(c_x), dim(c_x)) ∪ {c_x}   ▷ queue at distance d(c_x).
5:              l(c_x) ← QUEUED
6:          else
7:              l(c_x) ← UNQUEUED
8:          end if
9:      end for
10:     i ← 0  ;  d_max ← max(d(X))
11:     repeat
12:         i ← i + 1
13:         m ← false          ▷ If cell complex was modified restart loop over all distances
14:         for d ← (0,0), ..., (0,2), (1,0), ..., (d_max, 2) ∧ ¬m do   ▷ and dimensions.
15:             T ← Q(d)  ;  Q(d) ← ∅   ▷ Take all cells matching distance and dimension.
16:             for all c_t ∈ T do                              ▷ First, parallel check.
17:                 l(c_t) ← UNQUEUED
18:                 if ∃{c_t, c_u} deletable in X then
19:                     l(c_t) ← CANDIDATE
20:                     m ← true
21:                 end if
22:             end for
23:             if m then
24:                 for all c_t ∈ T do                          ▷ Sequential rechecking
25:                     if l(c_t) = CANDIDATE then              ▷ of marked candidates.
26:                         l(c_t) ← UNQUEUED
27:                         if ∃C = {c_t, c_u} deletable in X then
28:                             X ← X \ C
29:                             for all c_v ∈ {c_y ∈ X | c_y ≺ c_t ∨ c_y ≺ c_u} do   ▷ Incident cells
30:                                 if l(c_v) = UNQUEUED then   ▷ are queued if required.
31:                                     Q(d(c_v), dim(c_v)) ← Q(d(c_v), dim(c_v)) ∪ {c_v}
32:                                     l(c_v) ← QUEUED
33:                                 end if
34:                             end for
35:                         end if
36:                     end if
37:                 end for
38:             end if
39:         end for
40:     until ¬m                                   ▷ Continue until nothing changed.
41: end procedure
```

Implementation of thinning

Deletion tests are performed $O(n+m)$ times during thinning, with $n = |P|$ the number of voxels of the object P and m counting the number of multiple checks of one voxel, which depends on the topology of the voxel object. For large n an efficient implementation is crucial.

In the grid point model, tests for simplicity, for being part of a sub-iteration, and for being a local end-point can all be decided in the A_{26} neighborhood. A direct implementation of the discussed characterizations probably would be complex and inefficient. A solution could be to build a look-up table indexed by the neighboring configuration. By symmetry considerations, the size of the table might be reduced.

A much more elegant solution for evaluating local tests is based on binary decision diagrams, as proposed by Robert and Malandain (1998). Local tests can be considered boolean functions over 26 variables (or 27 variables if the central voxel itself is included). The possible outputs of such a function can be efficiently encoded as a rooted, directed, acyclic graph: the **binary decision diagram** (BDD). Any reference implementation may be used to compute the truth table from which optimized representations can automatically be generated. Here, the BuDDy library from http://buddy.sourceforge.net was used (see also Cohen, 2004). The resulting BDD is output as C source code and directly included in the implementation.

In the grid cell model all tests can easily be computed based on the integer coordinates of the grid cells. The characterization of deletable pairs was implemented as boolean tests without further optimization.

2.3.2 Two-Dimensional Skeletons: Geodesic Boundary Distance

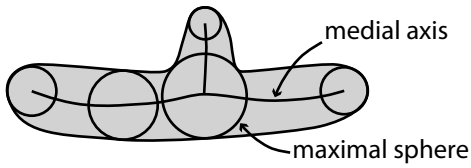
Thinning can be used to compute two-dimensional skeletons but it is susceptible to noise if preservation of geometric features based on local decisions is integrated. Homotopic skeletons of objects with cavities naturally contain two-dimensional voxel surfaces, which separate the enclosed background component from the outside. Thinning, as presented in the previous section, is able to compute them. But pure homotopy is rarely sufficient in applications. Geometric properties and reconstructability are often equally or more important. Those could be established by local tests for surface boundaries as in Borgefors et al. (1999); Jonker (2002, 2005). But these approaches have the disadvantage of being susceptible to noise.

Distance transformations, which encode global information of the voxel object, introduce a »global view« that may help to devise methods more robust to noise. Building on previous work, Malandain and Fernández-Vidal (1998) present a measure to characterize skeleton voxels based on the distance transformation. They propose to reconstruct homotopy in a second step by homotopic thinning. Costa (1999) proposes a measure based on the geodesic distance along the boundary of the object. Ogniewicz and Kübler (1995) use a similar measure to compute hierarchical Voronoi diagrams in two dimensions. Dey and Sun (2006) introduce the »medial geodesic function« to define and compute curve skeletons in three dimensions, which is is similar in spirit to the

approach introduced below. The guiding idea in all cases is to include a global view into the characterization of skeleton voxels. Distance transformations provide this view.

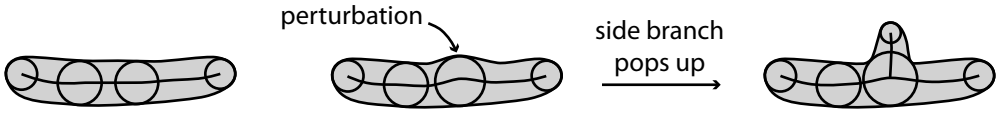
The medial axis is a mathematical object that is often used as a starting point for defining skeletons. In the continuous view $\mathbb{R}^3$, the medial axis is defined by the loci of maximal inscribed spheres touching the boundary of the object at two or more points. The set of nearest point in the background is denoted as $\Gamma(x) = \{y \in \bar{P} \mid d(x,y) = d(x,\bar{P})\}$. The **medial axis** M is now defined as all points with at least two nearest background points, that is $M = \{x \in P \mid |\Gamma(x)| \geq 2\}$:

Figure 2.27: Object, medial axis, and maximal spheres.



It is well known that the medial axis is unstable under perturbation of the boundary (see Attali et al. (2004) for a detailed review):

Figure 2.28: Instability of the medial axis under boundary pertubations.

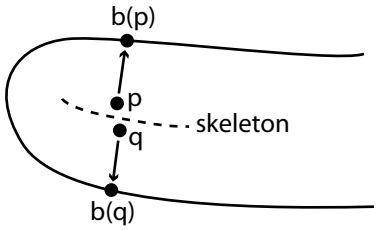


This property inhibits a direct use of the continuous medial axis to define skeletons for real-world voxel objects as their boundary always contains perturbations.

The discussion below introduces a graded representation of the medial axis, which associates an importance measure with every point. In a first step, this idea is developed in the continuous view. In a second step, a method is described operating on voxel grids, which follows the lines of the continuous view. The importance measure allows to select voxels skeletons stable under small changes of the boundary.

The basic idea is to use the nearest background points of a pair of points to decide if they are separated by the skeleton:

Figure 2.29: Two points and their nearest boundary points.

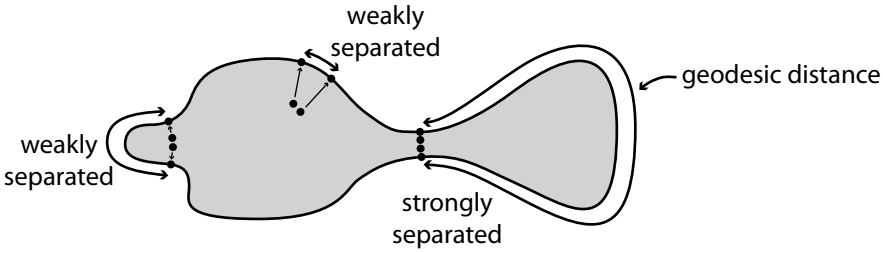


Assume a distance transformation that is able to provide not only the value to the nearest background point but also the coordinates of this background point. It is not uniquely defined for points on the medial axis as two (or more) background points are at the same distance. One of them is chosen in these cases. Analyzing the nearest background points $b(p)$ and $b(q)$ of a pair of neighboring points p and q reveals if they are separated by the

skeleton surface. Loosely spoken, if the points are on two distinct sides of the object, they are separated. This decision can be based on the location of $p, q, b(p)$ and $b(q)$. Malandain and Fernández-Vidal (1998) propose to analyze the distance of $b(p)$ and $b(q)$ and the angle formed by $b(p), p \approx q, b(q)$. Near to the boundary or at small structures the proposed measure is susceptible to noise. Ogniewicz and Kübler (1995) and similarly Costa (1999) integrate measures based on the shape of the object's boundary.

The idea developed here, which is also presented in Prohaska and Hege (2002), is to measure how much the object separates two background points by the geodesic distance along the boundary. The idea is illustrated below:

Figure 2.30: Degree of separation measured by geodesic distance.

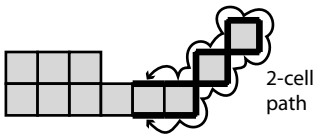


Small variations of the surface have little influence on the geodesic distance. This gives hope that a measure can be derived, which is robust to noise. A more precise definition in the continuous view is omitted. Instead we directly skip to voxel representations.

The geodesic distance of the nearest background points shall be computed in the voxel representation. For a binary object P the **nearest background point transformation** $b : P \to \bar{P}$ maps each voxel to the nearest background voxel (by a chamfer path). The geodesic distance along the boundary can be approximated by paths in the one-voxel-thick background layer $L = \{q \in \bar{P} \mid q \text{ is 26-adjacent to } P\}$. At convex parts of the object, paths in L give a good approximation of the geodesic distance. At concave parts a path may underestimate the geodesic distance by »taking a short-cut« as discussed below.

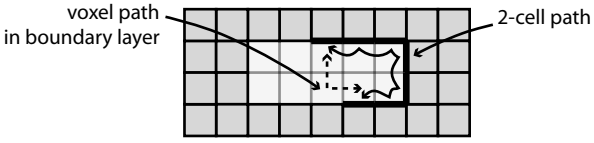
Difficulties may arise with the details of estimating the geodesic distance. The grid cell model could describe the boundary of the object exactly by 2-cells incident with an object and a background voxel:

Figure 2.31: 2-cell path along the boundary of a voxel object.



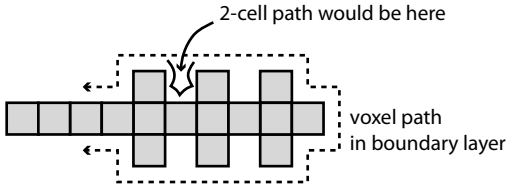
Computing a geodesic distance would require definitions of adjacency and paths in sets of 2-cells. In the following example this would be a real advantage:

Figure 2.32: Short cutting voxel path.



Here, the background boundary layer fails to approximate the geodesic distance because the two boundaries can be short cut by a »straight« voxel path. But we are only interested in cases where the object separates the two boundaries, thus connecting them along a straight path is not possible. Nonetheless, propagation in the boundary background voxel layer may yield different results from propagation in the 2-cell boundary at concave parts of the object:

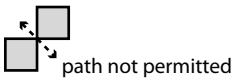
Figure 2.33: Another short cutting voxel path.



Such deviations from the correct value are ignored in the following. You may assume the resolution of the binary image is high enough such that at least one background voxel separates concave parts and the paths in L cannot take a short-cut. If this is not true for the input data computing paths in sets of boundary 2-cells should be considered.

Putting the discussed difficulties aside, the geodesic distance may be approximated in a one-voxel-thick background layer. The **geodesic distance** $g : L \times L \to \mathbb{N}$ of two background points p and q in the background layer L is given by the length of the shortest 6-path connecting them in L. If no path exists, the distance is ∞ by definition. Note, we need to respect the correct background adjacency relation (6-adjacency) to correctly separate two sides of the object:

Figure 2.34: Forbidden diagonal background path.



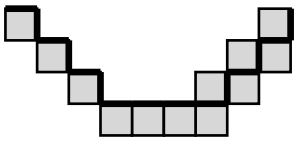
The characterization of a skeleton is now given by a threshold on the geodesic distance. Here the grid cell model proves superior to the grid point model because it allows to explicitly represent two-dimensional structures. A value is associated with every 2-cell of the object's grid cell complex. Cells above a chosen threshold belong to the skeleton. Two points of the continuous case are associated with a pair of 6-neighbors p, q. The geodesic distance between the two nearest background points is associated with the 2-cell $c^2_{\{p,q\}}$ incident with both voxels. A threshold selects the skeleton 2-cells. In summary:

- For each voxel $p \in P$ the nearest background voxel is given by $b(p)$.
- For two voxels p, q in the background layer $L = \{q \in \bar{P} \mid q \text{ 26-adjacent to } P\}$, their distance $g(p, q)$ is given by the length of the shortest 6-path connecting them in L.
- The skeleton importance of a 2-cell $c^2_{\{p,q\}}$ is given by $g(b(p), b(q))$.
- All 2-cells above the threshold t together with their lower dimensional neighboring cells form the skeleton Y_t.

The selected 2-cells have no one-to-one correspondence to voxels. Nonetheless, a representation of the skeleton in the grid point model can be achieved as follows. Each

2-cell $c^2_{\{p,q\}}$ has at least one incident 3-cell contained in the object's grid cell complex. If only one 3-cell, say $c^3_{\{p\}}$, is incident the grid point p will be included in the grid point skeleton. If two 3-cells $c^3_{\{p\}}$ and $c^3_{\{q\}}$ are incident, only the smaller grid point $p < q$ will be included in the grid point skeleton. $p < q$ is defined by component-wise comparison starting with the lowest index. Unfortunately the selected voxels do not necessarily form a thin set. They often contain non-simple voxels that could be removed without loosing geometric information:

Figure 2.35: Representing grid cell surfaces by voxels.



The left »staircase« results in a thin set but not the right one. A post-processing step can be applied removing all simple voxels while preserving geometry (see Borgefors et al., 1999).

Following the above rules implicitly allows to perform all computations in the grid point model. The importance of a voxel is computed

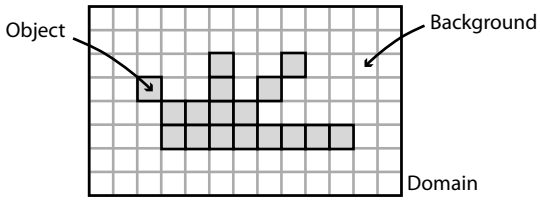
- by considering the three 6-neighbors with lower coordinates;
- by considering each of the three other 6-neighbors only if it is a background voxel;
- and taking the maximum of the geodesic distances.

Similar to thinning, the utility of the geodesic distance based skeletons depends on the architecture of the original object. Figure 4.5 illustrates skeletons of plate-like structures. The threshold on the geodesic distance selects skeletons that are stable to noise. Note however, in contrast to thinning homotopy equivalence is not guaranteed. For example the rod-like parts in Figure 4.7 are not stable with increasing thresholds. The skeletons in row two and three from the top fail to capture the rod-like parts. Thinning was used to solve this deficiency in the bottom two rows (see next section for more details). Sphere-like parts cause spiky sheets in the skeleton as depicted in Figure 4.6. Though topology is in general not preserved, cavities are a topological feature that is retained in all geodesic distance based skeletons. Figure 4.8 illustrates an example with plate-like structures attached to a hollow sphere. In this case the structure of the object is well reflected by the skeleton.

Implementation

A naive implementation of the geodesic distance based skeleton would require runtime quadratic with the input size. The next paragraphs give some details on how to efficiently implement the proposed measure. Assume a voxel object P with $k = |P|$ foreground voxels. P is stored as an array representing a sub-domain $D \subset \mathbb{Z}^3$ containing $n = |P|+|\bar{P}|$ voxels. The sub-domain is assumed to be sufficiently large, so that background voxels $\bar{P}$ surrounds P everywhere:

Figure 2.36: Object which does not touch boundary of computation domain.

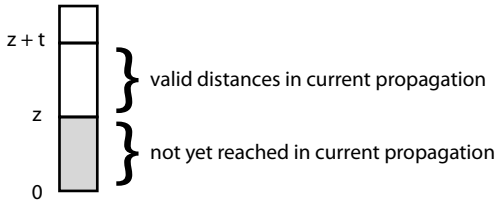


The nearest background voxel transformation is computed with chamfer propagation in $O(n)$ time. A naive implementation would now locate the two nearest background voxels for each of the $O(3k)$ 2-cells and compute the geodesic distance. In the worst case this would require a propagation through the complete background layer by Verwer's bucked sorted shortest path algorithm (Verwer et al., 1989), which may be of size $O(k)$ as well. Hence the overall runtime would be $O(k^2)$. Input data in our case is usually large and $O(k^2)$ is too slow.

The runtime can be reduced when limiting the distance of propagation and assuming well formed input data. A first general consideration is that we know an upper bound on the distances $g(p, q)$, which allows to reduce the complexity. Values above the importance threshold t need not be further differentiated. Thus we may restrict the propagation to a subset of L of worst case size $O(t^3)$ resulting in $O(kt^3)$ overall runtime. A practical value of t is 100, thus $t^3 = 10^6$, which still is a huge constant. Typically the background layer L forms a two-dimensional surface and not a three-dimensional volume. Thus propagation up to the maximum value t in such »well formed« input takes only t^2 steps. Many object voxel pairs map to boundary points located near each other. In these cases propagation will terminate even earlier. Voxels with a high importance value make up only a fraction of the k object voxels (approximately k divided by the mean radius $\bar{r}$ of maximal spheres). Hence, the typical runtime is of order $kt^2/\bar{r}$, which is linear in input size, however with a large but practical constant $t^2/\bar{r}$.

Unnecessary initialization of the arrays used to store results must be avoided. A naive implementation of the propagation of $g(p, q)$ starts with initialization of an array representing the background layer. The array has size $O(k)$. Computing $g(p, q)$ would be bound by the initialization of this array, and the above arguments would be invalid. A carefully chosen encoding scheme for the current distance transformation rooted at p avoids the initialization. For each new point p the geodesic distance is encoded in a different integer range $[z, z + t]$:

Figure 2.37: Encoding of distance values.



The range is advanced for every new base point. The array is only initialized once at the beginning of the computation, or on overflows of the integer type used for encoding the distances.

2.3.3 Mixed Dimensional Skeletons

A fusion of thinning and the skeleton characterization based on geodesic distances allows to extract voxel and grid cell skeletons containing rod-like and plate-like parts. In a first step a plate-like skeleton is computed. Its elements are locked during a subsequent thinning step guided by a distance transformation. The resulting skeleton faithfully represents one and two-dimensional parts of the object. The lower two rows in Figure 4.7 depict skeletons computed in this way. Still, the input must not contain volumetric, sphere-like parts as the discussed methods would fail to represent them properly.

The geodesic distance along the boundary can also guide thinning directly to compute one-dimensional skeletons of objects with plate-like parts. The thresholding step is skipped but instead the geodesic distance values are used as the guiding distance in SmoothDOHT. The geodesic distance increases towards the center of plate-like structures. Thinning is thus guided towards the center of these structures resulting in a one-dimensional skeleton centered within the plates. Figure 4.9 compares such skeletons with the results of standard distance ordered thinning. Although differences are minor they are clearly visible. The geodesic distance based skeletons are centered within the plates while the skeletons computed by standard distance ordered thinning are located at the thickest parts of the structure.

The overall computation time may increase dramatically compared to the thresholding method depending on the structure of the object. When using the geodesic distance to guide thinning no threshold is used and, therefore, propagation for computing the geodesic distance can not be terminated early.

Summary

- Skeletons provide a low dimensional representation capturing the shape of an object. Desirable properties of skeletons are homotopy equivalence with the object, thinness, medialness, and reconstructability.
- Skeletons can be computed by thinning and boundary propagation, medialness function based methods, wave front propagation and shock detection, or geometric methods.
- The grid point model represents voxel objects as points connected by an adjacency graph.
- The grid cell model represents voxel objects as a cell complex formed by zero-, one-, two-, and three-dimensional cells. It needs eight times the storage space of the grid point model. The grid cell model is superior to the grid point model for algorithms that need detailed control of topology.
- Thinning erodes the boundary of a voxel object by local operations to compute a deformation retract of the object in linear time. Erosion is guided by a distance transformation encoding the distance to the nearest background voxel.

- End-points of tubular structures can be retained during thinning. The sensitivity of the end-point detection needs to be controlled to avoid spurious branches.
- Thinning was extended to the grid cell model. A characterization of pairs of grid cells that can be deleted during thinning is simpler than characterizations of voxels that can be deleted in the grid point model.
- A measure indicating skeleton plates centered in the object was established based on the geodesic distance along the object boundary of two nearest boundary voxels of a pair of voxels. Skeletons are either selected by a threshold or a combination with thinning is used. Computing the measure requires runtime linear in the number of voxels but including a large constant given by the square of the threshold.
- Geodesic distance based quantities seem to be well suited to robustly identify one- and two-dimensional skeletons. A sound theoretical foundation of this observation is yet missing.

3 Piecewise Linear Geometric Representation of Voxel Skeletons

This chapter's primary goal is to generate piecewise linear geometry for voxel and grid cell skeletons. Such skeletons, as computed by the techniques discussed in the previous chapter, are the input to the methods presented now. The locations of voxel and grid cell skeletons are restricted to the underlying integer sets of the binary images used to represent them. In the process of generating piecewise linear geometry the discrete positions are smoothed and approximations of continuous one- and two-dimensional geometries are generated, which are represented as line sets and triangle meshes. These are particularly suitable for rendering on contemporary graphics hardware.

A secondary goal is to convert skeletons to more abstract representations. In the case of pure one-dimensional skeletons, for example, abandoning the geometric locations gives rise to an interpretation as graphs of junction nodes connected by edges, without direct geometric meaning.

Several authors discuss approaches to establish a notion of a surface completely defined in the grid point model with properties comparable to surfaces in the continuous domain. All the following approaches' goal is to establish methods based solely on grid points after transferring notions from the continuous domain. Morgenthaler and Rosenfeld (1981) introduce digital surfaces in the grid point model. They describe conditions on a set of voxels for having the Jordan property, that is the property to separate the background in two components. Couprie and Bertrand (1998) use an extended definition, which includes the case of so-called Morgenthaler surfaces. Based on a framework for digital topology presented by Ayala et al. (2002), Ciria et al. (2004, 2005) propose to transfer topological properties from continuous analogs to characterize digital surfaces.

In contrast, the goal of this chapter is to interpret voxel representations of surfaces with boundaries as continuous geometries and give piecewise linear approximations thereof. The voxel surfaces with boundaries introduced below *do not* separate the domain into disjoint regions as level sets of smooth functions do. Note, this task is different from constructing iso-surfaces of (continuous) scalar functions defined by a three-dimensional image. Iso-surfaces represent interfaces between regions below and above a threshold and can, for example, be approximated by the Marching Cubes algorithm (Lorensen and Cline, 1987) or similar algorithms with more detailed topology control (Lachaud, 2000).

Here, we need to deal with non-closed surfaces. Wang et al. (2005) discuss how to reconstruct non-manifold surfaces from point clouds. As an intermediate representation they use a voxel representation. They reconstruct geometry by building a connectivity graph and triangulating loops of this graph as presented in Azernikov et al. (2003) and Azernikov and Fischer (2004). Their idea is similar to our idea in Prohaska and Hege

(2002), which is presented in an extended version below. Building on our work, Fujimori et al. (2005) propose to locally label voxels in the neighborhood of a skeleton voxel and define an interface surface which is triangulated by locally applying the Marching Cubes algorithm.

For voxel input data, utilizing the voxel representations can be superior to geometric methods based on the Voronoi diagram, which are an alternative for constructing a geometric representation of a skeleton. Geometric methods start from a point sampled representation of the boundary and compute a geometric skeleton. They seem to be a natural choice if a geometric representation is requested. But as briefly discussed in Section 2.1 geometric methods are more expensive in terms of computation time and required storage than voxel based algorithms. Hence combining voxel based methods and geometric methods may be beneficial.

The methods presented below provide a link from a voxel representation to a geometric representation of a skeleton. Results of voxel based skeletonization is accepted as input and can be converted to a geometric representation. The resulting geometry may either be post-processed or used directly for rendering.

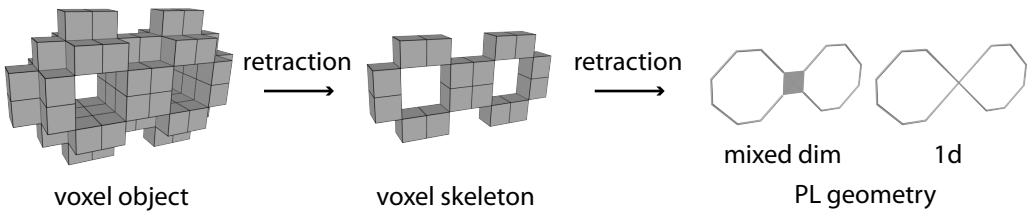
We will discuss geometry generation separately for voxel skeletons and for grid cell skeletons. Voxel skeletons are more challenging because as volumetric objects they fail to capture lower dimensions faithfully in some cases. Grid cell skeletons on the other hand do represent all dimensions explicitly and geometry generation turns out to be straight forward.

3.1 Geometry of Voxel Skeletons

We start with a voxel object in the grid point model built of one- and two-dimensional parts and construct topologically simple, low-dimensional, piecewise linear geometry per local neighborhood.

The general idea is to retract a voxel configuration to geometry:

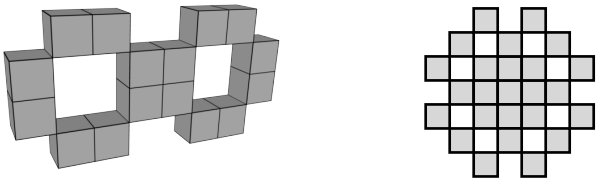
Figure 3.1: Retracting voxel objects to voxel skeletons to PL geometry.



This can be seen as an extension of the retraction used to compute voxel skeletons. The retraction may either stop if a two-dimensional representation is reached, or it may continue to a one-dimensional result. The desired dimension may be chosen depending on the needs of the application.

The conversion to one- and two-dimensional geometry should gracefully handle complex voxel configurations. As noted earlier, such configuration may be encountered in voxel skeletons:

Figure 3.2: Complex voxel configurations.



The interpretation of voxel skeletons may be unclear at junctions (left) and for some configurations, thinning stops before all interior voxels are deleted yielding skeletons containing voxels without background neighbor (right). The desired dimension of the geometry will be specified to enforce a certain geometric interpretation.

The result shall be represented as piecewise linear geometry of dimension up to two, embedded in three dimensions, that is a simplicial complex. Instead of giving a general definition we restrict the discussion to the needed dimensions. A **simplicial complex** $C = (V, E, T)$ is a tuple of three sets describing vertices, edges, and triangles. $V = \{v_i \in \mathbb{R}^3 \mid i = 1 \dots n\}$ contains the vertices of the geometry. $E = \{e_{ij} = (v_i, v_j)\} \subset V^2$ describes edges connecting two vertices each. $T = \{t_{ijk} = (v_i, v_j, v_k)\} \subset V^3$ holds the triangles formed by three vertices each. The edges of each triangle must also be contained in E, while E may contain edges that are not incident with a triangle. Examples of simplicial complexes are triangle meshes and line sets with piecewise straight line segments. Both map well to contemporary graphics hardware for rendering.

A local construction solely based on a neighborhood of a voxel shall be used. Similar to the Marching Cubes algorithm (Lorensen and Cline, 1987), local constructions could be stored in a look-up table and stitched together easily. This makes computations efficient. Restricting the construction to a local neighborhood has also a more fundamental reason. The principal location is already represented by the voxel skeleton. Thus the geometric representation should only locally deviate from the structure of the voxel skeleton. This can be ensured by a local construction.

To represent the topology of the voxel skeleton, the construction shall yield a deformation retract of the voxel configuration. Artificial loops, handles or cavities must be avoided. Again the voxel skeleton carries all principal information. The geometric representation must respect the topology of the voxel skeleton, which is ensured by a retraction.

