Skeletonization and Segmentation of Binary Voxel Shapes

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de
Technische Universiteit Eindhoven, op gezag van de
Rector Magnificus, prof.dr.ir. C.J. van Duijn, voor een
commissie aangewezen door het College voor
Promoties in het openbaar te verdedigen op
donderdag 12 februari 2009 om 16.00 uur

door

Dennie Reniers

geboren te Breda

Dit proefschrift is goedgekeurd door de promotoren:

prof.dr. A.C. Telea
en
prof.dr.ir. J.J. van Wijk

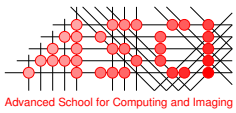
A catalogue record is available from the Eindhoven University of Technology Library

ISBN: 978-90-386-1496-0

Eerste promotor:
prof.dr.ir. A.C. Telea (Rijksuniversiteit Groningen)

Tweede promotor:
prof.dr.ir. J.J. van Wijk (Technische Universiteit Eindhoven)

Kerncommissie:
prof.dr. K. Siddiqi (McGill University, Canada)
dr. R.C. Veltkamp (Universiteit Utrecht)
prof.dr. M.T. de Berg (Technische Universiteit Eindhoven)



The work in this thesis was supported by NWO (Netherlands Organization for Scientific Research) under grant number 612.065.414.

The work in this thesis has been carried out in the research school ASCI (Advanced School for Computing and Imaging). ASCI dissertation series number: 173.

Typeset in LaTeX.
Printed by Eindhoven University Press.
Cover design by D. Reniers

Contents

Preface

This dissertation is the result of research that I conducted between January 2005 and December 2008 in the Visualization research group of the Technische Universiteit Eindhoven. I am pleased to have the opportunity to thank a number of people that made this work possible.

I owe my sincere gratitude to Alexandru Telea, my supervisor and first promotor. I did not consider pursuing a PhD until my Master's project, which he also supervised. Due to our pleasant collaboration from which I learned quite a lot, I became convinced that becoming a doctoral student would be the right thing to do for me. Indeed, I can say it has greatly increased my knowledge and professional skills. Alex, thank you for our interesting discussions and the freedom you gave me in conducting my research. You made these four years a pleasant experience.

I am further grateful to Jack van Wijk, my second promotor. Our monthly discussions were insightful, and he continuously encouraged me to take a more formal and scientific stance. I would also like to thank Prof. Jan de Graaf from the department of mathematics for our discussions on some of my conjectures. His mathematical rigor was inspiring. I am greatly indebted to the Netherlands Organisation for Scientific Research (NWO) for funding my PhD project (grant number 612.065.414). I thank Prof. Kaleem Siddiqi, Prof. Mark de Berg, and Dr. Remco Veltkamp for taking part in the core doctoral committee and Prof. Deborah Silver and Prof. Jos Roerdink for participating in the extended committee.

Our Visualization group provides a great atmosphere to do research in. In particular, I would like to thank my fellow doctoral students Frank van Ham, Hannes Pretorius, Lucian Voinea, Danny Holten, Koray Duhbaci, Yedendra Shrinivasan, Jing Li, Niels Willems, and Romain Bourqui. They enabled me to take my mind of research from time to time, by discussing political and economical affairs, and more trivial topics. Furthermore, I would like to thank the senior researchers of our group, Huub van de Wetering, Kees Huizing, and Michel Westenberg. In particular, I thank Andrei Jalba for our fruitful collaboration in the last part of my work.

On a personal level, I would like to thank my parents and sister for their love and support over the years, my friends for providing distractions outside of the office, and Michelle for her unconditional love and ability to light up my mood when needed.

Dennie Reniers
December 2008

Chapter 1

Introduction

Many computer science disciplines deal with representations of two-dimensional (2D) or three-dimensional (3D) geometric objects, called shapes. These shapes are often characterized by their geometry only: color and texture do not play any role. Shapes can be represented in various ways. In computer-aided design (CAD) for example, engineers often manipulate shapes by means of their boundary. The boundary is represented as a polygonal mesh, consisting of vertices, edges, and faces. In medical imaging, shapes are not created manually, but are produced automatically from real-world objects by an acquisition device, such as an MRI-scanner. These shapes are often represented by elements that enumerate their interior, called volume elements, or voxels. Scanners usually associate with each voxel a grayscale value that represents material density or some other measure. A *binary voxel shape* is a shape representation that views the world as black and white: it distinguishes between interior and exterior voxels only.

A voxel shape is a low-level description of the shape that represents every detail. These details are not necessary for a high-level understanding of the shape. In order to abstract from the shape, various shape descriptors have been proposed. In 1967, Blum introduced the *skeleton*, which transforms a shape into another one that is of a lower dimensionality than the shape it describes: a 1D and 2D structure for 2D and 3D shapes, respectively. The skeleton of a 2D shape can be seen as the stick-figure representation

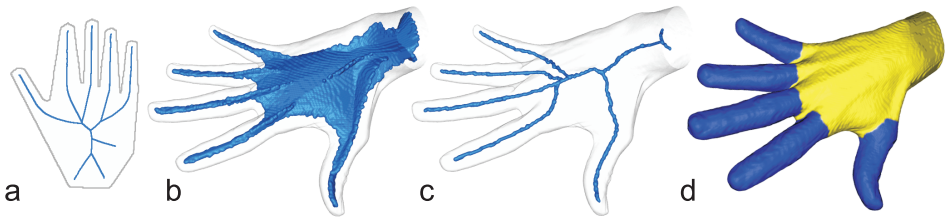


Figure 1.1: Example skeletons of (a) a 2D shape and (b) a 3D shape. (c) The curve skeleton. (d) An example shape segmentation.

of that shape. It is centered within the shape and captures the symmetry of the object: it can be considered as a curved and branching axis of symmetry. Instead of analyzing the shape directly, an algorithm may analyze its skeleton and so reduce the dimensionality of the problem. Skeletonization is still a topic of ongoing research for three reasons. First, computing the skeleton in an exact manner has turned out to be a difficult and computationally expensive problem, especially for 3D shapes. Second, according to its formal definition, the skeleton is notoriously sensitive to boundary noise, which one can address in various ways. Because of the first two reasons, most methods compute an approximation of the skeleton. Third, new skeleton-like structures have been proposed to alleviate some of the shortcomings of the Blum skeleton. In particular, the *curve skeleton* presents a truly compact 1D structure for 3D shapes. Figure 1.1 depicts the skeleton of a 2D hand shape, and the Blum and curve skeleton for a 3D hand shape.

At a very high level, shapes can be understood as consisting of distinct parts. The 3D hand shape in Figure 1.1 can be considered to consist of five fingers and the palm. Clearly, humans are very good at understanding and recognizing shapes. For a fantasy object one can often identify the different parts, even if they have no apparent functionality. This indicates that the identification of parts can in many cases be done by considering only the shape itself: no contextual information or a priori knowledge is required. In computer science, *shape segmentation* is the task of identifying the logical parts of a shape. Logical parts might correspond to patches having certain geometric properties, such as quasi-flatness. Alternatively, they might correspond to the meaningful parts of a shape, using principles from cognition and vision. Both segmentation types are still active research topics. One of the challenges is to create methods that are robust to noise and treat the shape in a multiscale fashion, something that humans naturally do. By scale, we mean one of the measurable dimensions of the shape, such as circumference, area, or volume. Fine scales correspond to shape details, such as bumps or wriggles, whereas coarse scales give the overall shape features, such as the fingers of a hand. We argue that it is beneficial to approach the problem of shape segmentation by using skeletons, as they already provide a high-level description of the shape. Looking at Figure 1.1(c) for example, the structure of the curve skeleton seems to reflect the part structure of the shape: each part has a corresponding curve-skeleton branch. The problem is then how to extract this part structure from the curve skeleton and map it to the surface, so that we obtain a shape segmentation as shown in Figure 1.1(d).

1.1 Objective

The research question that we address in this thesis is twofold:

- How can we efficiently compute curve and surface skeletons of binary voxel shapes so that they can be used effectively as shape descriptors in various applications?
- How can we use these skeletons to create different types of shape segmentations that benefit from the desirable properties of the skeleton?

An important requirement for the effective use of skeletons as shape descriptors is robustness to boundary noise. By creating a multiscale skeleton representation this problem

is handled automatically as boundary noise can be considered as small scale boundary features. Our aim is thus to develop a skeletonization method that produces both multiscale curve and surface skeletons. The shape segmentation methods are based on these multiscale skeletons and conveniently inherit the robustness and multiscale properties.

The techniques we develop all work on digital shapes, called binary voxel shapes in 3D. Digital shapes are commonly used in the medical imaging and discrete geometry fields, and have already been used in many existing skeletonization methods, most notably in the class of thinning methods. Skeletonization and segmentation of binary voxel shapes brings its own difficulties. The resolution of the data is typically low, the data contains discretization artifacts, often contains noise, and surface normals are not readily available. Therefore, we use integral operations rather than differential operations when possible, as integral operations are more robust for voxelized and noisy shapes. It is important to note that our developed methods are not limited to voxel shapes. Most if not all of the algorithmic elements in our techniques have their counterparts in other shape representations. In any case, there are algorithms to convert between any two shape representations.

All our methods are straightforward to implement and are presented with full pseudocode. We show that our methods work robustly and produce good results on practical real world examples, including ones containing large amounts of surface noise.

1.2 Outline

The remainder of this thesis is structured as follows. For each chapter we indicate the related publications.

In Chapter 2, we detail on skeleton definitions and their desirable properties. Skeletons are well-defined mathematical objects, but they are difficult to compute exactly according to their formal definition. Therefore, most approaches satisfy a subset of the desirable properties that are desired by the intended application. In addition, applications may require additional properties that standard skeletons do not exhibit. We give an overview of these properties and requirements. We give special attention to the robustness and multiscale requirements, as we consider them fundamental in their effective use as shape descriptors.

Chapters 3-7 contain the main contributions of this dissertation. In Chapter 3 we concentrate on skeletonization of 2D shapes. We first improve upon an existing skeletonization method, the Augmented Fast-marching Method, for 2D shapes. Then, we detail on the computation of the *Tolerance-based Feature Transform* (TFT), which can be used to alleviate discretization artifacts in skeletonization, or to obtain new multiscale shape descriptors [67]. We present and discuss four TFT algorithms and quantitatively and qualitatively compare them on efficiency and the accuracy of the results. This work was first published in [113].

In Chapter 4, we concentrate on 3D shapes and develop a new method to compute robust and multiscale curve and surface skeletons in a unified manner. To achieve this, we define and compute a novel importance measure, called the *collapse measure*, which assigns to each skeleton point its importance in representing the original shape. The collapse measure we present could be considered a non-trivial extension of the potential

residual of Ogniewicz and Ilg [94] from 2D to 3D. A characterizing property of the collapse measure is that it is based on global properties of the shape, in contrast to the local nature of many existing measures. As a result, the collapse measure has a notion of spatial scale, which reflects a natural way in which shape details are perceived according to their geometric dimension, and a robust skeleton, or simplified skeleton, is obtained by simply thresholding the collapse measure. We compare the quality of our simplified skeletons with existing approaches and give several applications. Part of this work was published in [115].

In Chapter 5, we use the definition of our collapse measure to produce a partitioning of our simplified surface skeletons into their respective sheets: the 2D surface components of which the surface skeleton consists. To partition the skeleton, we first have to compute the sheet intersection curves. We show this is not possible by using simple local topological operations and present a solution. This work was published in [112].

In Chapters 6 and 7 we zoom in on one particular application of our multiscale skeletons, namely 3D shape segmentation. In Chapter 6 we use the simplified curve skeleton to compute part-type segmentations, which divide the shape into high-level, meaningful, segments. We associate with each curve-skeleton point a potential part-cut: a border between parts. The curve-skeleton junctions indicate where part cuts should be placed. The part cuts induce a segmentation of the shape into meaningful segments. The segments inherit the robustness to noise from the curve skeleton. The part cuts are piecewise geodesic, resulting in smooth, non-jaggy segment borders, and the segmentations are pose-invariant. This work was published in [109] and [110].

In Chapter 7, we use the simplified surface skeleton to compute a surface classifier on the boundary, which presents a robust alternative to classical curvature-based classifiers. Based on this detector we present a patch-type segmentation, which, in contrast to the part-type segmentations, consists of low-level, quasi-flat patches. This work was published in [107] and [111].

In Chapter 8 we summarize our findings and present conclusions. We also give suggestions for future work. Figure 1.2 illustrates the chapters and their results, their interdependencies, and suggested reading order.

A list of publications related to this work can be found in the appendix.

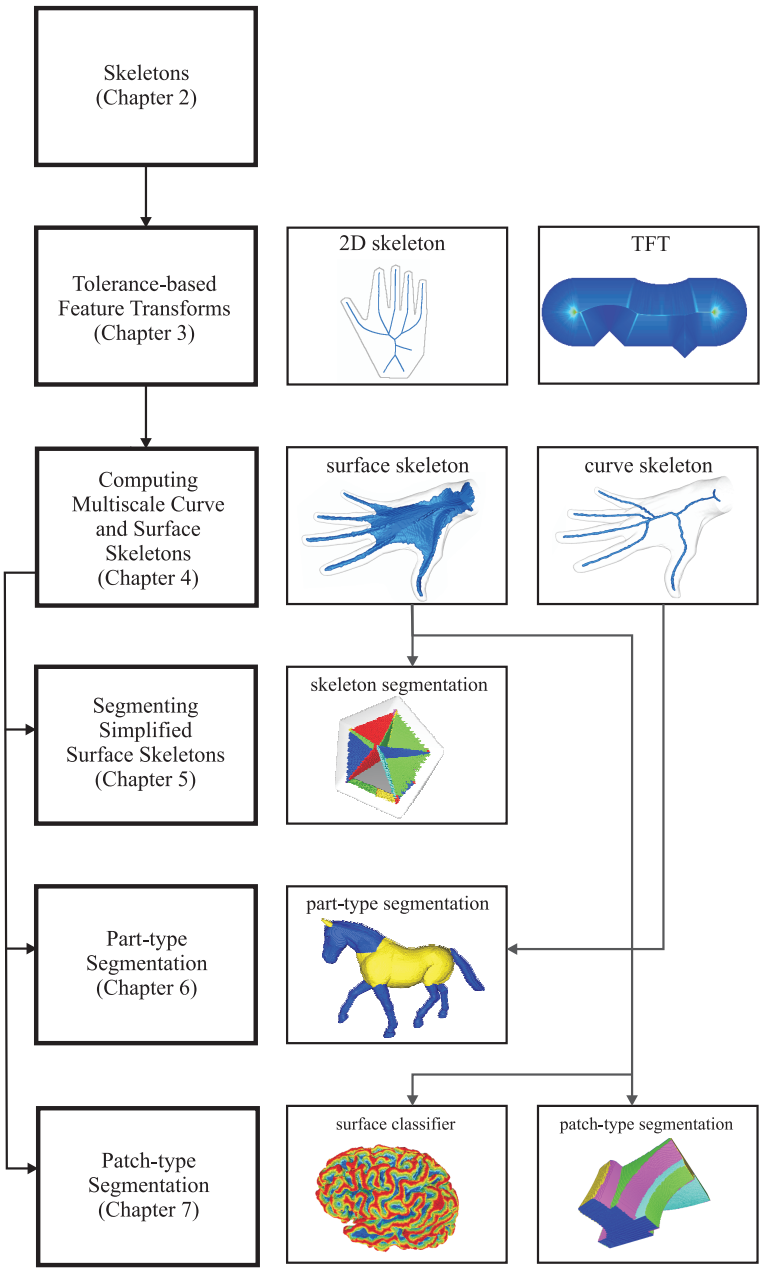


Figure 1.2: The chapters, their results, and their interdependencies.

Chapter 2

Skeletonization

This chapter discusses skeletons and skeletonization approaches. We list the most important skeleton properties, and additional requirements that their applications may have. Many of the properties and skeletonization techniques have been identified and discussed by Cornea [23], especially with regard to curve skeletons. This chapter adds some new properties and also discusses surface skeleton properties. We further detail on the notion of importance measures, and we explain how a suitable importance measure ensures skeletons possessing many desirable properties.

2.1 Shape representations

A *geometrical object* or *shape* Ω is a bounded, connected n-dimensional open set of $\mathbb{R}^n$. Its boundary $\partial\Omega$ is a connected, orientable, finite, and closed surface. In this work, we restrict ourselves to two and three-dimensional shapes ($n \in \{2, 3\}$). As the boundary is connected, this definition does not allow shapes with cavities, which are regions exterior to the shape that are fully enclosed by interior regions. In 3D, shapes with tunnels are allowed. The number of tunnels in a shape is called its *genus*.

Computer algorithms that work on shapes need to use a particular *shape representation* in order to represent shapes using a finite amount of data. Analytical shape representations have the advantage that they represent shapes exactly. However, it is difficult to design algorithms for them, as the mathematics can become complex when higher-order functions are involved, and the resulting algorithms are generally time consuming. In discrete shape representations, the shape is represented by a finite set of simple elements, such as points, line segments, and/or planar faces.

A *gray-scale image* is a n-dimensional array in which each element has a value between 0 and 1. In 2D, elements are called pixels, whereas in 3D, elements are called voxels. A gray-scale image can be obtained by sampling a real object, using e.g. a digital camera, or it can be obtained by sampling a virtual geometrical object. The resolution of the array determines the accuracy with which the original object can be represented. Each element in the array is associated with a lattice point in $\mathbb{Z}^n$. Hence, in the rest of this

thesis "points" may also denote pixels and voxels, depending on the context. An element in a *binary image* has either value 0 or 1. The binary image represents a *digital shape* Ω: the 1-values are considered the interior of the shape, whereas the 0-values can be considered the exterior. A binary image can easily be obtained from a gray-scale image by image segmentation (not to be confused with shape segmentation). The most basic image segmentation clamps the values in a gray-scale image to 0 and 1, with respect to a certain threshold, but more sophisticated image-segmentation methods are used in practice [29]. A 3D binary image consists of volume elements, called *voxels*. The set of interior voxels is called a *voxel shape*.

The study of topological properties of binary images is called *digital topology* [66], from which we repeat a small number of concepts here. In the 2D lattice, two (distinct) points are said to be 8-adjacent when each of their two coordinates differs by at most 1, and are 4-adjacent when precisely one coordinate does. In the 3D lattice, two points are 26-adjacent when each of their three coordinates differs by at most 1, and 6-adjacent when precisely one coordinate does. An n-path between two points is a list of n-adjacent points connecting those points. A set of points is said to be n-connected when there is an n-path between each two of them. In this work, a 2D shape Ω is a 4-connected subset of $\mathbb{Z}^2$, whereas a 3D shape is a 6-connected subset of $\mathbb{Z}^3$. We only consider binary images for which the interior and exterior voxels are two connected sets of voxels. In 3D, this excludes shapes with cavities, but it does not exclude shapes with tunnels. The boundary $\partial\Omega$ of a digital shape is defined as those interior points in Ω that are adjacent to an exterior point. The boundary neighborhood of a boundary point consists of all 26-adjacent points in $\partial\Omega$. Note that the cardinality of such a boundary neighborhood may vary from point to point.

In this thesis we use digital shapes to build our skeletonization and segmentation methods on. Other common representations of 3D shapes include unorganized point sets, which are typically produced by a range scanner, and polyhedral meshes, typically produced by CAD software. These are boundary representations: they enumerate elements of the shape surface rather than its volume. We argue that a volumetric representation is more natural than a boundary representation in the context of 3D skeletonization, because the skeleton is defined to lie inside the shape's volume. Using voxel shapes, we can conveniently represent and enumerate the volume elements, boundary elements, and skeleton elements uniformly by using the same voxel representation, yielding straightforward methods and implementations. Indeed, a large class of existing skeletonization methods act on voxel shapes, namely the thinning methods [68]. We would like to mention that other representations can be used as input to our algorithms when they are voxelized as a pre-processing step (e.g. using binvox [84]). Finally, our method does not contain any operations that are conceptually limited to voxel shapes, so that they can be adapted to other shape representations, such as polyhedral models.

We consider a digital shape to be the discretized version of a corresponding continuous shape. In order to give geometrical and topological guarantees on the skeletonization and segmentation results, we assume that the digital shape captures all the important geometrical and topological properties of the original shape. This assumption is not always trivial to satisfy. If the original shape has very thin parts, or some other small-scale details such as bumps or wriggles, the sampling grid resolution and position must be chosen

carefully in order to capture these parts. We do not concern ourselves here with these sampling problems and assume a good approximation by the voxel shape. Shapes should have a minimum thickness that is larger than the maximum distance between adjacent lattice points ($\sqrt{3}$ in 3D), so that the shape interior is always a connected set of voxels.

A *shape descriptor* represents the shape at a high level of abstraction. Instead of representing the original shape as accurately as possible, a shape descriptor extracts the essence or important features of the shape from the perspective of the application at hand. Good shape descriptors are discriminative: they assign different descriptions to different shapes. Quantitative shape descriptors strongly abstract from the shape by reducing it to one or more scalar values [29]. Simple examples include the volume, perimeter, or eccentricity of the shape. They are useful when shapes need to be compared to each other with respect to the most distinguishing features. In contrast, geometric shape-descriptors are fine abstractions that reduce the shape to a simplified or canonical representation, which is still a geometrical structure. Some geometric descriptors even allow the (approximate) reconstruction of the shape they describe, something quantitative descriptors do not allow. An example is the skeleton, which is discussed in the next section.

2.2 Skeleton definitions

Skeletons are geometric shape-descriptors. Their main features are that they are centered within the shape, and capture the topology and geometry of the shape in a compact manner. Although skeletons and related concepts have been known in mathematics for hundreds of years [70], they have gained interest in computer science since 1967 when defined by Blum [13] for the purpose of biological applications. They have since then been adopted by many fields in computer science, such as computer-aided design, visualization, computer graphics, and medical analysis. For example, skeletons are used in surface smoothing [118], volumetric animation [42], and feature-based visualization [104]. The skeleton can also be used to design objects in CAD applications [141]: one first creates a skeleton to specify the essential form, after which it is "inflated" to give the object its substance. An overview of the history and applications of skeletons is given by Leymarie [70].

Skeletonization The process of computing skeletons is called *skeletonization*. Conceptually, skeletonization can be seen as a function $\mathcal{S}^{n,m}$ that assigns to an n-dimensional shape Ω embedded in an $\mathbb{R}^n$ space, generically a set of m-dimensional components, that are embedded in the same space $\mathbb{R}^n$. By generically, we mean that for some shapes the components may be of a lower dimension than m. In this work, we restrict n to at most 3. We further make the restriction that $n > m$, which is a natural one, as it ensures that the skeleton is an abstraction from the shape. The larger the difference $n - m$, the higher the abstracting power of the skeleton. In the following, we discuss two skeleton types: the Blum skeleton $\mathcal{S}^{n,n-1}$ and the curve skeleton $\mathcal{S}^{3,1}$.

Blum skeleton The Blum skeleton $\mathcal{S}^{n,n-1}$, simply denoted $\mathcal{S}$ as it can be considered the standard skeleton definition, reduces the original n-dimensional shape by just one di-

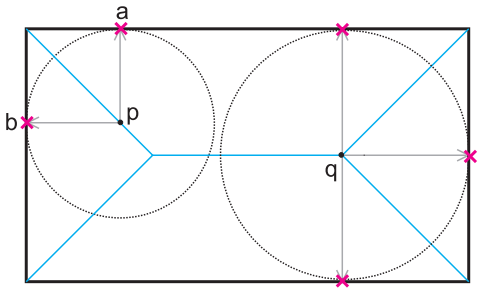


Figure 2.1: Feature points a, b of a skeleton point p.

mension. We next give a formal definition of the Blum skeleton for a continuous shape Ω. We particularize the skeleton definition and related results to digital shapes in later chapters. The Blum skeleton $\mathcal{S}$ of object Ω with boundary $\partial\Omega$ is defined as those points p in Ω that have at least two boundary points a, b at minimum distance of p:

$$\mathcal{S} = \mathcal{S}^{n,n-1}(\Omega) = \left\{ p \in \Omega \;\middle|\; \exists a, b \in \partial\Omega, a \neq b, \|p-a\| = \|p-b\| = D(p) \right\}, \quad (2.1)$$

where $D : \Omega \to \mathbb{R}_+$ is the *distance transform*, assigning to each object point the minimum distance to the boundary:

$$D(p) = \min_{k \in \partial\Omega} \|p-k\| \,. \quad (2.2)$$

In both equations, $\|\cdot\|$ denotes the Euclidean distance. In the past, approximate Euclidean distances have often been used, such as chamfer distances [117]. However, these have largely been made obsolete by efficient distance transforms [26]. Another reason to use alternative distance functions is to obtain different types of skeletons [129]. In this work, we confine ourselves to the common Euclidean distance. The points a, b at minimum distance from a skeleton point p are called *feature points* [50] of p (see Figure 2.1). Other names found in the literature are anchor points or contact points [97]. We call the vectors $\vec{pa}$ and $\vec{pb}$ *feature vectors*. A well-known property of the feature vectors is that they are normal to the shape boundary [97]. The distance between the skeleton point and its feature points can be stored together with skeleton. Because it allows reconstruction of the original shape from the skeleton [43], such a structure is called the skeleton or medial axis *transform* (MAT). One important property of the skeleton transform is its uniqueness: each shape has a unique skeleton transform.

Other names for the Blum skeleton are symmetry set, cut locus, and conflict set. Alternative names for 2D skeletons are medial axis and centerline. A single centerline is a single non-branching curve, useful as a camera path in virtual navigation [12]. These alternative names emphasize the centeredness property of the skeleton, but "skeleton" emphasizes the more general fact that it is a minimal representation of the shape. Hence, in the context of this thesis we prefer the name skeleton. Sometimes a small distinction is made between the skeleton and medial axis, differing only in the inclusion of the set of skeleton-branch endpoints. In this work, we make no such distinction.

Alternative skeleton definitions to the equidistant-point formulation from Eq. 2.1 exist [86]. Although these definitions are all essentially equivalent, they have resulted in different skeletonization approaches. One formulation defines the skeleton of a 2D shape as the locus of centers of maximally inscribed discs. A disc is maximally inscribed if it is completely inside the shape and is not fully included in another inscribed disc. Such a disc touches the shape boundary at two points: the feature points. In limit cases, the inscribed disc has more than two contact points. For a 3D shape, discs are simply replaced by balls. Another formulation of the skeleton uses the grass-fire analogy. Imagine a fire lighted on the shape boundary and the fire front moving at unit speed inward, in normal direction of the front. Then, the skeleton is defined as those points where the fire front meets itself. The arrival time of the front is equal to the inscribed disc radius. Finally, the skeleton can also be (equivalently) defined as the singularities in the distance-to-boundary field.

In case of a 2D shape, the skeleton $\mathcal{S}^{2,1}$ is a connected set of curves, which we call *branches* (see Figure 1.1(a)). The branches intersect in points called *junctions*. When the shape contains no holes, the skeleton is a tree. When the shape has holes, the skeleton is a graph, containing a loop for each hole in the shape. By the *symmetry-curvature duality theorem* [71], skeleton branches terminate at points of convex curvature maxima. Consequently, in case Ω is a circle, the skeleton reduces to a point: it contains no branches. For 3D shapes, the skeleton $\mathcal{S}^{3,2}$ is a connected set of 2D manifolds for the larger part, but might also contain 1D curves for cylindrical parts of the shape. Hence, the skeleton of a 3D shape is also known as medial surface, or *surface skeleton* as we refer to it. The surface skeleton is shown for a hand shape in Figure 1.1(b). As can be seen, the surface skeleton has a 2D structure inside the palm of the hand, but degenerates to curves for the cylindrical fingers.

Skeleton points can be classified by the number of contact points that the inscribed disc/ball has with the boundary [15, 46], or in other words, by the cardinality of their feature-point set. As can be seen in Eq. 2.1, each skeleton point has at least two feature points. In 2D, a skeleton point is generically a junction, an interior branch point, or a branch endpoint. Branch points have two feature points (point p in Figure 2.1), junctions have three or more (point q in Figure 2.1). The inscribed disc of an endpoint may partly coincide with the boundary, so that the feature points form one contiguous set. Non-generic cases are for example a junction having four feature points. These reduce to generic cases by a small local perturbation of the boundary. In 3D, the surface skeleton consists of manifolds, called *sheets*. Analogous to the 2D case, a surface-skeleton point is either an interior sheet point having two feature points, a sheet-intersection point having three, or a point on the skeleton rim having a contiguous set of feature points. Additionally, in 3D there might be points that have three sheet intersection curves coming together, but unlike the other three point types, these are isolated cases: they do not form connected structures.

Curve skeleton The *curve skeleton*, denoted $\mathcal{C}$, is a one-dimensional skeleton $\mathcal{S}^{3,1}$ of a 3D shape. The curve skeleton is a connected set of curves and can be seen as a stick-figure representation of the shape. Figure 1.1(c) depicts an example curve skeleton for the hand shape. Like the 2D skeleton $\mathcal{S}^{2,1}$, curve-skeleton branches tends to terminate at convex curvature maxima of the boundary. Curve skeletons do not have a widely accepted formal

definition. One of the main reasons for this lies in the difficulty to formally quantify a desirable property of the curve skeleton, namely that it should be locally centered with respect to the shape. For tubular shapes, this is easy to define and compute, but this becomes less clear for arbitrary shapes. Curve skeleton definitions usually share many of the surface skeleton properties, with the key difference that the curve skeleton consists of 1D curves instead of 2D surface components. Curve skeletons have a wide range of applications. In shape segmentation, the curve skeleton is very useful for extracting the part structure from a shape. In Chapter 6 we present such a segmentation approach. In virtual navigation, the curve skeleton is used to navigate the camera along a one-dimensional path that is centered within the shape, such as in virtual colonoscopy [12]. In shape matching and retrieval, the curve skeletons can be used as thin representations so that matching shapes can be more accurate than when using quantitative shape-descriptors, and they enable partial matching [132]. Cornea provides an extensive survey on curve-skeleton applications [24].

Finally, we mention some other geometric shape-descriptors related to skeletons. Skeletons are normally computed on binary images, but some methods compute skeletons directly from a gray-scale image, without requiring an image-segmentation step first [143, 96]. Related to the curve skeleton is the *Reeb graph*: a 1D structure encoding geometrical and topological properties of 3D shapes [125, 51]. It is constructed by following the evolution of the level sets of a real-valued function defined on the object boundary. The function should be chosen carefully to obtain a Reeb graph that is suitable for the application at hand. Finally, there are shape descriptors that capture more global symmetries than the skeleton does, such as planar symmetries [98]. The skeleton, in contrast, captures local point symmetries. The *shock scaffold* is a reduced-dimension summary of the skeleton [70]. It is the graph structure defined as the singularities in a flow on the skeleton, coinciding with the sheet intersection curves.

2.3 Properties

In this section we focus on the most important properties of Blum skeletons and curve skeletons, as they have been identified in literature. We also include properties of the skeletonization process itself, which can be equally important in practical applications. We distinguish between *intrinsic properties*, which follow from the formal skeleton definition, and *extrinsic properties*, which are required by the subsequent application that uses the skeleton as input. Some extrinsic properties may conflict with the intrinsic ones, and applications may establish extrinsic properties at the expense of one or more of the intrinsic ones. Strictly speaking, such methods do not compute the skeleton according to its formal definition, but as they produce similar results, it is useful to consider them as skeletonization methods for the sake of discussion and comparison. We give special attention to the conflicting instability property and the robustness requirement, as our skeletonization method we present in Chapter 4 is designed with this requirement in mind.

In Section 2.3.1 we discuss the properties of the Blum skeleton. In Section 2.3.2 we discuss the curve skeleton properties, which are related.

2.3.1 Blum skeleton properties

The following **intrinsic** properties of the Blum skeleton $\mathcal{S}$ follow from its formal definition (Eq. 2.1):

Topology preserving The skeleton is homotopic to the shape it describes. Only recently this was proved rigorously for all dimensions [74]. Because a shape (in our case) consists of a single connected component, homotopy implies that the skeleton consists of a single connected component. Naturally, the holes in the skeleton map one-to-one with holes in the shape. Homotopy should follow automatically from the skeletonization process itself, and should ideally not have to be ensured by some post-processing step.

Geometry preserving The skeleton captures the geometry of the shape. In the strong sense, this property dictates that the original object can be reconstructed from the skeleton [92]. The object boundary is formed by the envelope surface of the inscribed discs/balls with the stored radius. Note that homotopy is not required for reconstruction. In Figure 2.2, the shape consisting of two overlapping circles can be reconstructed from the two skeleton endpoints. The skeleton part that is needed for topology preservation is called a *ligature*. August et al. [9] argue that ligatures are the "glue" that connects the non-ligature parts, which represent the "substance" of the shape, and that a stable description of the shape can be given by the non-ligatures. If exact reconstruction is not required, a weak interpretation of geometry preservation entails that the skeleton captures all the important features of the shape.

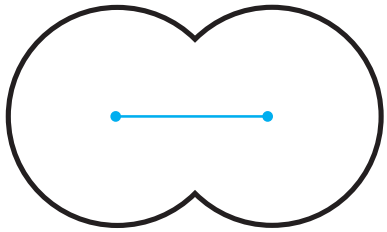


Figure 2.2: Topology versus geometry preservation.

Centered The skeleton is locally centered within the shape. Centeredness is needed for exact reconstruction.

Thin The skeleton is a thin representation of the original shape, that is, it has one dimension less than the original object. For digital shapes this property usually means that the skeleton should be at most one lattice point thick: the thinnest structure possible in a digital shape. We do not concern us with defining thickness formally, which is non-trivial for skeletons of 3D shapes and at points where skeleton branches or sheets meet.

It is often said that the thinness property conflicts with reconstructability [92]. For example, discretized balls centered at the 1-voxel thick skeleton of a box of uneven

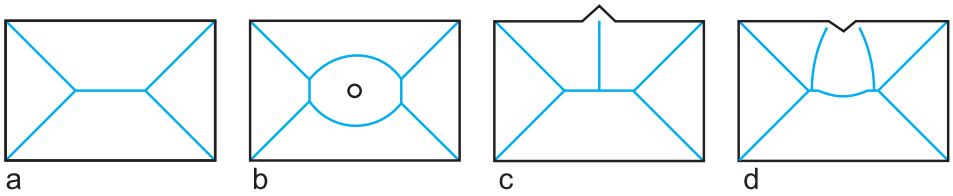


Figure 2.3: Skeleton of a noise-free rectangle (a), rectangle with hole (b), rectangle with bump (c), and rectangle with dent (d).

height do not touch the boundary in two places. Instead of requiring thinness everywhere, one could also require that the skeleton is 1-voxel thick when this does not conflict with the reconstructability property. Alternatively, one can allow that the reconstructed shape is, at each point, at most one voxel distance from the original shape.

Transformation invariant Skeletons are obviously invariant to isometric transformations of the shape, as these do not change the geometry and topology of the shape. Typically, this property is only violated by the shape discretization or any discretization steps in the skeletonization algorithm. For example, when the skeleton is computed by finding the singularities in the distance field, the skeleton is only invariant to transformation if exact Euclidean distances are used.

Unstable A undesirable property of the skeleton is that it is very unstable under noise in the shape. Formally stated, when we view skeletonization as a function $\mathcal{S}^{n,m}$, it is non-continuous according to the Cauchy definition of continuity. An arbitrary small perturbation ϵ of the shape Ω, caused by noise for instance, might result in a large difference δ in the skeleton $\mathcal{S}(\Omega)$. Indeed, strong geometry-preservation dictates that all object features are preserved, even non-essential features such as noise. From another point of view, it can be argued that the underlying cause of the sensitivity to noise is that the skeleton does not have a one-to-one correspondence with the boundary [59]. A single point on the boundary can generate many skeleton points, forming ligatures. Thus, changing a single boundary point can cause many axis points to change, creating the instability. In addition to noise, shape discretization may introduce instability in the skeleton. For example, the skeleton of a perfect circle is a single point, but the skeleton of a discretized disc does not consist of a single point because of the discretization inaccuracies.

We can distinguish between "salt-and-pepper" and boundary noise. Salt-and-pepper noise is a term used in image processing, referring to randomly occurring black and white pixels in an image. In digital shapes, this notion can be seen as random points that are removed from the shape. This type of noise typically changes the topology (the number of holes) of the shape: a single point removed from the shape interior causes a cavity and thereby a change in genus. Boundary noise is a less intrusive form of noise, in which only the boundary is affected by displacing boundary elements inward or outward along the boundary normal,

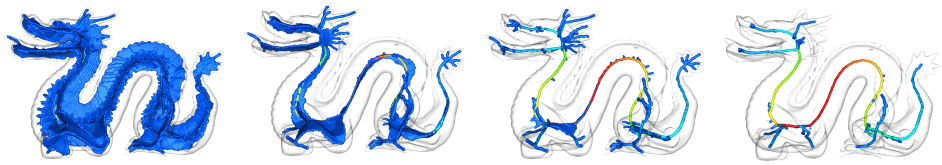


Figure 2.4: An example multiscale skeleton.

creating bumps and dents. The maximum distance from which an element can deviate from its true position determines the noise level. In general, this operation does not change the topology, except when it causes the boundary to self-intersect. Figure 2.3 shows the skeleton for a rectangle (a), the effect of salt-and-pepper noise (b), and boundary noise (c,d). In Figure 2.3(b), the rectangle contains a small hole due to salt-and-pepper noise. The resulting skeleton differs greatly from the skeleton of the non-noisy rectangle, and in fact has a different structure. A bump on the boundary causes a spurious branch in the skeleton (Figure 2.3(c)). A dent causes two spurious branches, because the dent causes two convex corners (Figure 2.3(d)).

The **extrinsic properties** result from requirements by the particular application. The following is an overview of the most common requirements.

Robustness The skeleton should be robust to noise and discretization artifacts. When a skeleton contains too many spurious parts, it is impractical to handle in any subsequent application. Moreover, these spurious branches represent insignificant features to which applications should be tolerant. Hence, practically all applications require robustness. The robustness requirement conflicts directly with the intrinsic property that the skeleton is unstable, and with the strong geometry-preservation property.

Consider Figure 2.3. It can be seen that by removing branches from the skeleton after it has been computed, which is called *pruning*, we cannot make the skeleton robust to the hole in the rectangle in (b). Thus, pruning cannot make the skeleton robust to salt-and-pepper noise. However, pruning could remove the spurious branches caused by boundary noise in (c) and (d). In this thesis, we assume that any noise in the shape does not affect its topology. In particular, we assume that the object does not contain salt-and-pepper noise, and boundary noise does not make the boundary self-intersect. The former is a reasonable assumption because salt-and-pepper noise can be easily eliminated by standard image processing operations such as dilation and erosion [29]. In Section 2.6 we discuss pruning methods in detail.

Multiscale representation Real objects exhibit different structures at different scales. For example, a person can be seen as a body with four limbs at a coarse scale, whereas at finer scales, the fingers and toes can be seen. At an even finer scale, hairs can be distinguished. The skeleton, being a shape descriptor, should account

for the multiscale nature of shapes [93]. Hence, it is desirable that a skeletonization method produces a family of increasingly detailed skeletons. Such a family could either be a finite or infinite set of skeletons, from which the user can select the skeleton at the desired scale. The multiscale requirement, linked to the above-mentioned notion of scale in which we consider the size of geometric features, is related to the robustness requirement in the sense that if the skeleton is multiscale, it can also be made robust. Indeed, noise can be considered as small-scale boundary features, that are only represented in the finest scales of the skeleton scale-space.

Skeletons can further be *hierarchical*, meaning that each coarse-scale skeleton is fully included in skeletons at finer scales. Hierarchical skeletons have as advantage that the correspondence between different scales follows automatically. Figure 2.4 depicts an example multiscale (and hierarchical) skeleton representation.

Efficient to compute The skeleton should be efficient to compute in terms of time and memory consumption. This is especially important for applications that require real-time skeleton computation, such as CAD applications. For shape retrieval, the skeletons of the shapes in the database can be precomputed, making efficiency less important. It has turned out that computing high-quality skeletons is a time consuming process, especially for 3D shapes. Some methods trade quality for speed, essentially computing skeleton approximations.

Easy to implement For practitioners, ease of implementation of a skeletonization method can be a reason to prefer it above others. It leads to quicker results and makes the method more useful in practice. An implementation of a complex skeletonization method might be offered as a black box by its designers, but such an approach makes it difficult to port, tune, or improve upon it. Ease of implementation is an important requirement, given that the vast majority of state-of-the-art skeletonization methods in existence are very complex and hard to replicate from research papers, and thus largely out of reach for the intended audience, the practitioners.

2.3.2 Curve skeleton properties

Recall that the curve skeleton is a 1D structure describing a 3D shape. It is conceptually related to the surface skeleton, and as such borrows many of its properties, but also adds some. As stated, all curve skeleton properties can be considered extrinsic because the curve skeleton does not have a widely-accepted formal definition. Nevertheless, we consider the properties that are related to the intrinsic surface skeleton properties also as intrinsic. We indicate the differences of the curve skeleton properties with those of the surface skeleton.

Thin Like the surface skeleton, the curve skeleton should be thin. The key difference here is that the curve skeleton should be of a lower dimension than the surface skeleton, namely 1D instead of 2D. Again, for voxel representations, this property might mean that curve skeletons should be at most one voxel thick. In this case,

junctions in its branching structure can easily be detected by a local topological analysis in a small neighborhood of the junction.

Topology preserving The curve skeleton should preserve the topology of the original shape. At the very least, the curve skeleton should be connected. Furthermore, for each tunnel in the shape there should be corresponding curve-skeleton loop around the tunnel. Naturally, a cavity in the shape cannot be represented as a cavity in the curve skeleton, as there is no cavity notion for curves. One could require that there is at least one curve-skeleton loop around the cavity [23], which seems unnatural as the location of such a curve would be rather arbitrary. In this work, we avoid these difficulties by ignoring shapes with cavities.

Geometry preserving The curve skeleton preserves geometry to a lesser extent than the surface skeleton. Reconstruction of the original object from the curve skeleton is in general not possible when storing only the inscribed ball radius. Whereas the surface skeleton transform can be considered a lossless compression of the shape, the curve skeleton is "lossy". As far as we know, no attempts have been made to allow reconstruction of the original shape using the curve skeleton by storing extra information. Another approach would be to construct curve skeletons with a large number of branches: the more branches, the more faithful the reconstruction. However, this would defeat the purpose of the curve skeleton as a high-level abstraction of the shape.

Concerning weak geometry-preservation, the curve skeleton is required to reach into all the salient parts of the shape and to terminate at prominent points of the shape boundary. Prominent points are locally convex points with two large principal curvatures of equal sign, i.e., elliptical points. Usually, the curve skeleton is not constructed explicitly in this manner, but this follows implicitly from the particular curve-skeleton algorithm.

Centered The curve skeleton should somehow be centered within the shape. Being a one-dimensional structure, the centeredness property allows methods a lot of freedom in placing the curve skeleton. Telea and Vilanova construct a curve skeleton that is centered with respect to axis-aligned cross-sections of the shape [137]. Cornea proposes that a curve-skeleton point p should be centered with respect to the cross-section of the boundary with the plane that is normal to the curve skeleton at p [23]. It seems natural to (partly) inherit the centeredness of the curve skeleton from the surface skeleton, by restricting the curve skeleton to some subset of the surface skeleton. Indeed, Dey and Sun define the curve skeleton as the 1D subset of surface skeleton that is in the middle of the surface according to a "medial geodesic function" [36]. However, it can be argued that a relaxed sense of centeredness is sometimes desired, as the sensitivity of the surface skeleton to noise affects the curve skeleton [23].

Applications may have additional requirements to the aforementioned intrinsic properties. Like surface skeletons, curve skeletons should ideally be robust to noise, provide a multiscale or hierarchical representation, and have a low cost of computation. Additionally, the following properties can be defined:

Skeleton-to-boundary mapping For many applications, the curve-skeletonization process should provide a skeleton-to-boundary mapping. Each interior, non-junction point on the curve skeleton should map to a simple closed curve on the boundary. A curve-skeleton segment then maps to a contiguous area delimited by the two curves corresponding to the segment endpoints. The mapping can be done as a post-processing step, as is done by Brunner [19], but the mapping is more meaningful if the curve skeleton definition intrinsically provides it. This correspondence between skeleton and object is important in animation for instance, where posing the skeleton, or *control skeleton*, is used to animate the object.

Given the weak geometry-preservation of the curve skeleton and the skeleton-to-boundary mapping, we should be able to deduce the part structure of the shape from the skeleton. "Component-wise differentiation" is the property that the logical parts of the object have a one-to-one correspondence with the logical components of the curve skeleton [23]. Cornea et al. suggest that using the curve skeleton branches that are delimited by junctions give the logical parts of the shape [25]. However, this may not always deliver intuitive results. In Chapter 6 we present a better method for extracting the part structure from the curve skeleton.