A natural interpretation is also expected to be smooth and the resulting geometry should be a manifold at as many points as possible. The manifold parts end in boundaries or are stitched at junctions. Lines and surfaces may meet at junctions resulting in line-line, line-surface, and surface-surface junctions.

The method presented below classifies the manifold type of voxels in a first step and generates geometry in a second step. Voxels are classified as *0-manifold* (points), *1-manifold*, *1-boundary*, *2-manifold*, *2-boundary* or *junction*. The classification is based on a cell complex in the 26-neighborhood of each voxel. Before executing the second step the desired dimension of the final result needs to be chosen. Either one-dimensional line sets can be enforced or a mixed 1d/2d representation can be allowed. Geometry is generated per dual cube and stitched together to form a global representation.

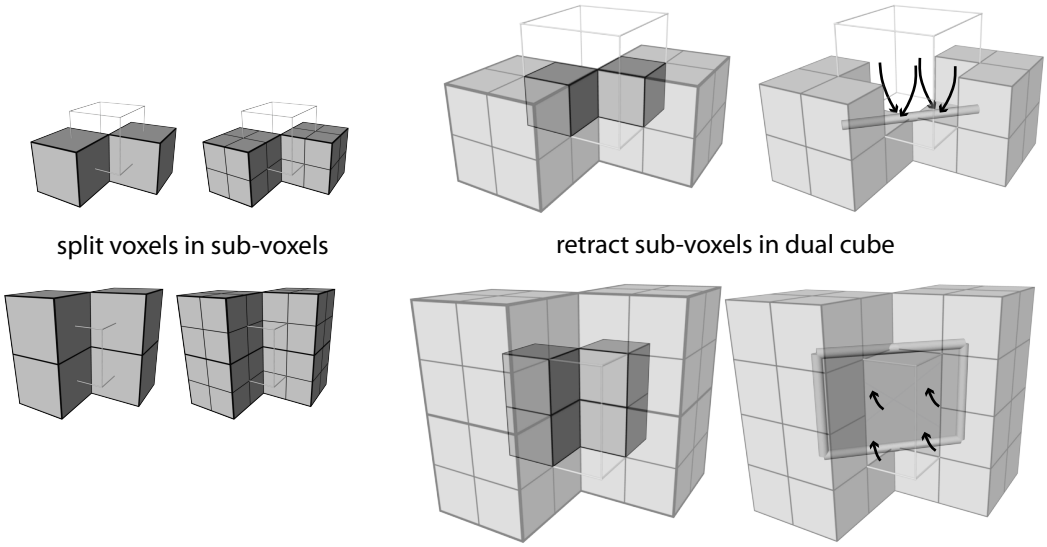


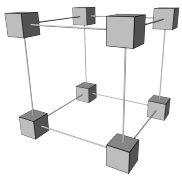
Figure 3.4: Two dual cube configurations and their deformation retracts. Voxel configurations; sub-voxels; sub-voxels contained in the dual cube; and the deformation retract are illustrated from left to right. In the top row, two voxels share a common edge; in the bottom row, four voxels share a common corner.

3.1.1 Manifold Type of Voxels Based on a Local Cell Complex

We will now construct a cell complex that is based on voxels as its 0-cells (vertices), while postponing the explicit geometric representation to the next section. No further vertices beyond the voxel locations are added at this point. Higher dimensional cells are described purely combinatorial by sets of lower dimensional cells. For example 1-cells (edges) are described as pairs of 0-cells and 2-cells (faces) are described as sets of 1-cells. Later, we will handle junction voxels and add more vertices to build a simplicial complex.

The construction is described in a **dual cube** formed by eight 26-adjacent voxels:

Figure 3.3: Dual cube.



All possible voxel configurations of a dual cube can be retracted by splitting voxels into eight sub-voxels. The sub-voxels are then collapsed to lower-dimensional cells. Figure 3.4 illustrates two cases. By inspecting all 256 cases, one can verify that the following construction yields a deformation retract of the sub-voxels in the dual cube for all cases.

The resulting cell complex provides sufficient information to investigate dimension and

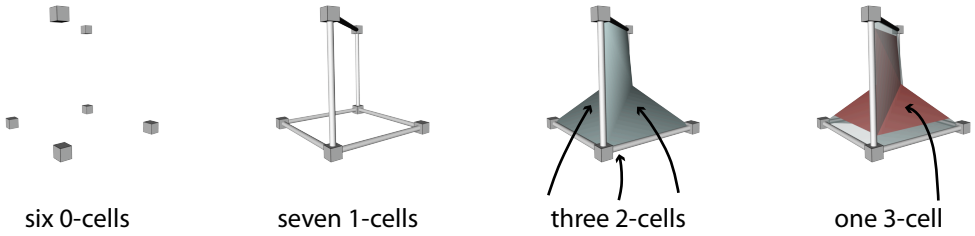


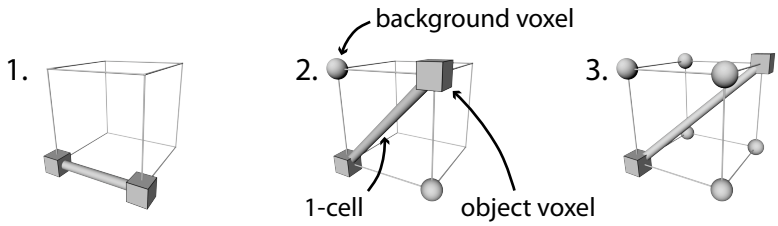
Figure 3.5: Local construction of a cell complex. Six 0-cells are connected by seven 1-cells yielding an Euler number of $\chi = -1$; thus at least two 2-cells are needed. Symmetry requires to add three of them. Now the Euler number computes to $\chi = 2$; thus a 3-cell is added. The cell complex is a deformation retract of the original six voxels. Note, this is an illustration of the combinatorial description of the cells. The details of the geometry have no further meaning beyond illustrative purposes.

manifoldness at each voxel. By analyzing the combinations of voxels used for describing the n-cells we can derive the dimension of the skeleton at each voxel and learn if it is manifold or not.

Instead of retracting the voxels we build the result starting from the vertices and attach higher dimensional cells as needed (see Figure 3.5). First, a 0-cell is added for each object voxel. 1-cells are attached until all 0-cells are connected. The rules when to attach a 1-cell are chosen to guarantee continuous transitions to neighboring dual cubes. Note, edges of a dual cube are shared by overall four dual cubes; faces by two dual cubes. Hence rules for adding a 1-cell edge or face of a dual cube must only depend on 0-cells located on the same edge or face.

The following rules are used to attach 1-cells:

Figure 3.6: Rules for adding 1-cells.



1. For each 6-adjacent pair of neighbors, a 1-cell is attached.

2. For two 18-adjacent neighbors on a face of the dual cube, which are not 6-adjacent, a 1-cell is attached only if the two other vertices of the face are background voxels.

3. For two 26-adjacent neighbors, which are not 18-adjacent, a 1-cell is attached only if they are the only two object voxels in the dual cube.

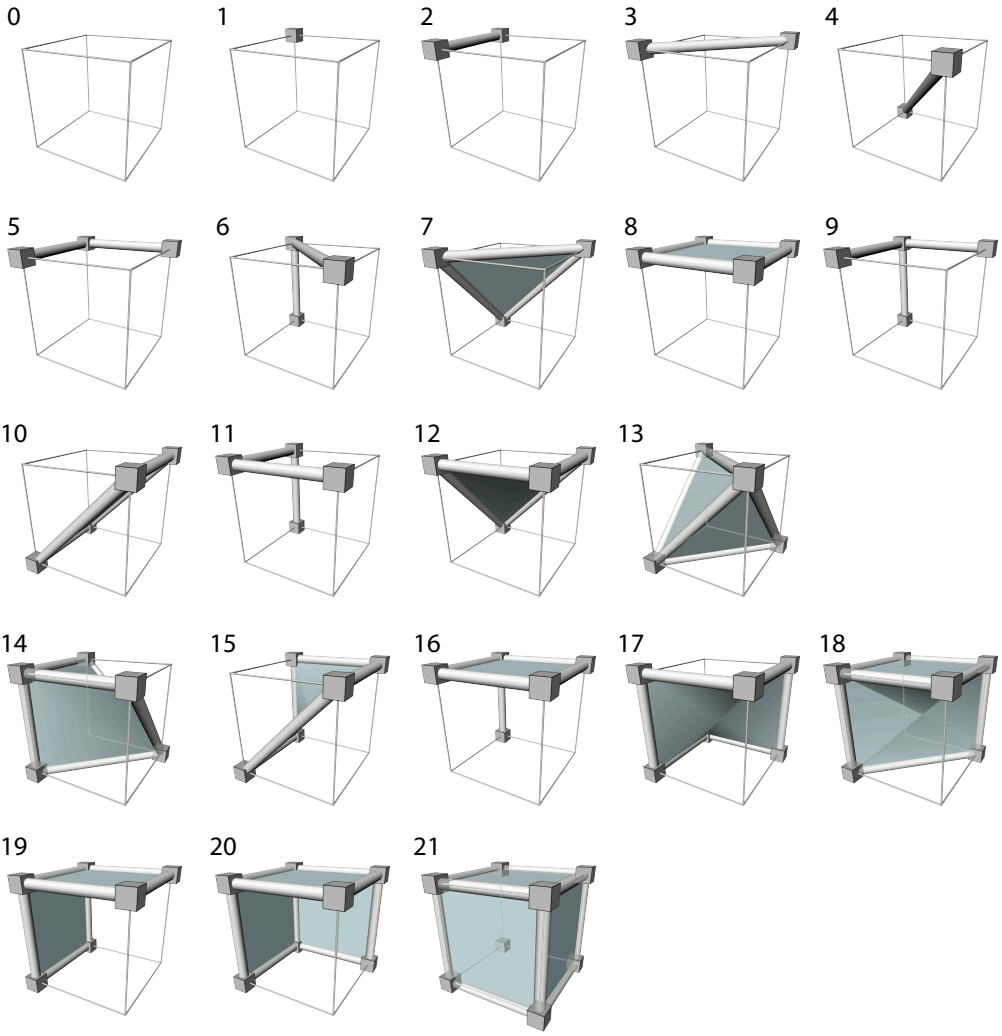


Figure 3.7: The 22 dual cell configurations and associated cell complexes. Cases 0–13 are numbered as in the original Marching Cubes paper. Cases 14–21 must be distinguished because color permutations, different from the Marching Cubes algorithm, are not a symmetry operation.

The 1-cells may form cycles and 2-cells are added filling the cycles to form a surface. Starting with the shortest cycles, 2-cells are attached until all edges are boundaries of a 2-cell. If several cycles have the same length, for symmetry considerations, a 2-cell is attached for each of them. The Euler characteristic χ of the resulting complex determines if a 3-cell is needed. If the complex has $\chi = 1$ it is homotopy equivalent to a point and no 3-cell is added. A 3-cell is attached if $\chi = 2$. Other cases do not occur.

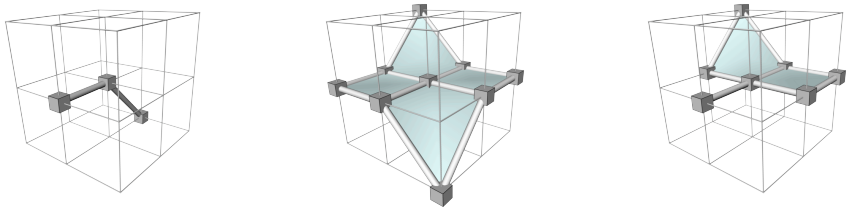
Symmetry operations allow to group the configuration into the 22 classes presented in Figure 3.7. Banks and Linton (2003) describe how to apply group theory to count these classes. In the case of voxel skeletons, the corners of a cube can be colored with two different colors (background, foreground) and spatial symmetry operations, but no

color symmetry operations, are allowed (see also Banks et al. (2004) for an extended version of the paper). In contrast to the Marching Cubes algorithm, permutations of the colors would break symmetry. The different classes are illustrated in Figure 3.7 by one representative member of each group.

Four cases still contain three-dimensional parts and should be retracted further. Cases 13, 14, and 18 contain volumes that can easily be collapsed. This is possible because the dual cube contains boundary surfaces at which the retraction may continue. The volume in case 21 can not be collapsed because no face incident with the background is available in the dual cube to start the retraction from. We postpone this discussion and concentrate on a combinatorial construction based purely on the original grid points and the shortest edges required to maintain connectivity as described above. Cases 13, 14, and 18 are further discussed in Section 3.1.3.

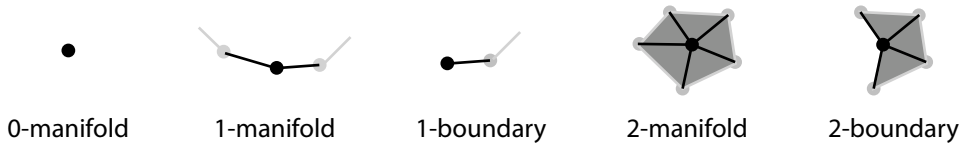
The per-dual-cube construction is the building block of a global construction, which we now analyze in a 26-neighborhood. Eight dual cubes' constructions can be glued by identifying common voxels. The higher dimensional cells can be labeled by their incident 0-cells, which are directly associated with voxels. The result forms a cell complex surrounding a voxel in its 26-neighborhood. Some cases are illustrated below:

Figure 3.8: Examples of cell complexes in a 26-neighborhood.



The central voxel can be classified based on the local neighborhood of the associated 0-cell in the following way:

Figure 3.9: Manifold types of 0-cells in a cell complex.



- If the 0-cell is not incident with a higher dimensional cell, the voxel is **0-manifold**.
- if the 0-cell is incident only with 1-cells and no higher dimensional cells:
  - if it is incident with two 1-cells, the voxel is **1-manifold**;
  - if it is incident with one 1-cell, the voxel is a **1-boundary**.
- If the 0-cell is incident with 2-cells but no 3-cell:
  - if all incident 1-cells are incident with exactly two 2-cell, the voxel is **2-manifold**

  - if two incident 1-cells are incident with one 2-cell each and the remaining 1-cells are incident with two 2-cells each, the voxel is a **2-boundary**.

- Otherwise, the voxel is a **junction**, which is further classified if the 0-cell is incident with at least one 1-cell that is not incident with a higher dimensional cell. In this case the junction is called a **1-x-junction**.

At this point we know for each voxel the dimension and manifoldness of a geometric representation of the voxel skeleton. For example, we can decide that a voxel represents a 1-manifold part, which will be converted to a line. Or we can decide, that a voxel represents a 2-boundary that will be converted to a surface boundary in the simplicial complex.

At some voxels the result may contradict our expectations. We may, for example, find two-dimensional parts in voxel skeletons that we expected to be purely one-dimensional. This can be caused by locked configurations as described in Chapter 2. And we may find clusters of junction voxels. For junction voxels the details of the geometric representation are unclear at this point.

Implementation

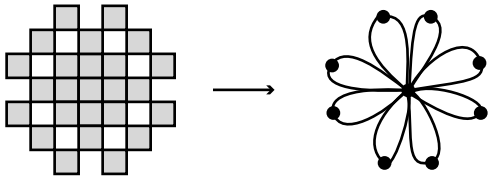
Even in a larger configuration, the classification of each point is only based upon its 26-neighborhood and can be efficiently encoded as a binary decision diagram. A naive, potentially slow implementation of the above construction can be used to answer questions like »is this voxel 0-manifold?« based on the voxel's 26-neighborhood. The answers are fed to a library computing a binary decision diagram encoding the truth table of the answers (see also Page 35). The binary decision diagram can then be converted to C source code and included in the implementation to efficiently classify voxels.

3.1.2 Line Sets and Graphs

This section's goal is to enforce a one-dimensional geometric representation of a voxel skeleton. For example based on prior knowledge about the object being analyzed one could know that the result must be one-dimensional. The following construction allows to enforce such an interpretation.

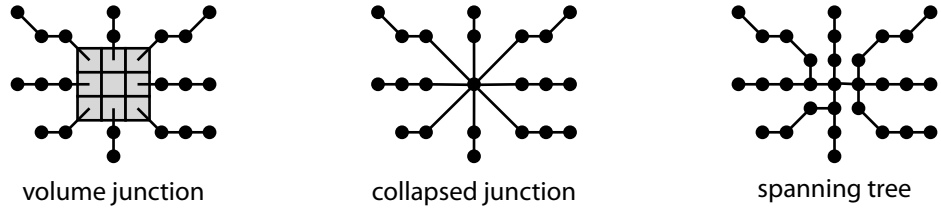
The classification of voxels allows to distinguish pure one-dimensional from other parts. 1-cells ending on both ends in 1-manifold or 1-boundary voxels are represented as edges connecting the centroids of the two voxels. Connected components of junction, 2-manifold, and 2-boundary voxels form a single 0-cell. All 1-cells ending in any of the connected component's voxels are attached to this 0-cell, yielding a graph:

Figure 3.10: Interpretation of a complex junction configuration.



A geometric realization can either present junctions as higher dimensional geometry and leave interpretation to the human viewer; or it may collapse voxels to a single point. Attached 1-cells are connected to this point either by straight lines; or by lines along voxel paths, which are found by choosing one voxel as the root, building a spanning tree of the adjacency graph in the connected component, and retaining only branches running to attached 1-cells:

Figure 3.11: Options for representing junctions.

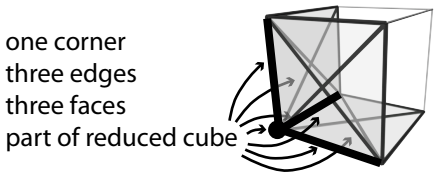


Only the second approach, collapsing branching regions to a single point, is discussed in more detail below.

But first all edges connecting two voxels are generated with help of a lookup-table. The configuration of object voxels in a dual cube determines the constructed 1-cells as discussed in the previous section. A lookup table of size 2^8 can store all possible results.[1] In a single scan over all dual cubes this table is used to build a graph representation of the object. 0-cells are the graph's nodes; 1-cells form its edges.

Because faces, edges and corners are shared between two, four and eight dual cubes, care must be taken to generate each edge exactly once. Assigning each point of space unequivocally to a single dual cube provides a solution. One corner with its three incident edges and faces together with the volume form a **reduced dual cube**:

Figure 3.12: Reduced dual cube.



The other boundary elements of a dual cube are part of a neighboring reduced dual cube. 1-cells are only stored in the lookup table if they are completely contained (except for their endpoints) in the reduced dual cube. For instance, an edge located on the top face of a dual cube would be rejected. It would be generated in the neighboring reduced dual cube to the top. Connectivity to the neighboring dual cubes is established by the common 0-cells. With each node of the constructed graph the type of the 0-cell is stored.

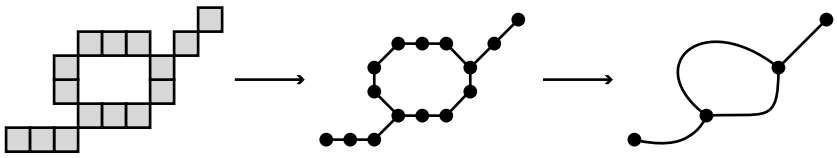
Non-1-manifold regions are now collapsed to a single point by computing the following equivalence relation: Two nodes are equivalent if they are both non-1-manifold and connected by an edge. Sets of equivalent nodes are replaced by a single node located at their centroid. Equivalent nodes can be computed by propagating along edges to non-1-manifold nodes. A scan over all nodes reveals all equivalence classes.

[1] Symmetries could be identified to reduce the number of cases. But the size of the table is already small.

Note, collapsing junction regions in the describe way may break homotopy equivalence if the junction nodes form a surface enclosing a cavity. This is a desired behaviour as we decided to enforce a one-dimensional interpretation of the voxel configuration.

The resulting graph can be further post-processed. Sequences of vertices incident with exactly two edges (1-manifold vertices) can be interpreted as the geometric realizations of a single edge of a graph with fewer vertices:

Figure 3.13: Voxels, graph, simplified graph.



Geometry of edges can be smoothed by filtering the positions of consecutive vertices. A gaussian weighted average over a number of neighboring vertices of the same edge yields good results. The number of vertices functions as a kernel size of a smoothing filter. The smoothness of the result can be controlled by tuning the kernel size and by applying multiple filtering passes.

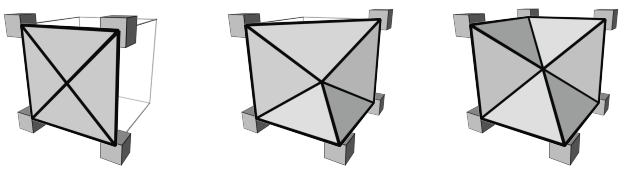
Any line rendering method is suitable for displaying the resulting geometry. Examples of existing methods are rendering lines as truncated cones (Hahn et al., 2001), which may be efficiently approximated on modern GPUs (Stoll et al., 2005); convolution surfaces (Oeltze and Preim, 2005); or illuminated lines (Zöckler et al., 1996; Mallo et al., 2005). All illustrations in this thesis were rendered using truncated cones. See for example Figure 4.1, that displays renderings of line skeletons generated from voxel skeletons. The close-up view illustrates a junction that was collapsed to a single point.

3.1.3 Surfaces

The next goal is to construct a triangulated surfaces from skeletons in grid point model. We no longer collapse 2-manifold voxels to a single point but represent them by surface patches. Starting from the characterization of voxels described on Page 49, all two-dimensional structures are triangulated and attached at junctions connecting them to other surface patches or lines.

The triangulation is built for each dual cube and stitched along the dual cube's faces. 2-cells are triangulated by adding the centroid of its vertices and adding triangles formed by the edges of the 2-cell and the centroid:

Figure 3.14: Triangulations of 2-cells.

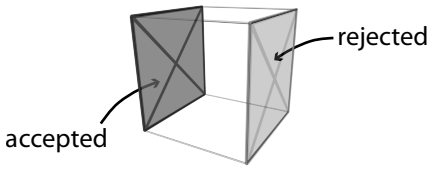


Identifying common vertices on the shared faces of the dual cubes established connectivity of the surface. The configuration of the vertices of a face completely

determines the resulting geometry on that face. Hence continuous stitching is straight forward. Identifying the vertices of the triangles across dual cubes allows to connect the triangles to form a connected mesh.

Reduced dual cubes ensure that each triangle is generated exactly once, as it was the case for line sets in the previous section. Triangles are only generated for 2-cells that are completely contained (except for their boundary) in the reduced dual cube:

Figure 3.15: Rejecting triangles based on the reduced dual cube.

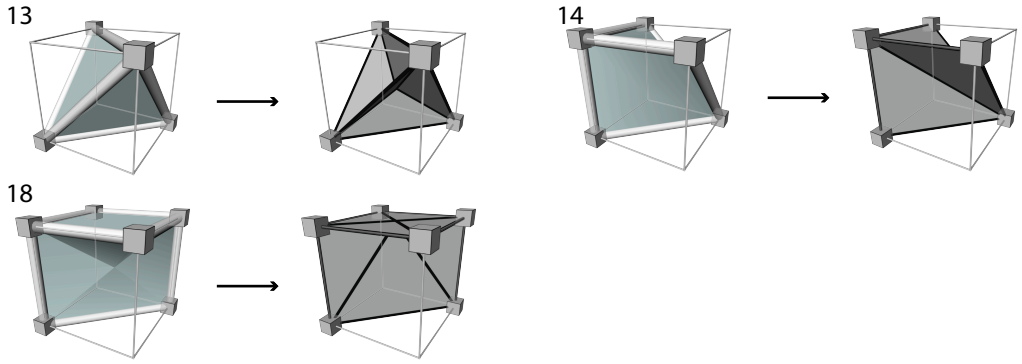


For example, a 2-cell consisting of the right face would be rejected. It would be triangulated in the neighboring dual cube.

2-cells can be classified according to their incident 0-cells as being manifold or not. If all incident 0-cells are 2-manifold or 2-boundary the cell is **all-manifold**; otherwise it is **not-all-manifold**. This classification carries over to the triangles. The type of the eight voxels in a dual cell completely determines the result. Because voxels can take three states (background; 2-manifold or 2-boundary; other) when highlighting manifold and non-manifold parts, a lookup table of size 3^8 is required to store all configurations of resulting all-manifold triangles and not-all-manifold triangles. A scan over all dual cubes builds the complete surface by referencing the lookup table.

Cases 13, 14, and 18 may be triangulated in an optimized way to avoid enclosed volumes. This can be achieved by changing the way edges are introduced. The modified triangulations are presented below:

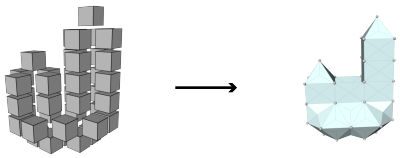
Figure 3.16: Improved triangulations of case 13, 14, 18.



Case 18 requires to deal with the vertex introduced on the face of the dual cube (in the center of the top face in the illustration). This vertex needs to be identified in the neighboring dual cube to establish connectivity of the triangle mesh, which can be achieved using a $2 \times 2 \times 2$ sub-divided voxel grid to store identifiers of the generated triangle vertices. The original grid points are located at even indicies in the subdivided grid. The additional vertices have two odd coordinates.

At this point we are able to convert voxel skeletons to a piecewise linear representation. For example one of the voxel configurations used earlier is triangulated as follows:

Figure 3.17: Triangulation of a complex voxel configuration.

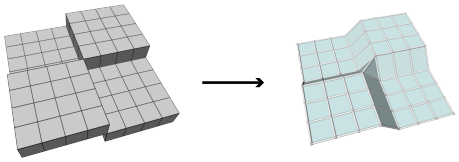


Note however, we failed to retract the solid configuration in case 21 to a lower dimension. The retraction fails in the dual cube because no boundary to the background is available. One might try to resolve the case in a larger neighborhood. But solid configurations can be infinitely large. So there will always be unresolved cases. Grid cells, as discussed in the next section, provide a more promising solution.

The classification of voxels discussed in Section 3.1.1 could be modified and based on the optimized triangulation. This would change the classification of some junction voxels. For example in case 18 vertices that have not been before may now be classified as manifold after collapsing the enclosed volume. The original classification overestimates the number of junction voxels but never falsely classifies voxels as manifold. Hence we can safely keep the original classification.

Self-folding voxel configurations are an open problem. The geometric realization represents self-folding as surfaces with non-manifold parts:

Figure 3.18: Triangulation of a »self-folding« voxel configuration.



The question remains open how to handle these cases more gracefully. A human viewer would consider the non-manifold parts being artificially introduced by the discrete nature of the representation and would probably expect a smooth manifold surface in the above example.

Post-processing, such as smoothing and triangle decimation, can be applied to the triangle mesh. See Bade et al. (2006) for a comparison of smoothing methods. Decimation algorithms, like for example presented by Garland and Heckbert (1997), can be used to reduce the triangle count. Vertices representing 1-x-junctions should be kept fixed during post-processing to preserve the location of attaching points of 1-manifold parts. Experiments revealed that the triangle mesh allows aggressive decimation of triangles while preserving a good approximation of the overall structure.

Note, the interpretation of a grid point configuration as a cell complex introduces 1-cells that are different from the standard (6-, 18-, 26-) adjacency relations. Connections to 18-neighbors and 26-neighbors are only conditionally added. In many cases, the resulting

connectivity graph is locally embeddable in two dimensions, which allows to construct a surface in theses cases. The graph *is not* one of the well known 6-, 18-, 26-adjacency relations but its edges are object dependent. Thus direct application of algorithms based on a standard adjacency relations fails. For example, after classifying voxels into different manifold types one could ask which 2-manifold voxels form a connected component such that they are represented as a single, connected manifold part in the geometric representation. One could hope to answer this question by applying region growing with one of the well known adjacency relations. But this approach fails because a data dependent adjacency relation would be needed, which would change with the local object configuration. Processing would be achievable but a lot of annoying details would need to be resolved. The question remains how to build a local interpretation as a piecewise linear surface with boundary of a 26-connected voxel skeleton while avoiding those difficulties. The heart of the problem is that a local sub-graph of the 26-adjacency relation is not planar (it can not be embedded in a plane) even for configurations that would naturally be interpreted locally as surfaces. Choosing the 6-adjacency relation for object voxels would be another way to avoid the problem. But 26-adjacency for the foreground is the far more common choice. Note, in the grid cell model similar problems do not arise.