Junction detection It can be argued [23] that methods that compute curve-skeleton junctions before or during skeletonization, produce better junctions than methods that compute them afterwards. In the former case, the junctions can be considered more meaningful as they are intrinsic to the curve skeleton definition, rather than a by-product.

Visibility The visibility property prescribes that each boundary point is visible from a curve-skeleton point: each skeleton point can be connected by a straight line that is completely inside the object [23]. This is important in virtual colonoscopy [55] for instance, in which the camera is navigated along the curve skeleton: the doctor should be able to easily inspect all areas of the colon.

Smoothness The curve skeleton should be smooth. Usually, the curves should be visually smooth, meaning that the curves have a continuous second-order derivative. This too is important in virtual endoscopy, to ensure that the camera moves smoothly, allowing easy inspection. Note that smoothness can also be achieved by a post-processing step and need not be intrinsic to the skeletonization process itself. The smoothness property may conflict with the centeredness property for some shapes.

Uniform Approaches that compute both the surface and curve skeleton should treat them in a uniform manner. Both skeletons should share the same set of properties, if applicable. For example, both the curve and surface skeletons should have a (related) multiscale representation. As opposed to methods combining curve and surface skeletons in an ad-hoc manner, a uniform definition and computation of both skeletons is preferable.

Inclusion For curve skeletons it might be desirable that they are fully included in the surface skeleton, $\mathcal{C} \subseteq \mathcal{S}$, motivated by the fact that curve skeletons can be considered

a limit case of surface skeletons for objects with local circular symmetry (e.g. a tube).

2.4 Skeletonization approaches

As stated in Section 2.3, an important requirement on both surface and curve skeletons is that they should be robust to noise to be useful in practice. Hence, skeletonization approaches usually take special precautions to ensure robustness of the skeleton. There are two main options [122] (see Figure 2.5):

- Simplification of the input shape, typically by smoothing.
- Simplification of the skeleton, called *pruning*.

Some approaches might use both simplification options (e.g. [130]), but most typically use one. Skeletonization approaches that use none of the options inevitably deliver unstable skeletons.

Simplification of the input shape rids the shape of noisy features so that no skeleton branch will be generated for them. A common approach is to smooth the shape, for example using the geometric heat equation. As additional advantage, multiscale skeletons can be obtained by varying the smoothing parameter. However, it has been shown that smoothing may introduce new skeleton structures [10]. Another problem is that in case the skeletonization method works on digitized images, pruning may still be needed afterward because the discretization itself causes spurious branches. We concentrate solely on pruning methods in this thesis.

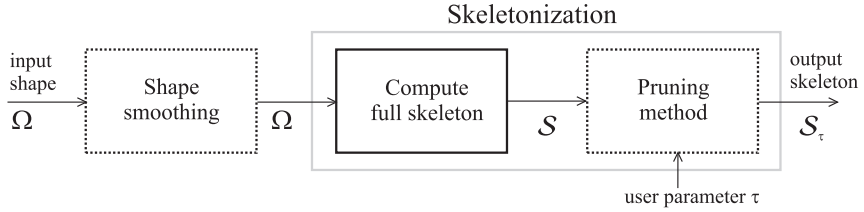


Figure 2.5: Schematic overview of the skeletonization process.

Simplification of the full skeleton $\mathcal{S}$ is called pruning. We denote the pruned skeleton $\mathcal{S}_\tau$, where τ indicates the pruning degree. The pruned skeleton is a subset of the skeleton. The quality of the pruned skeleton $\mathcal{S}_\tau$ thus depends on the full skeleton $\mathcal{S}$. In some algorithms, pruning is done while computing the skeleton, whereas in others, pruning is done after computing the full skeleton.

A large number of skeletonization approaches have been developed in the past. Several criteria can be used to characterize them:

- **Dimensionality** Methods work on 2D or 3D shapes, or both. The skeleton produced is embedded in the same space as the input shape. Higher-dimensional skeletonization approaches are rare and we do not consider them here.

- **Skeleton type** In 3D, methods may output either a surface skeleton, a curve skeleton or both. Methods that output a curve skeleton sometimes do this by taking the curve skeleton as a subset of the surface skeleton (e.g. [16, 36]), so these methods can be considered to compute both. Other methods output a curve skeleton directly (e.g. [78, 25]), without computing a surface skeleton first.
- **Input shape representation** Methods commonly work on either a boundary point sampling or a digital shape. Less common are methods that work directly on polyhedral shapes (e.g. [40, 130]) or spline-gons (2D shapes whose boundary is a closed spline, e.g. [121]). Although shape representations can be converted into each other, an approach can be characterized by the representation it naturally works on.
- **Output shape representation** Methods may output the (simplified) skeleton as a digital shape, a polyhedral representation, or consisting of a set of smooth curves (e.g. [121]). For methods working on digital shapes the output representation is usually equivalent to the input representation, but this is not necessarily the case. In [25] for example, the input is a voxel shape, whereas the output is a curve skeleton consisting of smooth curves.
- **Properties of the output** The produced skeletons may differ in the intrinsic and extrinsic properties they possess. We give some examples. Whereas most methods output connected (topology-preserving) skeletons, others do not (e.g. [40, 43]). Most methods take precautions to deliver robust skeletons, at least to some degree, but others do not (e.g. [16]). Some methods focus on producing smooth curve skeletons [1, 25], at the expense of centeredness.
- **Pruning method** The particular pruning method used also characterizes an approach. In Section 2.6 we discuss pruning methods in detail.
- **Techniques used in skeleton computation** Techniques employed in the skeletonization process are also considered characteristic. Common techniques used in skeletonization are: thinning, Voronoi diagram, distance field, and general field. We discuss these techniques in the next section.

2.5 Skeletonization techniques

In this section we discuss some techniques commonly found in skeletonization algorithms. For each technique we mention some idiosyncratic methods.

2.5.1 Thinning

Thinning, also called erosion or boundary peeling, is a process aimed at producing thin versions of digital shapes while maintaining their topology. It relies on concepts from digital topology and as such works on digital shapes [66]. By essentially simulating the fire front propagation in the grass-fire skeletonization formulation, the resulting thin structures are similar to skeletons. Thinning methods exist for both curve and surface skeletons. Simple points are iteratively removed from the set of object points Ω, where a *simple*

point is a lattice point whose removal does not change the topology of the shape [116]. Simple points are always located on the boundary of the object: removal of a point in the middle of the shape introduces a hole and violates homotopy. Removal of a simple point typically introduces new simple points. After removing all simple points, a thin version of the shape remains. Because simple points can be characterized locally, using so-called *templates*, which are small voxel blocks, thinning is an efficient process in terms of computation time and memory requirements.

Clearly, the order in which simple points are removed plays an important role in the final result, especially with respect to the rotational-invariance and centeredness of the result. Standard thinning methods typically do not produce centered results (e.g. [16]). Parallel methods aim at resolving this issue, by considering multiple points at once for removal instead of one by one. Another approach is to use the distance transform as the order in which the simple points are considered for removal [102].

Using only the notion of simple points, thinning methods would only preserve the topology of a shape and discard the geometrical information. All shapes without holes would be reduced to a single point, which is clearly an undesired result. To ensure geometry preservation, endpoints are identified that are on the tips of the significant skeleton parts. Endpoints will never be removed in the thinning process as to maintain the shape's essential geometry. Clearly, the precise manner of endpoint classification is important to the quality of the result. Unfortunately, characterizing endpoints cannot be done completely locally: from a local point of view, a bump on the boundary might be due to noise or might be a feature. Hence, although thinning methods itself are a purely local, their speed advantage is diminished when they are used for skeletonization purposes.

Note that in thinning methods, the feature points of a skeleton point are typically not stored in the thinning process. Therefore, it is difficult to prune the skeleton based on the feature points, and pruning has to be done based on properties of the skeleton itself, such as branch length (as in e.g. [18]). Svensson and Sanniti di Baja compute the curve skeleton using thinning [131], obtaining first the surface skeleton and then the curve skeleton. Their pruning step considers the branches, which are separated by junctions, one at a time and start with the peripheral branches to maintain homotopy. Branches are removed based on the ratio between the number of significant voxels and the total number of branch voxels. A voxel is considered significant if it is on a sheet intersection curve.

Lam et al. [68] provide a comprehensive survey on skeletonization methods by thinning.

2.5.2 Voronoi-diagram

Voronoi-based skeletonization methods [95, 6, 4, 37] use the Voronoi diagram to compute Blum skeletons. A Voronoi diagram is a decomposition of the space into cells based on *sites*, in such a way that each point in the cell is closest to its cell's site rather than to any other site. In skeletonization, boundary points or edges are used as sites. As cell borders (Voronoi edges) are equidistant to two sites, the relation with skeletons becomes clear: the skeleton is some subset of the edges in the Voronoi diagram. Voronoi edges that fall outside the shape are discarded. Points are usually used as sites, as it makes the Voronoi-diagram computation much simpler by using only intersections of half-spaces.

If a polyhedral input is used, the boundary needs to be sampled first. A problem of using points as sites is that, because an edge is generated between each two boundary points, a lot of edges need to be removed or pruned. The more sampling points, the more spurious branches the skeleton contains, which may look counter-intuitive at first sight, but becomes clear if we follow the Voronoi-diagram definition. Pruning methods for Voronoi techniques not only need to deal with boundary noise, but also needs to deal with these false edges. Methods that use higher-order Voronoi diagrams, which include not only points but also faces as sites, do not have the problem of false edges (e.g. [27, 130]).

In general, computing the Voronoi diagram is a computationally and memory intensive process, especially for large numbers of 3D sites. Exact arithmetic must be used to prevent approximation errors and degenerate cases can occur. Finally, the resulting structure can be complex to handle and might need to be faired first [52].

2.5.3 Distance field

The skeleton can be defined as the singularities in the Euclidean distance-to-boundary field, i.e., the distance transform. Singularities are the points where the field is non-differentiable. When the distance field is seen as a height map, the singularities can be seen as the mountain ridges and peaks. Niblack et al. [92] detect the local maxima and saddle points in the distance transform. The local maxima are connected to each other by uphill climbing in the distance transform, forming a connected skeleton. Wright and Fallside [144] apply a roof-profile detector from image processing on the distance transform. The filter size can be increased to detect the skeleton of large-scale features. The points are linked by an edge following scheme.

Ge and Fitzpatrick [45] identify problems with methods that detect the centers of maximal discs (CMDs) on exact Euclidean distance maps. The authors argue that the discrepancy between continuous and discrete inscribed discs gives problems, and provide a solution with a more accurate, but more costlier, CMD detection. The original object can be exactly reconstructed from these CMDs, however they are not guaranteed to form a connected skeleton, because of the discretized nature of the distance transform. Again, a steepest ascent method is used to connect CMDs and to produce homotopic skeletons.

Dimitrov et al. [39] measure the average outward flux of the gradient vector field of the Euclidean distance function. The limiting behavior of the average outward flux of this vector field through the boundary of a shrinking region is different for skeleton points than for non-skeleton points. This property lies at the heart of the Hamilton-Jacobi skeletonization algorithm [127]. Later, this work was extended to compute surface skeletons [17]. Thinning is used to assure connectedness of the result. Dimitrov et al. further show that the feature angle and skeleton-point type are related to the average outward flux under shrinking circular regions [39].

Schirmacher et al. [119] simulate the grass-fire model in continuous space to compute surface skeletons. The vertices of a polyhedral shape are propagated inward in its normal direction, that is, along the gradient of the distance transform. A vertex stops moving when it hits another part of the boundary. Vertices that come within a certain distance of each other are merged so that the shrunken boundary can directly be used as the skeleton output.

Telea and Van Wijk [136] compute the distance field of a 2D shape by simulating the front evolution using the fast marching method [120]. The key to this method is to integrate the observation that skeleton points are generated by the collapse of compact boundary segments during the front evolution. The 2D skeleton is detected by setting a threshold on the detected segment length. This algorithm has been extended to compute curve skeletons by slicing the object boundary along three orthogonal axes and running the 2D algorithm on each slice [137].

Gagvani et al. [43] present a "volume thinning" method for 3D shapes, based on detecting maxima in the (non-Euclidean) distance transform. This method favors reconstruction over connectivity of the skeleton. By relaxing the reconstruction criterion, skeletons of varying density can be obtained.

An important challenge of most field-based methods is their inherent reliance on detecting singularities of the computed field. First, such singularities typically involve higher-order derivatives, which can be unstable to compute on coarse-sampled images. Smoothing methods can be used to alleviate instabilities, but this has the undesired effect of removing details and also blurring the exact skeleton location. Moreover, the value of singularity detectors such as divergence is very weak over some skeletal configurations, when propagating fronts meet under very obtuse angles. This leads to delicate thresholding schemes which risk to yield thick and/or disconnected skeletons.

2.5.4 General field

For curve-skeletonization methods, the problem of the distance field is that it is not discriminative enough to obtain a 1D structure. For instance, the distance field of a box has the same values over the center surface-skeleton sheet. More general fields can be defined to resolve this. Another reason to use general fields is to obtain smoother skeletons: the distance field is not smooth near the skeleton by definition.

Ahuja and Chuang [1] define a repulsive force field inside the 2D shape, as if the shape boundary is charged with an electric charge. Particles are injected at certain seed points of the boundary and are followed. Cornea et al. [25] use a potential field for voxel shapes to compute multiscale curve skeletons with a fixed number of hierarchy levels. At the coarsest scale, the core skeleton is constructed by finding the critical points in the field, and are connected by integrating path lines over the vector field. Second, points of low-divergence are used as seeds to add new skeleton segments. At the finest scale, boundary points of high curvature are detected and are used as seeds to construct curve-skeleton segments. A desirable property of potential-field methods is that the resulting skeletons are much smoother because of the averaging effect that the field has. However, due to the nature of the field many boundary points affect a single object point, which may put the skeleton off-center. Another drawback is that seed points must be carefully chosen to obtain accurate, noise-free skeletons.

Ma et al. [78] center radial basis functions on vertices of a polyhedral input to create a smooth distance field inside the shape. A gradient descent algorithm is applied to each boundary vertex to locate the local maxima in the field. The local maxima are then connected by employing an active contour model to compute a curve skeleton of the 3D shape.

Zhou and Toga [146] find the curve skeleton of a voxel shape by computing the distances to a single seed point, for which usually a cusp is chosen. Points with similar distances are clustered, and replaced by a single point centered within the cluster, determined by the standard distance transform.

Hassouna and Farag [49] present a wave propagation method. The key idea of this method is to propagate wave fronts with different speeds from a seed curve-skeleton point. The first front propagates with a moderate speed to capture the object topology, while the second one propagates much faster at central points such that curve skeleton intersects the fronts at those points of maximum positive curvature.

2.5.5 Other techniques and hybrid approaches

The above mentioned techniques present a general classification of techniques and is by no means exhaustive. Other techniques have also been used. Shaked and Bruckstein [121] construct the medial axis from a smooth 2D shape by placing skeleton endpoints by analyzing boundary curvature, that is, according to the symmetry-curvature duality theorem [71], and tracing out the skeleton by solving first order differential equations. Kimmel et al. [64] present a similar method using this theorem. Li et al. [72] use a curve-skeletonization method which contracts edges in the input mesh, in order of their Euclidean length. The resulting curve-skeleton branches are disconnected and are connected in a post-processing step.

Methods might also combine techniques. For example, Bouix et al. [17] use a divergence-based operator to detect singularities in the distance transform, after which thinning is used to obtain a connected surface skeleton. Dey and Sun [36] give a formal definition for the curve skeleton computed by means of the *medial geodesic function* that is defined on the surface skeleton. This function is low near the periphery of the surface skeleton and increases toward the center. The function mimicks a distance function and is used to define the curve skeleton as the local maximum ridge of that function. The authors first compute a surface skeleton using a Voronoi-based method [37]. The medial geodesic function is computed on the edges of the skeleton mesh. A divergence-based measure, similar to [126], is used to detect the singularities of this function. The surface-skeleton mesh is eroded by collapsing mesh faces, edges, and vertices until a 1D connected structure remains.

2.6 Pruning methods

Pruning methods aim to make skeletons robust to noise. They output a pruned skeleton $\mathcal{S}_\tau$, with some pruning parameter τ, of a noisy shape Ω^τ, with noise parameter τ. The pruned skeleton $\mathcal{S}_\tau$ should be similar to the skeleton $\mathcal{S}$ of the original non-noisy shape Ω (see Figure 2.6). Ideally, there exists a pruning function such that $\mathcal{S}_\tau(\Omega^\tau) = \mathcal{S}(\Omega)$ for all shapes and for all noise functions. Unfortunately this is not possible, because pruning is restricted to removing spurious branches from the skeleton and is not able to relocate the skeleton [140]. For example, the dent in the rectangle boundary in Figure 2.3(d) also causes a dent in the skeleton. Pruning can remove the spurious branches due to the dent,

but will leave the dent in the skeleton intact. Hence, pruning can be used to produce skeletons that are topologically equivalent to the skeleton of the original non-noisy shape, that is, having the same branching structure.

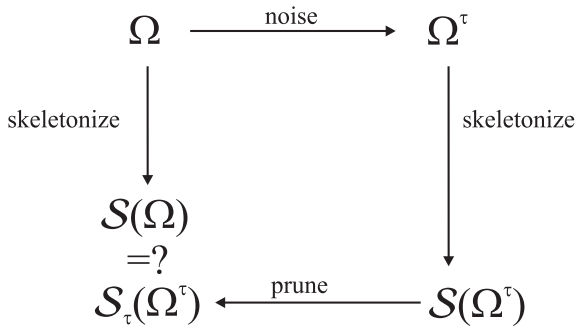


Figure 2.6: Schematic overview of pruning

Most pruning methods combine an *importance measure* with a *pruning strategy* [122]. The importance measure assigns to each skeleton point its significance in representing the shape. Small values should correspond to skeleton points that represent small-scale object details, whereas large values correspond to points that represent the main structure of the object. The pruning strategy is then used to remove skeleton points with a preference for the least important points, while maintaining the desirable properties of the skeleton. Such a strategy often incorporates a user parameter to specify the degree of pruning. We can identify four desirable **pruning requirements**:

- **Topology preserving** The pruning method should not disconnect the skeleton, in order to fulfill the skeleton homotopy property.
- **Weak geometry-preservation** Clearly, the original object cannot be fully reconstructed from the pruned skeleton: skeleton parts representing noise are removed. However, the pruned skeleton should still represent the important features of the shape. Note that the other intrinsic properties cannot be violated by the pruning method, as pruning only removes skeleton points. Therefore, the pruned skeleton is as centered, thin, and transformation invariant as the original.
- **Continuous** Pruning methods should be continuous, or stable: a small change in the pruning parameter should result in a small change in the resulting skeleton. Otherwise, the method is not very intuitive to use. Furthermore, a pruning method should have as few parameters as possible, and their effect upon which points get removed should be intuitive.
- **Generic** The pruning method should be as general as possible. It should apply to all kinds of shapes, and it should not be limited to any particular dimension, shape topology, or shape representation.
- **Efficient** The pruning method should be computationally efficient. Ideally, the pruning strategy has linear complexity in the number of skeleton points.

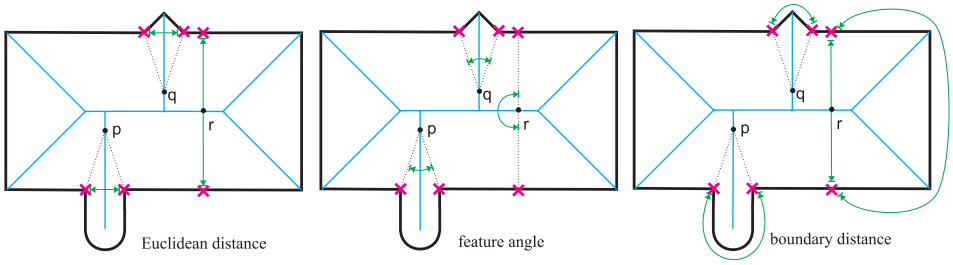


Figure 2.7: Illustrations of three importance measures: the local Euclidean-distance and feature-angle measures, and the global boundary-distance measure.

Common importance measures Various importance measures have been proposed in the past for both 2D and 3D skeletons. A frequently used importance measures is the *feature-angle measure*, which is proportional to the angle that the feature vectors make [14]. The smaller the angle, the less important the point is in representing the shape. The angle is proportional to the velocity with which the skeleton points emerge in the grass-fire model [122]. Indeed, when two meeting fire fronts are parallel, the feature angle attains its maximum of 180 degrees, and the skeleton unfolds with infinite speed. Blum and Nagel proposed the *boundary-axis ratio* [15], defined as the limiting ratio of boundary length to skeleton length, computed at each skeleton element. Another importance measure is the *radius measure*, using the radius of the inscribed disc as importance [32], which assumes that the skeleton inside small shape features has a relatively small distance to the boundary. The radius of the inscribed disc associated with a boundary point is also called the local feature size [3]. A related measure is the *Euclidean-distance importance measure*, which looks at the Euclidean distance between feature points [80]. It can also be useful to combine measures to strengthen them [80]. The *maximal-thickness measure* is defined as the maximal thickness of the erosion implied by the difference in the reconstructed shapes of the pruned and non-pruned skeleton [5]. Another measure is the *flux measure* introduced by Siddiqi et al. [126], which measures the average outward flux of the distance-field gradient on the boundary of a shrinking region (see Section 2.5.3). The *boundary-distance measure* was first introduced by Ogniewicz and Ilg [94] as the so-called "potential residual". The boundary-distance measure assigns to each point p the (minimum) distance along the boundary between the feature points of p. It thus measures the length of the boundary segment that "collapses" or folds onto the skeleton point. Skeleton points corresponding to small-scale boundary features have a small collapse, whereas points "deep inside" the object have a large collapse.

Figure 2.7 illustrates a selection of the mentioned importance measures.

Whereas some of the above-mentioned measures use only local information, others incorporate more global information. Local measures use only object properties from a small, limited vicinity of every object point. The feature angle, radius, and Euclidean-distance measures are local. They are incapable of distinguishing between locally identical configurations, such as those sketched in Figures 2.8. The pruning strategy must be used to enforce connectivity of the skeleton, and make sure that point q is removed be-

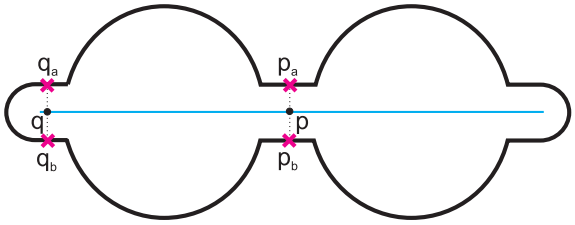


Figure 2.8: A 2D shape and its skeleton. Points $p, q \in \mathcal{S}$ and their feature points p_a, p_b and q_a, q_b respectively have locally identical configurations. Globally however, p is considered more important than q.

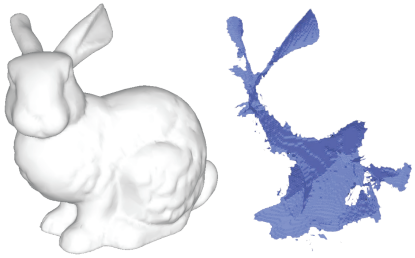


Figure 2.9: Surface skeleton (right) of the Stanford bunny (left) using the feature-angle measure (image from [40]).

fore point p. Global measures are not restricted to local information and can differentiate, in principle, between the two situations. The maximal thickness and boundary-distance measures are examples of global measures. In Figure 2.7, points p, and q both receive the same importance from the Euclidean-distance and feature-angle measures, but not from the boundary-distance measure. The only drawback of global measures is that they are more expensive to compute than local measures.

Pruning strategies Let $\rho : \Omega \to \mathbb{R}_+$ denote an importance measure. One of the simplest pruning strategies is the *thresholding strategy*, which simply puts a threshold τ on ρ. A pruned or *simplified skeleton* $\mathcal{S}_\tau$ is obtained by only keeping the points in the skeleton for which $\rho > \tau$:

$$\mathcal{S}_\tau(\Omega) = \left\{ p \in \mathcal{S}(\Omega) \;\middle|\; \rho(p) > \tau \right\}. \tag{2.3}$$

When ρ is a local measure, such as the feature angle, $\mathcal{S}_\tau$ will often contain disconnected parts. Hence, the thresholding strategy is sometimes called naive. Careful tweaking of τ may yield a connected skeleton, but this is not possible in general, and is undesirable as it puts a burden on the user. Figure 2.9 shows a surface skeleton of the Stanford bunny pruned using the feature angle [40]. Although the threshold was carefully chosen, the skeleton is still rather noisy and also contains holes, which should not be there as the bunny consists of a single connected component.

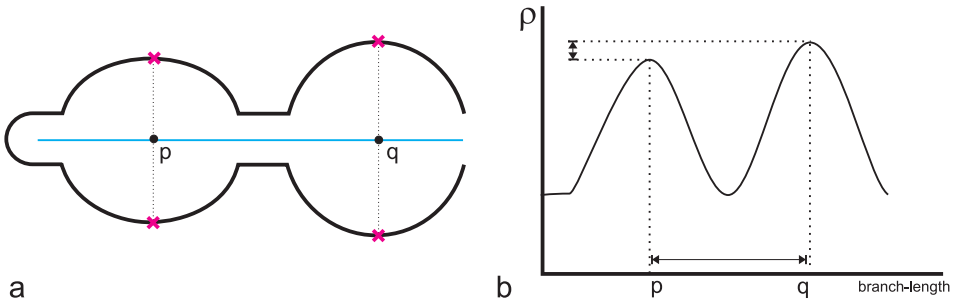


Figure 2.10: (a) A shape and its unpruned skeleton. (b) Pruned branch length in relation to threshold τ. The pruning method used is Euclidean feature-distance ρ in combination with an erosion strategy. This pruning method is not continuous.

To enforce the homotopy of the skeleton, an *erosion strategy* can be used on the skeleton. Points whose importance ρ is above the threshold τ are never removed by the erosion process. Points below the threshold are only removed if their removal is topologically invariant. Although thinning is a form of erosion, erosion is used here in the broader sense of removing skeleton elements while maintaining homotopy, and is not restricted to using digital topology. Thinning as an erosion strategy is for example used by Bouix and Siddiqi [17] to obtain connected curve and surface skeletons using the flux measure. Sud et al. [130] use the feature-angle measure in combination with an erosion step on the Voronoi-based skeleton. Erosion adds a computational cost, but the overhead is small as erosion is an efficient process. A more important drawback of an erosion strategy is that it makes the pruning non-continuous: a small change in the threshold might result in a significant change in the skeleton. This is illustrated in Figure 2.10, where the feature-distance measure ρ for a shape has two local maxima that differ only slightly, but are far apart on the branch.

Shaked and Bruckstein [122] propose the *rate pruning* strategy to assure the homotopy and continuity requirements. In rate pruning, pruning starts simultaneously from the medial axis endpoints and continues inwards along the medial axes. Instead of using the importance of skeleton points directly, the importance measure is merely used as a cue for the pruning speed. A high importance indicates slow pruning, whereas low importance indicates fast pruning. By taking the derivate of a measure along the 1D skeleton, one can go from the thresholding strategy to the rate pruning strategy. The authors formulate existing importance measures in the new strategy to unify discussion and facilitate comparison of measures. Rate pruning does not satisfy the genericity requirement on pruning methods, as it cannot be readily extended to 3D: the direction along which to prune is not clear as the skeleton of 3D shapes consists of 2D manifolds.

Monotonic importance measures To assure a continuous pruning method, without resorting to the non-generic rate pruning strategy, one could design an importance measure that is monotonically ascending. Such a measure has only a single local maximum (namely the global maximum), which could be considered the center of the skeleton

and thus of the object, and has no local minima (except at the skeleton endpoints). A monotonic importance measure in combination with the thresholding strategy is topology-preserving by construction. Additionally, such a pruning method will usually be continuous: the measure ρ can only contain plateaus, but these are generally small compared to the distance between local maxima of ρ in Figure 2.10(b). Another important advantage is that a monotonic measure that incorporates global criteria basically provides a multiscale skeleton representation. This multiscale representation is furthermore hierarchical, because pruning only removes points. Ogniewicz and Kübler [95] have observed that the monotonic boundary-distance measure gives rise to a skeleton scale-space. By progressively increasing τ, the noisy skeleton parts corresponding to noisy boundary features, which could also be considered small-scale features, are removed first. For higher values of τ, simplified skeletons are obtained that only represent features of at least size τ.

Unfortunately, although the thresholding strategy can also be applied in 3D, the boundary-distance measure cannot be readily extended to 3D in such a way that it is monotonically ascending. In Chapter 4 we detail on this problem and present a global importance measure, called the collapse measure, that copies the desirable properties of the boundary-distance measure to 3D shapes. Key to the definition of this measure is the incorporation of the curve skeleton, in addition to the surface skeleton.

Chapter 3

Tolerance-based Feature Transforms

Feature and distance transforms are important elements in many skeletonization methods. The definition of the skeleton (Eq. 2.1) relies on the identification of different boundary points located at exactly the same distance from the skeleton in order to produce connected, centered, and pixel-thin skeletons. As an extension to the normal feature transform, the Tolerance-based Feature Transform can be used to compute exact distance transforms [90], as a compact descriptor of shapes at different spatial scales [67], or to alleviate the effects of discretization in skeletonization. In this chapter, we compare four simple-to-implement methods for computing TFTs on binary images. Of these methods, the Fast Marching TFT and Euclidean TFT are new. We quantitatively and qualitatively compare all algorithms on speed and accuracy of both distance and feature results. Our analysis can be used beyond the application in skeletonization to help practitioners in the field to choose the right method for given accuracy and performance constraints. Although this chapter deals with 2D images, the presented algorithms can be straightforwardly extended to voxel shapes. A second goal of this chapter is to gain understanding of the accuracy limitations of an existing 2D skeletonization method, the AFMM [136]. This method combines the known Fast Marching Method for computing DTs together with the boundary-distance measure for computing robust and multiscale skeletons of 2D shapes. As we generalize the AFMM to compute 3D skeletons in Chapter 4, it is important to understand (and correct) its accuracy limitations.

3.1 Distance and feature transforms

A distance transform (DT) computes, for each point p of an object Ω, the distance $D(p) = \min_{q \in \partial\Omega} \|p - q\|$ to the nearest boundary-point q, called a *feature point* of p. Figure 3.1 shows an example distance transform.

The *feature transform* (FT), also known as the nearest-neighbor transform [26], as-

Figure 3.1: The distance transform of an object. The pixel intensity denotes distance to the object boundary.

signs to each object point $p \in \Omega$ the feature set $F(p)$:

$$F(p \in \Omega) = \left\{ q \in \partial\Omega \;\middle|\; \|q - p\| = D(p) \right\}, \tag{3.1}$$

where $\|q - p\|$ is the Euclidean distance between p and q. A *simple* FT computes only one feature per pixel, which is sufficient for some applications. We define the *tolerance-based* FT (TFT) as a map that assigns to each pixel all the feature points within a certain distance from the nearest feature:

$$F_\epsilon(p \in \Omega) = \left\{ q \in \delta\Omega \;\middle|\; \|q - p\| \leq D(p) + \epsilon \right\}, \tag{3.2}$$

where ϵ is a user-defined tolerance. One use of the TFT is to compute exact Euclidean DTs, as first observed by Mullikin [90] (see also Section 3.5). A second use of the TFT is to alleviate discretization artifacts in the normal feature transform for the purpose of skeleton computation. Figure 3.2(a) shows the medial axis endpoint for a continuous shape. The inscribed disc touches the boundary in a half-circle. In the discrete case in Figure 3.2(b), the four feature voxels in F are not a connected set. Using the TFT with an ϵ of 1 in Figure 3.2(c), the feature voxels of the endpoint correctly forms the discrete equivalent of a half-circle.

Figure 3.3 shows the TFT for an object computed using four different tolerances ϵ. The pixel intensity denotes the feature set size $|F_\epsilon|$. The feature sets of four selected pixels are shown using white line segments. For $\epsilon = 0$, it can be seen that pixel p has only one feature because the horizontal rectangle is of even height. Using a tolerance $\epsilon \geq 1$, the feature set $F_\epsilon(p)$ contains features from both sides of the rectangle.

Kovács et al. [67] introduced the D_ϵ-function, which is essentially equivalent to what we call TFT here. They argue that local maxima of D_ϵ are most important as they contain large amounts of boundary information, and together provide a compact medial descriptor. Furthermore, by varying ϵ shapes can be described at different spatial scales. They discuss the D_ϵ function in the context of human perception, but do not detail on efficient and accurate computation of the function, as we do this in chapter.

Overall, both DTs and FTs are used as a component in many applications from many different domains [26], ranging from image processing, pattern recognition, and shape

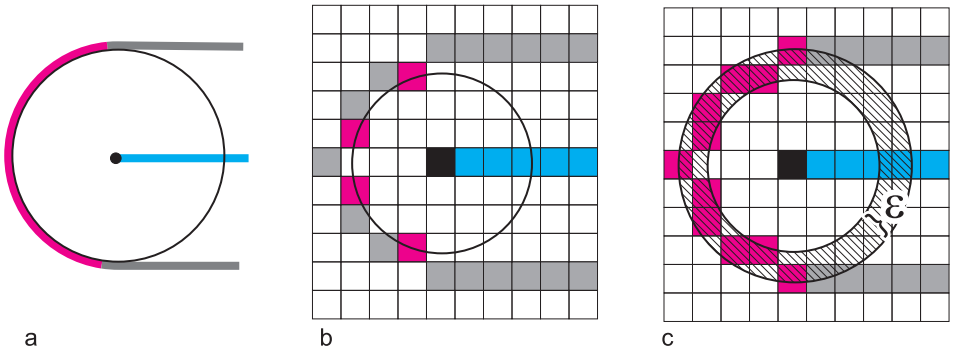


Figure 3.2: (a) Feature points in $\mathbb{R}^2$ form a connected set. (b) Feature points in $\mathbb{Z}^2$ form a disconnected set. (c) Tolerance-based feature points in $\mathbb{Z}^2$ form a connected set again.

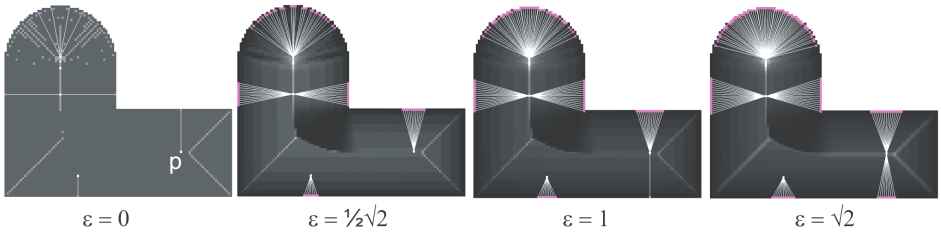


Figure 3.3: The TFT of an object using four different tolerances $\epsilon = 0, \frac{1}{2}\sqrt{2}, 1, \sqrt{2}$.

representation and modeling, to path planning, computer animation, skeletonization, and optimization algorithms.

DT and FT algorithms can be classified by the order in which they process the image pixels. *Raster scanning* algorithms (e.g., [31]) sequentially process pixels in scanline order, needing multiple passes in which pixels are assigned new minimum distances. Ordered propagation methods (e.g., [103]) reduce the number of distance computations needed by updating only pixels in a contour set, which propagates from the object boundary $\delta\Omega$ inwards. *Ordered propagation methods* accommodate (distance-based) stopping criteria easier than raster scanning ones, thus being more efficient for some applications. A well-known class of ordered propagation methods are level set and Fast Marching Methods (FMM) [120], which evolve the contour $\delta\Omega$ under normal speed (see Section 3.2). Although the FMM does not compute an exact Euclidean DT, the speed function it uses can be locally varied to compute more complex DTs, e.g., anisotropic, weighted, Manhattan, or position-dependent ones [120, 129]. Recent FMM extensions compute an FT [136, 137]. However, this is only a simple FT, and can be quite inaccurate in many cases. Applications using this feature set, such as skeletonization, can deliver wrong results, as pointed out by Strzodka and Telea [129].

As the above outlines, DT and FT methods have many, often subtle, trade-offs, which are not obvious to many practitioners in the field. In this chapter, we discuss several com-

petitive DT, FT, and TFT methods. Some of these methods extend existing ones, while others are new. We quantitatively and qualitatively compare the results of all methods with the exact TFT computed by brute force, and discuss the computational advantages and limitations of every method. The goal of our analysis is to provide a quantitative, practical guideline for choosing the "right" DT or (T)FT method to best match real-world application requirements, such as precision, performance, completeness, and implementation complexity. Such a method can form the basic building block for further 2D and 3D skeletonization methods, and other image processing methods.

This rest of this chapter is structured as follows. In Section 3.2 we discuss the FMM and we detail on its inaccuracies. In Section 3.3, we modify the existing Augmented Fast Marching Method (AFMM) to yield exact simple FTs, and illustrate its use by skeletonization applications. In Section 3.4, we extend this idea to compute TFTs by adding a distance-to-feature tolerance. In Section 3.5, we analyze Mullikin's raster scanning DT, and get insight into how to set our TFT tolerance to compute exact DTs. In Section 3.6, we present a novel method, called ETFT, based on a propagation order different from the FMM. In Section 3.7, we compare our new ETFT with the related graph-search method of Lotufo et al. [77], and also extend the latter to compute TFTs. Finally, we quantitatively compare all of the above methods (Sec. 3.8) and come to a conclusion (Sec. 3.9).

3.2 Fast Marching Method (FMM)

Level set methods are an Eulerian approach for tracking contours evolving in time. The fast marching method (FMM) [120] treats the special case of contours with constant sign speed functions. The arrival time $T : \mathbb{R}^2 \to \mathbb{R}_+$ of the contour at a point p is given by the solution of $\|\nabla T\|V = 1$ with $T = 0$ for the initial contour $\delta\Omega$. With a unit speed function V, we obtain the Eikonal equation:

$$\|\nabla T\| = 1 \,, \tag{3.3}$$

whose solution is the Euclidean DT of $\delta\Omega$. The FMM efficiently computes T using the idea that $T(p)$ depends only on the T values of p's neighbors $N(p)$ for which $T(N(p)) < T(p)$. The FMM builds T from the smallest computed T values by maintaining the pixels in the evolving contour, or narrow band, sorted on T. Pixels are split into three types, or in other words, can have three different *flags*: *known* pixels p^{K} have an already computed T; *temporary* pixels (p^{T}) have a T subject to update; and *unknown* pixels (p^{U}) have not yet been assigned a T value. Invariant is $T(p^{\mathrm{K}}) \leq T(p^{\mathrm{T}}) \leq T(p^{\mathrm{U}})$. Initially, all pixels on $\delta\Omega$ are known to have $T = 0$ and their neighbors are set to temporary. Next, the temporary pixel p with smallest T becomes known (as its T cannot be influenced by other pixels), its unknown neighbors $N^{\mathrm{U}}(p)$ become temporary, and their T values are updated based on their own known neighbors, until all pixels become known. Figure 3.4 depicts an iteration of the FMM. For a contour of length $B = |\delta\Omega|$ and area $N = |\Omega|$ pixels, the FMM needs $O(N \log B)$ steps, because it visits each object pixel once, and keeping the narrow band sorted on T in each iteration needs $O(\log B)$ steps.

The DT computed by the FMM is not exact. Errors occur due to the approximation of the gradient ∇T, usually of first or second order, the former being the most common.

Table 3.1: Differences between (approximate) FMM and exact Euclidean distances D ($\%e$: ratio of erroneous pixels to object pixels; $\max e$: maximum error; $\bar{e}$: average error).

image	img.size	$\max D$	$\%e$	$\max e$	$\bar{e}$
bird	238×370	49.82	89%	0.679	0.142
leaf	410×444	70.63	84%	1.013	0.290
dent	464×397	134.06	15%	1.210	0.082

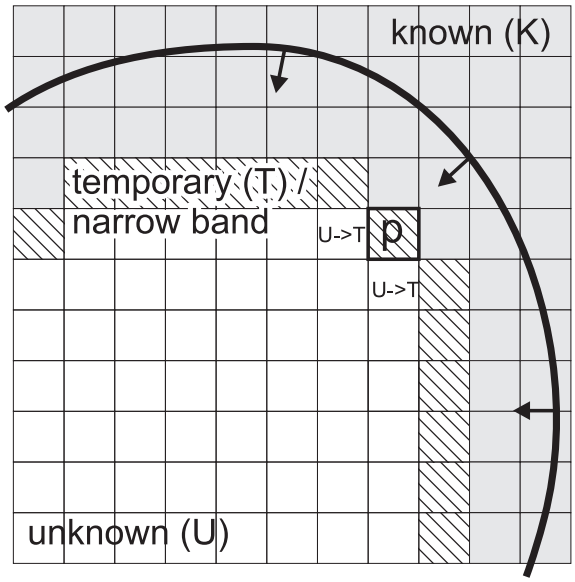


Figure 3.4: An iteration in the fast marching method.

Moreover, the errors are accumulated during the propagation. Sethian briefly treats the FMM accuracy [120, Sec. 12.3], but no comments are made on the implications for real-world applications. Figure 3.5 and Table 3.1 show differences between the FMM and the exact DT for some typical shapes. High errors (bright areas in Figure 3.5) seem to "diffuse" or spread away from boundary concavities. Indeed, a temporary point at a narrow band concavity has just one known neighbor N^{K}, so its distance is updated from a single known $T(N^{\mathrm{K}})$ value. A point at a narrow band convexity has several known neighbors, so its distance T is updated using more information. The maximal DT error can easily exceed 1 pixel (cf. Table 3.1), and can become arbitrarily large (depending on the image size). As we shall see in the next section, such errors can easily lead to incorrect skeletons when the FMM is used in skeletonization.

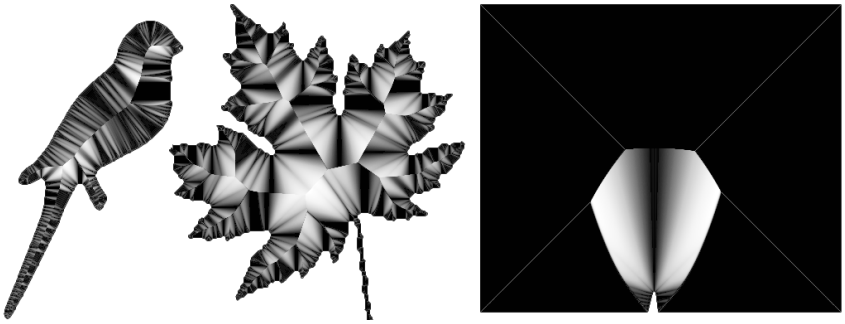


Figure 3.5: Differences between the (approximate) FMM distance and the exact Euclidean distances for the 'bird', 'leaf', and 'dent' images. Black indicates no error, white indicates the maximum error. See Table 3.1 for the exact values.

3.3 AFMM Star

The Augmented FMM (AFMM) [136] computes one feature per pixel by propagating an arc-length parameterization U of the initial boundary $\delta\Omega$ together with FMM's distance value T. $U(p)$ basically identifies a feature point $F(p)$. When a narrow band pixel p is made known and its unknown direct 4-neighbors $a \in N_4^{\mathrm{U}}(p)$ are added to the narrow band, $U(a)$ is set to $U(p)$. After propagation has completed, for all points q where U varies with at least τ over $N_4(q)$, a segment of at least length τ from the original boundary collapses. Hence, the above point set $\{q\}$ represents a (pruned) skeleton of Ω with τ as the pruning parameter. This is precisely the boundary-distance measure as described in Section 2.6.

AFMM's complexity remains the same as for the FMM, namely $O(N \log B)$, where N is the number of object pixels and B the boundary size, because it adds just the propagation of one extra value U. Similar methods exist for digital images [29] and for polygonal contours [95], respectively.

However efficient and effective for computing simple FTs and 2D skeletons, the AFMM has several accuracy problems when computing U, as can be easily seen from the resulting skeletons. Errors show up as skeleton branches having the wrong angle, are too thick, or are disconnected (e.g., Figure 3.6 left). Such errors occur when a skeleton branch emerges from a convex shape part delimited by two sharp concavities such as the leaf's stem. The angle of the continuing branch, which is a ligature, is incorrect. The reason is that the value $U(a)$ is determined by only *one* pixel $p \in N_4(a)$, namely the p that is first made known.