3.2 Geometry of Grid Cell Skeletons

Converting grid cell skeletons to piecewise linear geometry is much simpler than converting grid point configurations. Grid cell skeletons are already represented as a cell complex. Hence the first step, prescribing an interpretation as a cell complex, is not needed.

The neighborhood of a cell is analyzed to reveal junctions and manifold parts. The rules classifying 0-cells (Page 49) are directly applicable. 1-cells can be classified similarly:

- If a 1-cell is not incident with any higher dimensional cell, the 1-cell is **1-manifold**.
- If a 1-cell is incident only with 2-cells but no 3-cells:

  if it is incident with exactly two 2-cells, the 1-cell is **2-manifold**;

  if it is incident with exactly one 2-cell, the 1-cell is **2-boundary**.
- Otherwise, a 1-cell is a **junction**.

For 2-cells the following rules apply:

- If a 2-cell is not incident with a 3-cell, the 2-cell is **2-manifold**.
- Otherwise, the 2-cell is a **junction**.

Note, 3-cells are not contained in grid cell skeletons, thus all 2-cells are 2-manifold. In the same way as in the grid cell model, 1-manifold 1-cells can be further classified by their incident 0-cells. If all incident 0-cells are 1-manifold or 1-boundary, the 1-cell is

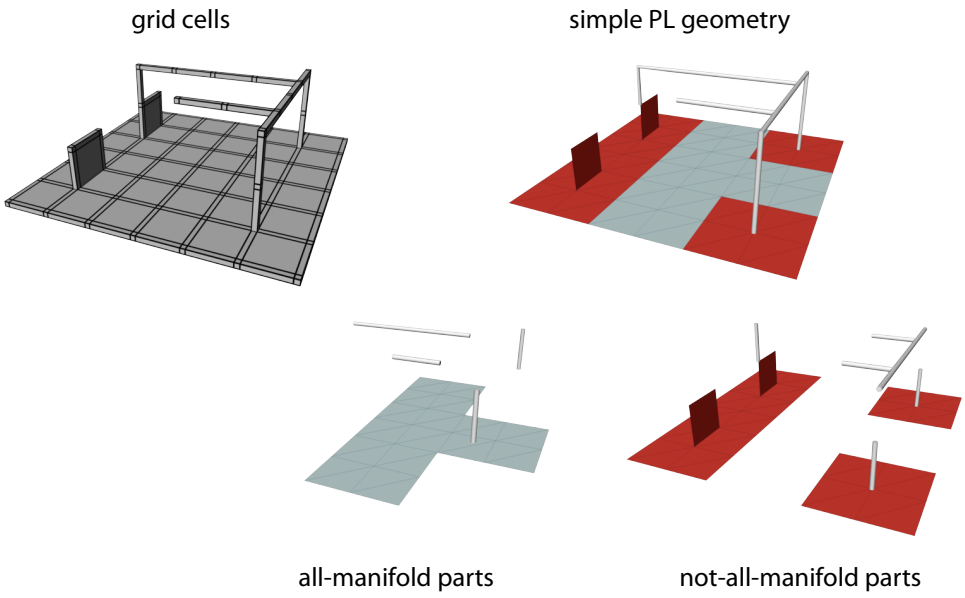


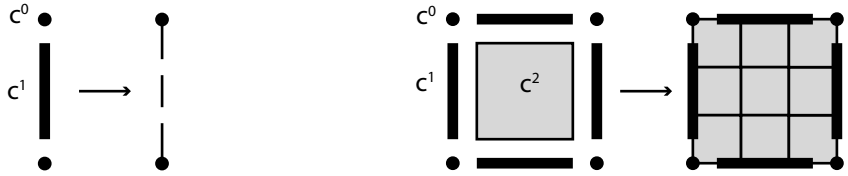
Figure 3.19: Simple piecewise linear geometric representation of a grid cell skeleton. The grid cell skeleton is depicted on the left. Geometry, depicted on the right, is decomposed into all-manifold and not-all-manifold parts, which can be rendered separately.

all-manifold, otherwise it is not-all-manifold; the same applies to 2-manifold cells and incident 2-manifold 0-cells.

Simple piecewise linear geometry representing the grid cell skeleton is constructed by representing 1-manifold 1-cells as straight line segments connecting the two incident 0-cells and 2-manifold 2-cells as quads built by their four incident 0-cells. Each triangle or line segment can be further classified as all-manifold or not-all-manifold. This allows highlighting different parts of the structure. Post-processing, such as smoothing and triangle decimation, is applied as described in the previous subsection. See Section 4, in particular Figure 3.19 and Figures 4.5–4.7, for examples.

Detailed piecewise linear geometry uses more geometric primitives to represent the organization of grid cell complexes in manifold parts attached to junctions. In contrast to simple piecewise linear geometry, each 1-cell is represented by at least one all-manifold line segment; each 2-cell by an all-manifold triangle:

Figure 3.20: Subdivision scheme for grid cells.

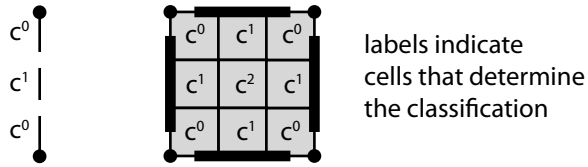


The organization of the cell complex is captured in more detail at the cost of an increased number of line segments and triangles. Distinct elements are used for representing

manifold parts and for representing their attachment to junctions. As 1-manifold 1-cells are attached to two 0-cells, three distinct elements are needed; one for representing the manifold part and one for each end attached to a 0-cell. 2-manifold 2-cells are attached to four 1-cells and four 0-cells. Overall nine distinct elements are needed. Three line segments represent a 1-cell. Nine tiles represent a 2-cell. Vertices are located at the positions of 0-cells and at one third and two thirds the distance to the next 0-cell in each direction. Hence a $3 \times 3 \times 3$ subdivision of the 0-cells' positions generates all potential vertex locations of above scheme. This allows to establish continuity to neighboring cells by associating vertices with integer positions in the subdivided grid. A table maps them to vertices in the piecewise linear geometry.

The classification of 0-, 1-, 2-cells canonically determines a classification of line segments and tiles:

Figure 3.21: Classifying sub-divided grid cells.



Manifold parts, junctions and attaching of cells can be highlighted explicitly as illustrated in Figure 3.22.

The grid cell model's capability of explicitly representing all dimensions makes geometry generation simpler compared to the grid point model. Grid cell skeletons explicitly are pure one-dimensional, pure two-dimensional, or mixed-dimensional. Three-dimensional cells are never part of a grid cell skeleton. Hence no further retraction is needed during geometry generation: 1-cells are straight forwardly represented as line segments; 2-cells as quads.

Cells in the grid cell model are restricted to be aligned with the three principal axes, whereas 26-neighbors in the grid point model have more directional freedom. But post-processing the geometry is needed in both cases to achieve a smooth representation. Therefore the restriction to principle directions is not a serious limitation. Simply more smoothing might be needed in the grid cell case.

In rare cases, grid skeletons may »spiral« around a straight voxel line:

Figure 3.23: Voxel line, grid cell line, smoothed geometry.

This can occur if cells have the same distance value to the boundary. In such cases the result of thinning dependes on the details of the processing order. This case is rarely observed and can be resolved by smoothing the constructed geometry as illustrated at the right.

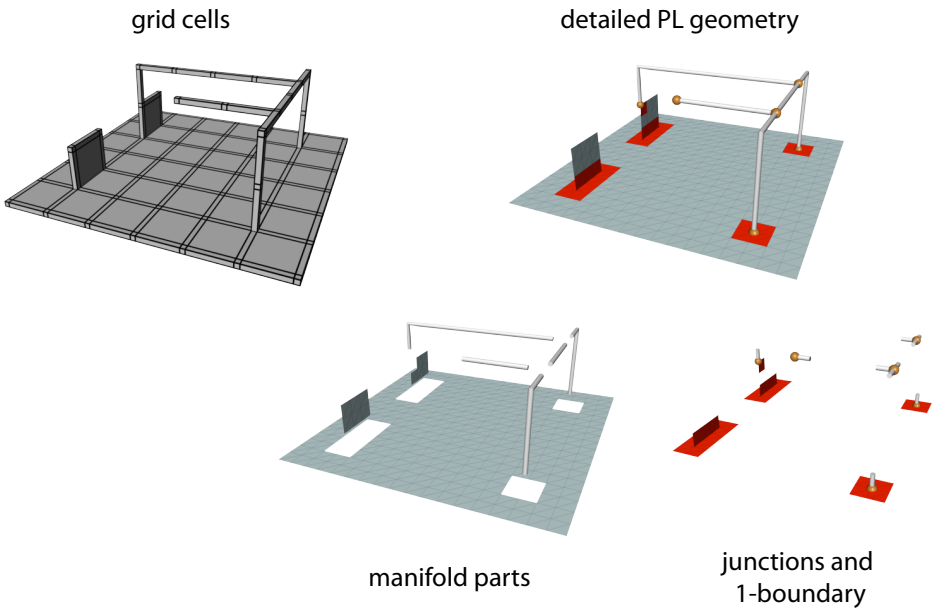
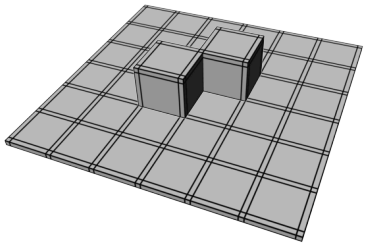


Figure 3.22: Detailed piecewise linear geometric representation of a grid cell skeleton. The grid cell skeleton is depicted on the left. Geometry, depicted on the right, is decomposed into all-manifold and not-all-manifold parts, which can be rendered separately. The detailed representation is capable of illustrating junction detached from each other and the topology of manifold parts.

Grid cell skeletons may contain self-folding parts, similar to voxel skeleton:

Figure 3.24: Self-folding grid cell skeleton.



One root of the problem is that four 2-cells can be attached to one 1-cell, with two pairs of them forming manifolds. A different grid type with generic point locations could be a way to resolve the problem of self-folding. For instance, the body centered cubic grid could serve as a base for a grid cell model with only three 2-cells meeting at each 1-cell.

Summary

- A novel geometry generation scheme for representing voxel skeletons by piecewise linear geometry was presented. Voxels in the grid point model are locally retracted

in each dual cube to a low-dimensional simplicial complex, that is triangle meshes and piecewise straight lines.

- Thinking specifically about voxel skeleton configurations instead of generic voxel configurations helps to deal with the topological implications.
- One-dimensional results can be enforced, if required, by collapsing junctions to a single point.
- Grid cells can be directly converted to geometry. A subdivided geometry can be used to give a detailed picture representing manifold parts and their attachments to each other separately.
- Smoothing and triangle decimation is applied to the generated geometry before it is used for interactive rendering.

4 Skeletons of Synthetic Examples

This chapter presents results of skeletonization applied to synthetic test objects. Input test objects with varying noise levels were constructed on a 256^3 regular grid. A combination of three major structures—rods, plates and spheres—was used to build the test objects. Initial shapes were manually drawn on a 3d voxel grid and smoothed by a gaussian filter. A threshold selected the final object voxels. Two distorted versions of each shape were generated in addition. Random noise of two different frequencies was added to the scalar function before filtering and threshold selection.

Figure 4.1 presents one-dimensional voxel skeletons in the grid point model. The input object contains only rods. Smooth distance ordered thinning (Page 26) was applied. Afterwards geometry was generated as described in Section 3.1.2 and smoothed by a weighted averaging of neighboring vertices. The inset depicts a voxel configuration which would be initially interpreted as a local loop and collapsed to a single point during geometry generation. The end-point detection rule (Page 29) is varied from retaining no end-points to end-points of length 1, 2, and 5. Retaining end-points of length 1 makes the skeleton highly sensitive to noise on the object surface; whereas all other cases result in robust skeletons. No further pruning of spurious side branches is needed.

Figure 4.2 presents similar results for computations of one-dimensional skeletons in the grid cell model. Thinning in the grid cell model (Page 33) was used followed by generation of simple piecewise linear geometry (Page 56) and smoothing. The inset shows the local configuration in the grid cell model corresponding to the configuration shown in Figure 4.1 for the grid point model. The one-dimensional structure is explicitly resolved. Importance of local cell end-points (Page 33) is 0 (no end-points retained), 2, 4, and 6. An importance of 2 yields results practically unusable, because many spurious branches are included in the skeleton. Higher values result in robust skeletons representing only the major features of the object. The results do not visually differ from the skeletons computed in the grid point model (Figure 4.1).

Figure 4.3 explores the influence of architectural elements on the skeletons computed by distance ordered thinning. Results are only presented for the grid point model; grid cell skeletons exhibit the same principal behaviour. Plate-like structures are represented by the computed skeleton to some extent. The topological features are reflected, while details of the geometry, like for example the main location and direction of the plates, are not. The situation is similar for a sphere-like part. Cavities however are a topological feature and thus always preserved by thinning. The result is no longer one-dimensional but contains a surface enclosing the cavity.

In Figure 4.4, strong noise was added, which causes topological changes of the original object. Cavities and loops may be introduced and are reflected by the skeleton because thinning strictly preserves homotopy equivalence. This limits thinning's robustness to

noise.

The next two examples illustrate the pure two-dimensional case. The object in Figure 4.5 contains plate-like structures only. A grid cell skeleton is computed based on the geodesic boundary distance (see Section 2.3.2) followed by generation of simple piecewise linear geometry (Page 56). The top two results show raw versions of geometry for a threshold of 100 and 200 on the chamfer-(3, 4, 5)-distance along the boundary. The results in the third row were smoothed and not-all-manifold triangles are highlighted in red. The number of triangles is approximately 150,000. The decimation algorithm by Garland and Heckbert (1997) was applied to reduce the triangle count to approximately 3,000 in the last row. The same processing is used in Figure 4.6 on an object containing a sphere like part. Spikes originating from the center of the sphere are induced in the skeleton. The skeleton fails to effectively represent the sphere like part.

In Figure 4.7 an object combining rod-like and plate-like shapes was processed in the grid cell model. Plate-like parts are extracted in a first step (second and third row with threshold 100 and 200) followed by thinning to extract the rod-like parts (fourth row). The resulting skeleton represents the different kinds of architectural elements.

Cavities are preserved by geodesic distance based skeletons as illustrated in Figure 4.8. The inside boundary of the cavity can not be connected to the outside boundary by a path on the surface. Hence the geodesic distance always detects cavities independently of the chosen threshold.

Figure 4.9 compares thinning ordered by the distance to the boundary with thinning ordered by the geodesic distance. Distance to the boundary pushes the skeleton to the thickest parts of the object while the geodesic distance pushes the skeleton to the center of plate-like structures.

More examples based on »real world« image data are presented in Chapter 6.

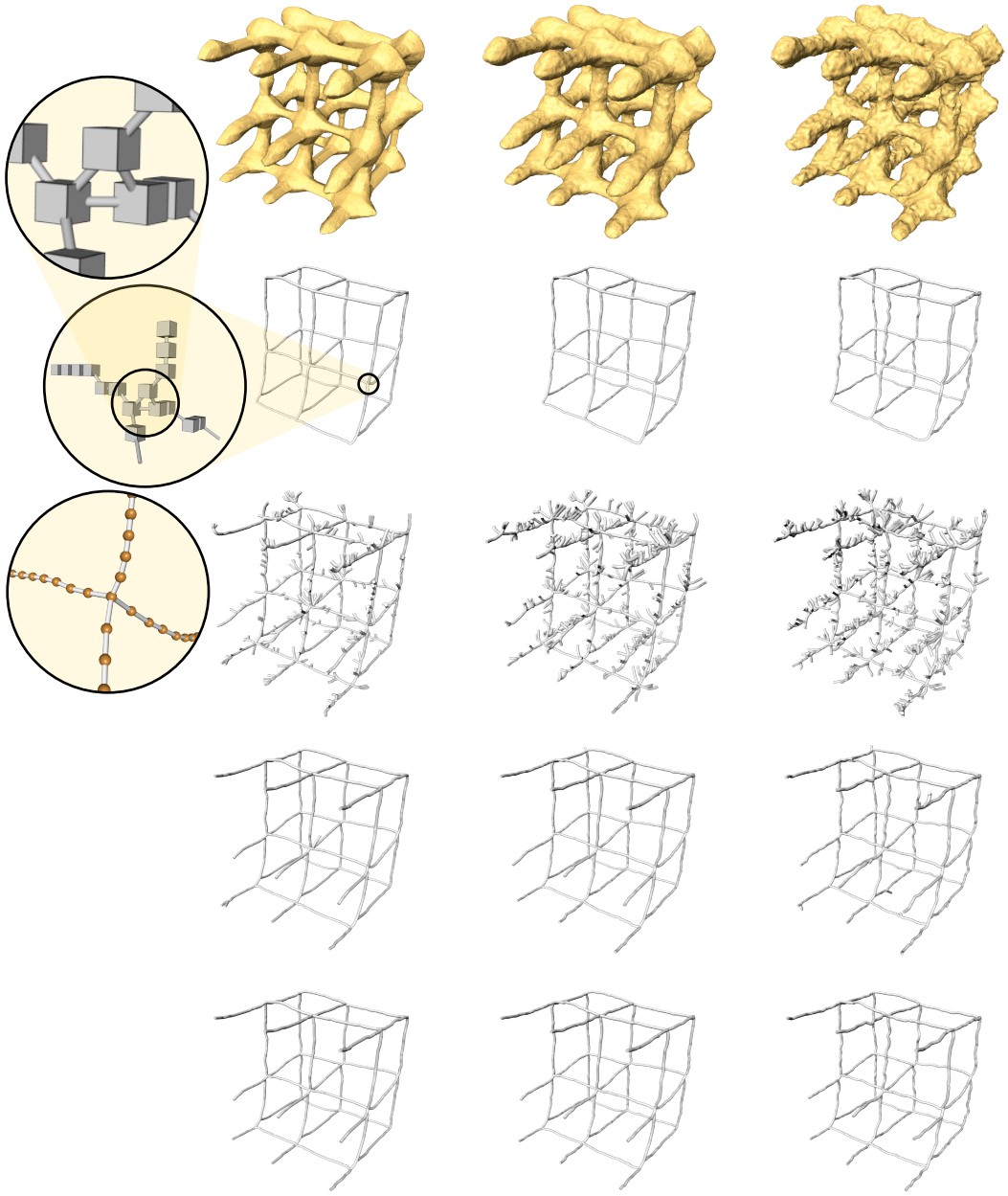


Figure 4.1: Skeletons of rod-like structure computed in the grid point model. From left to right, noise of increasing intensity is added. From top to bottom, the importance of local end-points is 0 (no end-points), 1, 2, and 5. The middle and top insets show close-up views of the grid point configuration at a junction. The bottom inset displays the generated geometry after collapsing the junction to a single point. This Figure is referenced on Page 29, 33, 52, 61, 79.

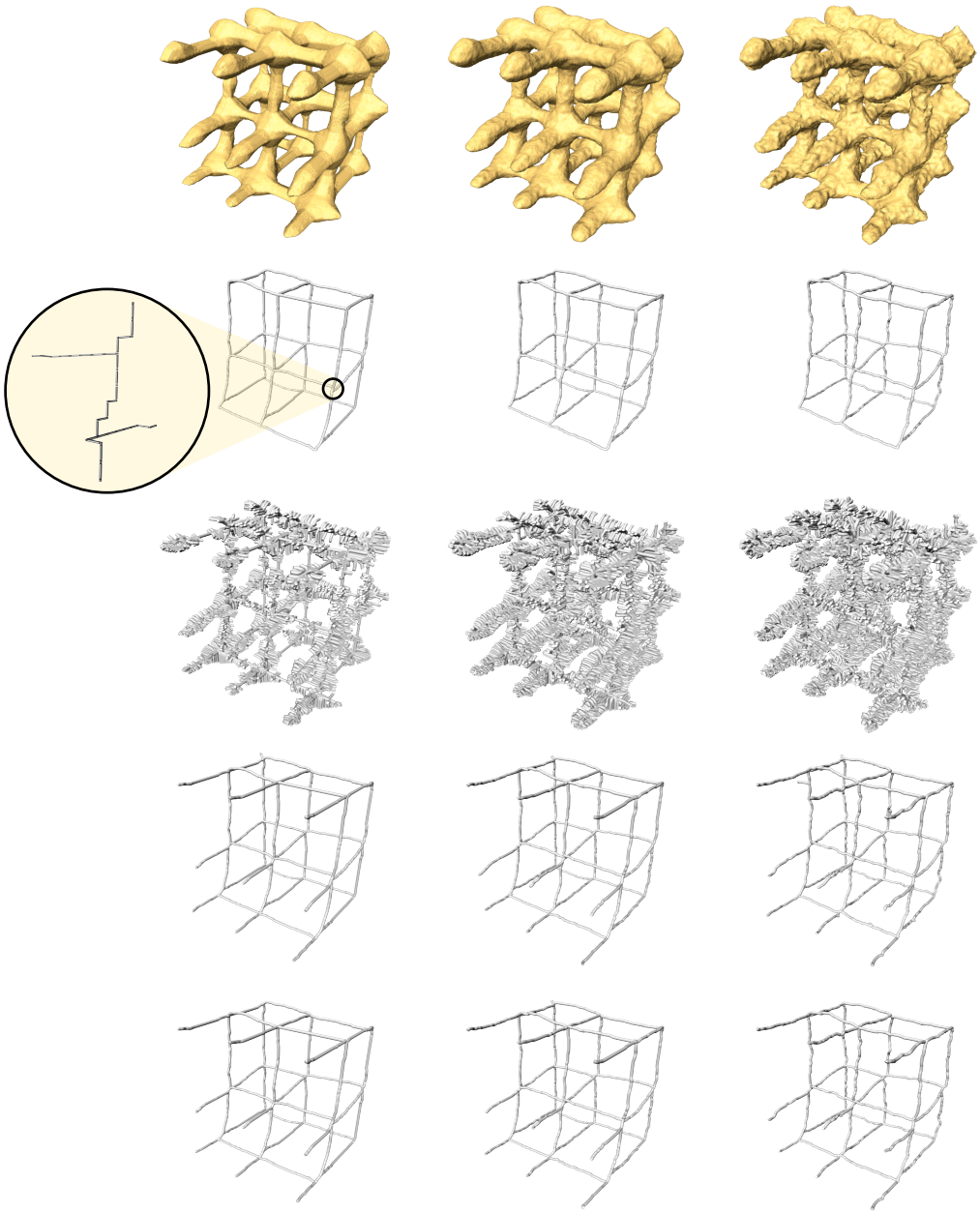


Figure 4.2: Grid cell skeletons of rod-like structure. From left to right, noise of increasing intensity is added. From top to bottom, the importance of local end-points is 0 (no end-points), 2, 4, and 6. The inset shows a close-up view of the grid cell configuration at a junction. This Figure is referenced on Page 33, 61.

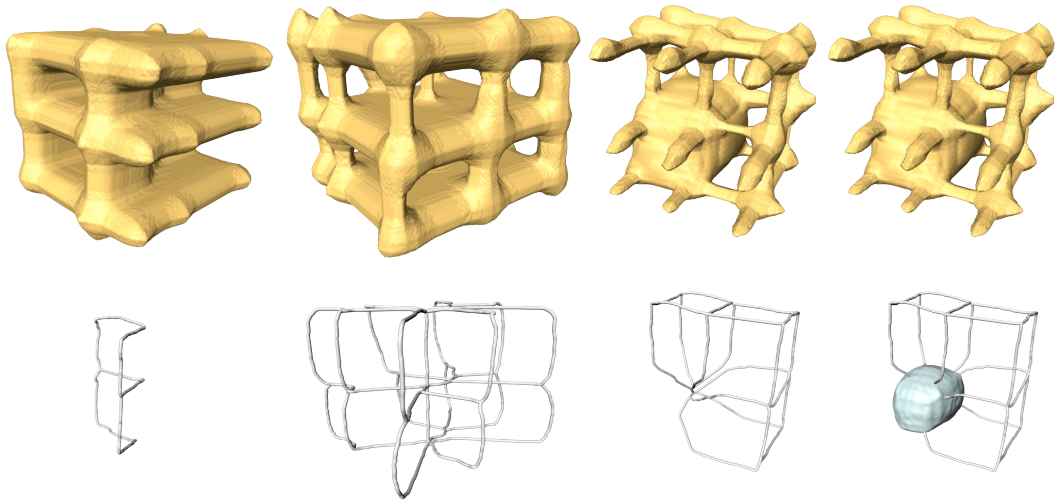


Figure 4.3: Influence of architecture on distance ordered thinning. The skeletons in the bottom row were computed from the object in the top row by thinning in the grid point model. The rightmost object contains a cavity inside the sphere. This Figure is referenced on Page 29, 61.

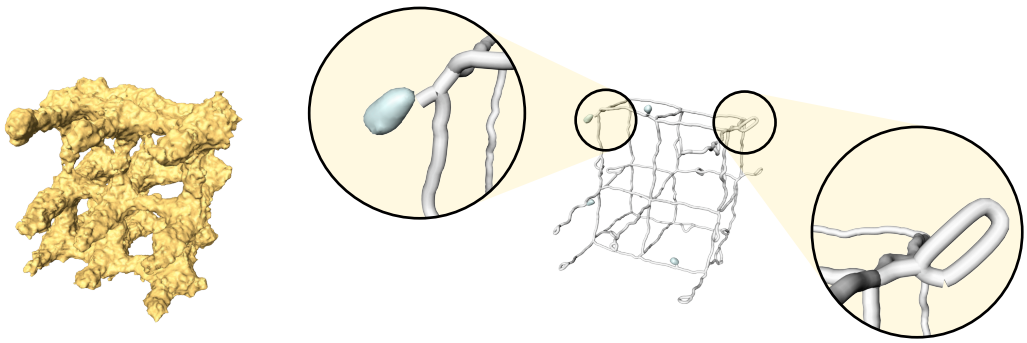


Figure 4.4: Influence of strong noise on thinning. No end-point were retained but noise can create loops and cavities, which are reflected by the skeleton, limiting thinning's robustness to noise. This Figure is referenced on Page 29, 61.