We propose to solve this problem as follows. For a point a that is made temporary, we set $F(a)$ (or equivalently $U(a)$) to the closest feature among the neighbor's feature sets, i.e.:

$$F(a) = \underset{q \in F(N_8^{\mathrm{K},\mathrm{T}}(a))}{\arg\min} \|q - a\| \,. \tag{3.4}$$

This method, which we call AFMM Star, solves AFMM's inaccuracy problems, i.e., yields a reliable simple FT method and subsequently a correct 2D skeletonization method.

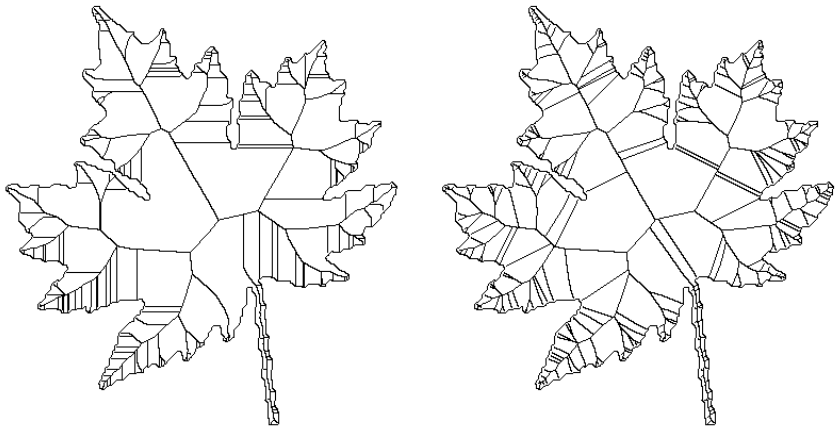


Figure 3.6: AFMM skeletonization errors (left). AFMM Star skeletonization (right).

We observe that AFMM Star computes robust, thin, and connected skeletons for arbitrarily complex noisy 2D boundaries (e.g., Figure 3.6 right).

One remaining problem is that we use the numerically inexact FMM DT (cf. Sec. 3.2). For practical applications, e.g. skeletonization, incorrect skeleton points will occur only where the FMM DT error exceeds 1 pixel. From Table 3.1, we see that this happens only at a very few pixels of relatively large objects. This gives a quantitative estimate of the AFMM Star limitations.

3.4 Fast Marching TFT

The AFMM Star is a simple FT, i.e., it computes just one feature per point. However, some applications, such as angle-based skeletonization [40] require all features to be found. Moreover, tolerance-based FTs are desired for the reasons outlined in Section 3.1.

We now propose the novel Fast Marching Tolerance-based Feature Transform (FMTFT) which computes for each pixel p a feature set F_ϵ whose size depends on a user-defined distance *tolerance* ϵ, as defined in Eq. 3.2. The pseudocode is shown in Figure 3.7. The distances D^{f} are computed using the FMM [120]. The function $fmmdistance$ (line 15) is based on the arrival times of the known neighbors of a (see [136] for an implementation of this function). We initialize the feature set $F(p) = \{q \in \delta\Omega \;\big|\; \|q - p\| \leq \epsilon\}$ for all $p \in \delta\Omega$. When the distance of a point a is updated during the FMM evolution, we simultaneously construct a candidate feature set C (line 16):

$$C(a) = \bigcup_{q \in N_s^{\mathrm{K},\mathrm{T}}(a) \cup \{a\}} F(q)\,, \tag{3.5}$$

where s is the neighborhood size. Next, let the distance $D(a) = \min_{q \in C(a)} \|q - a\|$. $D(a)$ is more accurate than the FMM distance $D^{\mathrm{f}}(a)$, because it is computed directly

1: **for** each $p \in \Omega \cup \bar{\Omega}$ **do**
2: **if** $p \in \bar{\Omega}$ **then**
3: $flag(p) \leftarrow \mathtt{K}, D^{\mathrm{f}}(p) \leftarrow 0$
4: **else if** $p \in \delta\Omega$ **then**
5: $flag(p) \leftarrow \mathtt{T}, D^{\mathrm{f}}(p) \leftarrow 0, F(p) \leftarrow \{q \in \delta\Omega \mid \|q - p\| \leq \epsilon\}$
6: **else if** $p \in \Omega \wedge p \notin \delta\Omega$ **then**
7: $flag(p) \leftarrow \mathtt{U}, D^{\mathrm{f}}(p) \leftarrow \infty$
8: **end if**
9: **end for**
10: **while** $\exists_q flag(q) = \mathtt{T}$ **do**
11: $p \leftarrow \underset{q:flag(q)=\mathtt{T}}{\arg\min} D^{\mathrm{f}}(q)$
12: $flag(p) \leftarrow \mathtt{K}$
13: **for** each $a \in N_4^{\mathtt{U},\mathtt{T}}(p)$ **do**
14: $flag(a) \leftarrow \mathtt{T}$
15: $D^{\mathrm{f}}(a) \leftarrow fmmdistance(a, N_4^{\mathtt{K}}(a))$
16: $C \leftarrow \bigcup_{q \in N_s^{\mathtt{K},\mathtt{T}}(a) \cup \{a\}} (F(q))$
17: $D(a) \leftarrow \min_{q \in C} \|q - a\|$
18: $F(a) \leftarrow \{q \in C \mid \|q - a\| \leq D(a) + \epsilon\}$
19: **end for**
20: **end while**

Figure 3.7: Fast Marching TFT (FMTFT).

as the distance from a to its nearest feature, while D^{f} is computed incrementally by a first-order approximation of the gradient. D^{f} is used only to determine the propagation order (line 11), as for the AFMM Star (Sec. 3.3). The tolerance-based feature set $F(a)$ is constructed by *trimming* C in line 18:

$$F_\epsilon(a) \leftarrow \left\{ q \in C \;\middle|\; \|q - a\| \leq D(a) + \epsilon \right\} . \tag{3.6}$$

Thus, this algorithm assumes that the feature set of a pixel a can be determined from the feature sets of a's neighbors. Statement 3.6 also occurs in all other to-be-discussed methods (line 30 in Figure 3.8, line 17 in Figure 3.10, and line 16 in Figure 3.11).

The accuracy of D is influenced by the neighborhood size s and the tolerance ϵ. D can be made more accurate by increasing s. In general however, D cannot be made exact no matter the choice of s. This is because the Voronoi regions of feature pixels are not always 4 or 8-connected sets on a discrete grid [26]. In contrast, ϵ *can* be set so that all distance errors are eliminated. This was also observed by Mullikin, in a related context, as detailed in the next section.

3.5 ϵ-Vector Distance Transform

Mullikin presents a scan-based algorithm for computing exact Euclidean DTs [90]. He first identifies pixel arrangements for which Danielsson's scan-based vector distance transform (VDT) with 4-neighborhoods [31] yields inexact distances. The problem of the VDT is that it stores only one feature. In Figure 3.9, the VDT computes that $F(q) = \{a\}$ and $F(r) = \{b\}$. For p, one of the nearest features from its 4-neighbors is taken. Thus

```
1: for each p ∈ δΩ do
2:     F(p) ← {q ∈ δΩ | ||q − p|| ≤ ε}
3: end for
4: for y from 0 to N − 1 do
5:     for x from 0 to M − 1 do
6:         update( (x,y), (x-1,y-1) ) if s = 8
7:         update( (x,y), (x,y-1) )
8:         update( (x,y), (x+1, y-1) ) if s = 8
9:         update( (x,y), (x-1, y) )
10:    end for
11:    for x from M − 1 downto 0 do
12:        update( (x,y), (x+1,y) )
13:    end for
14: end for
15: for y from N − 1 downto 0 do
16:    for x from 0 to M − 1 do
17:        update( (x,y), (x-1, y+1) ) if s = 8
18:        update( (x,y), (x, y-1) )
19:        update( (x,y), (x+1, y+1) ) if s = 8
20:        update( (x,y), (x-1, y) )
21:    end for
22:    for x from M − 1 downto 0 do
23:        update( (x,y), (x+1,y) )
24:    end for
25: end for
26: procedure update(a,b)
27: if a, b ∈ Ω then
28:    C ← F(a) ∪ F(b)
29:    D(a) ← min_{q∈C} ||q − a||
30:    F(a) ← {q ∈ C | ||q − a|| ≤ D(a) + ε}
31: end if
32: end procedure
```

Figure 3.8: ϵ-Vector Distance Transform (ϵVDT). The image has dimensions $M \times N$.

$F(p) \in F(q) \cup F(r) = \{a, b\}$, while the actual nearest feature is $F(p) = \{c\}$. This situation occurs when there are three object pixels a, b, c so that $\|\vec{aq}\| < \|\vec{cq}\|$, $\|\vec{br}\| < \|\vec{cr}\|$, $\|\vec{cp}\| < \|\vec{ap}\|$, and $\|\vec{cp}\| < \|\vec{bp}\|$, i.e., when the hatched area contains a grid point. Mullikin proposes, in his ϵVDT, to store *all* nearest features, and additionally all features at a distance within a certain tolerance ϵ. Essentially, ϵVDT computes tolerance-based feature sets as defined in Eq. 3.2. Mullikin shows that an exact distance transform is obtained when $\epsilon \geq \sqrt{D}/D$, where D is the number of spatial dimensions. This result can also be used for the other TFT methods discussed in this chapter.

The ϵVDT computes tolerance-based feature sets only as a means to compute exact DTs. Mullikin does not detail on the accuracy of the feature sets themselves [90]. Moreover, he uses only 4-neighborhoods as these are sufficient for exact Euclidean distances. We extended ϵVDT to also use 8-neighborhoods (see pseudocode in Figure 3.8). These are useful for improving the feature set accuracy, as shown in Table 3.2. The pseudocode can be found in Figure 3.8. The ϵVDT is compared to the other methods in Section 3.8.

Besides the fact that the ϵVDT uses a scan-based approach, another conceptual difference with the FMTFT is that it uses a write formalism instead of a read formalism [142].

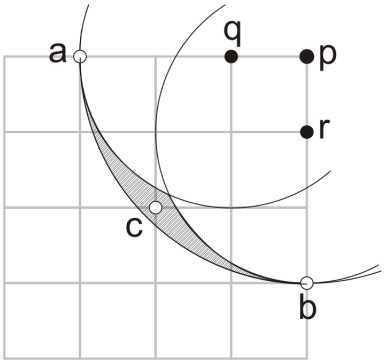


Figure 3.9: Points a, b, and c are boundary points. Point p, q, r are object points.

Whereas in the FMTFT the candidate feature set C of a pixel a is constructed by reading from all neighboring pixels (line 16 in Figure 3.7), the ϵVDT writes information from a single neighbor to a (line 28 in Figure 3.8).

3.6 Euclidean TFT

The FMTFT visits points in the order of the inaccurate FMM distances (Sec. 3.4). Although this keeps the original FMM advantage of using different speed functions (if so desired), an erroneous propagation order potentially influences the distance and feature set accuracy (Sec. 3.1). The idea comes thus naturally to design an ordered propagation FT which visits the points in order of the accurately computed distances. We present the pseudocode of this new method, called Euclidean TFT, in Figure 3.10. The neighborhood size s (line 15) and tolerance ϵ (line 17) have the same meaning as for the FMTFT and ϵVDT discussed above. The initialization is the same as for the FMTFT, it also uses a read formalism, and the propagation is still in the order of increasing distances. However, where the FMTFT propagates in the order of D^{f} (Figure 3.7, line 11), the ETFT propagates on the more accurate distances D (Figure 3.10, line 11).

We found out that the above exact-distance propagation order yielded a comparable speed and accuracy, posing no advantage over the FMM order. Thus, we made the following change. Whereas the FMTFT updates all pixels $a \in N^{U,T}(p)$ (Figure 3.7, line 13), the ETFT was made to update only pixels $a \in N^{U}(p)$ (Figure 3.10, line 13). Now the ETFT updates pixels only once, trading accuracy for speed (see Table 3.2).

3.7 Graph-search TFT

The FMTFT and ETFT resemble the graph-search approach of Lotufo et al. [77]. However, the graph-search method uses a write formalism, and propagates only one feature per pixel, i.e., it is a simple FT. We extended Lotufo's algorithm to a TFT, so that it can be readily compared to the FMTFT, ϵVDT, and ETFT methods. Figure 3.11 gives the

1: **for** each $p \in \Omega \cup \bar{\Omega}$ **do**
2: **if** $p \in \bar{\Omega}$ **then**
3: $flag(p) \leftarrow \mathtt{K}, D(p) \leftarrow 0$
4: **else if** $p \in \delta\Omega$ **then**
5: $flag(p) \leftarrow \mathtt{T}, D(p) \leftarrow 0, F(p) \leftarrow \{q \in \delta\Omega \mid \|q - p\| \leq \epsilon\}$
6: **else if** $p \in \Omega \wedge p \notin \delta\Omega$ **then**
7: $flag(p) \leftarrow \mathtt{U}, D(p) \leftarrow \infty$
8: **end if**
9: **end for**
10: **while** $\exists_q flag(q) = \mathtt{T}$ **do**
11: $p \leftarrow \underset{q:flag(q)=\mathtt{T}}{\arg\min} D(q)$
12: $flag(p) \leftarrow \mathtt{K}$
13: **for** each $a \in N_4^{\mathtt{U}}(p)$ **do**
14: $flag(a) \leftarrow \mathtt{T}$
15: $C \leftarrow \bigcup_{q \in N_s^{\mathtt{K},\mathtt{T}}(a) \cup \{a\}} (F(q))$
16: $D(a) \leftarrow \min_{q \in C} \|q - a\|$
17: $F(a) \leftarrow \{q \in C \mid \|q - a\| \leq D(a) + \epsilon\}$
18: **end for**
19: **end while**

Figure 3.10: The Euclidean TFT algorithm (ETFT).

pseudocode for this extension, called the Graph-search TFT (GTFT). Now the differences between the ETFT and GTFT methods become visible. While GTFT uses only the flags T and K and updates all neighboring pixels a of p flagged as T (Figure 3.11, line 13), ETFT also uses the flag U and only updates these pixels (Figure 3.10, line 13). Since there are in general less pixels flagged U in ETFT than T in GTFT, ETFT updates less pixels per iteration. However, the update of a pixel in ETFT involves more work as all neighbors of a are used (Figure 3.10, line 15), whereas in GTFT only p is used (Figure 3.11, line 14). The running time differences are detailed in the next section.

3.8 Comparison

There are several differences between the four presented algorithms. The first is that ϵVDT is not an ordered-propagation approach and as such does not use flags, unlike the others. Second, a distinction can be made between the use of read versus write formalisms [142]. The FMTFT, ETFT, and ϵVDT all use the read formalism. The features of a are determined by reading the neighbors of a. This mimics a convolution. The GTFT on the other hand uses a write formalism. The feature set of a is a combination of the feature sets of a and p. The third difference lies in the number of times pixels can be updated. In the FMTFT and GTFT, narrow-band pixels may be updated multiple times. In the ϵVDT, each pixel is also updated more than once. In the ETFT, however, each pixel gets its value only once because only unknown points are written to. The differences between the ordered-propagation approaches are illustrated in Figure 3.12.

Unlike DT methods, the computational complexity of (T)FT methods depends on the feature set sizes and is therefore strongly input dependent. For example, the center of a circle is a worst case, as its feature set contains all boundary points. The feature set

for each $p \in \Omega \cup \bar{\Omega}$ **do**
  if $p \in \bar{\Omega}$ **then**
    $flag(p) \leftarrow \mathtt{K}, D(p) \leftarrow 0$
  else if $p \in \delta\Omega$ **then**
    $flag(p) \leftarrow \mathtt{T}, D(p) \leftarrow 0, F(p) \leftarrow \{q \in \delta\Omega \mid \|q - p\| \leq \epsilon\}$
  else if $p \in \Omega \wedge p \notin \delta\Omega$ **then**
    $flag(p) \leftarrow \mathtt{T}, D(p) \leftarrow \infty$
  end if
end for
while $\exists_q flag(q) = \mathtt{T}$ **do**
  $p \leftarrow \underset{q:flag(q)=\mathtt{T}}{\arg\min} \; D(q)$
  $flag(p) \leftarrow \mathtt{K}$
  for each $a \in N_s^{\mathtt{T}}(p)$ **do**
    $C \leftarrow F(a) \cup F(p)$
    $D(a) \leftarrow \min_{q \in C} \|q - a\|$
    $F(a) \leftarrow \{q \in C \mid \|q - a\| \leq D(a) + \epsilon\}$
  end for
end while

Figure 3.11: The Graph-search TFT algorithm (GTFT).

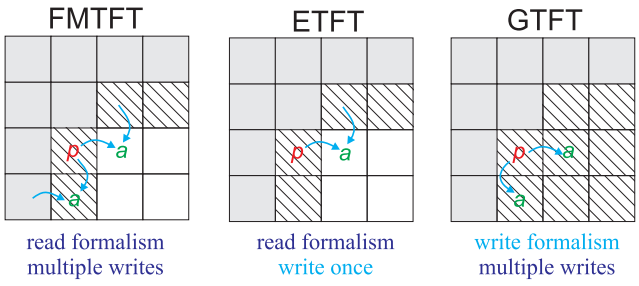


Figure 3.12: The differences between the various algorithms.

size for points inside convex object regions increases with distance to boundary. When updating a pixel p, the whole candidate feature set for p must be inspected. For N image pixels, and B boundary pixels, this poses a worst case of $O(N(B + \log B))$ for the three propagation-based methods and $O(NB)$ for ϵVDT. Luckily, average real-world images are far from this worst case. It is difficult to mathematically characterize the average input image. Nevertheless, to give more insight into real-world running times, we empirically compare all discussed TFT methods on speed and accuracy of both distances and feature sets. We use images that are often used as typical input for image processing algorithms (e.g. [126, 9]).

We implemented all methods and ran them on a Pentium IV 3GHz with 1 GB RAM. Some design decisions had to be made here. For the propagation-based methods (FMTFT, ETFT, GTFT) we used a priority queue to efficiently find the temporary pixel at minimum distance to the boundary. Feature sets are stored as STL multimaps [91] containing (distance,feature) pairs, so that merging two feature sets takes $O(n \log n)$ time (as we must avoid duplicates), while trimming the candidate set takes $O(\log n)$. To prevent

Figure 3.13: Locations of feature errors for the leaf image, $\epsilon = \sqrt{2}$. From left to right: FMTFT8, ϵVDT8, GTFT8, and ETFT8. The boundary and erroneous pixels are thickened for better display.

floating point precision problems when performing the statement in Eq. 3.6, it is needed to evaluate $\|q - p\| \leq D(p) + \epsilon + \phi$ instead, where ϕ is larger than the minimum representable difference between two floating point numbers. However, ϕ must be chosen smaller than $\left|\frac{1}{2}\min_{p,q,r\in\Omega} \|\vec{pq}\| - \|\vec{pr}\|\right|$: half of the minimum difference between two distances that can occur on the grid. Alternatively, integer arithmetic can be used when the equations were rewritten. In our experiments, the use of ϕ improved the accuracy by two orders of magnitude.

Table 3.2 compares the distances D^m and feature sets F^m produced by the methods m to the exact distances D^e and features F^e, calculated using a brute-force approach. The table shows measurements on the 'leaf' image, and cumulative measurements on 10 different images. The considered methods are the FMM, FMTFT, GTFT, ϵVDT, and ETFT. We do not consider the AFMM Star, as it is a simple FT and thus superseded by the more general FTs. For all methods, except the FMM, we ran the 4 and 8 neighborhood variants, and used 4 different tolerances: 0, $\frac{1}{2}\sqrt{2}$, 1, and $\sqrt{2}$. For the FMM, we used only the first-order distance gradient approximation (as mentioned in Sec. 3.2), which needs just the 4-neighborhood. The variants are denoted as, e.g., FMTFT4 $\epsilon 0$ for the Fast Marching TFT using a 4-neighborhood and zero tolerance.

Table 3.2 shows that the distance errors decrease by increasing either the neighborhood size s or the tolerance ϵ. As previously noted, increasing the neighborhood size does not always eliminate all errors. Indeed, all methods produce one error for the leaf image with $s = 8$ and $\epsilon = 0$. As predicted, using $\epsilon = \frac{1}{2}\sqrt{2}$ eliminates all errors for all methods; higher tolerances are not useful for computing exact distances. Finally, our novel method ETFT4 $\epsilon\frac{1}{2}\sqrt{2}$ is the fastest of all considered methods.

We next examine the accuracy of the computed feature sets. We compare feature sets by comparing the average relative differences between a method's features and the exact (brute-force method) features, denoted in column "$\bar{e_r}$". Let the relative error e_r of a pixel p be $e_r(p) = \left|\frac{|F^m(p)|-|F^e(p)|}{|F^e(p)|}\right|$, then, $\bar{e_r}$ is the average of e_r over all pixels $p \in \Omega$. The tolerance ϵ is not only a means to compute exact distances, but is also a user parameter for computing feature sets. Unlike for distances, relaxing the tolerance ϵ increases the errors for feature sets. Indeed, it is more difficult to identify all features that are within

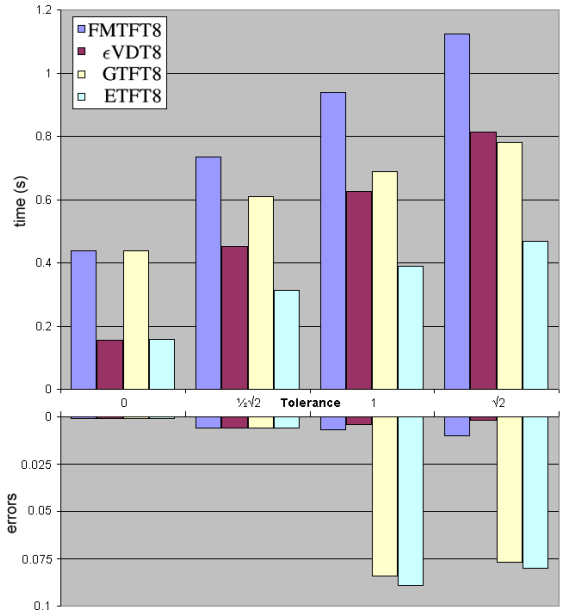


Figure 3.14: Timings and relative error (e_r) of the 8-neighborhood variants for the 'leaf' image.

$D(p) + \epsilon$ for higher ϵ. For $\epsilon > 0$, none of the considered methods deliver the complete feature set as determined by the brute force application of Eq. 3.2, although some have only a few erroneous pixels. From Table 3.2, we see that the 8-neighborhood variants have the best accuracy ($< 0.1\%$). Of these, ETFT8 is the fastest (see also Figure 3.14). For applications needing maximum accuracy, ϵVDT8 is the method of choice. Although ϵVDT8 has a better complexity, it is probably slower because of the hidden constant: the image is scanned twice. Finally, we illustrate the locations of the pixels with erroneous feature sets for the leaf image in Figure 3.13.