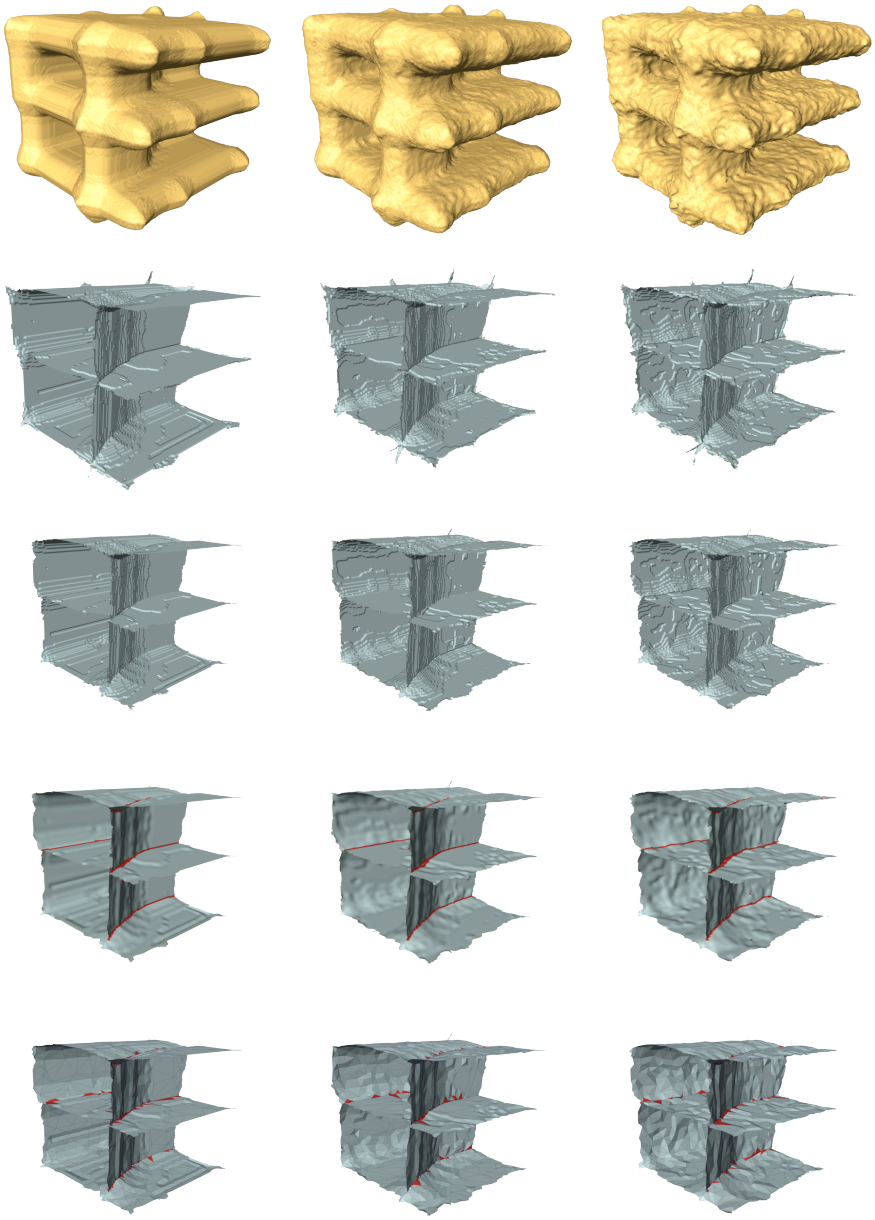


Figure 4.5: Grid cell skeletons of plate-like structure. From left to right, noise of increasing intensity is added. From top to bottom, the second and third row show pure two-dimensional skeletons with a threshold of 100 and 200 on the chamfer-(3, 4, 5)-distance along the boundary. In the fourth row, geometry is smoothed, and in the fifth row, the triangle count is decimated to 2% of the original number of triangles. This Figure is referenced on Page 39, 56, 62, 79.

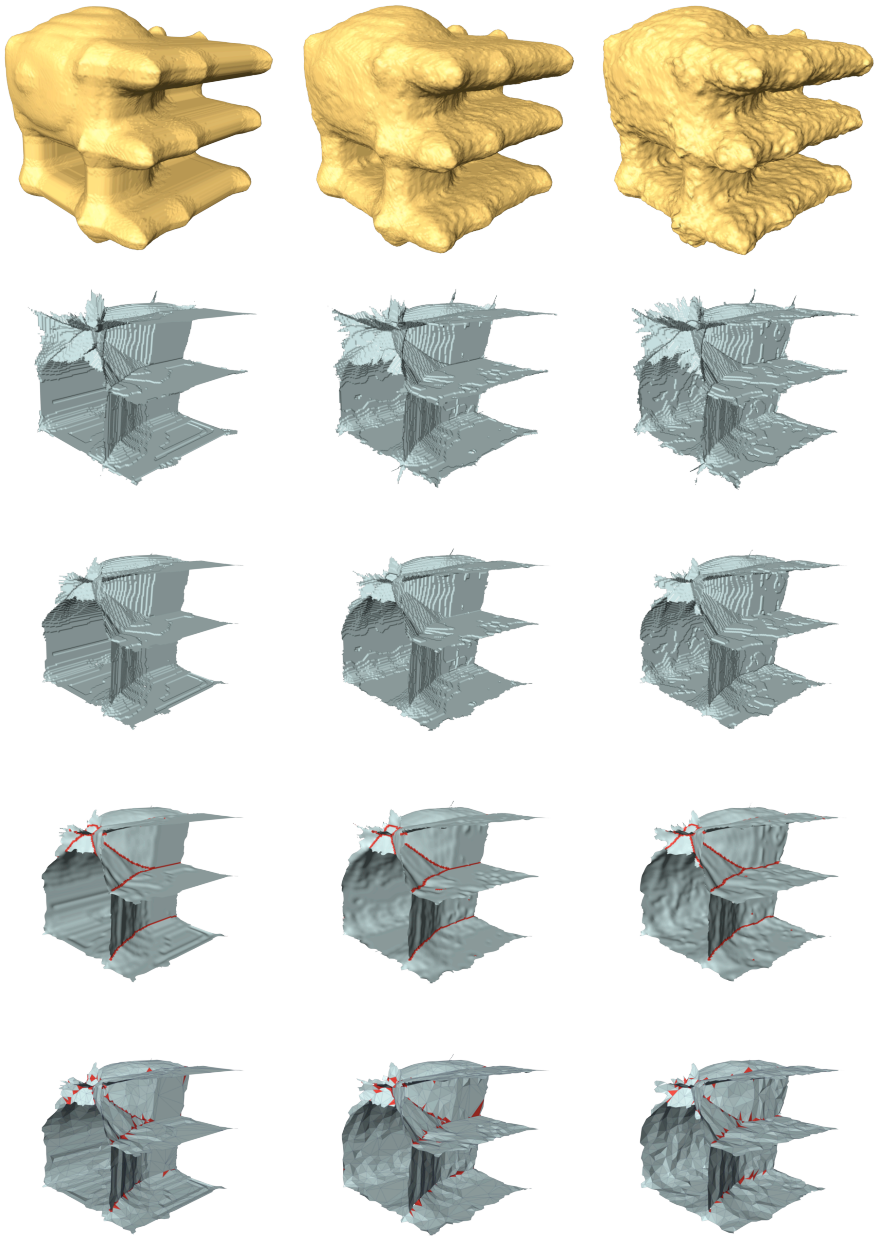


Figure 4.6: Grid cell skeletons of a structure including a sphere-like part. From left to right, noise of increasing intensity is added. From top to bottom, the second and third row show pure two-dimensional skeletons computed using thresholds of 100 and 200 on the chamfer-(3, 4, 5)-distance along the boundary. In the fourth row, geometry is smoothed, and in the fifth row, the triangle count is decimated to 2% of the original number of triangles. This Figure is referenced on Page 39, 62.

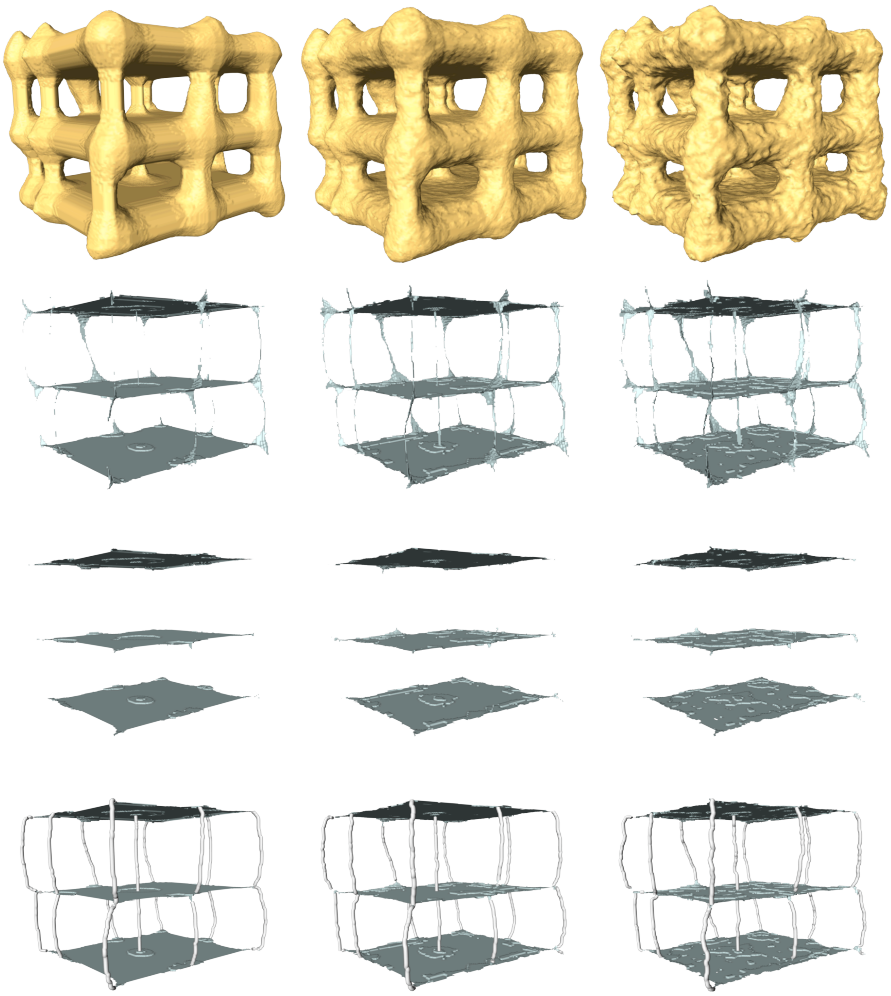


Figure 4.7: Grid cell skeletons of a mixed rod- and plate-like structure. From left to right, noise of increasing intensity is added. From top to bottom, the second and third row show pure two-dimensional skeletons computed using thresholds of 100 and 200 on the chamfer-(3, 4, 5)-distance along the boundary. The bottom row shows mixed dimensional skeletons for threshold 200. This Figure is referenced on Page 39, 41, 56, 62.

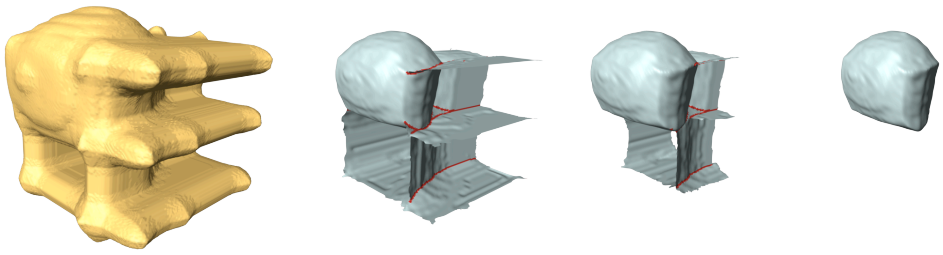


Figure 4.8: The influence of cavities on a geodesic distance based skeleton. The skeletons were computed using a threshold of 200, 400, and 1000 on the chamfer-(3, 4, 5)-distance along the boundary. The cavity is a stable feature reflected by the skeleton independently of the threshold. This Figure is referenced on Page 39, 62.

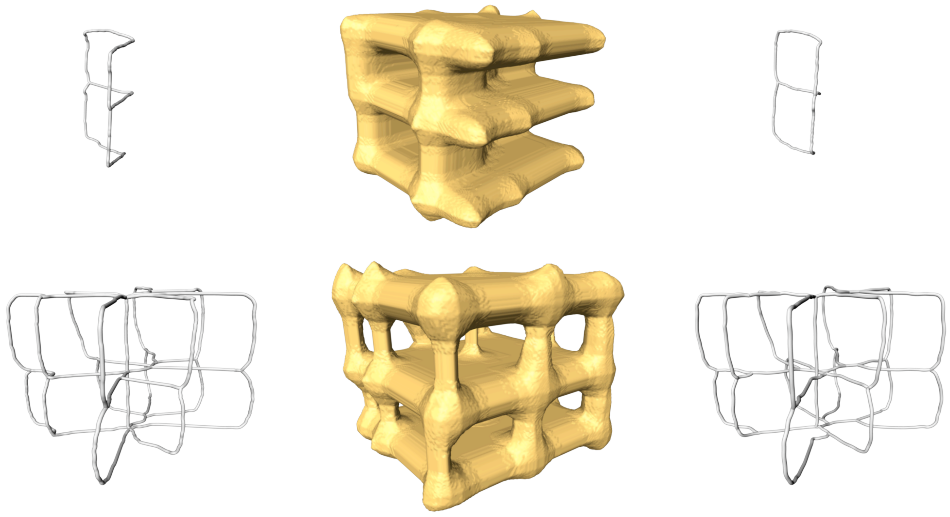


Figure 4.9: Comparison of thinning ordered by distance and by geodesic distance. The results of distance ordered thinning are depicted on the left; the result of geodesic boundary distance ordered thinning on the right. This Figure is referenced on Page 62.

5 External Memory Algorithms for Computing Skeletons

This chapter extends perviously presented algorithms to out-of-core processing. In the consequence, massive voxel objects of any size that fit in secondary storage can be processed with a limited amount of main memory.

External memory algorithms and data structures (Vitter, 2001) address the problem of efficiently processing data that exceeds the size of main memory. Applying established algorithms to such input data is not straight forward. Input data initially resides on disk. Part of it needs to be loaded into main memory, processed, and results stored to disk, before the next part is loaded.

A major difference to in-core processing is that data is transferred in blocks between disk and main memory. The effective bandwidth accessing consecutive locations versus random seeks differs dramatically. Access costs are highly non-uniform. The same is true over the full memory hierarchy (disk–RAM–cache–CPU) as apparent from Table 5.1. Disk access times are four orders of magnitude larger than access times to main memory whereas bandwidth changes only one order of magnitude. Thus the influence of access time on the runtime of algorithms is most significant when disk access is required.

This observation is captured in the **external memory model** used for analyzing algorithms (Vitter, 2001). The objective is to devise algorithms with optimal runtime, optimal required space, and an optimal number of block transfers. Data of size N is initially stored on disk, which is assumed to have infinite capacity. The disk is organized in blocks of size B. For computation, data needs to be transferred to a cache (the main memory) of size $M \ll N$. Data is transferred between disk and cache as full blocks. The CPU has access to any storage element in the cache at uniform cost (uniform cost RAM). But the CPU can not directly access data on disk; see Figure 5.1 for an illustration. For example access to a single byte not currently in the cache requires transfer of a complete block of size B. Scanning once through the data can be achieved with $\lceil N/B \rceil$ block transfers if data is scanned in the same order as it is stored on disk. But in the worst case, a scan may require N block transfers if only one element from each block is used and the block gets evicted from the cache before the next byte is required. Because B is typically a large constant—for example 4 kByte in many operating systems—the benefits of an optimal number of accesses are high.

Known algorithms need to be analyzed and modified to run with minimal performance loss due to out-of-core data storage. The first step is to understand the data access patterns. Then, when possible, the algorithm should be redesigned to maximize data access locality. This may require to devise a data storage layout consistent with the access pattern, thus amortizing the cost of individual I/O operations over several memory access operations.

Table 5.1: Latencies and bandwidths in the memory hierarchy. Typical numbers for size, access time, and bandwidth in 2001; adapted from Hennessy et al. (2003), p. 394.

	Register	Cache	Main Memory	Disk
Size	< 1 kB	< 16 MB	< 16 GB	> 100 GB
Access time [ns]	0.25–0.5	0.5–25	80–250	5,000,000
Bandwidth [MB/s]	20,000–100,000	5000–10,000	1000–5000	20–150
Managed by	Compiler	Hardware	Operating System	Operating System

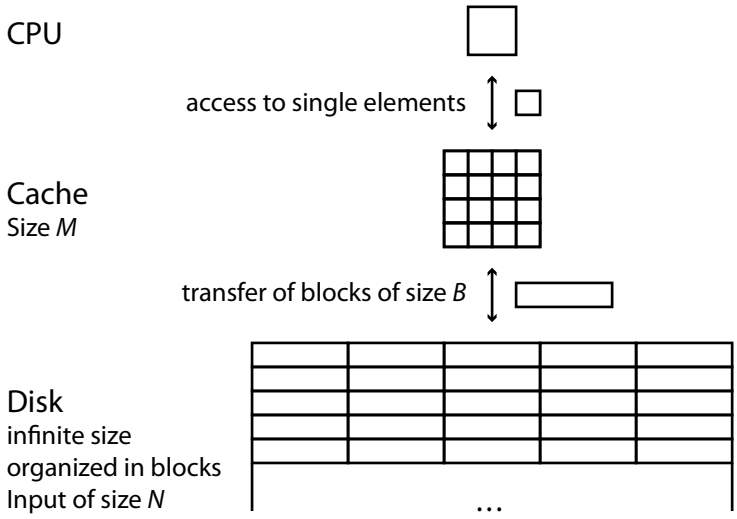


Figure 5.1: External memory model. Input data of size N is initially stored on disk, which is assumed to have infinite capacity. The disk is organized in blocks of size B. For computation blocks of data are transferred to a cache of size $M \ll N$. The CPU can only access data in the cache.

Algorithms may be optimized for a specific cache size M or block size B. More recently, cache-oblivious algorithms (Frigo et al., 1999) opened a new way of considering these problems. Their goal is to optimize for any kind of memory hierarchy containing caches, without needing to know details of the hierarchy, such as cache or memory sizes. Silva et al. (2002) give a review of external memory algorithms in visualization. Algorithms presented include many algorithms for geometry processing. The following discussion focuses on data stored on regular structured grids as all algorithms presented throughout this thesis process data stored on such grids.

One way to deal with massive data in visualization is explicitly managing application's memory and I/O. Cox and Ellsworth (1997) analyze application controlled demand paging of data decomposed into small cubes stored one per page, which is often denoted as chunking (see Sarawagi and Stonebraker, 1994). They conclude that many visualization algorithms only need a small fraction of the data; using small pages and cubed storage is favorable. As operating systems page size is often fixed, as their argument goes, application support is needed. Law et al. (1999) discuss how to organize a filtering pipeline that allows efficient, multi-threaded processing of large structured data sets. Their system

decomposes input data into blocks which are sent through the pipeline on demand. They discuss handling of intermediate data in caches and handling of block boundaries for filters with known kernel size. They conclude that explicit management of cache sizes and domain decomposition is superior to relying on virtual memory. Bergeron et al. (2005) propose iterators, which integrate knowledge about the applications' access pattern to control pre-fetching of structured data. They argue this would lead to »significant performance improvements while hiding details of out-of-core access«. Other systems and APIs dealing with structured data are the Active Data Repository (Kurc et al., 2001); NetCDF (UNIDATA, 2004), especially an efficient implementation based on MPI-IO (Thakur et al., 2002) as discussed by Li et al. (2003); and HDF5 (NCSA, 2003), on which the implementations of algorithms presented below are based upon. All these formats allow to store data in cubed chunks and to efficiently access sub volumes.

A different approach is to rely on virtual memory managed by the operating system but reorganize data in a cache oblivious layout (Frigo et al., 1999). This organization allows efficient access independently of the details, such as page size. Pascucci and Frank (2003) use this approach successfully for slicing of large structured grids. Yoon et al. (2005) present an application to mesh layouts.

The following discussion is most closely related to Law's system in using explicit data management: Blocks of data will be explicitly loaded, processed, and stored back to disk. An implementation should be based on an file format that stores three-dimensional chunks of data. As long as the dimensions of blocks loaded into main memory are large in each direction compared to the chunks on disk, overhead caused by the file format is low. HDF5's (NCSA, 2003) chunking capabilities provide a suitable implementation, which was used as a basis for implementing the presented algorithms.

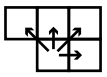
The following discussion is limited to the grid point model, but could easily be extended to the grid cell model.

5.1 Chamfer Distance Transformation

The discussion below presents an extension of the chamfer distance transformation algorithm introduced in Section 2.3 to external memory. Eight block-wise scans are required, compared to two raster scans in the standard algorithm. The discussion is presented in two dimensions first. An extension to three dimensions is straight forward.

Chamfer propagation takes the value at a pixel, adds weights to a neighboring pixel, and propagates a value if the sum is smaller than the already stored value at the neighbor. In the forward scan, from bottom left to top right, values are propagated to the right and the top including diagonal pixels:

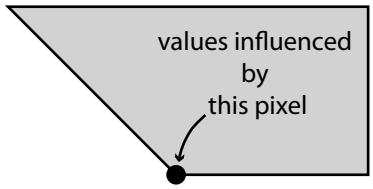
Figure 5.2: Chamfer mask.



After visiting all pixels in scan-line order, influence of a pixel extends to an area with a shape determined by the propagation mask and, in principle, is not limited in distance.

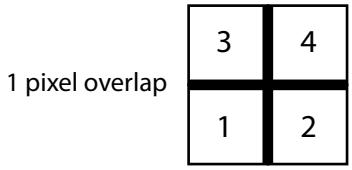
The following illustration depicts the area influenced by one pixel during a forward scan:

Figure 5.3: Area of influence of a chamfer propagation.



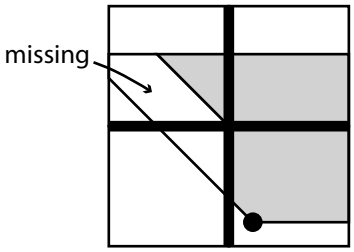
Efficient out-of-core processing can be achieved by block-wise processing with a carefully chosen processing order that visits blocks multiple times. Path propagation is achieved by decomposing data in blocks with one pixel overlap:

Figure 5.4: Overlapping blocks.



Visiting these blocks in scan-line order 1, 2, 3, 4 propagates all path to the right and top; but diagonal paths to the top left may be cut at block boundaries and results would differ from the original algorithm:

Figure 5.5: Propagation interrupted by block boundaries.



Visiting blocks in each scan-line a second time in reverse order resolves this deficiency. Thus blocks in the example above would be processed in order 1, 2, 1, 3, 4, 3. Now all paths are correctly propagated. The same argument applies to the backward scan. Overall, four scans over blocks with one pixel overlap compute a globally correct chamfer distance transformation in two dimensions.

In three dimensions, eight scans, as described in Algorithm 4, are needed. The three-dimensional chamfer masks (see Figure 2.14) propagate values to all voxels in the top direction. Therefore blocks need to be visited row by row as in two dimensions and each layer of blocks also row by row in reverse order. The example layer would be processed in the following order: 1, 2, 1, 3, 4, 3, 1, 2, 1. The exact number of blocks processed is $2 \cdot n_z \cdot (2n_x - 1) \cdot (2n_y - 1)$, with n_x, n_y, n_z the number of blocks in each dimension.

The presented algorithm provides a solution to computing distance transformations of massive input data. We used a simple chamfer approximation in our presentation. A

Algorithm 4 External memory chamfer distance transformation. The number of blocks in each direction is denoted by n_x, n_y, n_z. The procedures ForwardPropagateBlock and BackwardPropagateBlock are placeholders for standard chamfer scans.

```
1:  procedure ExternalChamferTransformation
2:      for z = 1 ... n_z do
3:          for y = 1 ... n_y ... 1 do
4:              for x = 1 ... n_x ... 1 do
5:                  ForwardPropagateBlock(x, y, z)
6:              end for
7:          end for
8:      end for
9:      for z = n_z ... 1 do
10:         for y = n_y ... 1 ... n_y do
11:             for x = n_x ... 1 ... n_x do
12:                 BackwardPropagateBlock(x, y, z)
13:             end for
14:         end for
15:     end for
16: end procedure
```

larger overlap between neighboring blocks can be used to apply larger chamfer masks, which provide a better approximation of euclidean distances (Fouard and Malandain, 2003). The block-wise processing scheme removes the limit on data size imposed by available main memory by increasing the processing time. In practice the standard raster scan algorithm could be used as long as data fits into main memory. The external memory variant would only be chosen for larger input data.

5.2 Thinning

This section presents an external memory extension of distance ordered homotopic thinning, as introduced in Section 2.3.1. The input is decomposed into blocks and each block is processed separately. Voxels with a lower distance to the block boundary than to the object boundary are kept fixed to ensure medialness of the resulting skeleton. They will be processed in a neighboring block. Overlap is adjusted accordingly.

Few other authors discussed block-wise skeletonization. Vossepoel et al. (1997) propose to skeletonize blocks independently of each other. In a second step, tracing of pixel lines and labeling of crossing points with the block boundary is used to connect all parts of the skeleton. Compared to our algorithm processing is quite complex. It remains unclear from their discussion if connecting the skeleton reliably succeeds in all cases. Pakura et al. (2002) process blocks without special precaution at the block boundaries and simply ignore processing artifacts. Thus, their skeletons may differ from the correct solution at block boundaries.

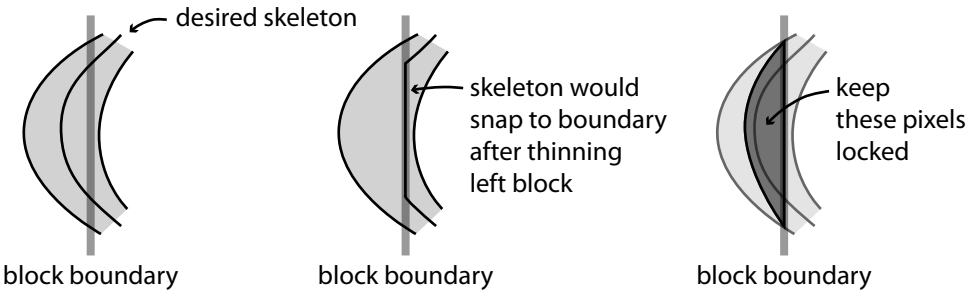


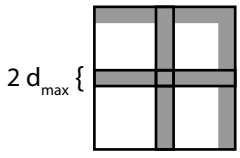
Figure 5.6: Example of a block boundary during thinning. Left: desired result; Middle: skeleton would be dislocated if left block was processed naively; Right: pixels are locked if their distance to the object boundary is larger than to the block boundary.

Our extension of thinning to external memory does not in general compute the exactly same result as globally applied SmoothDOHT; but the desired properties—homotopy, thinness, medialness—are preserved. Distance ordered thinning starts eroding voxels at all object boundaries simultaneously. Although only a local neighborhood is investigated before each removal, a block-wise decomposition for processing is not obvious because global ordering of voxels is necessary to ensure medialness. Block boundaries destroy the global ordering as becomes apparent shortly.

Applying thinning to overlapping sub-blocks can quite easily guarantee homotopy equivalence, while it is harder to guarantee medialness. Homotopy equivalence at block boundaries can be achieved by keeping the outmost layer of overlapping blocks fixed during processing. Figure 5.6 depicts a situation which could occur at a block boundary. Because one of the object's boundaries is not included in the sub-block, erosion would start only from one side of the object and would move past the medially located voxels to the block boundary and the skeleton would not be located in the center of the object.

Locking voxels that are nearer to the block boundary than to the object boundary resolves this issue. For those voxels medialness can not be established in the current sub-block. To compute a global skeleton sub-blocks need to be arranged such that every voxel is located more distant to the block boundary than to the object boundary in at least one sub-block. If an upper bound d_{max} on the distance transformation is known this is easily achieved by sub-blocks with $2d_{max}$ overlap. See illustration below and Algorithm 5 for details:

Figure 5.7: Block overlap for thinning.



We presented similar results in Fouard et al. (2004). There we proposed a more complex processing scheme handling block boundaries in different directions explicitly. The decomposition described here is simpler and straight forward to implement. Sub-blocks in both cases must be large enough—larger than $4d_{max}$—to allow a decomposition with

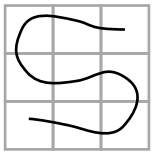
Algorithm 5 External thinning based on SmoothDOHT as detailed in Algorithm 2. A voxel object with a maximum distance transformation value d_{max} is processed in blocks of size s with overlap $2d_{max}$.

```
1:  procedure ExternalThinning
2:      for z = 0, s − 2d_max, 2 · (s − 2d_max), ... do
3:          for y = 0, s − 2d_max, 2 · (s − 2d_max), ... do
4:              for x = 0, s − 2d_max, 2 · (s − 2d_max), ... do
5:                  load subblock at x, y, z of size s^3
6:                  L ← {p|p voxel in sub-block at x, y, z of size s^3 ...
7:                          ... ∧ (d(p) > distance to upper block boundary ...