3.9 Conclusion

In this chapter, we both analyzed and extended several distance and feature transform methods for binary images. Our goal was to provide a guide for practitioners in the field for choosing the best method that meets application-specific accuracy, speed, and output completeness criteria, with an eye on the application of distance and feature transforms in skeletonization methods. First, we perfected the existing simple FT method AFMM to deliver more accurate results (AFMM Star, Sec. 3.3). We next extended this method to a new tolerance-based feature transform, FMTFT, that allows e.g. overcoming undesired sampling effects when computing skeletons (Sec. 3.4). Next, we discussed three other easy-to-implement TFT methods: the existing ϵVDT (Sec. 3.5), the new ETFT (Sec. 3.6),

Table 3.2: In each row: distances D^m and feature sets F^m of method m are compared to the exact distances D^e and features F^e. Left table: method comparison for the 'leaf' image. Right table: cumulative comparison of 10 different images. For distances (D), we show: the number of erroneous pixels ($\#e$), maximum distance error ($\max e = \max_p |D^m(p) - D^e(p)|$), and average distance error $\bar{e}$. For features (F) we show: the number of pixels for which feature counts are different ($\#e$), and the average relative error "$\bar{e_r}$" (see text). Timings are denoted in seconds in column t.

	D^m			F^m		
method m	$\#e$	$\max e$	$\bar{e}$	$\#e$	$\bar{e_r}$	t
FMM	29381	1.01	0.25	170	0.182%	0.31
FMTFT4 $\epsilon 0$	18	0.19	0.00	320	0.309%	0.31
ϵVDT4 $\epsilon 0$	22	0.19	0.00	218	0.147%	0.13
GTFT4 $\epsilon 0$	22	0.19	0.00	218	0.147%	0.28
ETFT4 $\epsilon 0$	18	0.19	0.00	317	0.304%	0.13
FMTFT8 $\epsilon 0$	1	0.04	0.00	1	0.001%	0.44
ϵVDT8 $\epsilon 0$	1	0.04	0.00	1	0.001%	0.16
GTFT8 $\epsilon 0$	1	0.04	0.00	1	0.001%	0.44
ETFT8 $\epsilon 0$	1	0.04	0.00	1	0.001%	0.16
FMTFT4 $\epsilon \frac{1}{2}\sqrt{2}$	0	0.00	0.00	12428	8.452%	0.50
ϵVDT4 $\epsilon \frac{1}{2}\sqrt{2}$	0	0.00	0.00	540	0.125%	0.31
GTFT4 $\epsilon \frac{1}{2}\sqrt{2}$	0	0.00	0.00	31835	23.120%	0.31
ETFT4 $\epsilon \frac{1}{2}\sqrt{2}$	0	0.00	0.00	26914	16.744%	0.19
FMTFT8 $\epsilon \frac{1}{2}\sqrt{2}$	0	0.00	0.00	34	0.006%	0.73
ϵVDT8 $\epsilon \frac{1}{2}\sqrt{2}$	0	0.00	0.00	34	0.006%	0.45
GTFT8 $\epsilon \frac{1}{2}\sqrt{2}$	0	0.00	0.00	34	0.006%	0.61
ETFT8 $\epsilon \frac{1}{2}\sqrt{2}$	0	0.00	0.00	34	0.006%	0.31
FMTFT4 $\epsilon 1$	0	0.00	0.00	7674	3.879%	0.66
ϵVDT4 $\epsilon 1$	0	0.00	0.00	162	0.025%	0.42
GTFT4 $\epsilon 1$	0	0.00	0.00	22306	11.805%	0.41
ETFT4 $\epsilon 1$	0	0.00	0.00	15916	7.763%	0.25
FMTFT8 $\epsilon 1$	0	0.00	0.00	44	0.007%	0.94
ϵVDT8 $\epsilon 1$	0	0.00	0.00	32	0.004%	0.63
GTFT8 $\epsilon 1$	0	0.00	0.00	250	0.084%	0.69
ETFT8 $\epsilon 1$	0	0.00	0.00	284	0.089%	0.39
FMTFT4 $\epsilon\sqrt{2}$	0	0.00	0.00	17654	8.083%	0.75
ϵVDT4 $\epsilon\sqrt{2}$	0	0.00	0.00	142	0.018%	0.55
GTFT4 $\epsilon\sqrt{2}$	0	0.00	0.00	36718	19.525%	0.42
ETFT4 $\epsilon\sqrt{2}$	0	0.00	0.00	29446	14.286%	0.30
FMTFT8 $\epsilon\sqrt{2}$	0	0.00	0.00	43	0.010%	1.13
ϵVDT8 $\epsilon\sqrt{2}$	0	0.00	0.00	16	0.002%	0.81
GTFT8 $\epsilon\sqrt{2}$	0	0.00	0.00	273	0.077%	0.78
ETFT8 $\epsilon\sqrt{2}$	0	0.00	0.00	279	0.080%	0.47

D^m	F^m		
$\Sigma\#e$	$\Sigma\#e$	$\overline{\Sigma\bar{e_r}}$	Σt
147938	1202	0.306%	1.47
674	2792	0.576%	1.58
685	1889	0.304%	0.64
685	1891	0.304%	1.28
674	2779	0.573%	0.61
144	267	0.072%	2.09
145	217	0.054%	0.89
145	217	0.054%	2.11
144	267	0.072%	0.83
0	35828	3.888%	3.31
0	2285	0.093%	2.60
0	128672	21.655%	1.89
0	146949	17.137%	1.19
0	412	0.010%	5.16
0	392	0.009%	3.86
0	392	0.009%	3.59
0	410	0.010%	2.20
0	23854	1.835%	4.39
0	415	0.023%	3.42
0	107543	13.522%	2.53
0	89708	8.963%	1.76
0	151	0.004%	6.63
0	89	0.002%	5.24
0	1611	0.164%	4.36
0	1404	0.129%	2.81
0	59747	4.562%	5.27
0	286	0.015%	4.77
0	169396	20.611%	2.64
0	152655	14.830%	1.98
0	165	0.005%	8.39
0	27	0.000%	6.81
0	1105	0.124%	5.25
0	1130	0.157%	3.58

and the GTFT extension of Lotufo's graph-searching method (Sec. 3.7).

For computing exact distances, ETFT4 $\epsilon\frac{1}{2}\sqrt{2}$ is the fastest of the considered methods. Although there are other, faster, exact DT methods, e.g., [82], the ETFT4 $\epsilon\frac{1}{2}\sqrt{2}$ can accommodate early distance-based termination and has a simple implementation (cf. Figure 3.10). For computing feature sets, all methods produce fairly accurate results ($< 0.1\%$ errors) for tolerances even up to $\sqrt{2}$. ϵVDT8 is the most accurate, while ETFT8 is the fastest. Finally, FMTFT8 is still useful, as it is the only considered method that can handle different speed functions.

Chapter 4

Computing Multiscale Curve and Surface Skeletons

As indicated in Section 2.6, the boundary-distance importance measure, introduced by Ogniewicz and Ilg [94], provides many desirable properties for the computation of 2D skeletons, such as robustness and connectedness of the skeletons, and it additionally provides a multiscale description of shapes. Its only shortcoming is that there is no immediate extension to 3D. In this chapter we propose the *collapse measure*, which generalizes the boundary-distance measure to 3D, inheriting its desirable properties.

4.1 Introduction

In this chapter we develop a skeletonization approach that computes multiscale curve and surface skeletons in a unified manner, by combining a global and monotonically-ascending importance measure with a simple thresholding strategy, controlled using a single user parameter. Our computed skeletons possess many of the desirable properties as discussed in Section 2.3. The monotonicity of the importance measure is guaranteed by combining the straightforward generalization of the boundary-distance measure with a new measure on the curve skeleton. The monotonicity ensures connectedness of the skeleton and intuitiveness of the pruning. The global nature of the measure provides robust skeletons, and by increasing the pruning parameter further, progressively simplified skeletons are obtained, providing a multiscale shape-descriptor. Another nice property is that the particular curve skeleton definition that we use facilitates shape segmentation, which is discussed in Chapter 6. We present a voxel-based implementation of our method. Because our measure is of a global nature, it is more costly to compute than purely local measures (see Chapter 2 for a comparison between local and global measures). We provide several optimizations to alleviate this. We demonstrate the algorithm on several real-world examples and suggest applications.

The structure of this chapter is as follows. In Section 4.2 we recall the 2D boundary-distance measure and present it as an advection process. In Section 4.3 we present our

collapse measure for continuous shapes, ignoring the non-generic cases. This continuous model motivates our voxel-based algorithm and is presented in Section 4.5.1. In Section 4.5.2 we discuss how our method handles the non-generic cases. Section 4.5.3 discusses how to handle shapes with tunnels, and Section 4.5.4 presents optimizations. Section 4.6 presents results of our algorithm. Section 4.7 discusses how our method fulfills the desirable skeleton properties, and compares it with existing methods. Section 4.8 concludes this chapter.

4.2 The 2D boundary-distance measure

Let us recall the boundary-distance measure, first introduced by Ogniewicz and Ilg [94], and also used by the AFMM (Chapter 3.3). For each skeleton point, which generically has two feature points, the measure is defined as the length of the shorter boundary-curve component delimited by these two feature points. Note that we assume that the shape has no holes. In case of a hole, there are two boundary components and the two feature points of skeleton-loop points lie on either component. A workaround for this has been presented [136], but we do not consider it here.

The boundary-distance measure is monotonic, where monotonicity means that the measure has only a single local maximum on the skeleton. This property can be seen to hold as follows. Feature vectors never intersect by definition. If they would intersect, the intersection point would be a skeleton point itself according to definition Eq. 2.1. Assuming a genus 0 shape, the feature points of a skeleton point p must always both be included in the connected boundary-component induced by the feature points of another skeleton point q. This inclusive relationship ensures the monotonicity of the boundary-distance measure ρ. We call the single local maximum the *root*, as the skeleton of a genus 0 shape is a tree. Interestingly, the root could be considered the 0D skeleton $\mathcal{S}^{2,0} \in \mathcal{S}^{2,1}$.

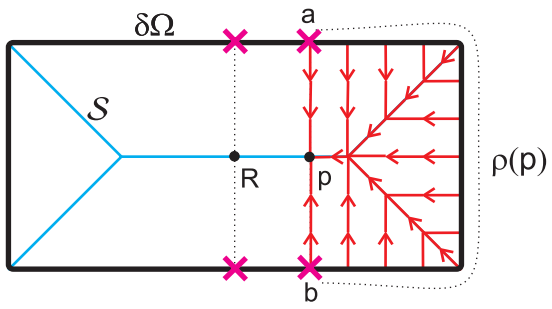


Figure 4.1: The boundary-distance measure seen as an advection process. A rectangle shape Ω, its skeleton $\mathcal{S}$, the root R, a skeleton point p, and p's feature points a, b. The collapse of p is formed by the origins of the trajectories (red) through p.

The boundary-distance measure can be seen in the light of an advection model, as follows. The skeleton of Ω can be defined as those points where the distance transform D is non-differentiable. Governed by the eikonal equation, the trajectory of each boundary

point follows the gradient field ∇D and ends at the skeleton $\mathcal{S}$, where ∇D is undefined. We can define a flow vector field $\mathbf{V}$ that extends ∇D to the skeleton, such that trajectories do not end at $\mathcal{S}$. This is illustrated in Figure 4.1. On non-skeleton points, $\mathbf{V}$ is equal to the distance field gradient $\mathbf{V} = \nabla D$. On $\mathcal{S}$, $\mathbf{V}$ advects boundary points toward a unique sink, the root R. The measure ρ for a point p can be defined as the number of particles that flow through p, which for the skeleton points is equivalent to the boundary-distance measure.

4.3 Extending the boundary-distance measure to 3D

We now present an extension of the boundary-distance measure to 3D. We present the model for continuous shapes first, and apply it to digital shapes in Section 4.5.

When extending the measure ρ to a 3D shape Ω with a 2D boundary $\partial\Omega$, we face the problem that there are still only two feature points per skeleton point, but two feature points do not split the boundary surface into two connected components. However, there is a straightforward generalization possible, as was also observed by Costa [28] and Prohaska and Hege [100]. We interpret the length of the smaller boundary-component in 2D as the shortest distance between the feature points along the boundary, which can be readily extended to the 3D shape as the shortest geodesic on the 2D boundary. Recall the definition of the feature transform $F : \Omega \to \mathcal{P}(\partial\Omega)$ (Eq. 3.1):

$$F(p \in \Omega) = \left\{ q \in \partial\Omega \;\middle|\; \|p - q\| = D(p) \right\}, \tag{4.1}$$

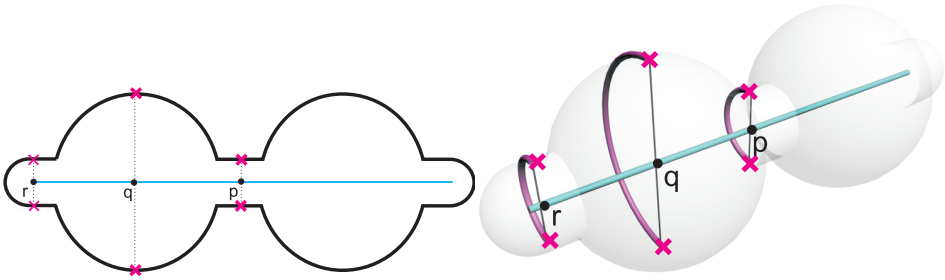
where $\|\cdot\|$ denotes Euclidean distance and D is the Euclidean distance-to-boundary function. We assume for now exactly two feature points for each point p. Let $\rho_{\mathcal{S}} : \mathcal{S} \to \mathbb{R}_+$ denote the importance measure which assigns to each surface-skeleton point p the shortest-geodesic length between the two feature points $a, b \in F(p)$:

$$\rho_{\mathcal{S}}(p \in \mathcal{S}) = \text{length}(\gamma(a, b)), \tag{4.2}$$

where $\gamma(a, b)$ is the shortest geodesic on $\partial\Omega$ between a and b, and length$(\cdot)$ denotes curve length. For some skeleton points the shortest geodesic between a and b will not be unique, in which case we assume γ takes one of them. This does not matter for the definition of $\rho_{\mathcal{S}}$ as these geodesics are necessarily of equal minimum length.

Intuitively, it can be seen that $\rho_{\mathcal{S}}$ is small on the skeleton rims, and $\rho_{\mathcal{S}}$ becomes larger when further away from the rim. A surface-skeleton point which has both its feature points on small-scale features of the boundary will receive a small importance, as these feature points cannot be far apart on the boundary. Otherwise, the shape feature would not be considered small. For skeleton points which have their feature points far apart geodesically, the shortest geodesic connecting them must pass multiple small-scale features or a single large-scale feature, so that they can be considered more important. From another point of view, $\rho_{\mathcal{S}}$ quantifies how different the two feature points a, b are, which relaxes the requirement that $a \neq b$ in the skeleton definition of Eq. 2.1 by putting a threshold on $\rho_{\mathcal{S}}$.

Note that the importance is measured in terms of curve length along the boundary both for 2D and 3D shapes. There is obviously also a difference: in 2D the feature points of a



(a) 2D dumbbell shape. $\rho(p) > \rho(q) > \rho(r)$. (b) 3D dumbbell shape $\rho_{\mathcal{S}}(p) = \rho_{\mathcal{S}}(r) < \rho_{\mathcal{S}}(q)$.

Figure 4.2: Extending the boundary-distance measure.

skeleton point p divide the boundary into two connected components. The importance of a point p is expressed as the size of the smaller component, and the unit of the measure has the same dimension as the boundary. This distance happens to coincide with the shortest path along the boundary between the two feature points. In 3D however, we take the length of the shortest geodesic and the measure is of a different dimensionality than the boundary itself.

Due to this dissimilarity, the problem of the proposed measure $\rho_{\mathcal{S}}$ is that it is not monotonic. Again, by monotonicity we mean that the skeleton has only a single local maximum on $\mathcal{S}$, called the root. Consider Figure 4.2(a), showing a dumbbell shape, comprised of two discs and a connection between them. In the 2D case, the boundary-distance measure clearly yields a monotonic ρ such that $\rho(r) < \rho(q) < \rho(p)$. Imagine the 3D shape obtained by revolving the 2D shape around its horizontal axis. By construction, the surface skeleton is a 1D structure, so that it could also be considered the curve skeleton of the shape. Due to the rotational symmetry, each (curve and surface) skeleton point has a contiguous ring of feature points. This is not a stable situation: slightly stretching the shape along an axis other than the horizontal one, reduces the number of feature points to two diametrically opposed feature points along the non-stretched axis, which is a stable situation. Let us assume such an infinitesimal stretching. Then it can be seen that the monotonicity of $\rho_{\mathcal{S}}$ is violated, because we have that $\rho_{\mathcal{S}}(p) = \rho_{\mathcal{S}}(r) < \rho_{\mathcal{S}}(q)$ in our 3D shape (Figure 4.2(b)). Indeed, this extension of the boundary-distance measure cannot distinguish between point p and r, as the boundary has locally similar configurations. In 2D, ρ can be considered a global measure, as it considers all of the boundary-curve points, but in 3D $\rho_{\mathcal{S}}$ can be considered a quasi-global approach: it considers a large part of, but not the whole boundary surface.

In order to obtain a truly global and monotonic measure, we need to define a different measure. The key idea is to split the boundary into two components, as we did with the 2D boundary-distance measure, and next define the measure as a surface area, rather than a curve length. In our (slightly stretched) 3D dumbbell, due to symmetry considerations, the two feature points of a skeleton point p admit two shortest geodesics (necessarily of equal length), together forming a ring around the object. Let Γ be the set of shortest

geodesics associated with a point p:

$$\Gamma(p) = \bigcup_{a,b \in F(p)} \gamma(a,b) \; . \tag{4.3}$$

A simple closed curve Γ is also called a *Jordan curve*. The *Jordan curve theorem* dictates that a Jordan curve splits the boundary into two connected components (if the shape has no tunnels). We take the area of the smaller boundary component as our new measure $\rho_{\mathcal{C}}$. Let $C(p)$ denote the set of connected components in $\partial\Omega$ induced by the Jordan curve Γ of p, i.e., the *component set* of p. Curve-skeleton points generate at least two boundary components. Our new measure $\rho_{\mathcal{C}}$ is defined as:

$$\rho_{\mathcal{C}}(p \in \mathcal{C}) = \text{area}(\partial\Omega) - \max_{c \in C} \text{area}(c) \; . \tag{4.4}$$

We take the inverse of the largest component area in order to handle non-generic cases in which $|C| \geq 3$, as we see in Section 4.5.2. Now, it can be seen that this new measure has the desired monotonicity for the three points on the dumbbell shape: $\rho_{\mathcal{C}}(r) < \rho_{\mathcal{C}}(q) < \rho_{\mathcal{C}}(p)$. We later discuss the monotonicity of $\rho_{\mathcal{C}}$ for all shapes.

Curve skeleton definition We constructed our 3D dumbbell shape in such a manner that the curve skeleton coincides with the surface skeleton and its location is known due to symmetry considerations. We did not yet define the curve skeleton, we only defined a measure on it. Our new measure can only be defined if the two feature points of the curve skeleton admit two shortest geodesics. Therefore, we now define the curve skeleton $\mathcal{C}$ as those points that admit a Jordan curve:

$$p \in \mathcal{C} \Leftrightarrow |\Gamma| \geq 2 \; . \tag{4.5}$$

Notice the connection with the definition of the Blum skeleton (Eq. 2.1). The curve skeleton is defined here in terms of feature points that are connected with each other using two shortest geodesics: the curved-surface equivalent of straight lines. Points on the Blum skeleton are connected to the boundary by straight lines of minimum length.

It is not obvious that Eq. 4.5 yields a connected curve skeleton for all possible shapes. However, it seems reasonable to assume, as follows. Curves evolve continuously over the shape surface when subjected to a continuous deformation. The surface skeleton also deforms continuously under the same deformation. From these two assumptions, we conjecture that a curve on $\mathcal{S}$ on which each point admits two shortest geodesics between its feature points, evolves continuously over $\mathcal{S}$. We argue that because our dumbbell shape with connected curve skeleton is homotopic to any other genus 0 shape, the curve skeleton of any genus 0 shape is connected.

Dey and Sun [36] independently presented a very similar curve skeleton definition, namely as the singularities of $\rho_{\mathcal{S}}$, which they call the *medial geodesic function*. They conjecture that the singularities correspond to those points that admit two shortest geodesics. They prove that the singularities of the function consist of curves and/or isolated points and enforce connectedness of the extracted curve skeleton by applying an erosion operation to the surface skeleton in order of the medial geodesic function. However, we have

observed in practice that our computed curve skeletons are connected without any enforced connectivity step (see the results in Section 4.6). Other differences between both methods are discussed in Section 4.7.2.

In order to prove that $\rho_{\mathcal{C}}$ is monotonic for all shapes one has to prove that the component sets C have an inclusive relation, that is, any two components $c \in C(p), d \in C(q)$ for two points $p, q \in \mathcal{C}$ cannot partly overlap:

$$c \cap d \in \{\emptyset, c, d\} \; . \tag{4.6}$$

In other words, the Jordan curves associated with the curve-skeleton points never intersect (although they can partly coincide). This proof can be roughly sketched as follows. Take two curve-skeleton points $p, q \in \mathcal{C}$. Take that subset $\mathcal{C}_{pq} \subset \mathcal{C}$ that is between p and q. Because the skeleton is homotopy equivalent to the boundary [74], the skeleton can be continuously deformed into the boundary. A simple curve that lies completely inside one skeleton sheet can be continuously deformed into two simple curves on the boundary, one on each side of the shape. The feature points $F(\mathcal{C}_{pq})$ thus generate two curves l, m on $\partial\Omega$. Each shortest geodesic $\gamma \in \Gamma(\mathcal{C}_{pq})$ has one endpoint on l and the other endpoint on m. Such a geodesic γ does not intersect l or m in any other point as it would no longer be the shortest geodesic. Now, if a shortest geodesic $\gamma_p \in \Gamma(p)$ intersects $\gamma_q \in \Gamma(q)$, it needs to have (a multiple of) two intersection points. Although such an intersection might be possible, we have never observed this in our example shapes. In any case, non-intersecting Jordan curves are still guaranteed in this situation by taking the same shortest path between the intersection points for both Jordan curves $\Gamma(p)$ and $\Gamma(q)$. A more thorough proof of the inclusiveness property of components would also need to consider parts of the curve skeleton that do not lie within one sheet.