8:                                  ... ∨ p in block boundary layer)}
9:                  apply SmoothDOHT on sub-block with L kept fixed
10:                 store sub-block
11:             end for
12:         end for
13:     end for
14: end procedure
```

sufficient overlap. d_{max} is computed in one scan over the input distance transformation. In many applications voxel objects are »thin«, so d_{max} is small and sufficiently large sub-blocks fit into main memory. For all other cases further investigation of the algorithm would be needed.

In principle erosion can require a propagation further than d_{max} to carry over several sub-blocks boundaries. This is likely if no end-points are fixed. Multiple scans over intermediate results would be needed to erode the object to a single object in the following example:

Figure 5.8: Example requiring multiple thinning scans.



The center line would need to be eroded block by blockx to be able to finally retract it to a single point. In practical cases with end-point detection, one scan is sufficient to shrink the object to its center lines.

The discussed algorithm is capable of computing globally correct voxel skeletons of massive objects with a maximum thickness limited by the amount of main memory available for processing. The maximum thickness of a specific object can be evaluated from a global distance transformation to determine the required amount of main memory. Larger amounts of memory help to reduce overhead. In practice one could use the standard algorithm as long as data fits into main memory and only switch to the external

memory variant if needed.

5.3 Geodesic Boundary Distance

The next algorithm extended to external memory is the measure based on the geodesic distance presented in Section 2.3.2. The algorithm can be recast as an image filter with limited size, which allows to easily decompose the output domain in non-overlapping blocks. Each block is visited and part of the input domain enlarged by the kernel size s in each direction is loaded. The result in the output block does not depend on input data further away. Thus, one scan over the output blocks computes the full result:

Figure 5.9: Decomposition of output; margin on input.

The required kernel size, which depends on the maximal distance to the boundary and the importance value selecting the skeleton, is derived as illustrated in Figure 5.10. We assume an upper bound d_{max} on the distance transformation inside the object. The threshold t on the importance value selecting skeleton voxels is also known. For each output voxel the two nearest object boundary voxels need to be included in the input block. These are no further away than d_{max}. The geodesic distance is computed between the two nearest boundary voxels. Values beyond t can be cut. In the worst case a path shorter than t could run straight to the input block boundary and back to the second boundary voxel. Therefore, an additional border of $t/2$ is sufficient to hold all paths with length below t. Hence points further away in the input than $s = d_{max} + t/2$ do not influence an output voxel. s is the required kernel size. d_{max} and t must be small to allow loading of sufficiently large sub-blocks of the input into main memory.

Overhead of the block decomposition is small. The computation of the geodesic distance dominates runtime. While the distance transformation yielding the nearest object boundary voxel needs to be computed on the full sub-block, it is sufficient to evaluate the geodesic distance for neighboring pairs of output voxels. Because block decomposition does not change the number of output voxels, overhead is small. Nonetheless, large blocks are favorable because some overhead of the block-wise algorithm depends on the ratio of the kernel size to the block size.

The main algorithm can easily be combined with post-processing of the voxel skeleton and thinning. Thresholding and removal of two-voxel-thick parts (see Page 39) is applied before the output block is stored. A limited extension of the kernel size by the size of the masks used for removal tests is sufficient. The input block size is increased accordingly. A thinning step can follow to extract rod-like parts in which voxels identified as part of the two-dimensional skeleton would be kept fixed.

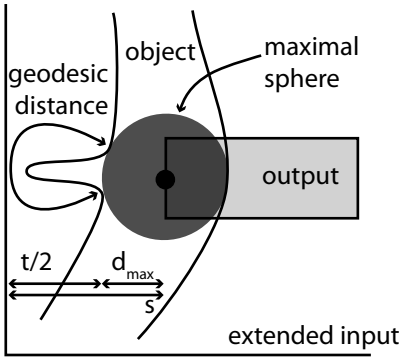


Figure 5.10: Margin for computing the geodesic distance based measure. The required input size to compute the two-dimensional skeleton is derived from a worst case configuration. A border of at least the distance to the nearest object boundary and half of the maximum geodesic distance, hence $s = d_{max} + t/2$, is always sufficient to compute correct results.

5.4 Geometric Representation of Voxel Skeletons

Classifying voxels and generating geometry per dual cube are local operations (see Section 3.1), which allows a straight-forward extension to block-wise processing. The only challenge is to establish continuity across block boundaries. One voxel overlap is required and previously generated vertices need be identified in a neighboring block.

A simple solution for establishing connectivity across block boundaries is to store vertex identifiers in a global table with storage for every block boundary slice. With block size s and (cubed) input data of size N such a table requires $N^{1/3}/s$ slices of size $N^{2/3}$ each. Hence the overall size is $\mathrm{O}(N/s)$. For large block-size s this is a practical amount of identifiers to store in main memory; although the data structure does not scale to larger size as it is linear in the input size. An improved data structure would only retain identifiers at boundaries between already processed blocks and yet unvisited ones. It would require only $\mathrm{O}(N^{2/3})$ storage but it is more awkward to maintain as it needs to be dynamically adjusted during the block scan.

The resulting geometry is assumed to fit into main memory. But it could as well be streamed to disk for further processing with external memory algorithms and hierarchical rendering techniques reviewed in Silva et al. (2002); or streaming geometry processing as discussed by Isenburg and Lindstrom (2005).

5.5 Timings

This section's objective is to measure the influence of block-wise processing on the overall runtime. Each method is measured three times: (1) in a standard implementation running on a single large array residing completely in main memory; (2) in a block-wise implementation with all data in main memory; (3) in a block-wise implementation with out-of-core data storage. The differences in runtime between (1) and (2) are related to the

Table 5.2: Timings for distance transformations and thinning. The table lists timings for chamfer distance transformation, thinning, and geodesic distance computation for an input data set of size 1280 × 1280 × 1280. The input data was processed in one large block (1); in block-wise scans with data completely in main memory (2); and in block-wise processing including disk I/O (3).

	(1) Standard	(2) Block-wise in-core	(3) Out-of-core
Chamfer distance	107 s	211 s	4647 s
Thinning	214 s	257 s	2195 s
Geodesic distance	9432 s	10717 s	11687 s

overhead resulting from block-wise processing. The differences between (2) and (3) are related to disk I/O.

Two test data sets of size 1280 × 1280 × 1280 were generated by replicating the data sets used in Figure 4.1 and Figure 4.5 five times in each direction. Approximately 3.5% voxels are set to object voxels in the object made of rod-like structures; approximately 16% voxels are set to object voxels to form the plate-like structure.

The overall size was chosen such that running all algorithms completely in main memory is possible. All test were run on a machine with two Opteron 250 and 16 GB of main memory running a Linux kernel. During the out-of-core measurement 14 GByte of main memory were blocked by a call to the *mmap* system call with the flag *MAP_LOCKED* to inhibit caching of I/O by the operating system; the source code from http://www.dunkel.dk/thesis/blockmem.c was used.

Table 5.2 lists execution times of chamfer distance computation and thinning of the object made of rod-like structures; and of the computation of geodesic distances along the boundary (with a threshold of 100 on the chamfer-(3, 4, 5)-distance) of the object made of plate-like structures. Runtimes of block-wise processing in main memory are larger than processing data in one block by a factor between one and two. Runtimes of out-of-core processing are mainly limited by the I/O capabilities. The CPU was rarely used at 100% in these cases. Optimizing I/O performance was not the goal of these timings. Data were read from and written to a single disk. Figure 5.11 displays I/O rates (data rate to and from disk) during processing measured with the Unix command *iostat*.

Computation of the chamfer distance transformation is I/O bound. It starts with a scan over the input data (reading 1 Byte per voxel) to initialize the chamfer distance transformation stored as short integer values (writing 2 bytes per voxel). This is indicated as (a) in the I/O-diagram. During (b) values are propagated, which requires approximately 8 scans. During each scan, 2 Bytes are read and 2 Bytes are written. Hence read and write rates are expected to be approximately the same. The sustained average rate for both cases is roughly 4000 kB/s. An estimation for the overall runtime is given by the number of voxels times eight scans times 2 Bytes per voxel divided by the average I/O rates. This computes to $1280^3 \cdot 8 \cdot 2 \,/\, 4000000 \approx 8000$ seconds. The measured runtime is smaller, which can be understood by caching of the operating system. The last blocks used during a forward scan are used first during a backward scan. Apparently not all caching effects were suppressed by blocking memory.

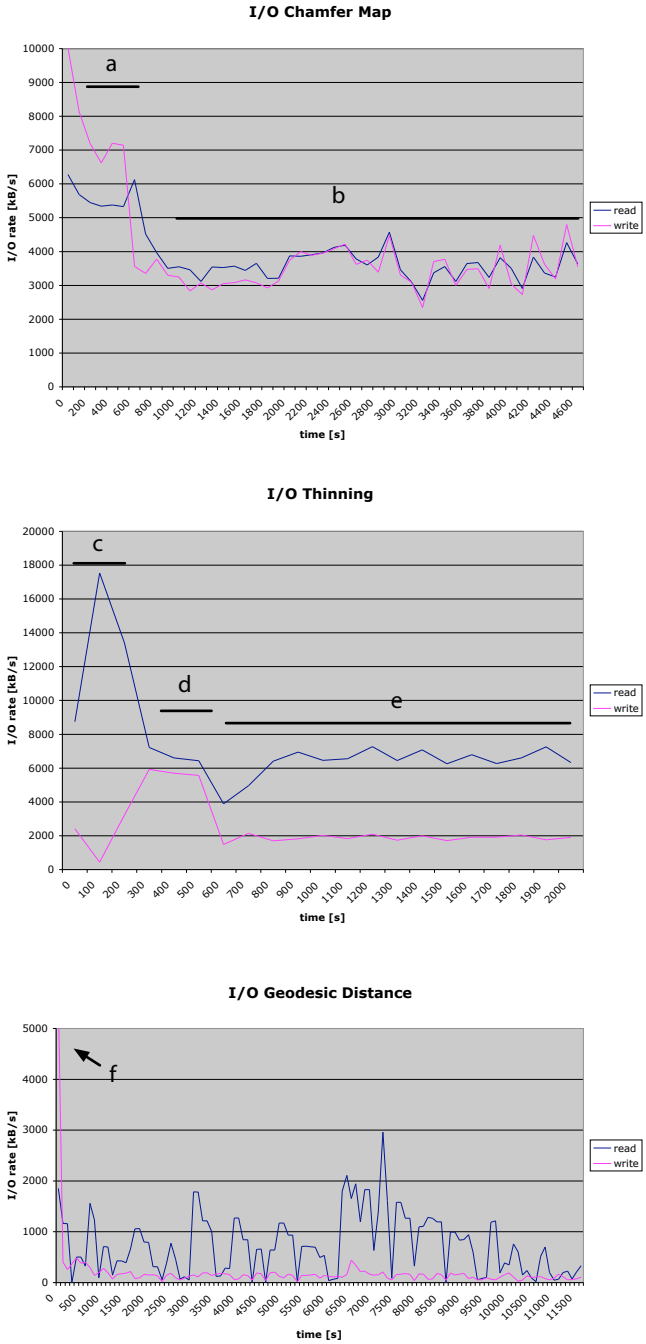


Figure 5.11: I/O rates during processing. Read and write disk I/O rates were measured during out-of-core processing of chamfer distance transformations, thinning, and geodesic distance computations. Note the different scales. The numbers indicate different stages of the algorithms, that are discussed in more detail in the main text.

Thinning is I/O bound. It starts with a scan over the input distance transformation to find the maximum distance value. In the diagram, indicated by (c), only read I/O takes place. The next step, indicated by (d), is to initialize output data (write 1 Byte per voxel) by scanning through the input once (read one Byte per voxel). Read and write rates are approximately equal. During (e) actual thinning takes place. Overall 3 Bytes (2 Bytes distance transformation, 1 Byte intermediate result) are read and 1 Byte of intermediate result is written back. As apparent in the diagram the read rate is three times higher than the write rate. A rough estimate of the runtime is given by dividing three times the number of voxels by the average read rate. This results to $1280^3 \cdot 3 / 6000000 \approx 1000$ seconds, which is in good concordance with the measured runtime of approximately 1300 seconds for the actual thinning.

Computation of the geodesic distance is, in contrast with the other two algorithms, bound by computations on the CPU. The average I/O rates are lower than in the previous measurements. The read rate is higher than the write rate as a larger block of the input domain is read but only a small sub-block contains valid results and is written to disk. The first spike, indicated at (f), is caused by initialization of boundaries of the result data set. It is not related to the main computations.

In summary, block-wise processing (ignoring I/O) only slightly increases runtimes. Chamfer distance computation and thinning tend to be I/O bound, while computation of the geodesic distance tend to be CPU bound.

Summary

- When data exceeds the size of available main memory, external memory algorithms must be used for processing. Block-wise data transfer between disk and main memory causes highly non-uniform costs for data access. Known algorithms need to be analyzed and modified to run with minimal performance loss.
- Block-wise processing of data combined with a chunked data storage format are the key to efficient out-of-core processing of three-dimensional voxel data. Algorithms need to be adjusted to correctly handle block boundaries.
- An algorithms was presented to compute chamfer distance transformations in approximately eight block-wise scans for any input.
- Thinning and the geodesic distance based measure are computed with a data dependent overlap at the block boundaries. The maximum thickness of the object must be small enough to allow processing with a limited amount of main memory.
- Skeletons are converted to piecewise linear geometry in one block-wise scan.
- Chamfer distance computation, thinning, and geometry generation tend to be I/O bound. Computing the geodesic distance based measure tends to be CPU bound.
- Execution times are acceptable for batch processing but not for interactive use.

6 Applications

This chapter presents two applications of the previously discussed skeletonization methods. One-dimensional skeletons of micro-vascular networks are extracted and analyzed in the first application. The second example extracts two-dimensional skeletons of bone micro-architecture.

6.1 Reconstruction and Visualization of Micro-Vascular Networks

In Cassot et al. (2006) we introduced a novel method for analyzing the morphology of micro-vascular networks to support understanding of micro-circulation in the human brain. A geometric representation of the network is reconstructed starting from images acquired through confocal microscopy. Results can be visually inspected and form the basis for further morphological and topological analysis.

For a discussion of the biomedical background please refer to Cassot et al. (2006) and references therein. The background is only briefly summarized here: Many interesting biological challenges are tightly linked to the anatomy of the micro-vascular architecture. Examples are understanding the distribution of pressure and wall shear stress; transport and exchange of oxygen; regulation of blood flow; angiogenesis and remodeling; the blood flow response to physical or neural activity; and, subsequently, the interpretation of hemodynamically based functional imaging methods. Only three-dimensional methods are capable of providing reliable data on highly complex networks.

The following presentation focuses on the contribution of techniques and algorithms presented in this thesis. Some algorithms, which were developed in joint work, are also presented in Fouard et al. (2006); Fouard (2005); and Fouard et al. (2004).

In a first step, a three-dimensional image of the micro-vascular network is acquired in multiple blocks by confocal microscopy, pre-processed, and stitched to a single large volume stored on disk. Approximately 300 µm thick sections of indian ink-injected human brain tissue are scanned with a resolution of $1.2 \times 1.2 \times 3$ µm in multiple blocks (roughly 100), each $512 \times 512 \times 200$ voxels large (z-direction is oversampled with a slice distance of 1.4 µm). Each block is preprocessed: median filtering removes salt-and-pepper noise; gauss filtering smoothes the boundaries of the vessels; the drop of illumination in deeper slices is corrected for by scaling image values with a linear function in z. The positions of the blocks relative to each other are available from the imaging device with a precision of 5 µm. The blocks have an overlap of approximately 50 voxels at each border, which allows to automatically align and merge them to one large volume. The

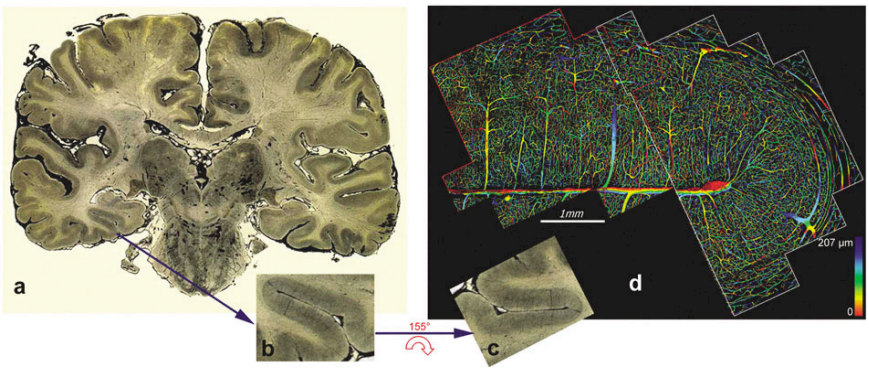


Figure 6.1: Data acquisition from a thick section of human brain tissue. (a) a complete slice of brain tissue; (b, c) the collateral sulcus in the temporal lobe; (d) depth coded projection of the zone reconstructed by confocal microscopy after aligning and merging sub-blocks.

imprecise location of the blocks is corrected by searching for a slightly shifted position that minimizes a correlation function of the grey values in the 50 voxel overlap. Afterwards, the blocks are sampled on one large structured grid, which is never loaded entirely into main memory. The resulting data sets have a size in the order of $6000 \times 6000 \times 200$. Figure 6.1 illustrates an input specimen and results after the described processing steps.

Except for few cases, which are discussed below, a simple threshold on the image data generates a voxel object representing the vessel network to which skeletonization methods are applied. External memory variants of distance transformations, thinning, and geometry generation as discussed in the previous chapters are used to extract a line representation of the vessels. All methods were integrated into the visualization system Amira (Stalling et al., 2005). The local thickness of the vessels is estimated by evaluating the distance transformation at each line vertex. Adjusted chamfer weights can be used to compute distance transformations on anisotropic grids (see Fouard and Malandain, 2005). Doing so may save space (and time) compared to resampling on an isotropic grid with identical voxel size in each direction. Postprocessing the inital graph results in a representation as vessel segments connected at braching nodes. Filling with indian ink is incomplete in some case and vessels are thus disconnected in the image data. Hence, end-points are preserved during thinning.

The resulting line sets contain vertices in the order of millions, which can be loaded completely into main memory. Figure 6.2 displays the results for a sub-volume. Rendering thin lines with thickness coded as color is possible at interactive frame-rates. Displaying thickness by truncated cones achieves interactive frame-rates only for subsets of the network.

The network structure is best analyzed in two different parts: (1) main vessel trees branching and (2) the net-like capillaries. Those two can be extracted from the reconstructed graph based on the diameter and manually identified root points of the vessel trees; Figure 6.3 depicts an example of such trees. In addition, various morphological measures, like density, orientation, and inter-capillary distance of the vessels and the frequency and space distribution of these measures can be quantitatively

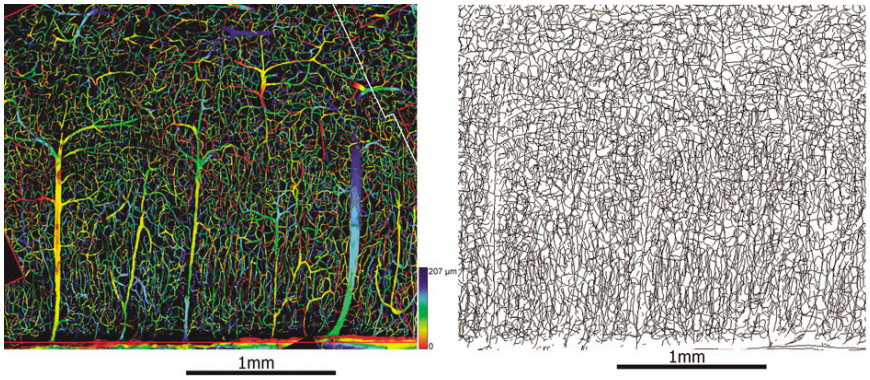


Figure 6.2: Part of a reconstructed micro-vascular network. A color-coded projection of the merged image data is displayed on the left and the corresponding skeleton on the right.

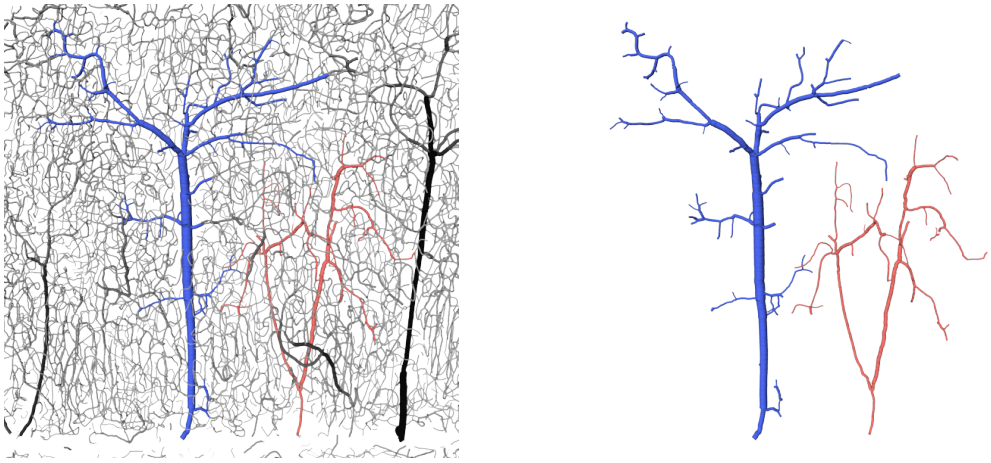


Figure 6.3: A vene tree and an artery tree extracted from a larger network of vessels. The vene tree is colored in blue; the artery tree in red. The left image displays the extracted vessel trees in the context of the surrounding network, which is removed on the right.

analyzed.

Interactive visualization was crucial in establishing the described methods. For example, the resulting reconstruction was validated by visual comparisons with the original image data. A projection of the image data side by side with the reconstructed network and three-dimensional rendering together with iso-surfaces of the image data proved particularly useful (see Figure 6.4). Good visual matching was observed throughout the network.

Some statistical errors in the reconstruction need to be dealt with. In few cases (approximately 4%) segments end unconnected, which is caused by missing signal in the input data. Incomplete filling of the vessels with ink was identified as the primary reason. For statistical analysis the error introduced by incomplete reconstruction is considered acceptable. Nonetheless, manual editing of the networks was explored as a second option.

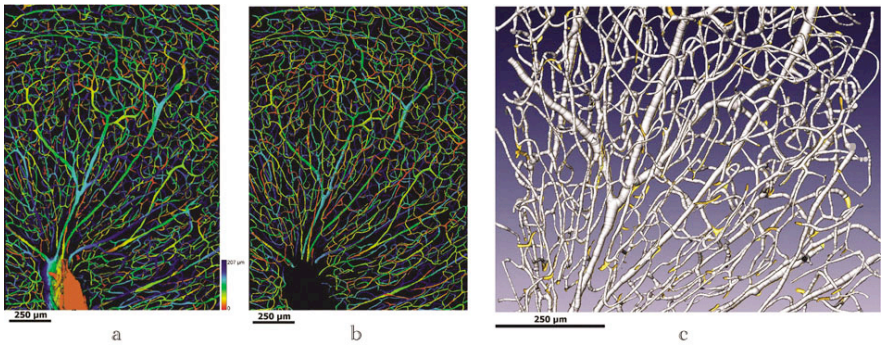


Figure 6.4: Comparison of the reconstructed network with the original image data. (a) projection view of the image data; (b) depth-coded visualization of the reconstructed line-set; (c) three-dimensional visualization of the line-set (white cylinders) and the original data (yellow iso-surface).

The system guides a user to end-points of unconnected segments and allows to select and connect nearby points. Segments can also be deleted or moved. If a highly accurate model of the network is needed, for example in simulations, the time needed for manual validation and editing may be accepted to eliminate all errors.

The results may also be improved by interacting at an early stage of the processing pipeline. Main vessels are often only partially filled. As a consequence, their representation in the voxel object is incomplete and diameters cannot be correctly estimated. Amira's image segmentation editor can be used to manually edit the binary image and fill the missing parts of the vessels. The network reconstruction now successfully extracts the required information. Similarly, uninteresting structures, for example the main vessel running in the center of the network, can be manually deleted.

The introduced methods are a first step towards a comprehensive, computerized analysis of massive micro-vascular networks in three dimensions and towards developing a complete model of micro-circulation. We are now able to create computer assisted visualizations of the vessel anatomy in huge vessel networks—a task that was manually performed in the past (see Figure 6.5 for an example). In addition, the computer assisted method supports a detailed quantitative analysis, which was not possible before. External memory algorithms allow to analyze huge micro-vascular networks on commodity machines usually available in research laboratories.

6.2 Visualization and Analysis of Bone Micro-Architecture

The applications discussed next are related to a research project on bone structure with the main objective of establishing a precise diagnostic method for quantifying alterations in the structural composition of human trabecular bone. A survey over the project is given in ESA MAP Team AO-99-030 (2005). Details about novel quantification methods of

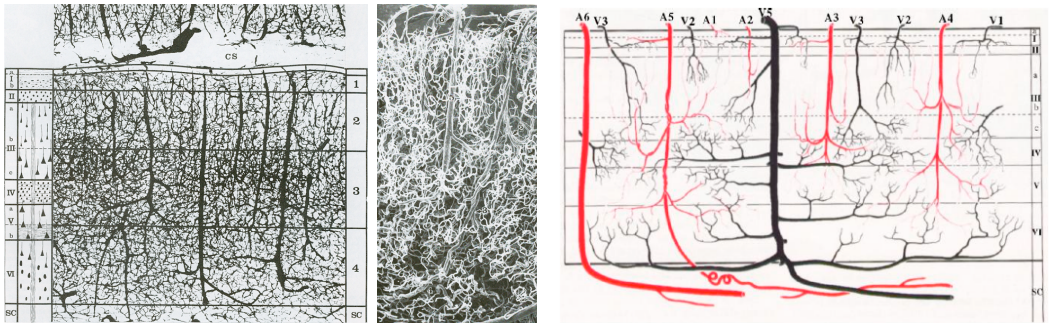


Figure 6.5: Microscope views and a hand-drawn illustration of a vascular network. Two views through a microscope (left) and a hand-drawn illustration (right) of the vessel anatomy in the human brain, replicated from Duvernoy et al. (1981).

two-dimensional Computed Tomography (CT) and three-dimensional micro-CT (µCT) images can be found in Saparin et al. (2005) and Zaikin et al. (2005). Details on data management and remote data access are presented in Prohaska et al. (2004); Prohaska and Hutanu (2005); and Kähler et al. (2005).

The following presentation restricts itself to the application of skeletonization methods to µCT data sets acquired during the project. The team collected a large number of µCT data sets. Proximal tibial bone biopsies were µCT-scanned in a Scanco µCT 40 scanner at a volume size of 20 µm (Thomsen et al., 2005). Entire vertebral bodies were µCT-scanned in a Scanco µCT 80 scanner before and after failure load testing at a voxel size of 38 µm. The typical size of an acquired image is 2048 × 2048 × 1000 voxels at 2 Bytes, resulting in roughly 8 GBytes of data per specimen. Almost 100 such data sets were acquired.

Input data was segmented to a cavity-free voxel object representing the calcified bone tissue. This was achieved by filtering (median, gauss) the input data and separating bone from background by a simple threshold. In general, bone structure is assumed to be free of cavities (Odgaard and Gundersen, 1993). Thus a connected component labeling algorithm was applied to detect cavities and relabel them to be part of the bone.