Collapse measure We now combine our new global measure $\rho_{\mathcal{C}}$ with the quasi-global measure $\rho_{\mathcal{S}}$ to come to a monotonic measure for the whole skeleton. For points on the curve skeleton we use the curve-skeleton measure $\rho_{\mathcal{C}}$, whereas we use the measure $\rho_{\mathcal{S}}$ for other surface-skeleton points $\mathcal{S} \setminus \mathcal{C}$. We obtain our new global importance measure $\rho : \mathcal{S} \rightarrow \mathbb{R}_+$ for 3D shapes, called the *collapse measure*:

$$\rho(p \in \Omega) = \begin{cases} \rho_{\mathcal{C}}(p) & \text{if } p \in \mathcal{C} \\ \rho_{\mathcal{S}}(p) & \text{if } p \in \Omega \setminus \mathcal{C} \end{cases} \tag{4.7}$$

Formally, the measures $\rho_{\mathcal{S}}$ and $\rho_{\mathcal{C}}$ cannot be combined because they are of a different dimensionality. The first expresses curve length, whereas the second expresses an area. By combining them, we essentially interpret $\rho_{\mathcal{S}}$ as an area, obtained by letting the shortest geodesics have a 1-unit width. Indeed, in our voxel-based setting, presented in Section 4.5, this is completely natural as curves are computed as voxel chains, and thus have a 1-voxel width.

To prove that ρ is monotonic we have to prove that for each point p there exists a path to the root R on $\mathcal{S}$ such that ρ is monotonically ascending along the path. We showed that this is true for the curve-skeleton points (Eq. 4.6). Remains to show this is true for the other surface-skeleton points. For each surface-skeleton point p, there should be a path toward a curve-skeleton point q along which $\rho_{\mathcal{S}}$ is ascending. Suppose that no such

path exist. This means that a point p trying to reach q is stuck in a local maximum of $\rho_{\mathcal{S}}$. But all local maxima are curve-skeleton points [36], as all singularities have to admit two shortest geodesics. Finally, when a path reaches the surface skeleton and continues along the curve skeleton, it should hold at the point of transition $q \in \mathcal{C}$ that $\rho_{\mathcal{C}}(q) \geq \rho_{\mathcal{S}}(q)$. This is obvious when we think of the surface area $\rho_{\mathcal{C}}$ for a point q as the summation of the Jordan curves of the smaller part of the curve skeleton delimited by q, which necessarily includes the curve $\Gamma(q)$ that generates the area.

Simplified skeleton After we have computed ρ for all points in the object Ω, we obtain a *simplified skeleton* $\mathcal{S}_\tau$ by simply applying the thresholding strategy. To uniformly handle objects of different sizes, we normalize ρ by dividing it by the total object surface area:

$$\mathcal{S}_\tau(\Omega) = \left\{ p \in \Omega \;\middle|\; \frac{\rho(p)}{\text{area}(\partial\Omega)} > \tau \right\}, \tag{4.8}$$

where τ is the desired simplification level. Note that we do not need to perform any post-processing step as our simplified skeletons are connected by default, as the collapse measure ρ is monotonic. If only the simplified curve skeleton is desired, denoted $\mathcal{C}_\tau$, one applies the same thresholding strategy to $\rho_{\mathcal{C}}$ on $\mathcal{C}$. Likewise, only the simplified surface skeleton can be computed by applying the thresholding strategy to $\rho_{\mathcal{S}}$. Unlike $\mathcal{C}_\tau$, these simplified surface skeletons, in which the curve skeleton does not play a role, are not guaranteed to be connected, especially for higher values of τ.

As we already mentioned, ρ is essentially of a higher dimensionality on the curve skeleton $\mathcal{C}$ than on the remainder of the surface skeleton $\mathcal{S} \setminus \mathcal{C}$. Whereas $\rho_{\mathcal{C}}$ on $\mathcal{C}$ (1D) denotes a collapsed area (2D), $\rho_{\mathcal{S}}$ on $\mathcal{S}$ (2D) denotes a curve length (1D). This means that, when we increase τ, the surface skeleton typically disappears completely even before the curve skeleton starts to get simplified. This is desirable in applications where the curve skeleton is considered more important. However, in other applications, this behavior may not be desired, as we show in Section 4.7. For such applications, we can "equalize" ρ by reducing the dimensionality of $\rho_{\mathcal{C}}$ from an area to a length by taking its square root. We denote the *equalized* skeleton by $\mathcal{S}'_\tau$. In contrast to $\mathcal{S}_\tau$, the equalized skeleton gets simplified uniformly both in its curve and in its surface components when τ is increased. Note that this is just one of the possible ways to equalize ρ. Other options leading to other applications are open to further study.

4.4 The collapse measure as an advection model

As explained in Section 4.2, the 2D boundary-distance measure can be seen to result from an advection process. A vector field can be defined inside the object in such a way that the measure is equivalent to the advected mass through the point. It would be nice if our collapse measure ρ can also be seen to result from some advection process. In fact, we conjectured this in earlier work [115]. The idea was that the points flowing through a surface-skeleton point p are precisely the points coming from its associated shortest geodesic. The feature points of p flow through p directly by means of the distance-field gradient ∇D, the other points on the geodesic are feature points of their respective

skeleton points and are advected through p by defining the flow field $\mathbf{V}$ on $\mathcal{S}$ appropriately. On the curve skeleton, $\mathbf{V}$ is simply defined as tangent to it toward the global root, which can be considered the 0D skeleton $\mathcal{S}^{3,0}$. We can see the definition of $\rho_{\mathcal{C}}$ as the result of an advection process, as $\mathbf{V}$ can be unambiguously defined on the curve skeleton, namely in direction of the root.

To make such a definition of $\mathbf{V}$ on $\mathcal{S}$ possible, we should have the following. If we take the shortest geodesic γ associated with a curve-skeleton point p and take a point a on γ, where a is the feature point of $q \in \mathcal{S}$, it should hold that the involute b of a (the other feature point of q) also lies on γ. Otherwise, $\mathbf{V}$ cannot be defined on $\mathbf{V}$: a should be advected through p but b should not. Although this is true for some simple objects, like a box, we have found that the conjecture does not hold for more complex objects. We conclude that, although the measure $\rho_{\mathcal{C}}$ can be seen to result from an advection model, the combined measure ρ cannot.

4.5 Algorithm

This section presents a voxel-based algorithm which implements the method described in Section 4.3. A strong point of our algorithm is that it works on the object and boundary voxels only and not on any derived structures. In particular, there is no need to compute $\mathcal{S}$ first: all object voxels are processed independently. This also makes a parallel implementation straightforward.

Section 4.5.1 discusses the implementation into more detail. Section 4.5.2 shows how the algorithm deals with the non-generic cases, something we have ignored thus far. Section 4.5.3 shows how we handle shapes with tunnels. Finally, in Section 4.5.4 we present some techniques that are not essential for the skeleton computation, but are used to speed-up the computation.

4.5.1 Algorithm details

Our algorithm works on a binary voxel shape Ω as described in Section 2.1. Let Ω be the set of interior voxels, and let $\partial\Omega$ be the set of boundary voxels. A boundary voxel-graph is defined in which the boundary voxels are nodes and two nodes are connected by an edge when their voxels are 26-adjacent. The pseudocode of the algorithm is shown in Figure 4.3, and the algorithm is illustrated in Figure 4.5. It consists of four stages: computing the (extended) feature transform, the shortest geodesics, the collapse measure, and finally the simplified skeleton. We discuss the stages in order.

First, we have to compute the feature transform F for the voxel shape. For this we use the 3D extension of Danielsson's algorithm [31], which is a raster-scanning approach. Although the method is not exact, this is not a problem, as the detection of the curve skeleton does not rely on the exact position of feature points. There are two problems with simply using the feature transform F thus obtained. The first problem is that, while surface skeleton points always have at least two feature points in $\mathbb{R}^3$, this is not necessarily so in discrete ($\mathbb{Z}^3$) space. In a box of even height for example, no voxels on the center surface-skeleton sheet have two feature voxels (Figure 4.4(a)). Second, even if for

1: compute feature transform F on Ω
2: **for** each object voxel $p \in \Omega$ **do**
3: $\quad \overline{F} \leftarrow \bigcup_{x,y,z \in \{0,1\}} F(p_x + x, p_y + y, p_z + z)$
4: $\quad \Gamma \leftarrow \bigcup_{a \neq b \in \overline{F}}$ shortestpath(a, b)
5: $\quad$ **if** Γ contains a Jordan curve **then** {curve skeleton voxel}
6: $\quad\quad C \leftarrow$ {connected components in $\partial\Omega \setminus \Gamma$}
7: $\quad\quad \rho(p) \leftarrow$ area$(\partial\Omega) - \max_{c \in C}$ area(c)
8: $\quad\quad \mathcal{C} \leftarrow \mathcal{C} \cup \{p\}$
9: $\quad$ **else** {surface skeleton or non-skeleton voxel}
10: $\quad\quad \rho(p) \leftarrow \max_{\gamma \in \Gamma}$ length(γ)
11: $\quad$ **end if**
12: $\quad \rho(p) \leftarrow \frac{\rho(p)}{\text{area}(\partial\Omega)}$
13: **end for**
14: $\mathcal{S}_\tau = \{p \in \Omega \mid \rho(p) > \tau\}$

Figure 4.3: Pseudocode of the complete skeletonization algorithm.

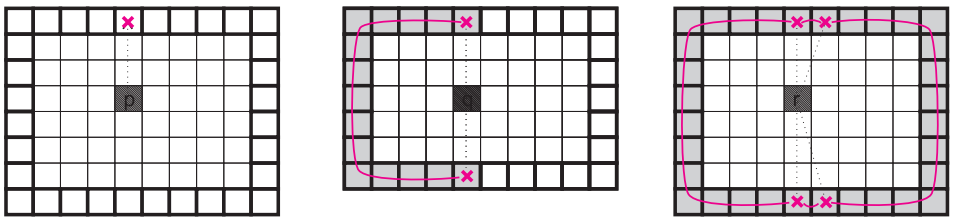


Figure 4.4: Problems of the discrete feature transform for skeletonization. The cross-section of a 3D box is shown. Feature points shown as crosses, shortest paths as magenta curves. The surface skeleton cannot be detected in p: p has only one feature point (a). The curve skeleton cannot be detected in q: q has only one shortest path (b). Both skeletons can be detected in r when using the extended feature transform (c).

all surface-skeleton points two feature points are obtained, we might find only one shortest path for intended curve-skeleton voxels because of the discretization (Figure 4.4(b)). Although we could use a tolerance-based feature transform (Chapter 3) to solve these discretization issues, we prefer the following solution as it yields less feature points. We introduce the *extended feature transform* $\overline{F}$. The extended feature set $\overline{F}(p)$ is obtained by merging the feature set of p with the feature sets in a $2 \times 2 \times 2$ neighborhood that includes p (Figure 4.3, line 3):

$$\overline{F}(p) = \bigcup_{x,y,z \in \{0,1\}} F(p_x + x, p_y + y, p_z + z) \tag{4.9}$$

Figure 4.4(c) shows that using $\overline{F}$ we obtain (at least) two shortest paths for curve-skeleton voxels that form a loop.

In the second stage of our algorithm, we have to compute the shortest paths between feature points (Figure 4.3, line 4). The shortest path is used as the discrete-space approximation to the shortest geodesics and is computed as a 26-connected path in the boundary graph, using Dijkstra's algorithm [38]. One can estimate the length of the original shortest geodesics, which the shortest path approximates, by summing the Euclidean distances between adjacent voxels in the path, yielding the weights 1, $\sqrt{2}$, and $\sqrt{3}$ for the three adjacency relations. We use the length estimator of Kiryati and Székely [65] instead, who suggest using the experimentally obtained weights of 0.9016, 1.289, and 1.615, as they generally better approximate curve lengths on digitized surfaces. We compute Γ as the set of all the shortest paths between all feature points in $\overline{F}(p)$ (Eq. 4.3). Although it seems computationally expensive to compute all shortest paths between all feature points, $\overline{F}$ is typically small (≤ 8), and most paths are between neighboring voxels so that Dijkstra's algorithm terminates quickly. Typically, two shortest paths of considerable size need to be computed for surface-skeleton voxels, and four for curve-skeleton voxels. Furthermore, we cache shortest paths to prevent computing the same shortest path twice. We detail on the caching scheme in Section 4.5.4.

In the third stage, we have to classify the object point p as being a curve-skeleton point or not. In our continuous model, a point p is on the curve skeleton $\mathcal{C}$ when it admits two shortest geodesics (Eq. 4.5). However, because of the extended feature transform, the shortest-path set Γ contains numerous shortest paths even for non-curve-skeleton points, thus we have to use a different detector. Because the voxels in the shortest-path set Γ form the discrete equivalent of a continuous Jordan curve, we could use the Jordan curve theorem to detect curve-skeleton points as points whose Jordan curve divides the boundary graph into at least two connected components. We took this approach in earlier work [115]. However, this has as disadvantage that it does not work for shapes with tunnels, and that computing connected components is a relatively slow process that should be avoided if possible. Therefore, we opt for a different approach. The voxels occupied by the discrete Jordan curve, consisting of several shortest paths, do usually not form a nice one-voxel thick ring. Instead, the ring might be thick at some places and contain small holes. Therefore, we slightly dilate the voxels in Γ on $\partial\Omega$ so that we obtain a thick surface-band Γ' centered at the noisy ring. The dilation is performed by adding points to Γ' that are within a short distance of Γ in the boundary graph (again using Dijkstra's algorithm and the length estimator). Next, we determine the number of connected border segments that Γ' has. If two or more segments are found, p is concluded to be a curve-skeleton point. Empirical studies on an extensive family of real-world 3D shapes show that a fixed (geodesic) dilation distance of 5 voxel lengths is enough to obtain two connected boundaries if Γ is the discrete representation of a Jordan curve. When three or more boundaries for Γ' are found, p can be considered a curve-skeleton junction (we use this in our part-type segmentation method that we present in Chapter 6). If we detect the point as a curve-skeleton point, we add it to the set of curve-skeleton points $\mathcal{C}$ (Figure 4.3, line 8). Even though we cannot compute the collapse measure $\rho_{\mathcal{C}}$ on the $\mathcal{C}$-loops for shapes with tunnels, as C has only 1 component on these $\mathcal{C}$-loops, it is possible to compute the curve skeleton $\mathcal{C}$. The Jordan curve detection by counting the border segments of the dilation Γ' assumes that the dilation does not change the topology of the ring. Therefore, our method does not support shapes with small tunnels, as a small tunnel

may provide a way to connect the two boundaries of the dilated path-set. Such tunnels could be removed by standard morphological operations for example.

After classification of p, we assign as importance to the curve-skeleton point the area of the smaller connected components in the boundary graph induced by $\Gamma(p)$. Although regular curve-skeleton points admit exactly two components, $|C| = 2$, for curve-skeleton junctions holds $|C| \geq 3$. Another reason why C might have more than two components is that a discrete Jordan curve consists of at least four shortest paths which might have (small) holes. Although such a shortest-path set does divide the surface into two components like the Jordan curve does, it may have small openings between the shortest paths themselves. Because of these two cases, we combine the areas of all the components except the largest (Figure 4.3, line 12). Although we could use a more accurate and sophisticated area estimator to measure the component's size, we simply take the number of voxels in the component as the area, as we found this yields good results in practice. If the object point is not on the curve skeleton, we assign the maximum shortest-path length in Γ (Eq. 4.7, and Figure 4.3, line 10). Note that we assign this length also to non-surface-skeleton points. In this manner, we treat all object points equally. The non-surface-skeleton points will be simplified before the surface-skeleton points for small values of τ. Experimental evidence suggests that using a τ of 5 (voxel lengths) discards all object points and delivers the full (non-simplified) surface skeleton.

Finally, after computing the importance ρ for each object-point p, we produce a simplified skeleton by keeping only points for which $\rho(p) > \tau$. In order to compute the equalized skeleton $\mathcal{S}'_\tau$, the algorithm is slightly modified by taking the square root of ρ on $\mathcal{C}$ (Figure 4.3, line 7), and normalizing by $\sqrt{\text{area}(\partial\Omega)}$ (Figure 4.3, line 12).

4.5.2 Non-generic cases

Our model from Section 4.3 assumed the generic case of a skeleton point having exactly two feature points and admitting one or two shortest geodesics between them. An example of such a generic curve-skeleton point is shown in Figure 4.6(a). However, our algorithm from the previous section can also deal with the non-generic cases, as explained in this section. We can distinguish two cases:

- Skeleton points having three or more feature points.
- Skeleton points having three or more shortest geodesics between their two feature points.

We begin explaining the first case. Recall that surface skeletons consist of manifolds with boundaries, called *sheets* and that sheets intersect in curves (Sec. 2.2). Points on these curves have more than two feature points. Figure 4.6(b) shows such a configuration in a box with a vertical ridge. The selected curve-skeleton point lies on the intersection curve of three sheets and thus has three feature voxels. This is a limit case. A point lying on one sheet has one pair of feature points, but a point lying on the intersection of three sheets has three pairs of feature points. Each feature pair shares each of its two feature points with an other pair, yielding three distinct feature points. No two feature points among these three admit two shortest geodesics between them, so that the curve skeleton

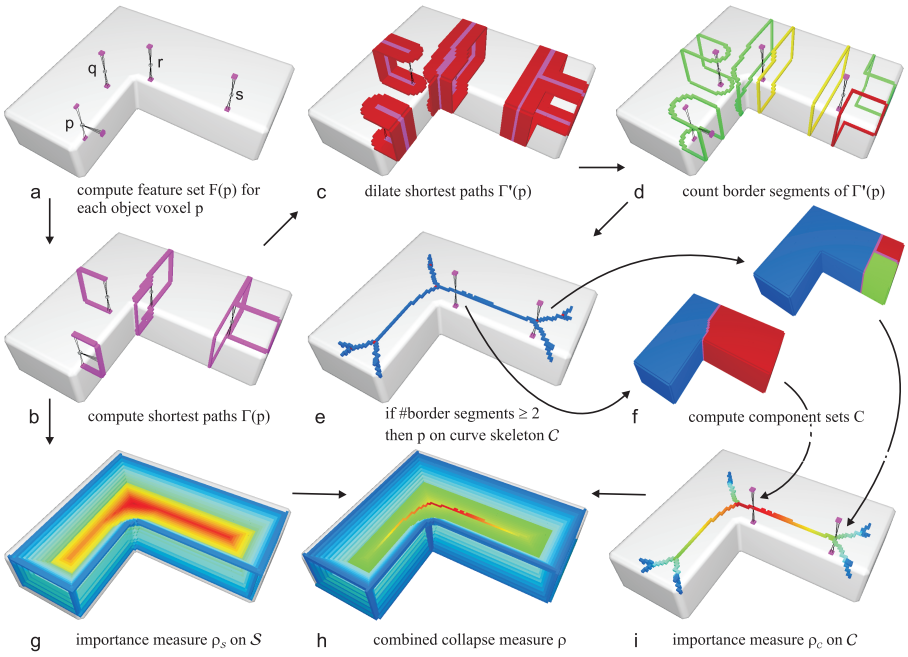


Figure 4.5: Illustration of our skeletonization algorithm. Four points p, q, r, s are selected. (a) The extended feature sets $\overline{F}$. (b) The shortest-path sets Γ, resulting in $\rho_{\mathcal{S}}$ shown in (g). (c) The dilated path sets Γ'. (d) The connected border segments of the dilated path-sets. (e) Point r and s are marked as curve-skeleton voxels, as we count at least two border segments for them. (f) We compute the component set C for each curve-skeleton voxel, resulting in measure $\rho_{\mathcal{C}}$ shown in (i). (h) The combined measure ρ.

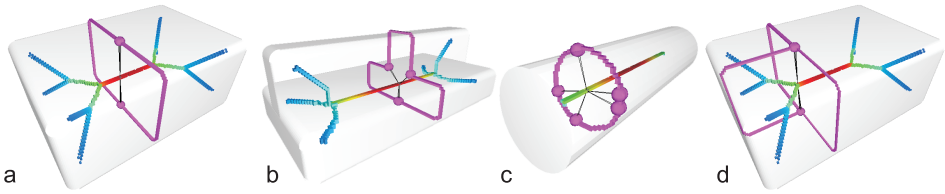


Figure 4.6: (a) Generic case. (b-d) Non-generic cases. The curve skeletons are shown with a rainbow color map encoding the collapse measure ρ. In each image, the shortest path set for a selected curve skeleton voxel is shown in magenta. Feature voxels are shown as spheres and are connected to the selected voxel using line segments.

cannot be detected using this criterion. However, our algorithm combines all shortest geodesics, so that we now obtain a (discrete) Jordan curve that splits the object surface into two components, correctly detecting the curve-skeleton point. Another example of a point having three or more feature points is that of a curve-skeleton point whose inscribed ball has a finite contact with the boundary. This yields a continuum of feature points, which in voxel space results in a limited number of feature voxels. Combining the shortest paths between all feature voxels again resolves the issue as it creates a Jordan curve that splits the object surface into two components. An example of this situation in a cylinder shape is shown Figure 4.6(c).

Combining all shortest geodesics might result in slightly wrong component sets in the uncommon occasion that a curve-skeleton point lies on the intersection curve of more than three sheets. In this case, not every two feature points and the associated shortest geodesic between them correspond to a sheet. For example, in the case of an extruded cross, one has four intersecting sheets and the intersection line coincides with the curve skeleton. There are 4 feature points, resulting in six shortest geodesics, where there should be 4. The resulting shortest-path set Γ contains some shortest paths that do not lie on the Jordan curve in the continuous case. The collapse measure might differ slightly from what it should be, violating monotonicity. However, we did not find this to be a detectable issue in any real-world example (see Sec. 4.6).

The second non-generic case is that of a point $p \in \mathcal{C}$ having three or more shortest geodesics between its two feature points, which happens at junction points of the curve skeleton (Figure 4.6(d)). This is a limit case, since each curve-skeleton point next to the junction does admit two shortest geodesics. The difference with the generic case is that we obtain more than two connected components in case of a junction point. The algorithm deals with this by always taking the area of $\partial\Omega \setminus c$ as the value of $\rho(p)$ (Figure 4.3, line 7), where c is the largest component in C.