Visualization of Bone Architecture

The skeletonization algorithms presented in this thesis can now be used to compute skeletons. Skeleton renderings are superior to iso-surface renderings in illustrating the prevailing architectural orientation of plates. A reason is that skeletons do not enclose a volume and allow to look through the structure whereas the view is blocked in iso-surface renderings. For example, Figure 6.7 displays extracted skeletons side by side with iso-surfaces. The depicted specimen is a tibial bone biopsy, which was scanned at a voxel size of 20 µm resulting in an image of size 450 × 450 × 1400. The cortical shell is located at the top. Two orientations rotated 90 degrees to each other are displayed. The bone structure is organized in plates with a prevailing orientation. This becomes most apparent on the left-hand side, where the viewing direction is chosen to be parallel with the bone

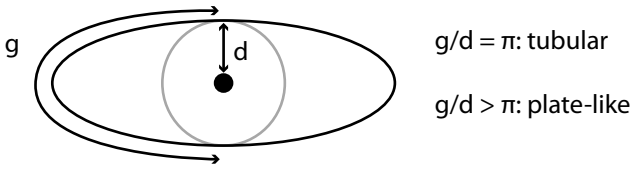
plates.

The skeleton also tends to better preserve the structural architecture after triangle decimation compared to the iso-surface. An iso-surface »wastes« triangles to enclose a volume whereas skeletons require only one layer of triangles per plate. The bottom row of Figure 6.7 displays aggressively decimated triangle meshes, to approximately 1% of the original triangle count. The skeleton more closely resembles visually the rendering generated with a high-resolution triangle mesh.

The distance transformation inside the object and the geodesic distance along the boundary provide useful measures for highlighting different structures. The distance transformation gives an estimation of the local thickness when evaluated at the skeleton location. For example, Figure 6.8 displays the same biopsy used above, now colored by the local thickness. In Figure 6.9 geometric primitives were filtered and only structures with a low value of the distance transformation (bottom) respectively high value (top) are retained. Filtering the skeleton provides an easy way to focus on various aspects of the structure. An iso-surface is not capable of displaying thickness in the same straight forward way because the distance transformation needs be evaluated at the center of the object but the iso-surface is defined at the boundary.

A simple measure indicating plate-like structures can be defined based on the geodesic distance. Some care is needed because the local thickness also contributes to the geodesic distance, which can be corrected for by using the ratio of the geodesic distance to the distance transformation:

Figure 6.6: Measuring deviation from the tubular case.



The ratio g/d increases the more the object deviates from a tubular structure. Figure 6.10 depicts this measure color-coded on the skeleton. In Figure 6.11 filtering was applied to focus on plate-like parts (see top right).

The opposite, filtering to highlight tube-like parts, does not work equally well. Towards the boundary of plate-like structures the ratio of geodesic distance to distance transformation behaves as if the structure was tubular. Because the ratio is evaluated on a purely local basis it fails to distinguish between points at the boundary of a larger plate-like structure and points in the center of a tubular structure.

To reduce the amount of geometric primitives further, the two-dimensional skeleton can be abandoned and replaced by a one-dimensional skeleton. To do so, thinning is guided by the geodesic distance as described in Section 2.3.3. The resulting line skeleton is centered withing the plate-like structures. The local thickness and the ratio with the geodesic distance, can again be used to highlight different structures. Figure 6.12 and Figure 6.13 depict the curve skeleton as circular sprites color-coded with the deviation from a tubular structure. The same structures can be revealed with the two-dimensional and with the one-dimensional skeleton (compare Figure 6.11 and Figure 6.13).

Skeletons can also serve as a basis for a decomposition of the structure into rods and plates. Using structural decomposition in bone structure analysis is proposed in Bonnassie et al. (2003) and Stauber and Müller (2006). Figure 6.14 displays an example based on the methods presented in this thesis applied to a bone sub-volume. A one-dimensional skeleton was computed in the grid-cell model and cut at its branching points. Each edge was colored and these colors were propagated to the volume such that every point is colored with the color of the nearest point on the curve skeleton. The coloring is evaluated on the skeleton to induce a decomposition into structural elements.

Analyzing Architectural Alterations

Another objective was to find structural alterations in bone micro-structure. The analysis is base on two μCT scans of human vertebral bodies acquired before and after compression testing.

In a first step the two scans need to be placed in a common coordinate system. The visualization system Amira (Stalling et al., 2005) is used to manually select matching reference points (landmarks) on an iso-surface representation of the bone micro-architecture. Figure 6.15 depicts a low resolution volume rendering of a μCT-scan, an iso-surface rendering, and selected landmarks. From the landmarks a rigid transformation is computed and applied to one of the two data sets.

Afterwards, both images are segmented, skeletonized, and the distance of the original skeleton to its counterpart after compression testing is computed. Figure 6.16 illustrates the process. A thick slice of the vertebra is selected and rotated to provide a side view. The two-dimensional skeleton provides the overall context. Now the one-dimensional skeleton of the original structure is integrated in the view. Skeleton points with a high distance to the structure after compression testing are rendered as circular sprites. All other skeleton points are suppressed. The distance is computed by a distance transformation seeded at the skeleton.

A sub-volume that shows lots of changes is inspected in more detail. The mixed dimensional skeleton is selected for display and part of the image data of the compressed bone is loaded and displayed as an iso-surface (bottom center). At the right the original structure is depicted. In the center of the image a micro-fracture occured. After compression testing the structure in the center is missing.

Analyzing such changes in micro-architecture is ongoing research and will hopefully provide insight in the mechanisms of bone fracture. For example Thurner et al. (2006) explore changes based on synchrotron CT images acquired under loading conditions. The techniques presented in this thesis provide advanced tools to analyze such structural changes and are a basis for further bio-medical investigations.

Summary

- Starting from massive image data acquired by confocal microscopy of human brain tissue, skeletonization was applied to reconstruct large micro-vascular networks as graphs of branching nodes connected by edges representing the vessels. Information on geometric location and local thickness is associated with the edges. The networks are used for visualization and quantification of the micro-vascular networks.
- Micro-bone architecture forms a network built of rod-like and plate-like parts. Micro-CT scanners provide detailed three-dimensional images of this architecture. Skeletonization was applied to construct one- and two-dimensional skeletons used in visualization of architecture and analysis of structural alterations.

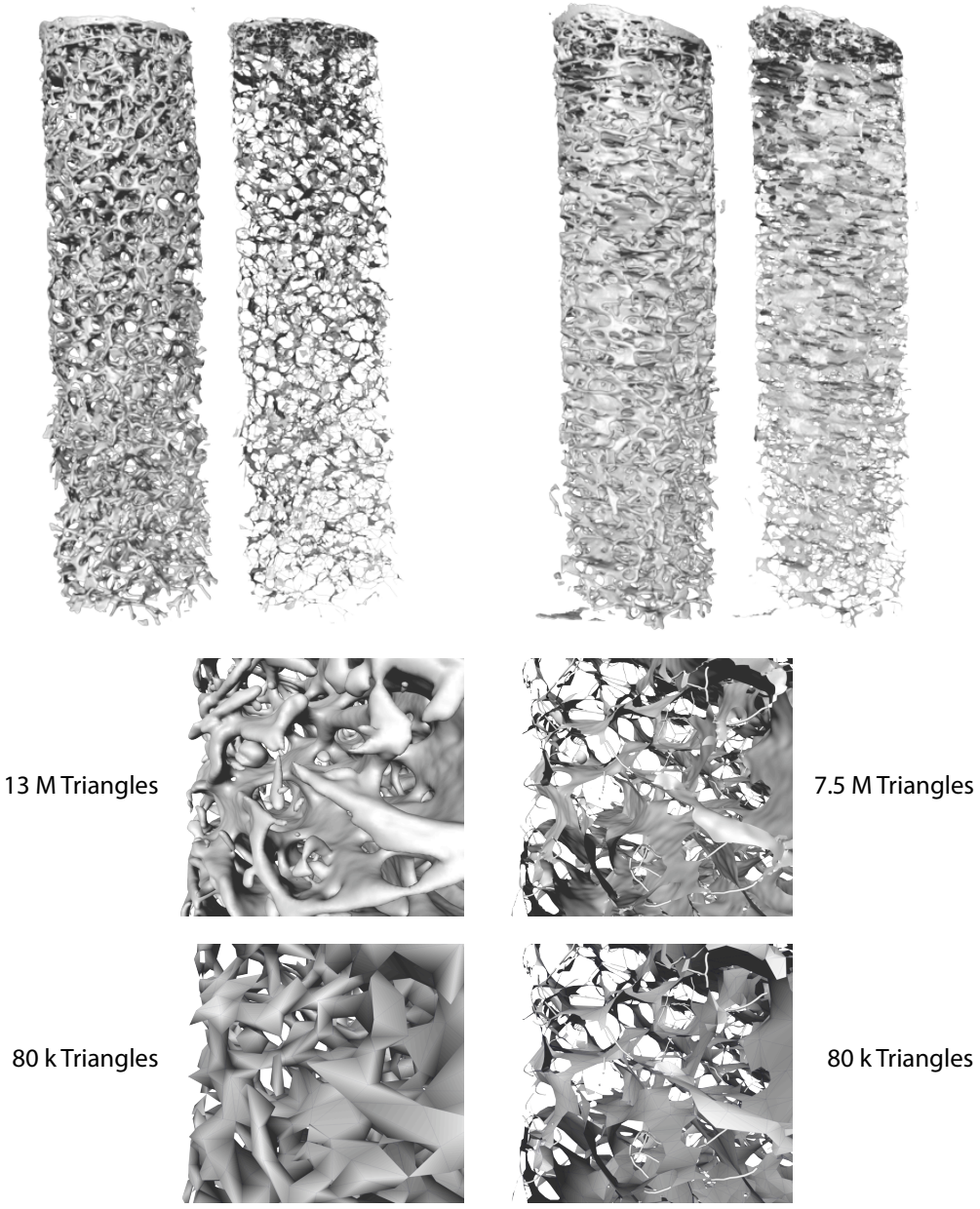


Figure 6.7: Iso-surfaces and skeletons of a bone biopsy. The iso-surfaces is displayed on the left in each pair; the skeletons on the right. In the bottom row the number of triangles was decimated as indicated in the figure. All other images use the higher triangle count.

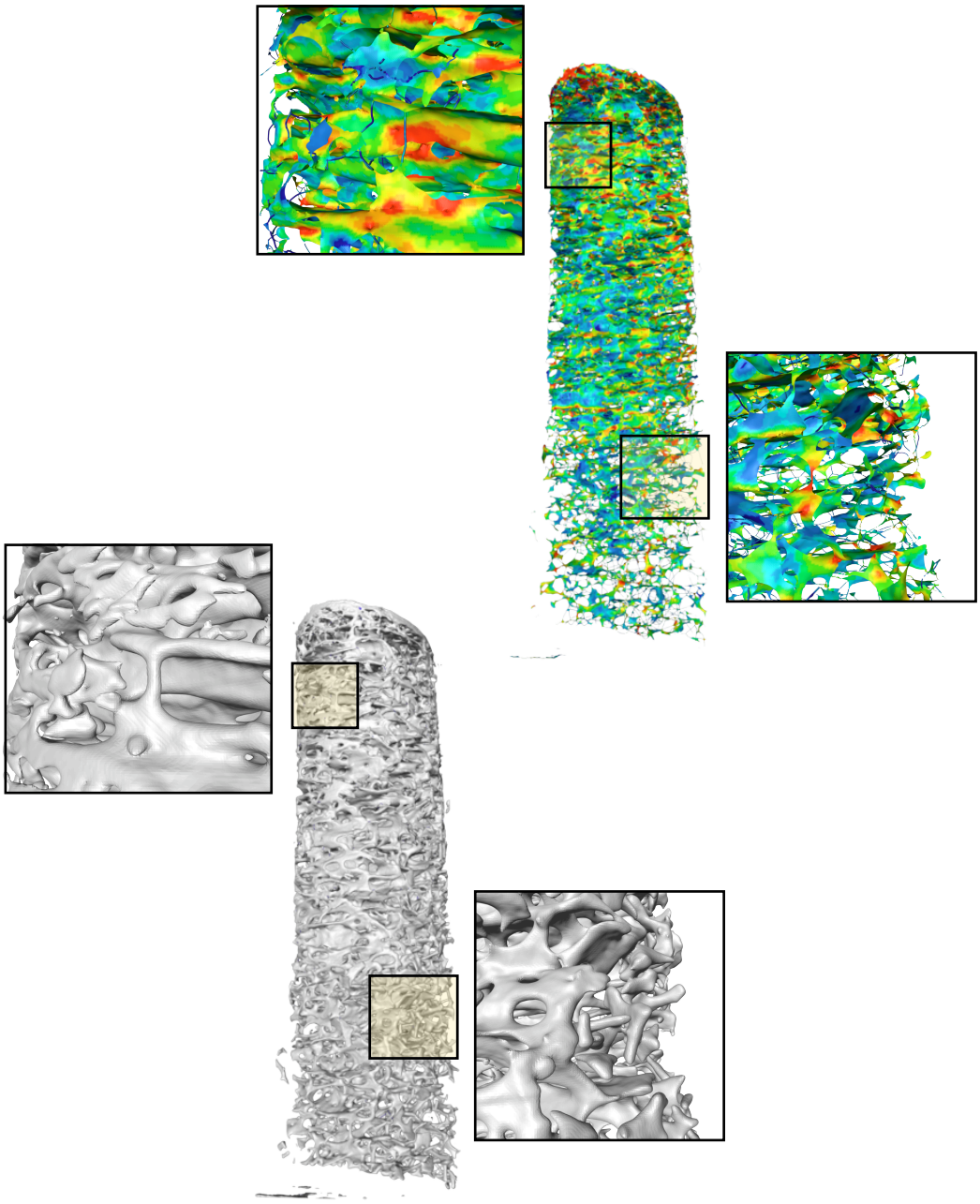


Figure 6.8: Local thickness color-coded on a skeleton. The local thicknessof the structure displayed in the bottom left is color coded on the skeleton at the top right. Blue indicates low values; red indicates high values.

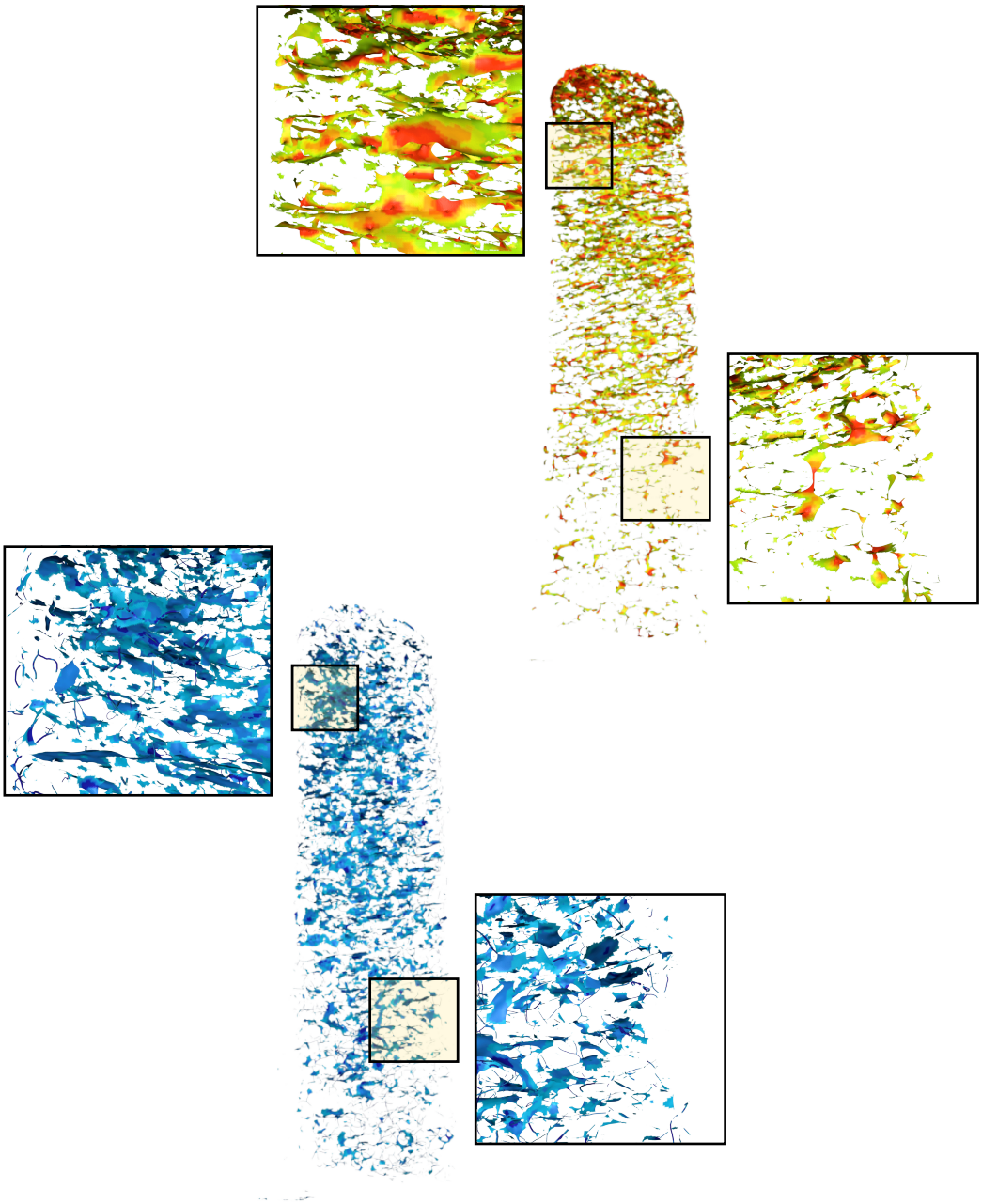


Figure 6.9: Skeleton filtered by local thickness. The structure is filtered for thin structures (bottom) and thick structures (top).

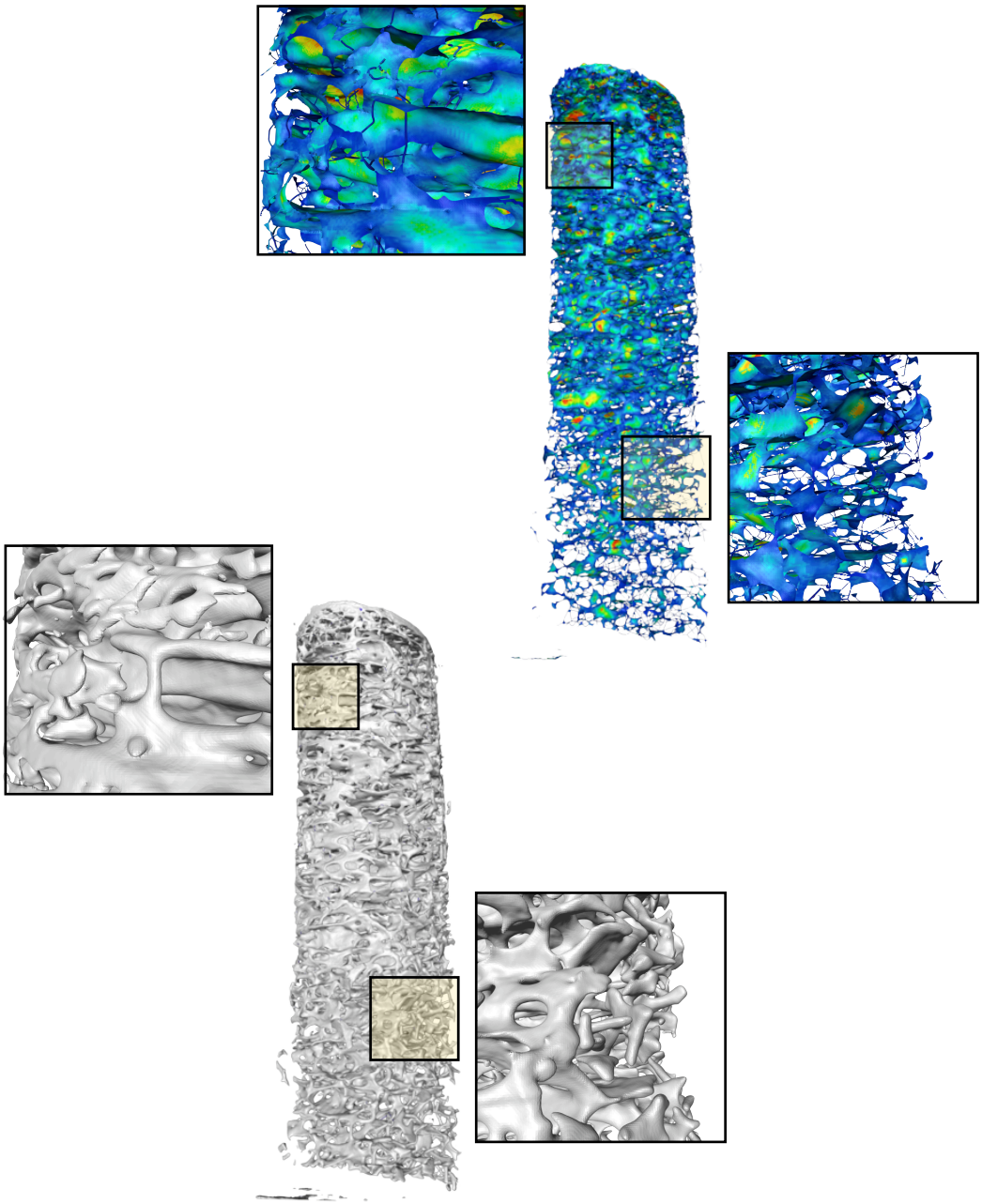


Figure 6.10: Deviation from tubular structure color-coded on a skeleton. The deviation from tubular structure is color-coded on the skeleton. Blue indicates low deviation; red indicates high deviation.

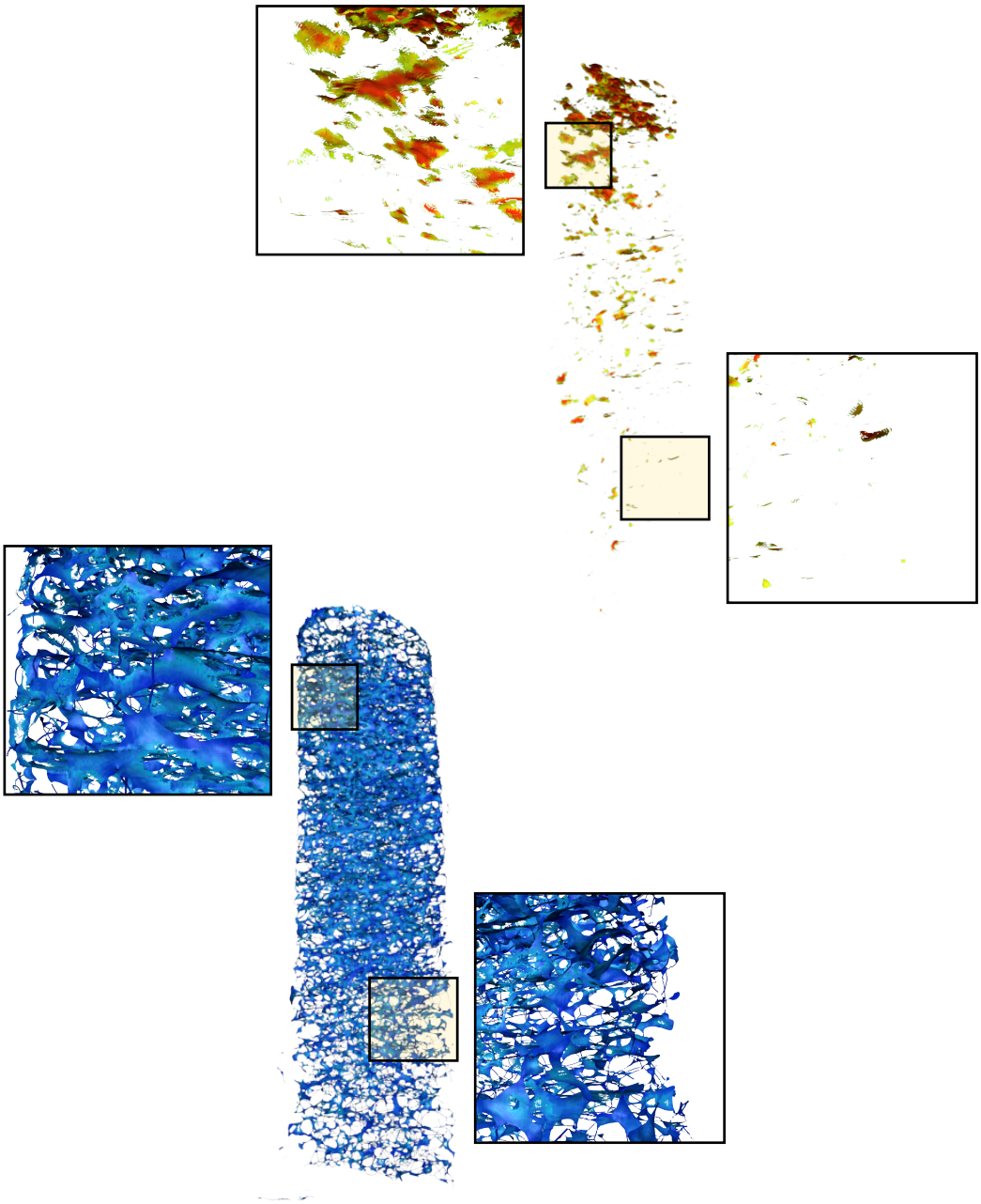


Figure 6.11: Skeleton filtered by deviation from tubular structure. The structure is filtered according to the deviation from a tubular structure. Parts with low deviation are displayed in the bottom left; parts with hight deviation in the top right.

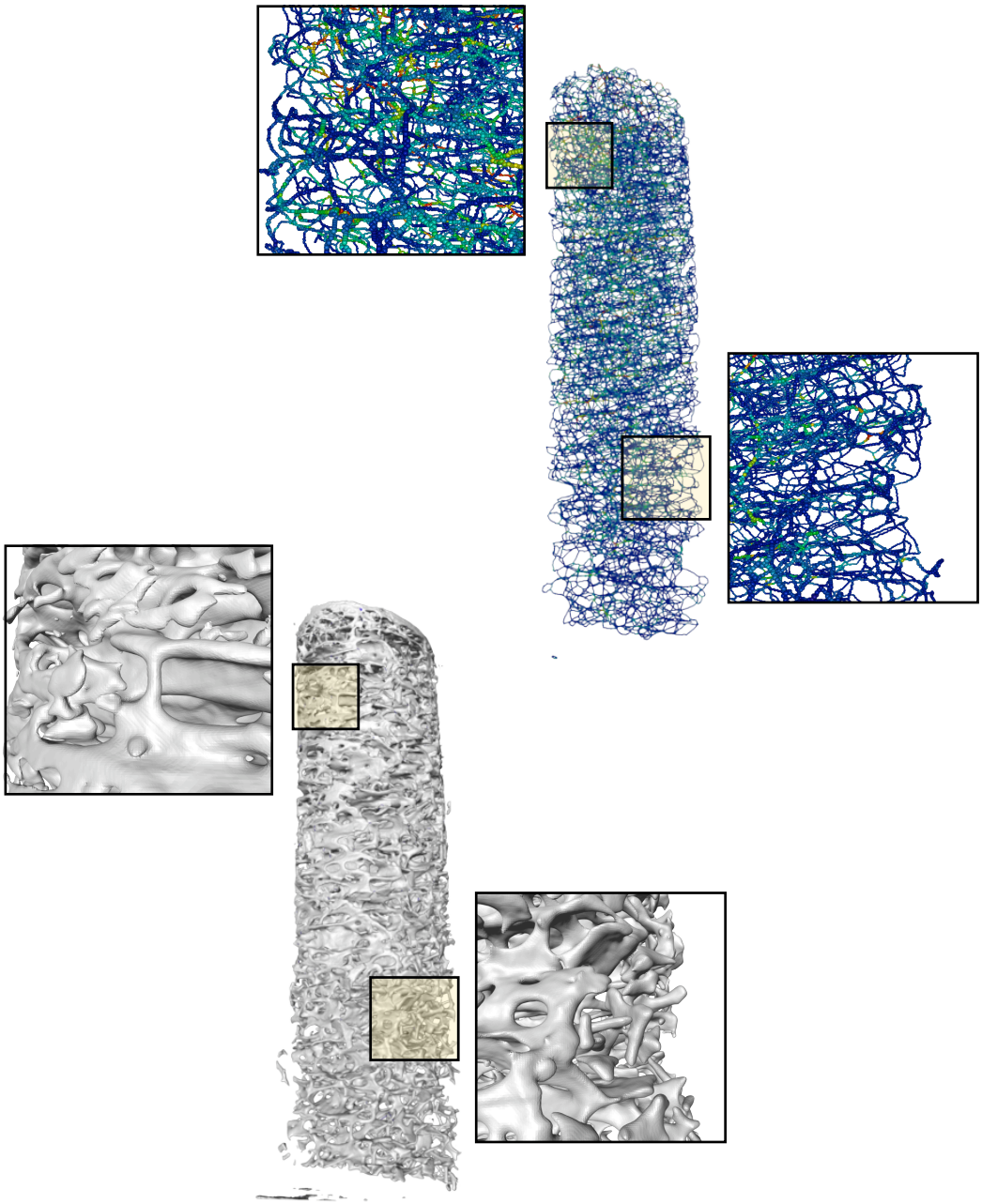


Figure 6.12: Deviation from tubular structure color-coded on a skeleton. The one-dimensional skeleton is rendered as circular sprites placed at the vertices of the skeleton.

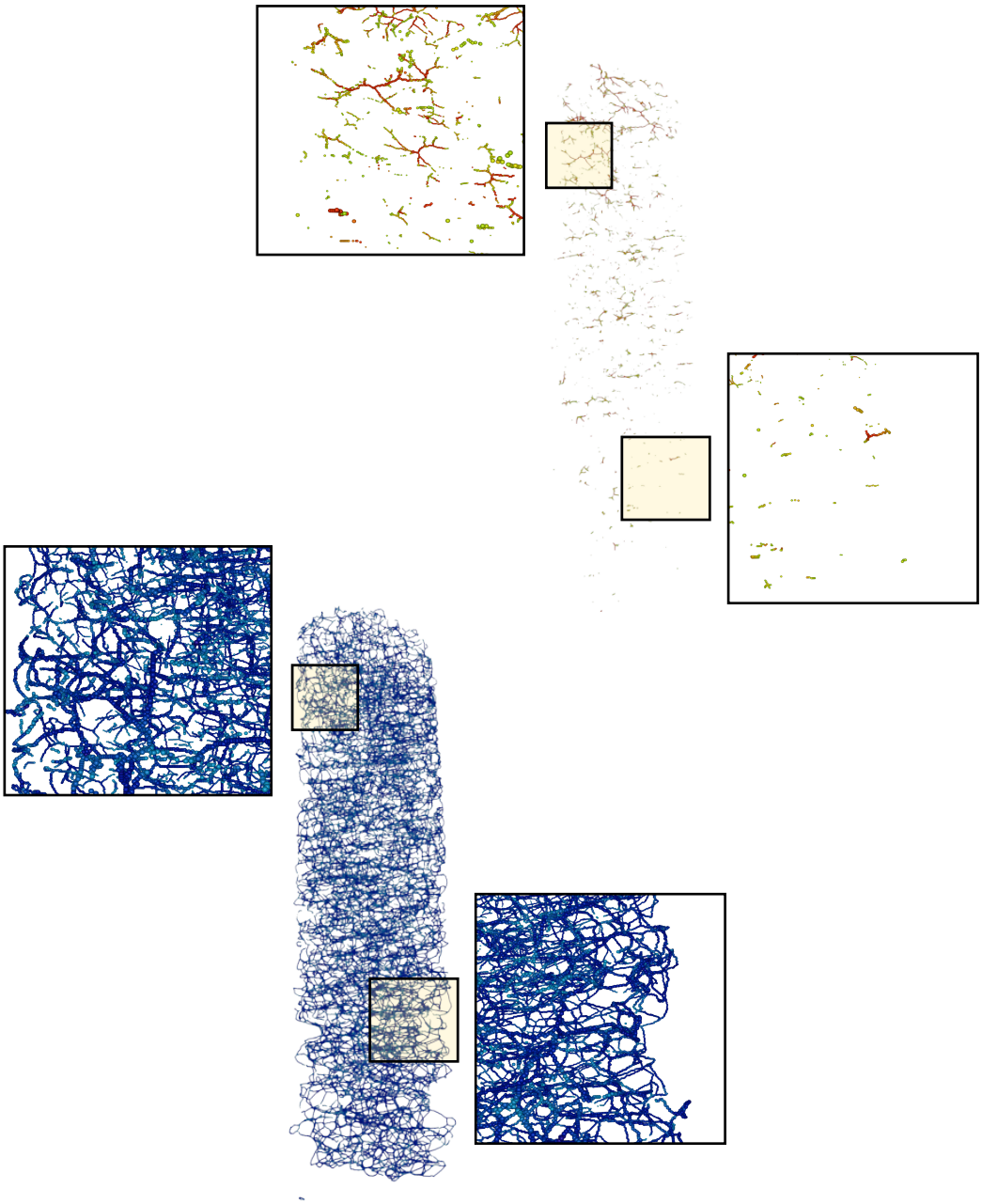


Figure 6.13: One-dimensional skeleton filtered by deviation from tubular structure. The skeleton is filtered to display only tubular structures (bottom) and plate-like structures (top).

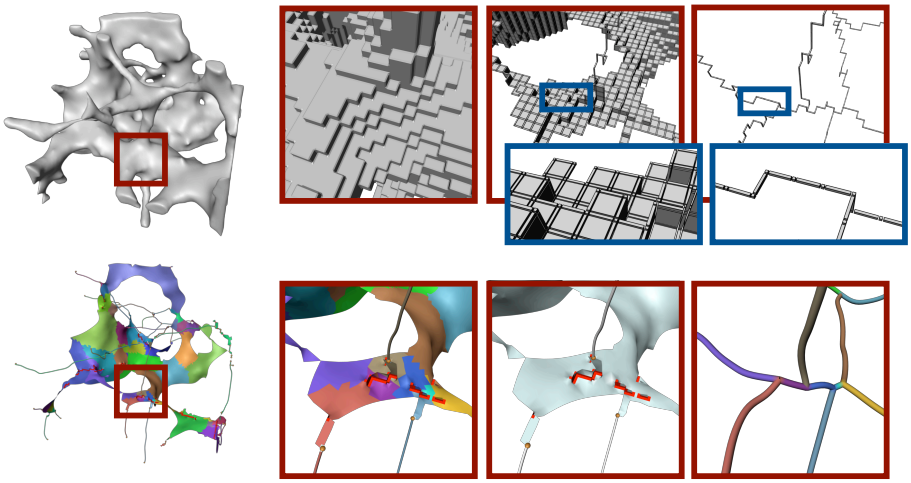


Figure 6.14: Bone structure decomposed into structural elements. An iso-surface rendering, the voxel object representing bone, a two-dimensional grid cell skeleton, and a one-dimensional grid cell skeleton are displayed in the top row from left to right. The bottom row displays geometric representations of skeletons. At the very right, the one-dimensional skeleton is represented as a graph with colored edges. These colors are propagated onto the two-dimensional skeleton (second from right) inducing a decomposition into elements (left). Junctions are indicated in red.

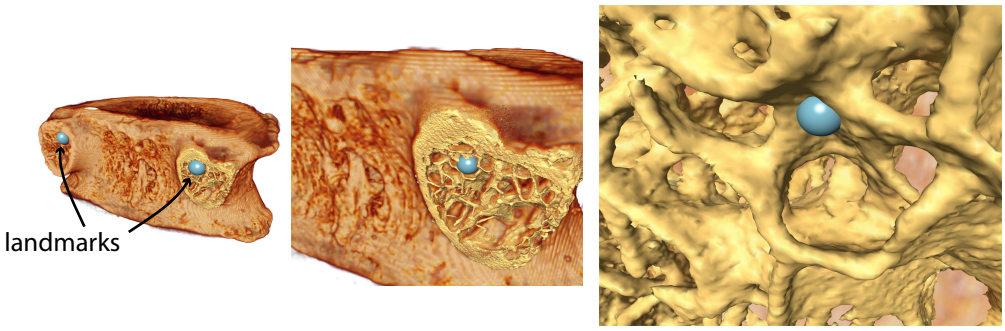


Figure 6.15: Landmarks on a human vertebral body used for registration. A volume rendering of a µCT-scan of a vertebral body, an iso-surface rendering, and selected landmarks are depicted. The landmarks are used to align the image data set with a second scan after compression testing.

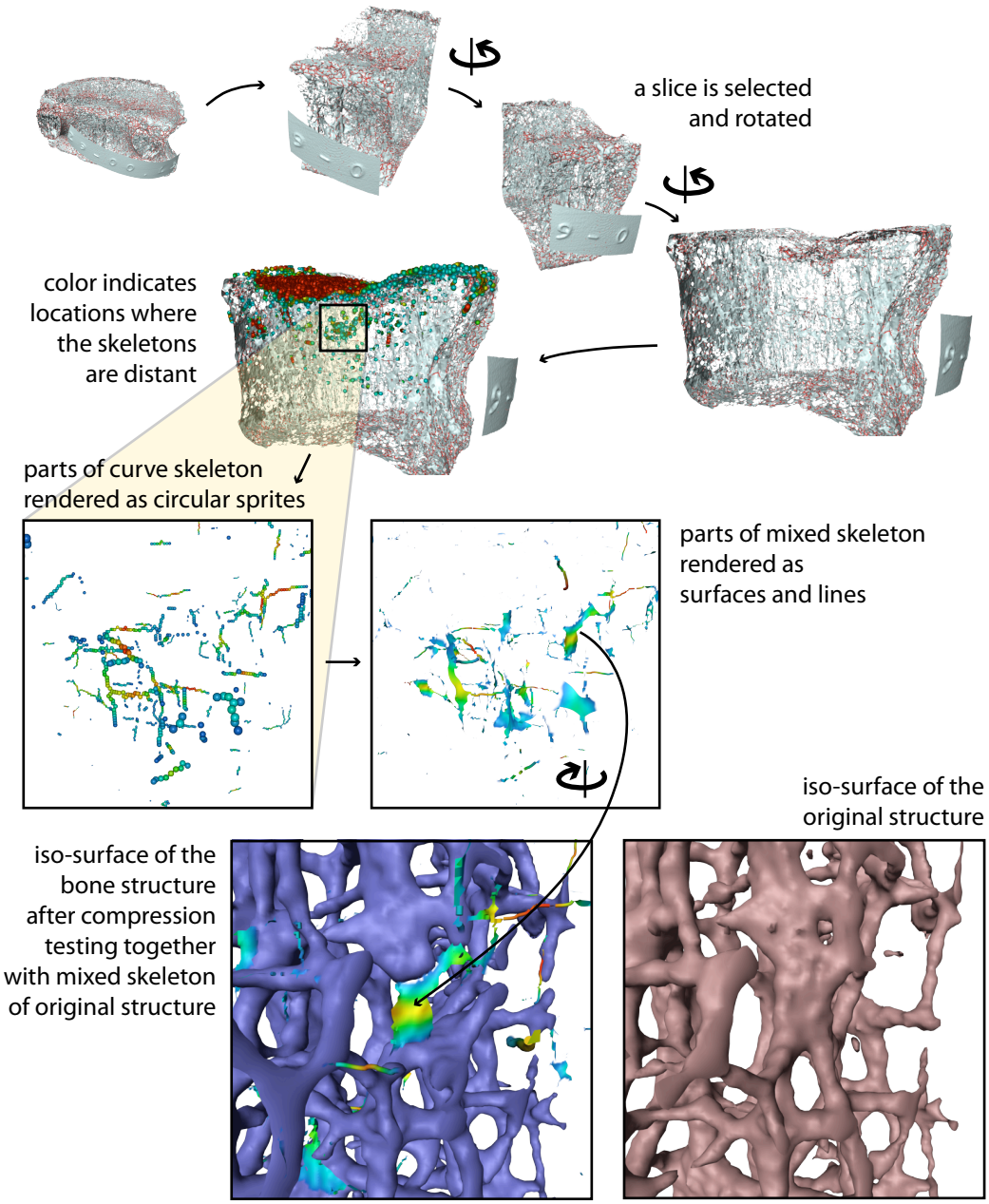


Figure 6.16: Search for micro-cracks in a human vertebral body. The two-dimensional skeleton gives an overview (red spots indicate junctions in the skeleton). A slice is selected and rotated to a side view. Locations with change are highlighted by circular sprites. An interesting sub-volume is inspected in more detail. At the bottom the structure after compression testing (blue) is displayed together with the skeleton of the original structure. In the center of the image a micro-fracture occurred. The original structure is displayed in red at the right.

7 Conclusions

This work presented skeletonization algorithms for analyzing and visualizing massive voxel objects with complex, network-like architecture; even if their size exceeds the amount of main memory and external memory algorithms are needed. The presented results help to visualize and analyze huge image data, which become abundant in science. Such image data often contain well defined objects that can be segmented resulting in voxel objects of massive size. Those voxel objects are the input to the presented algorithms.

The question »How to compute skeletons of massive voxel objects?« was solved for computing voxel curve skeletons by external memory algorithms for chamfer distance transformations and for distance ordered homotopic thinning, both requiring a constant number of scans over the data. A theoretical analysis predicted approximately four times the execution time of the standard algorithm for chamfer distance transformation and approximately the same execution time as the standard algorithm for thinning when ignoring I/O. Timing confirmed these predictions and revealed that the execution is I/O-bound on a standard workstation.

Chapter 5 discussed how a constant number of block-wise scans over the data stored in chuncked layout on disk is I/O-optimal (Vitter, 2001). The applied block-wise decomposition schemes are quite simple. Yet it was necessary to carefully analyze effects of block boundaries and to adapt the original algorithms to compute globally correct results when using a decomposition scheme with overlapping blocks. Many other image processing algorithm that can be decomposed into block-wise processing should also execute I/O-optimal when based on a chuncked storage layout.

As a general principle we can conclude that careful analysis of existing voxel based algorithms is superior to naive block-wise processing ignoring boundary effects, as for example proposed for thinning by Vossepoel et al. (1997) or Pakura et al. (2002). The upfront cost of devising modified algorithms is rewarded by data independent, globally correct results. In many cases the upfront cost should not be too high as a simple block-wise decomposition should often be sufficient.

In practice, we are now able to compute distance transformations and thinning on voxel objects of any size in reasonable time, as long as data fits on a disk. Because computations are I/O-bound and take too long to be effective for interactive use, the following recommendations should be followed in practice: 1) use standard algorithms as long as data fits into main memory and switch to external memory processing only if needed; and 2) choose all needed specifications based on a small sub-volume processed in main memory and execute all external memory processing as a batch job. Chapter 6 successfully applied the methods to visualization and analysis of vascular networks.

Future work should consider the end-to-end problem of computing the geometric representation of the skeleton from the original image data avoiding I/O all together,

except for one required scan over the input. Isenburg and Lindstrom (2005) explore a promising idea for geometry processing that may serve as a starting point. Their idea of spatial finalization in a stream processing pipeline could be transferred to processing of voxel objects and provide a foundation for practical algorithms solving the end-to-end problem for well formed input data requiring less I/O than the methods presented here.

Chapter 3 answered the question »How to render voxel skeletons?« by retracting voxels to homotopy equivalent piecewise linear geometry, that is triangles and piecewise straight lines, for rendering. The fact that a certain voxel configuration represents a skeleton was explicitly considered and justified the retraction of voxels to geometry. As a general principle the type of structure a voxel configuration represents should be taken into account when devising solutions. For voxel skeletons, a recommendation is to specifically consider the topological implications instead of devising solutions for generic voxel configurations.

Problems caused by naively tracing voxels based on the 26-adjacency relation are avoided by respecting homotopy equivalence through the retraction of voxels. »Spurious loops« at line junctions, for example, do not occur. Explicitly considering topology explains heuristics used for post-processing of voxel skeletons, like for example collapsing adjacent junction voxels to a single point.

Note however, two questions regarding the retraction scheme should be answered in the future: 1) The retraction scheme induces a data dependent adjacency relation, which may differ from the commonly used 6-, 18-, or 26-adjacency relations. The implications of this fact needs further research. 2) It was left open how to resolve the »solid configuration«, where all eight voxels of a dual cube belong to the skeleton. Obviously, this configuration can not be locally retracted to a lower dimensional representation because there is no interface surface to the background to start the retraction from. A solution may be to construct geometry in a larger neighborhood.

The question »how to render voxel skeletons?« is also related to the problem of »how to represent one-dimensional and two-dimensional structures and junctions by voxels?«, which was addressed by the discussion of thinning in the grid cell model. Section 2.3.1 revealed clear advantages of decomposing a voxel into its vertices, edges, faces and the enclosed volume opposed to treating voxels as indecomposable points with an adjacency relation as in the grid point model. The main reason of this advantage is that the decomposition forms a topological cell complex, which is able to resolve topological configurations unambiguously.

An implication is that the grid cell model is superior to the grid point model for algorithms that need detailed control of topology. This is, for example, the case for skeletonization. A counter example is the chamfer distance transformation computing the length of the shortest path to the background. Here, the grid point model with the 26-adjacency relation is better suited because the edges to 26-neighbors at each voxel better approximate the euclidian distance than the six main directions parallel to the coordinate axes of the grid cell model. Note also, the grid cell model needs eight time the storage space of the grid cell model, which may limit its practical use if memory is scarce. In conclusion, considering to transfer known algorithms from grid points to grid cells may be beneficial if topology is at the heart of the algorithm. But not all algorithms

benefit, and some may be better based on the grid point model.

An interesting opportunity for future research is considering grid cells built from a different grid type to reduce the number of cells meeting at a lower dimensional cell to the minimal number induced by generic point locations. The number of faces meeting at one edge should be reduced from 4 to 3 and the number of edges meeting at a point from 6 to 4. The reduced number of cells should avoid degenerate cases as for example non-manifold self intersections of surfaces. The body centered cubic grid is a good candidate for starting further investigations.

Section 2.3.2 introduced a noise-insensitive measure based on the geodesic distance along the boundary as a solution to the problem of computing skeletons stable under variations of the object's boundary through, for example, noise. Experiments confirmed that the measure induces a family of skeletons graded by relevance. With increasing relevance value only the most prominent plate-like parts of the object survive in the skeleton, which is centered within the plates. The skeletons are not necessarily homotopy equivalent with the original object. A combination with thinning was presented, which allows to establish topological guarantees and to compute line skeletons directly from objects with plate-like parts.

The geodesic distance based measure has some advantages compared to other methods but is computational expensive. In contrast to angle based methods, like discussed by Malandain and Fernández-Vidal (1998), the proposed measure is able to ignore small variations of the boundary while retaining thin structures of the object that are geometrically important. Resulting skeletons directly represent only the most important structures while other methods often require pruning as a post-processing step. The quadratic worst case runtime, needed to compute the geodesic distance of all required pairs of points, is a major drawback.

The proposed measure seems to be well suited to define one-dimensional skeletons directly from objects containing plate-like structures. The main reason is that the geodesic distance seems to avoid two-dimensional plateaus with the same value, which cause difficulties in distance transformations. Thinning can therefore be effectively guided towards one-dimensional structures centered within plates. Recently, Dey and Sun (2006) discussed a very similar approach and proposed a definition of curve skeletons based on the geodesic distance. Loosely spoken, the geodesic distance seems to encode more information about the global shape of the object's boundary than a distance transformation does. Theoretical investigations of this experimentally observed property should be conducted in the future.

The geodesic distance based measure proved to be a useful tool for the visualization of plate-like structures as discussed in Chapter 6. In practice the geodesic distance based skeletonization is the recommended method if time for computing it can be afforded. A combination with thinning is needed if guaranteed homotopy equivalence with the original object is needed. The resulting voxel skeleton locations are better centered in plate-like structures than results of thinning guided by distance transformations.

Figure 7.1 summarizes the various methods and the decisions needed to select the appropriate method for your problem. All methods start from a voxel object and not from the original image data. Therefore the quality of the segmentation, that classifies the image

data in object and background, is critical for the quality of the resulting skeleton. You also need a priori knowledge about the architecture of the voxel object to select a suitable skeletonization method. For a generic architecture the methods should be only applied with care. You need to decide if the resulting skeletons shall be homotopy equivalent with the object or if homotopy equivalence is not required. Depending on your a priori knowledge and your decisions, the results will be a one-dimensional cell complex that can be converted to a graph; a mixed one/two-dimensional cell complex, represented by triangulated surfaces and lines; or a pure two-dimensional cell complex, represented by a triangulated surface.

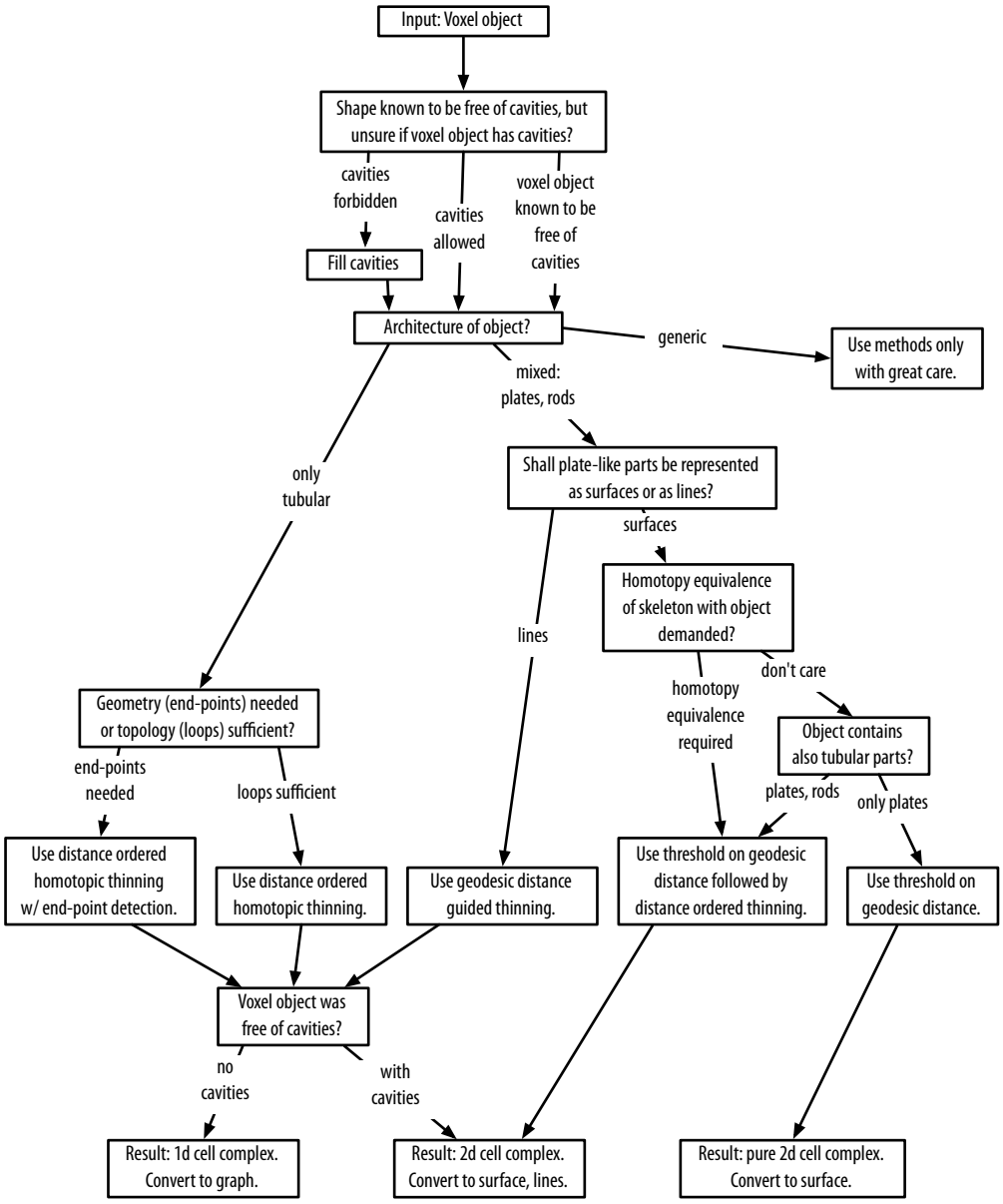


Figure 7.1: Selecting an appropriate skeletonization method. Decisions required to select an appropriate method for a specific combination of the type of architecture of the input voxel object and the desired properties of the resulting skeleton are summarized in the diagram.

7 Conclusions

Bibliography

G. H. Abdel-Hamid and Y.-H. Yang. Multiresolution skeletonization: An electrostatic field-based approach. In *Proc. IEEE Int. Conf. Image Processing (ICIP-94)*, pages 949–953. 1994.

N. Ahuja and J.-H. Chuang. Shape representation using a generalized potential field model. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 19(2):169–176, 1997.

P. S. Aleksandrov. *Combinatorial Topology*, volume 1. Graylock Press, 1956.

N. Amenta, S. Choi, and R. K. Kolluri. The power crust. In *Proc. 6th ACM Symp. Solid Modeling and Applications (SMA '01)*, pages 249–266. ACM Press, New York, NY, USA, 2001.

A. Amini, J. Jagadeesh, and B. McGinley. Three dimensional thinning of octree encoded objects. In *Proc. Fifteenth Southern Biomedical Engineering Conf.* 1996.

C. Arcelli and G. S. di Baja. A one-pass two-operation process to detect the skeletal pixels on the 4-distance transform. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 11(4):411–414, 1989.

D. Attali, J.-D. Boissonnat, and H. Edelsbrunner. Stability and computation of the medial axis — a state-of-the-art report. In T. Möller, B. Hamann, and B. Russell, editors, *Mathematical Foundations of Scientific Visualization, Computer Graphics, and Massive Data Exploration*. Springer-Verlag, 2004.

R. Ayala, E. Domínguez, A. R. Francés, and A. Quintero. Weak lighting functions and strong 26-surfaces. *Theoretical Computer Science*, 283(1):29–66, 2002.

S. Azernikov and A. Fischer. Efficient surface reconstruction method for distributed CAD. *Computer-Aided Design*, 36(9):799–808, 2004.

S. Azernikov, A. Miropolsky, and A. Fischer. Surface reconstruction of freeform objects based on multiresolution volumetric method. In *Proc. 8th ACM Symp. Solid Modeling and Applications (SMA '03)*, pages 115–126. 2003.

R. Bade, J. Haase, and B. Preim. Comparison of fundamental mesh smoothing algorithms for medical surface models. In *Simulation und Visualisierung*, pages 289–304. 2006.

D. C. Banks and S. A. Linton. Counting cases in marching cubes: Toward a generic algorithm for producing substitopes. In *Proc. IEEE Visualization '03*, pages 51–58. 2003.

D. C. Banks, S. A. Linton, and P. K. Stockmeyer. Counting cases in substitope algorithms. *IEEE Trans. Visualization and Computer Graphics*, 10(4):371–384, 2004.

R. D. Bergeron, P. J. Rhodes, T. M. Sparr, and X. Tang. Out of core visualization using iterator aware multidimensional prefetching. In *Proc. SPIE Visual Data Analysis '05*, volume 5669, pages 295–306. 2005.

T. M. Bernard and A. Manzanera. Improved low complexity fully parallel thinning algorithm. In *Proc. 10th Int. Conf. Image Analysis and Processing (ICIAP '99)*. 1999.

G. Bertrand and G. Malandain. A new characterization of three-dimensional simple points. *Pattern Recognition Lett.*, 15(2):169–175, 1994.

I. Bitter, A. E. Kaufman, and M. Sato. Penalized-distance volumetric skeleton algorithm. *IEEE Trans. Visualization and Computer Graphics*, 7(3):195–206, 2001.

I. Bitter, M. Sato, M. Bender, K. T. McDonnell, A. Kaufman, and M. Wan. CEASAR: Accurate and robust algorithm for extraction a smooth centerline. In *Proc. IEEE Visualization '00*, page 45. 2000.

H. Blum. A transformation for extracting new descriptors of shape. In W. Walthen-Dunn, editor, *Models for the Perception of Speech and Visual Form*, pages 362–380. MIT Press, Cambridge, MA, 1967.

V. G. Boltianskij and V. A. Efremovich. *Intuitive combinatory topology*. Springer, 2000.

A. Bonnassie, F. Peyrin, and D. Attali. A new method for analyzing local shape in three-dimensional images based on medial axis transformation. *IEEE Trans. Systems, Man, and Cybernetics, Part B*, 33(4):700–705, 2003.

G. Borgefors. Distance transformations in arbitrary dimensions. *Computer Vision, Graphics, and Image Processing*, 27:321–345, 1984.

G. Borgefors. Distance transformations in digital images. *Computer Vision, Graphics, and Image Processing*, 34(3):344–371, 1986.

G. Borgefors, I. Nyström, and G. S. di Baja. Computing skeletons in three dimensions. *Pattern Recognition*, 32:1225–1236, 1999.

S. Bouix. *Medial Surfaces*. Ph.D. thesis, School of Computer Science, McGill University, Montréal, 2003.

S. Bouix and K. Siddiqi. Divergence-based medial surfaces. In *Proc. 6th European Conf. Computer Vision (ECCV 2000)*, volume 1842 of *Lecture Notes in Computer Science*, page 603. Dublin, Ireland, 2000.

K. Brodlie, J. Brooke, M. Chen, D. Chrisnall, A. Fewings, C. Hughes, N. John, M. Jones, M. Riding, and N. Roard. Visual supercomputing—technologies, applications and challenges. In *Eurographics, State of the Art Report*. 2004.

C. A. Burbeck and S. M. Pizer. Object representation by cores: Identifying and representing primitive spatial regions. Technical Report TR94-048, University of North Carolina at Chapel Hill, Chapel Hill, NC, USA, 1994.

S. K. Card, J. D. Mackinlay, and B. Shneiderman. *Readings in Information Visualization: Using Vision to Think*. Morgan Kaufmann Publishers, San Francisco, 1999.

F. Cassot, F. Lauwers, C. Fouard, S. Prohaska, and V. Lauwer-Cances. A novel three-dimensional computer assisted method for a quantitative study of microvascular networks of the human cerebral cortex. *Microcirculation*, 13:15–32, 2006.

F. Chazal and A. Lieutier. The λ-medial axis. *Graphical Models*, 67(4):304–331, 2005.

J.-H. Chuang, C.-H. Tsai, and M.-C. Ko. Skeletonization of three-dimensional object using generalized potential field. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 22(11):1241–1251, 2000.

J. C. Ciria, A. de Miguel, E. Domínguez, A. R. Francés, and A. Quintero. A maximum set of (26, 6)-connected digital surfaces. In *Proc. Int. Work. Combinatorial Image Analysis (IWCIA 2004)*, pages 291–306. 2004.

J. C. Ciria, A. de Miguel, E. Domínguez, A. R. Francés, and A. Quintero. Local characterization of a maximum set of digital (26, 6)-surfaces. In *Proc. Discrete Geometry for Computer Imagery (DGCI 2005)*, pages 161–171. 2005.

H. Cohen. The BuDDy library & boolean expressions. *C/C++ Users Journal*, 2004.

N. Cornea, D. Silver, X. Yuan, and R. Balasubramanian. Computing hierarchical curve-skeletons of 3d objects. *The Visual Computer*, 21(11):945–955, 2005a.

N. D. Cornea, D. Silver, and P. Min. Curve-skeleton applications. In *Proc. IEEE Visualization '05*, page 13. 2005b.

L. Costa. Multidimensional scale space shape analysis. In *Int. Work. Synthetic-Natural Hybrid Coding and Three-Dimensional (3D) Imaging (IWSNHC3DI '99)*, pages 214–217. Santorini, Greece, 1999.

M. Couprie and G. Bertrand. Simplicity surfaces: a new definition of surfaces in z^3. In *SPIE Vision Geometry V*, volume 3454, pages 40–51. 1998.

M. Couprie and R. Zrour. Discrete bisector function and Euclidean skeleton. In *Proc. Discrete Geometry for Computer Imagery (DGCI 2005)*, pages 216–227. 2005.

M. Cox and D. Ellsworth. Application-controlled demand paging for out-of-core visualization. In *Proc. IEEE Visualization '97*. 1997.

O. Cuisenaire. *Distance Transformations: Fast Algorithms and Applications to Medical Image Processing*. Ph.D. thesis, Université catholique de Louvain, Laboratoire de Telecommunications et Teledetection, 1999. http://ltswww.epfl.ch/~cuisenai/papers/.

E. S. Deutsch. Thinning algorithms on rectangular, hexagonal, and triangular arrays. *Comm. ACM*, 15(9):827–837, 1972.

T. K. Dey and J. Sun. Defining and computing curve-skeletons with medial geodesic function. In *Proc. Symp. Geometry Processing (SGP '06)*. 2006.

H. M. Duvernoy, S. Selon, and J. L. Vannson. Cortical blood vessels of the human brain. *Brain Research Bulletin*, 7(5):519–579, 1981.

ESA MAP Team AO-99-030. Assessment of bone structure and its changes in microgravity. In A. Wilson, editor, *SP-1290: Microgravity Applications Programme: Successful Teaming of Science and Industry*, pages 282–305. European Space Agency, 2005.

F. Fol-Leymarie. *Three-Dimensional Shape Representation via Shock Flows*. Ph.D. thesis, Brown University, Providence, Rhode Island, USA, 2003.

C. Fouard. *Extraction de paramètres morphométriques pour l'étude du réseau micro-vasculaire cérébral*. Ph.D. thesis, Université de Nice – Sophia-Antipolis, 2005. http://www.inria.fr/rrrt/tu-1142.html.

C. Fouard and G. Malandain. Systematized calculation of optimal coefficients of 3-d chamfer norms. In I. Nyström, G. S. di Baja, and S. Svensson, editors, *Proc. Discrete Geometry for Computer Imagery (DGCI 2003)*, 2886, pages 214–223. LNCS 2886, Napoly, Italy, 2003.

C. Fouard and G. Malandain. 3-d chamfer distances and norms in anisotropic grids. *Image and Vision Computing*, 23(2):143–158, 2005.

C. Fouard, G. Malandain, S. Prohaska, and M. Westerhoff. Blockwise processing applied to brain micro-vascular network study. *IEEE Trans. Medical Imaging*, 25(10):1319, 2006.

C. Fouard, G. Malandain, S. Prohaska, M. Westerhoff, F. Cassot, C. Mazel, D. Asselot, and J.-P. Marc-Vergnes. Skeletonization by blocks for large 3D datasets: Application to brain microcirculation. In *IEEE Int. Symp. Biomedical Imaging: From Nano to Macro (ISBI '04)*. Arlington, Virginia, 2004.

M. Frigo, C. E. Leiserson, H. Prokop, and S. Ramachandran. Cache-oblivious algorithms (extended abstract). In *Proc. 40th Annual Symp. Foundations of Computer Science*, pages 285–397. IEEE Computer Society Press, 1999.

T. Fujimori, H. Suzuki, Y. Kobayashi, and K. Kase. Contouring medial surface of thin-plate structures using local marching cubes. *J. Computing and Information Science in Engineering*, 5(2):111–115, 2005.

N. Gagvani. Parameter-controlled volume thinning. *Graphical Models and Image Processing*, 61(3):149–164, 1999.

M. Garland and P. S. Heckbert. Surface simplification using quadric error metrics. *Computer Graphics*, 31:209–216, 1997.

Y. Ge and J. M. Fitzpatrick. On the generation of skeletons from discrete Euclidean distance maps. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 18(11):1055–1066, 1996.

P. Giblin and B. B. Kimia. A formal classification of 3d medial axis points and their local geometry. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 26(2):238–251, 2004.

P. J. Giblin and B. B. Kimia. Transitions of the 3d medial axis under a one-parameter family of deformations. In *Proc. 7th European Conf. Computer Vision (ECCV 2002)*, pages 718–734. Springer-Verlag, London, UK, 2002.

R. B. Haber and D. A. McNabb. *Visualization Idioms: A Conceptual Model for Scientific Visualization Systems*. IEEE Computer Society Press Tutorial, LosAlamitos, Calif., 1990.

H. K. Hahn, B. Preim, D. Selle, and H.-O. Peitgen. Visualization and interaction techniques for the exploration of vascular structures. In *Proc. IEEE Visualization '01*, page 295. 2001.

C. D. Hansen and C. R. Johnson. *The Visualization Handbook*. Elsevier, 2005.

A. Hatcher. *Algebraic Topology*. Cambridge University Press, 2002.

J. L. Hennessy, D. A. Patterson, D. Goldberg, and K. Asanovic. *Computer Architecture*. Morgan Kaufmann, 2003.

M. Hilaga, Y. Shinagawa, T. Kohmura, and T. L. Kunii. Topology matching for fully automatic similarity estimation of 3d shapes. In *Proc. SIGGRAPH 2001*, pages 203–212. ACM Press, New York, NY, USA, 2001.

M. Isenburg and P. Lindstrom. Streaming meshes. In *Proc. IEEE Visualization '05*. 2005.

K. Jänich. *Topology*. Springer, 1994.

C. Johnson, R. Moorhead, T. Munzner, H. Pfister, P. Rheingans, and T. S. Yoo, editors. *NIH-NSF Visualization Research Challenges Report*. U.S. National Institutes of Health, Bethesda, MD, U.S.A., 2006.

P. P. Jonker. Morphological operations on 3D and 4D images: From shape primitive detection to skeletonization. In G. Borgefors, I. Nystom, and G. S. di Baja, editors, *Proc. 9th Int. Conf. Discrete Geometry for Computer Imagery (DGCI 2000)*, volume 1953 of *Lecture Notes in Computer Science*, pages 1371–391. Springer, Berlin, Germany, 2000.

P. P. Jonker. Skeletons in n dimensions using shape primitives. *Pattern Recognition Lett.*, 23(6):677–686, 2002.

P. P. Jonker. Discrete topology on n-dimensional square tessellated grids. *Image and Vision Computing*, 23(2):213–225, 2005.

P. P. Jonker and A. M. Vossepoel. Mathematical morphology in 3D images: comparing 2D & 3D skeletonization algorithms. Summer School On Morphological Image and Signal Processing (27 - 30 September, Zakopane, Poland), Delft, Netherlands, 95.

R. Kähler, S. Prohaska, A. Hutanu, and H.-C. Hege. Visualization of time-dependent remote adaptive mesh refinement data. In *Proc. IEEE Visualization '05*, pages 175–182. 2005.

A. E. Kaufman, D. Cohen, and R. Yagel. Volume graphics. *IEEE Computer*, 26(7):51–64, 1993.

B. Kégl and A. Krzyżak. Piecewise linear skeletonization using principal curves. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 24(1):59–74, 2002.

E. D. Khalimsky, R. D. Kopperman, and P. R. Meyer. Computer graphics and connected topologies on finite ordered sets. *Topology and its Applications*, 36:1–17, 1990.

G. Klette. Simple points in 2d and 3d binary images. In *CAIP 2003*, volume 2756 of *Lecture Notes in Computer Science*, pages 57–64. 2003.

R. Klette and A. Rosenfeld. *Digital Geometry*. Morgen Kaufmann, San Francisco, 2004.

T. Kong. On topology preservation in 2-d and 3-d thinning. *Int. J. Pattern Recognition and Artificial Intelligence*, 9(5):813–844, 1995.

T. Y. Kong, R. Kopperman, and P. R. Meyer. A topological approach to digital topology. *American Mathematical Monthly*, 98(12):901–917, 1991.

V. Kovalevsky. Algorithms and data structures for computer topology. In G. Bertrand, A. Imiya, and R. Klette, editors, *Digital and image geometry: advanced lectures*, volume 2243 of *Lecture Notes in Computer Science*, pages 38–58. Springer-Verlag New York, Inc., New York, NY, USA, 2001.

V. A. Kovalevsky. Finite topology as applied to image analysis. *Computer Vision, Graphics, and Image Processing*, 46(2):141–161, 1989.

T. Kurc, Ü. Çatalyürek, C. Chang, A. Sussman, and J. Saltz. Visualization of large data sets with the active data repository. *IEEE Computer Graphics and Applications*, 21(4):24–33, 2001.

J.-O. Lachaud. Continuous analogs of digital boundaries: A topological approach to iso-surfaces. *Graphical Models*, 62:129–164, 2000.

J.-O. Lachaud. Coding cells of digital spaces: a framework to write generic digital topology algorithms. In A. Del Lungo, V. Di Gesù, and A. Kuba, editors, *Proc. Int. Work. Combinatorial Image Analysis (IWCIA 2003), Palermo, Italy*, volume 12 of *ENDM*. Elsevier, 2003.

L. Lam, S.-W. Lee, and C. Y. Suen. Thinning methodologies — A comprehensive survey. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 14(9):869, 1992.

C. C. Law, W. J. Schroeder, K. M. Martin, and J. Temkin. A multi-threaded streaming pipeline architecture for large structured data sets. In *Proc. IEEE Visualization '99*, pages 225–232. IEEE Computer Society Press, 1999.

T.-C. Lee, R. L. Kashyap, and C.-N. Chu. Building skeleton models via 3-d medial surface/axis thinning algorithms. *CVGIP: Graphical Models and Image Processing*, 56(6):462–478, 1994.

J. Li, W.-K. Liao, A. Choudhary, R. Ross, R. Thakur, W. Gropp, R. Latham, A. Siegel, B. Gallagher, and M. Zingale. Parallel netCDF: A scientific high-performance I/O interface. In *Proc. 2003 ACM/IEEE Conference on Supercomputing (SC '03)*, page 39. Phoenix, Arizona, 2003.

S. Loncaric. A survey of shape anlaysis techniques. *Pattern Recognition*, 31(8):983–1001, 1998.

W. E. Lorensen and H. E. Cline. Marching cubes: a high resolution 3D surface construction algorithm. In M. C. Stone, editor, *Proc. SIGGRAPH '87*, pages 163–170. Computer Graphics, Volume 21, Number 4, 1987.

C. M. Ma. A 3D fully parallel thinning algorithm for generatin medial faces. *Pattern Recognition Lett.*, 16:83–87, 1995.

C. M. Ma and M. Sonka. A fully parallel 3D thinning algorithm and its appllications. *Computer Vision and Image Understanding*, 64(3):420, 1996.

C.-M. Ma and S.-W. Wan. Parallel thinning algorithms on 3D (18,6) binary images. *Computer Vision and Image Understanding*, 80:364–378, 2000.

C.-M. Ma and S.-Y. Wan. A medial-surface oriented 3-d two-subfield thinning algorithm. *Pattern Recognition Lett.*, 22(13):1439–1446, 2001.

C.-M. Ma, S.-Y. Wan, and J.-D. Lee. Three-dimensional topology preserving reduction on the 4-subfields. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 24(12):1594–1605, 2002.

W.-C. Ma, F.-C. Wu, and M. Ouhyoung. Skeleton extraction of 3d objects with radial basis functions. In *Proc. Shape Modeling Int. (SMI '03)*, page 207. IEEE Computer Society, Washington, DC, USA, 2003.

S. Majumdar, M. Kothari, P. Augat, D. C. Newitt, T. M. Link, J. C. Lin, T. Lang, Y. Lu, and H. K. Genant. High-resolution magnetic resonance imaging: Three-dimensional trabecular bone architecture and biomechanical properties. *Bone*, 22(5):445–454, 1998.

G. Malandain and G. Bertrand. Fast characterization of 3D simple points. In *Proc. 11th Int. Conf. Pattern Recognition (IAPR)*. The Hague, The Netherlands, 1992.

G. Malandain and S. Fernández-Vidal. Euclidean skeletons. *Image and Vision Computing*, 16:317–327, 1998.

O. Mallo, R. Peikert, C. Sigg, and F. Sadlo. Illuminated lines revisited. In *IEEE Visualization*, page 3. 2005.

A. Manzanera, T. M. Bernard, F. Prêteux, and B. Loguet. Ultra-fast skeleton based on an isotropic fully parallel algorithm. In *Proc. 8th Int. Conf. Discrete Geometry for Computer Imagery (DGCI '99)*, volume 1568 of *Leture Notes in Computer Science*, pages 313–324. 1999a. Http://www-sim.int-evry.fr/People/Manzanera.html.

A. Manzanera, T. M. Bernard, F. Prêteux, and B. Loguet. A unified mathematical framework for a compact and fully parallel n-D skeletonization procedure. In *Proc. SPIE Conf. 3811 on Vision Geometry VIII*. 1999b. Http://www-sim.int-evry.fr/People/Manzanera.html.

A. Manzanera, T. M. Bernard, F. Prêteux, and B. Longuet. Medial faces from a concise 3D thinning algorithm. In *Proc 7th IEEE Int. Conf. Computer Vision*, volume 1, pages 337–343. 1999c. Http://www-sim.int-evry.fr/People/Manzanera.html.

V. Marion-Poty and S. Miguet. A new 2-D and 3-D thinning algorithm based on successive border generations. Research Report 94-20, Laboratoire de l'Informatique du Parallélisme, 1994.

B. H. McCormick, T. A. DeFanti, and M. D. Brown. Visualization in scientific computing. NSF Landmark Report, 1987.

D. G. Morgenthaler and A. Rosenfeld. Surfaces in three-dimensional images. *Information and Control*, 51:227–247, 1981.

NCSA. HDF5 - A new generation of HDF. 2003. http://hdf.ncsa.uiuc.edu/HDF5/.

I. Nyström, G. S. di Baja, and S. Svensson. Curve skeletonization by junction detectino in surface skeletons. In C. A. et. al., editor, *IWVF4*, volume 2059 of *Lecture Notes in Computer Science*, pages 229–238. 2001.

A. Odgaard and H. H. G. Gundersen. Quantification of connectivity in cancellous bone, with special emphasis on 3-D reconstructions. *Bone*, 14:173, 1993.

S. Oeltze and B. Preim. Visualization of vasculature with convolution surfaces: method, validation and evaluation. *IEEE Trans. Medical Imaging*, 24(4):540–548, 2005.

R. L. Ogniewicz and O. Kübler. Hierachic Voronoi skeletons. *Pattern Recognition*, 28(3):343–359, 1995.

M. Pakura, O. Schmitt, and T. Aach. Segmentation and analysis of nerve fibers in histological section of the cerebral human cortex. In *Proc. 5th IEEE Southwest Symp. Image Analysis and Interpretation (SSIAI '02)*, pages 62–66. 2002.

K. Palági and A. Kuba. A parallel 3D 12-subitaration thinning algorithm. *Graphical Models and image Processing*, 61:199–221, 1999.

V. Pascucci and R. J. Frank. Hierachical indexing for out-of-core access to multi-resolution data. In G. Farin, B. Hamann, and H. Hagen, editors, *Hierarchical and Geometrical Methods in Scientific Visualization*, page 225. Springer, 2003.

M. Peternell. Geometric properties of bisector surfaces. *Graphical Models*, 62(3):202–236, 2000.

S. M. Pizer, P. T. Fletcher, A. Thall, M. Styner, G. Gerig, and S. C. Joshi. Object models in multiscale intrinsic coordinates via m-reps. *Image and Vision Computing*, 21(1):5–15, 2003a.

S. M. Pizer, K. Siddiqi, G. Székely, J. N. Damon, and S. W. Zucker. Multiscale medial loci and their properties. *Int. J. Computer Vision*, 55(2-3):155–179, 2003b.

S. Prohaska and H.-C. Hege. Fast visualization of plane-like structures in voxel data. In *Proc. IEEE Visualization '02*, pages 29–36. Boston, Massachusetts, 2002.

S. Prohaska and A. Hutanu. Remote data access for interactive visualization. In *Proc. 13th Annual Mardi Gras Conf.: Frontiers of Grid Applications and Technologies*, pages 17–22. 2005.

S. Prohaska, A. Hutanu, R. Kähler, and H.-C. Hege. Interactive exploration of large remote micro-CT scans. In *Proc. IEEE Visualization '04*, pages 345–352. Austin, Texas, 2004.

C. Pudney. Distance-ordered homotopic thinning: A skeletonization algorithm for 3D digital images. *Computer Vision and Image Understanding*, 72:404–413, 1998.

J. Reddy and G. Turkiyyah. Computation of 3d skeletons using a generalized Delaunay triangulation technique. *Computer-Aided Design*, 27(9):677–694, 1995.

E. Remy and E. Thiel. Computing 3d medial axis for chamfer distances. In *Proc. 9th Int. Conf. Discrete Geometry for Computer Imagery (DGCI 2000)*, pages 418–430. Springer-Verlag, London, UK, 2000.

E. Remy and E. Thiel. Medial axis for chamfer distances: computing look-up tables and neighbuorhoods in 2d or 3d. *Pattern Recognition Lett.*, 23(6):649–661, 2002.

L. Robert and G. Malandain. Fast binary image processing using binary decision diagrams. *Computer Vision and Image Understanding*, 72(1):1–9, 1998.

P. Rüegsegger, B. Koller, and R. Müller. A microtomographic system for the nondestructive evaluation of bone architecture. *Calcified Tissue Int.*, 58:24–29, 1996.

K. Saha, B. Chanda, and D. D. Majumder. Principles and algorithms for 2-d and 3-d shrinking. Technical Report TR/KBCS/2/91, N.C.K.B.C.S. Library, Indian Statistical Inst., Calcutta, India, 1991.

P. K. Saha and B. B. Chaudhuri. 3D digital topology under binary transformation with applications. *Computer Vision and Image Understanding*, 63(3):418–429, 1996.

P. K. Saha, B. B. Chaudhuri, and D. D. Majumder. A new shape preserving parallel thinning algorithm for 3D digital images. *Pattern Recognition*, 30(12):1939–1955, 1997.

P. Saparin, J. S. Thomsen, S. Prohaska, A. Zaikin, J. Kurths, H.-C. Hege, and W. Gowin. Quantification of spatial structure of human proximal tibial bone biopsies using 3D measures of complexity. *Acta Astronautica*, 56(8):820–830, 2005.

S. Sarawagi and M. Stonebraker. Efficient organization of large multidimensional arrays. In *Proc. 10th Int. Conf. Data Engineering*, pages 328–336. IEEE Computer Society, Washington, DC, USA, 1994.

M. Sato, I. Bitter, M. Bende, A. Kaufman, and M. Nakajima. TEASAR: Tree-structure extraction algorithm for accurate and robust skeletons. In B. A. Barsky, Y. Shinagawa, and W. Wang, editors, *Proceedings of the 8th Pacific Graphics Conference on Computer Graphics and Application (PACIFIC GRAPHICS-00)*, pages 281–289. IEEE, Los Alamitos, CA, 2000.

D. Shaked and A. M. Bruckstein. Pruning medial axes. *Computer Vision and Image Understanding*, 69(2):156–169, 1998.

K. Siddiqi, S. Bouix, A. Tannenbaum, and S. W. Zucker. The hamilton-jacobi skeleton. In *Proc. 7th IEEE Int. Conf. Computer Vision*, volume 2, pages 828–834. 1999.

K. Siddiqi, A. Shokoufandeh, S. J. Dickinson, and S. W. Zucker. Shock graphs and shape matching. In *Proc. 6th Int. Conf. Computer Vision (ICCV '98)*, page 222. IEEE Computer Society, Washington, DC, USA, 1998.

C. T. Silva, Y.-J. Chiang, J. El-Sana, and P. Lindstrom. Out-of-core algorithms for scientific visualization and computer graphics. Course notes IEEE Visualization '02, 2002.

D. Stalling, M. Westerhoff, and H.-C. Hege. Amira: a highly interactive system for visual data analysis. In C. D. Hansen and C. R. Johnson, editors, *The Visualization Handbook*, chapter 38, pages 749–767. Elsevier, 2005.

M. Stauber and R. Müller. Volumetric spatial decomposition of trabecular bone into rods and plates—a new method for local bone morphometry. *Bone*, 38:475–484, 2006.

C. Stoll, S. Gumhold, and H.-P. Seidel. Visualization with stylized line primitives. In *Proc. IEEE Visualization '05*, page 88. 2005.

S. Svensson and G. Borgefors. On reversible skeletonization using anchor-points from distance transforms. *Journal of Visual Communication and Image Representation*, 10:379–397, 1999.

S. Svensson, I. Nyström, and G. S. di Baja. Curve skeletonization of surface-like objects in 3d images guided by voxel classification. *Pattern Recognition Lett.*, 23(12):1419–1426, 2002.

S. Takahashi, Y. Takeshima, and I. Fujishiro. Topological volume skeletonization and its application to transfer function design. *Graphical Models*, 66(1):24–49, 2004.

H. Talbot and L. Vincent. Euclidean skeletons and conditional bisectors. In *SPIE Visual Communications and Image Processing*, volume 1818, pages 862–876. 1992.

R. Thakur, W. Gropp, and E. Lusk. Optimizing noncontiguous accesses in MPI-IO. *Parallel Computing*, 28(1):83–105, 2002.

J. S. Thomsen, A. Laib, B. Koller, S. Prohaska, L. Mosekilde, and W. Gowin. Stereological measures of trabecular bone structure: comparison of 3D micro computed tomography with 2D histological sections in human proximal tibial bone biopsies. *J. Microscopy*, 218:171, 2005.

P. Thurner, P. Wyss, R. Voide, M. Stauber, M. Stapanoni, U. Sennhauser, and R. Müller. Time-lapsed investigation of three-dimensional failure and damage accumulation in trabecular bone using synchrotron light. *Bone*, 39(2):289–299, 2006.

M. Tory and T. Möller. Rethinking visualization: A high-level taxonomy. In *Proc. IEEE Symp. Information Visualization (INFOVIS '04)*, pages 151–158. 2004.

UNIDATA. NetCDF—network common data format. 2004. http://my.unidata.ucar.edu/content/software/netcdf.

J. van Wijk. The value of visualization. In *Proc. IEEE Visualization '05*, page 11. 2005.

B. J. H. Verwer, P. W. Verbeek, and S. T. Dekker. An efficient uniform cost algorithm applied to distance transforms. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 11:425, 1989.

L. Vincent. Efficient computation of various types of skeletons. In *SPIE Medical imaging V*, volume 1445, pages 297–311. San Jose CA, 1991.

J. S. Vitter. External memory algorithms and data structures: Dealing with massive data. *ACM Computing Surveys*, 33(2):209–271, 2001.

A. M. Vossepoel, K. Shutte, and C. F. P. Delanght. Memory efficient skeletonization of utility maps. In *Proc. 4th Int. Conf. Document Analysis and Recognition*, pages 797–800. Ulm (Germany), 1997.

J. Wang, M. M. Oliveira, and A. E. Kaufman. Reconstructing manifold and non-manifold surfaces from point clouds. In *Proc. IEEE Visualization '05*, pages 415–422. 2005.

S.-E. Yoon, P. Lindstrom, V. Pascucci, and D. Manocha. Cache-oblivious mesh layouts. In *Proc. SIGGRAPH '05*. ACM Press, 2005.

A. Zaikin, P. Saparin, J. Kurths, S. Prohaska, and W. Gowin. Modeling resorption in 2D CT and 3D μ-CT bone images. *Int. J. Bifurcation and Chaos*, 15(9):2995–3009, 2005.

Y. Zhou, A. Kaufman, and A. Toga. Three-dimensional skeleton and centerline generation based on an approximate maximum distance field. *Visual Computer*, 14:303–314, 1998.

M. Zöckler, D. Stalling, and H.-C. Hege. Interactive visualization of 3d-vector fields using illuminated streamlines. In *Proc. IEEE Visualization '96*, pages 107–113. San Fransisco, 1996.

A. Zomorodian. *Computing and Comprehending Topology: Persistence and Hierarchical Morse Complexes*. Ph.D. thesis, University of Illinois, Urbana-Champaign, 2001.

Index