4.5.3 Handling shapes with tunnels

So far, we have assumed that Ω has no tunnels, so that the curve skeleton $\mathcal{C}$ is a tree. The reason for this is that Jordan curve theorem does not hold everywhere for shapes with tunnels: it only holds for curve-skeleton points that are not on a curve-skeleton loop. These loop points have shortest geodesics that go from one side of the shape to the other side through at least one tunnel. Fortunately, we can detect the problematic $\mathcal{C}$-loop points: these are points in $\mathcal{C}$ whose component set C has cardinality 1. We would like to assign an importance to these loop points so that we can also create simplified skeletons for shapes with tunnels. In assigning an importance the most important constraint is that $\rho_{\mathcal{C}}$ should be monotonic. Imagine a curve skeleton with a single loop with multiple $\mathcal{C}$-branches connected to it (Figure 4.7). Each branch endpoint that lies on the loop has a valid $\rho_{\mathcal{C}}$ assigned to it. The easiest way to create a monotonic $\rho_{\mathcal{C}}$ is to assign to all loop points the same value, namely the maximum $\rho_{\mathcal{C}}$ value among the branch endpoints. It makes sense to assign the same value to all loop points: why should one loop point be preferred over the others? When simplifying the skeleton by increasing τ, all $\mathcal{C}$-branches but the most important one are removed first, after which the whole loop is completely removed.

When we see $\rho_{\mathcal{C}}$ as the result of an advection process, the loops also present a problem.

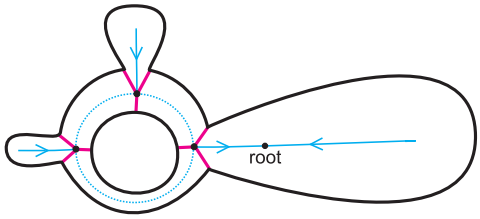


Figure 4.7: A curve-skeleton loop with three branches. Shortest-path sets are shown for the branch endpoints. The flow of mass toward the root is indicated by arrows.

From each curve-skeleton point there should be a path that ends in a unique sink (the root). When the advected particle encounters a loop, it is not clear which direction should be taken. Our solution means that the particle makes a full tour on the loop before it leaves to the branch containing the root. Another way to look at it is that the particle takes both routes. Choosing a particular direction over the other one leads to different solutions, resulting in non-uniform values of $\rho_{\mathcal{C}}$ on the loop. For example, in Figure 4.7 one might want to guide the mass through the thick part of the loop, as it could be considered more important than the thin part. The result would be that when increasing τ, the loop is first opened (somewhere) on the thin part. The choice of the opening is rather arbitrary and it can be hard to generalize this idea to multiple loops. Although these alternatives are interesting, we have not pursued this direction and leave it as future work.

4.5.4 Optimizations

Our algorithm considers all object voxels and computes multiple shortest paths for each of them. The global nature of the importance measure makes it slow to compute when compared to local approaches (e.g., the feature-angle measure). To improve the execution time, we have implemented several optimizations.

First, instead of using the regular Dijkstra's algorithm, we use A* graph-search algorithm [48]. When using Dijkstra's algorithm to compute a shortest path between two nodes (voxels) a and b in the boundary voxel-graph, the shortest paths are found between a and every intermediate node c, in the order of path length from a to c. The algorithm stops when b is reached. The A* algorithm modifies Dijkstra's algorithm by improving the order in which nodes are visited based on a heuristic function h, in such a way that point b is likely to be found earlier. Instead of using just the length of the path ac as the cost function for determining the order of points, A* takes as cost function the path length from a to c plus the heuristic function h, estimating the distance from c to b. In our case, in which the graph represents the boundary of a geometrical object, we use the Euclidean distance between c and b as h. The Euclidean distance is a lower bound for the shortest-path length, and thus makes for a suitable heuristic.

Second, we cache all shortest paths computed to prevent computing the same shortest path twice. The main reason for caching is that we compute the shortest paths between all feature points in the extended feature set. By construction, the feature sets of neighboring points have some feature points in common. We store for each pair of feature points the

length of the shortest path and the voxels it occupies. The size of the cache is a user parameter and presents a trade-off between speed and memory usage. In our experiments a cache size of 50 MB was used, storing a maximum of 13 million voxels, enough to store the paths for the shapes shown in this chapter, which presents a speed-up factor of 3 to 4 approximately. For the resolutions we used (see Section 4.6), larger cache sizes do not improve speed significantly. We have considered using the cached shortest paths to speed up Dijkstra's algorithm itself. If in Dijkstra's algorithm we encounter a point c for which the shortest path to the target point b is already known, we are done. However, it is rare that the shortest path has already been computed to the target feature point, as feature points are seldom shared between skeleton points, besides those due to the extended feature transform.

Third, we create a spatial subdivision scheme on the boundary graph to speed-up the connected component computations necessary for computing $\rho_{\mathcal{C}}$. Computing the sizes of the connected components C induced by the shortest-path set Γ for a point p in a naive manner requires us to visit each boundary point. Doing this for each curve-skeleton point implies a heavy computational burden. To alleviate this, we create, after having constructed the boundary graph, a (simple) spatial subdivision on the boundary graph in the form of a tree-data structure. Each node in the tree represents and stores a compact area (set of voxels) on $\partial\Omega$, that is completely included in its parent node's area. The top node in the tree covers the whole boundary. The 8 child nodes of a parent node are constructed by dividing 8 seed nodes evenly over the node, which are then grown so that they cover the whole of the parent area. Our experiments indicated that the placement does not need to be optimal to obtain a good speed-up. The subdivision process is done recursively until there is a leaf node for each voxel. After the subdivision, for each node the neighboring nodes at the same level are determined. Querying the data structure with a shortest-path set Γ as input and the induced components sets as output works as follows. First, all nodes in the tree that cover Γ are marked. For this purpose, both Γ and the node area are represented as sorted lists so that intersections can be computed efficiently. The idea is that the tree prevents us from visiting all boundary voxels as we can consider large areas of the boundary at once, by means of their representative nodes. Only for nodes that intersect Γ we need to traverse down the tree. After all nodes occupied by Γ are marked, the connected component labeling begins. We label nodes breadth-first. The idea is that nodes marked as covering Γ are never labeled. When a marked node is tried to be labeled, the algorithm traverses down the tree and recursively labels the children instead. When possible, that is, when the parent node is not marked, the algorithm traverses up the tree as far as possible and continues labeling. Finally, the tree is traversed from the top and the connected components are output in the form of sets of labeled nodes. The area (number of voxels) of an output component is computed by summing the voxel set sizes of the nodes.

Due to the spatial subdivision scheme, our algorithm gains in speed because not every voxel needs to be visited when finding connected components. Only in the vicinity of Γ is the tree traversed down to the lowest level. The complexity is improved from $O(b)$ to $O(\log b)$, where b is the number of boundary voxels. For the complexity of the complete algorithm, see Section 4.7.1.

The speed-ups obtained using these techniques are indicated for two shapes in Ta-

Table 4.1: Table with optimization measurements. Timings are wall-clock times, in seconds.

shape	without any	with caching	with A*	with subdivision	with all
Hand256	773	545	510	96	60
Rockerarm256	1877	1161	1148	1336	170

Table 4.2: Table with measurements. See the text for details.

shape	dim	$\|\Omega\|$	$\|\partial\Omega\|$	#paths	#comp.	paths t	comp. t	mem	total t
Bird512	262x132x488	717k	95k	750k	73k	126s	59s	303M	196s
Cow256	78x146x244	549k	44k	357k	47k	9s	14s	125M	28s
Cow384	115x217x366	1,904k	102k	811k	151k	32s	102s	319M	148s
Cow512	154x289x488	4,575k	183k	1,441k	345k	75s	302s	592M	411s
Dino512	159x453x487	3,218k	177k	1,413k	231k	61s	156s	590M	246s
Dragon512	214x336x488	5,656k	320k	2,664k	491k	152s	254s	747M	475s
Hand512	488x205x342	3,137k	167k	1,348k	211k	132s	171s	570M	332s
Homer512	162x488x276	3,947k	191k	1,517k	324k	62s	207s	561M	299s
Horse512	488x421x228	4,900k	214k	1,698k	349k	117s	289s	716M	449s
Plane	217x304x98	545k	110k	912k	71k	257s	57s	306M	323s
Mobile384	366x224x348	1,211k	175k	1,418k	123k	71s	83s	482M	171s
Octopus384	366x259x335	610k	76k	680k	64k	16s	24s	321M	54s
Rockerarm256	246x126x76	528k	67k	541k	58k	68s	31s	176M	105s
Rockerarm384	366x188x112	1,860k	154k	1,197k	156k	301s	142s	399M	461s

ble 4.1. Clearly, the spatial subdivision delivers the largest speed-up, as it is an improvement of the algorithm complexity, whereas the other two are not. On the other hand, this latter optimization is only used on the curve skeleton, whereas the other two also speed up the surface skeleton computation.

4.6 Results

We have implemented our algorithm in C++. Our software can be downloaded at http://www.win.tue.nl/~dreniers. We ran our software on a Pentium 4, 3GHZ with 1GB of RAM. As input we used several complex polygonal meshes [2], voxelized using binvox [84] for various resolutions. We used object resolutions ranging up to 512^3 voxels. For each object the resolution that we used is appended to the object's name. Various measurements are shown in Table 4.2. Column "dim" indicates the dimension of the object. Columns "$|\Omega|$" and "$|\partial\Omega|$" indicate the number of boundary and object voxels respectively. Columns "#paths" indicate the number of computed shortest paths, and "#comp." indicates how many times the connected components are computed (Figure 4.3, line 6), whereas "paths t" and "comp. t" show the total wall-clock time for both. The last column shows the total wall-clock time for the whole algorithm to complete. As can be seen, our multiscale skeletons can be computed within 10 minutes for all objects. Note that setting the user parameter τ to obtain skeletons simplified to different degrees can be done in real-time. Column "mem" shows the peak memory usage, which is below 800 MB for the objects we considered. We did not optimize for memory usage: it could be reduced easily

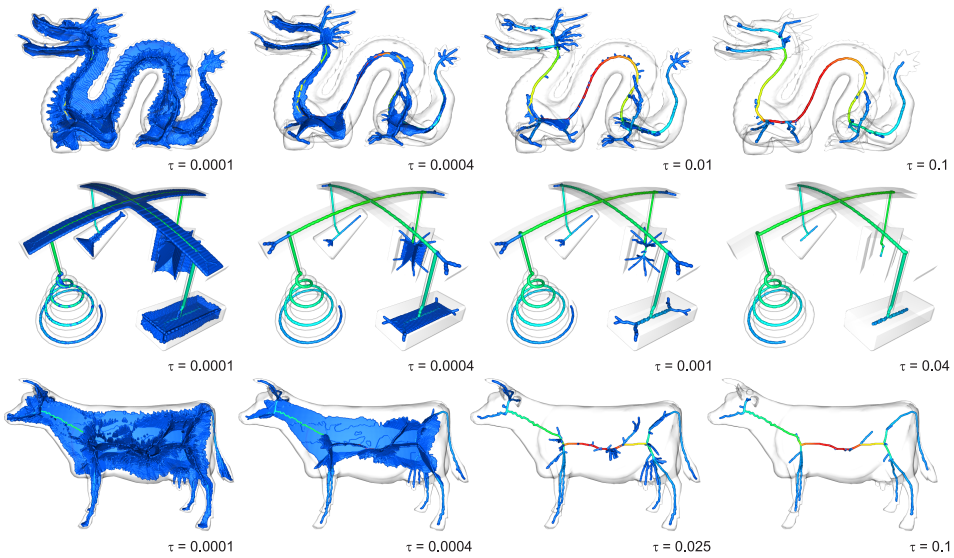


Figure 4.8: Simplified skeletons $\mathcal{S}_\tau$ of the Dragon, Mobile, and Cow objects at four thresholds τ. The importance measure is visualized using a rainbow color map.

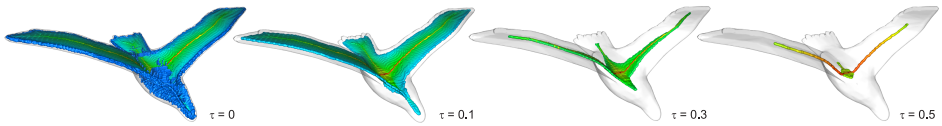


Figure 4.9: Simplified equalized skeletons $\mathcal{S}'_\tau$ for the Bird object.

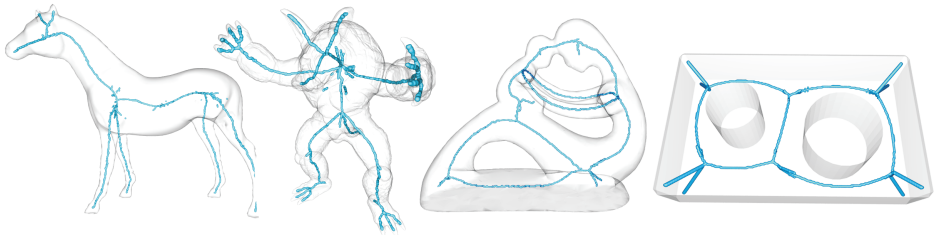


Figure 4.10: The non-simplified curve skeleton $\mathcal{C}$ for several objects.

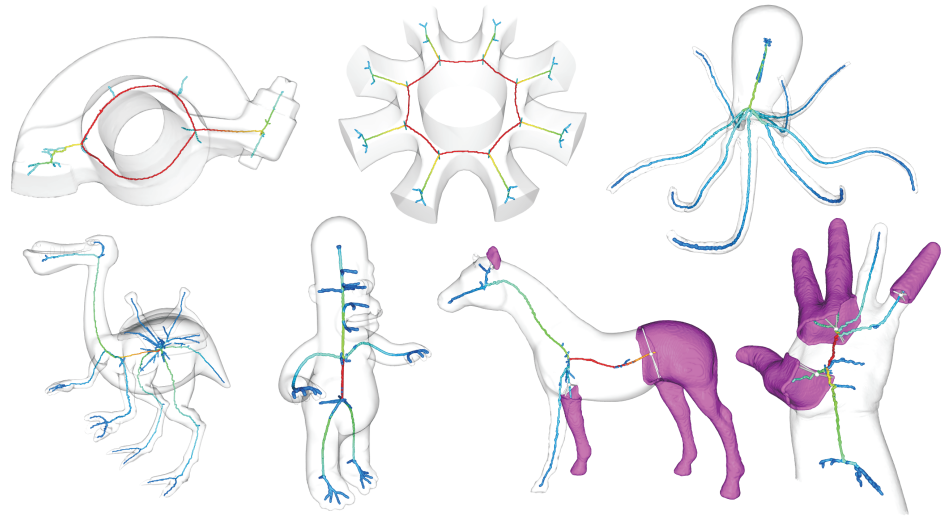


Figure 4.11: The importance measure $\rho_{\mathcal{C}}$ for several shapes, visualized using a rainbow color map. For the Hand and Horse objects we show the areas (magenta) associated with several selected curve-skeleton voxels (white). The line segments indicate feature voxels of the selected voxels.

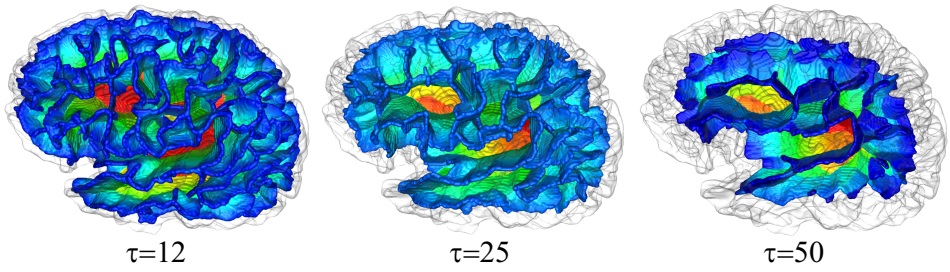


Figure 4.12: Simplified surface skeletons of a human brain for three simplification levels.

by not storing intermediate results as we do now. All results shown have been rendered using a raytracer [99] for better display. The voxel surfaces are rendered using implicit surfaces.

Figures 4.8 shows the simplified skeletons $\mathcal{S}_\tau$ of three objects as computed by our algorithm, with the indicated settings of τ. The importance measure is visualized using a rainbow color map, mapping 0 to blue (i.e., unimportant skeleton points) and 1 to red (i.e., central, important skeleton points). We observe that the non-generic cases in the Mobile object, such as the cylinder and extruded star, are handled well. The surface skeleton is mainly blue because its importance measure is significantly smaller than that of the curve skeleton, as explained in Section 4.3. In order to highlight the variation of ρ on $\mathcal{S} \setminus \mathcal{C}$ we can better use the equalized skeleton $\mathcal{S}'_\tau$. Figure 4.9 shows $\mathcal{S}'_\tau$ for the Bird object.

It is interesting to see how $\mathcal{S}_\tau$ and $\mathcal{S}'_\tau$ progress differently when increasing τ. With $\mathcal{S}'_\tau$, the surface and curve skeleton are simplified more simultaneously, especially near the periphery of the skeleton. This can be useful for some applications, as explained in the next section.

Figure 4.11 shows the curve skeleton and associated collapse measure $\rho_\mathcal{C}$ in isolation for various objects. We observe that the extracted curve skeletons reach into fine structures like the tentacle tips of the Octopus object and the Dinopet's fingers, are centered with respect to the object surface, and exhibit very little wiggle noise. Figure 4.10 shows several non-simplified curve skeletons $\mathcal{C}$. We observe some unimportant disconnected parts in the curve skeletons, the reason for which is explained in Section 4.7.1. Note again that these results are obtained without any thinning or other post-processing step enforcing connectivity.

Finally, Figure 4.12 shows the simplified surface skeletons for a human brain, obtained by segmenting a 3D grayscale image obtained using an MRI scanner. The brain has a complex folding surface, which can be problematic for skeletonization methods as the resulting skeletons are very complex. The curve skeleton is not shown as it is not meaningful for shapes like these that do not have a clear part structure. Because the brain is convex from a global point of view, the simplified surface skeleton is still connected. As can be seen, our method is able to produce the structurally complex surface skeleton of the brain with no problems.

More results of our method are available online [106], including animations of $\mathcal{S}_\tau$'s progression.

4.7 Discussion

Because curve and surface skeletonization methods differ in the precise skeleton definition that they use, the object representation they work on, and the applications they target, they are usually evaluated and compared by their skeleton properties as discussed in Section 2.3. In Section 4.7.1 we discuss how our approach satisfies these desirable properties. In Section 4.7.2, we compare our approach to some other approaches by visual inspection of the results, and discuss how several key aspects of our approach relate to other approaches.

4.7.1 Properties

The curve and surface skeletons are *invariant* under isometric transformations modulo the voxel discretization. The importance measure is slightly affected by object rotations because the shortest-path length computation by using Dijkstra's algorithm is. Alternatively, more accurate methods for computing shortest paths could be used [63]. Both skeletons are up to two voxels thick due to the use of the extended feature transform; a simple post-processing step could be added to make it one voxel *thin*, if desired. The surface skeleton is *centered*, as far as the discretization allows it, because it is defined as those points having at least two feature points. The curve skeleton is centered on the surface skeleton with respect to the importance measure $\rho_\mathcal{S}$, and is thus also centered within the object.

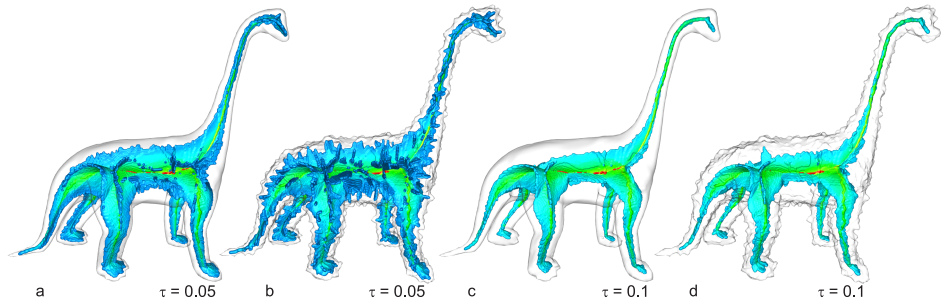


Figure 4.13: The skeletons can be made robust by increasing the threshold τ.

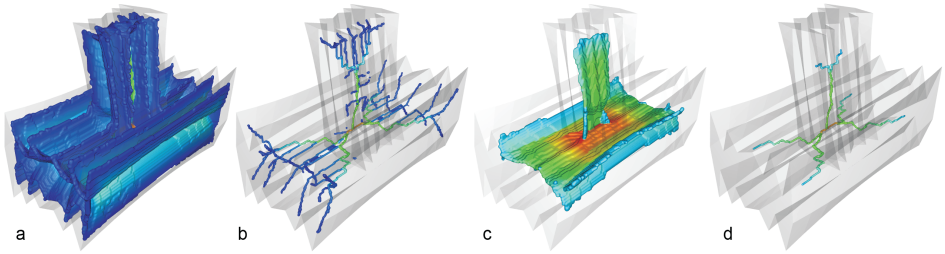


Figure 4.14: (a,b) Unstable surface and curve skeleton under ripple noise. (c,d) Robust skeletons.

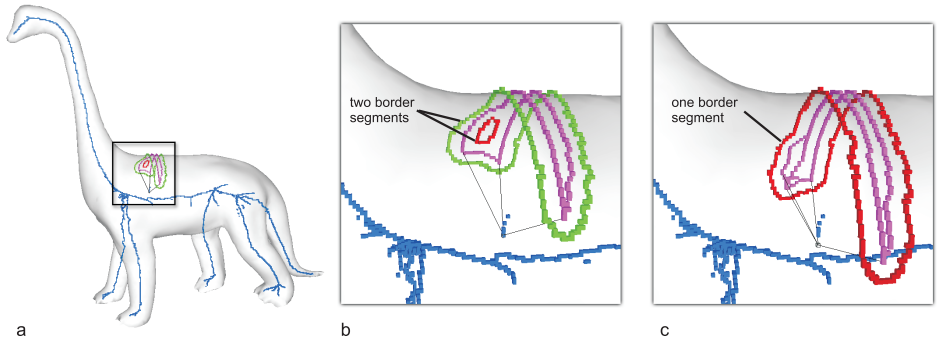


Figure 4.15: (a) Illustration of insignificant disconnections in non-simplified curve skeleton $\mathcal{C}$ of the Dino shape. (b) Selected voxel is detected as $\mathcal{C}$-voxel. (c) Selected voxel not classified as $\mathcal{C}$-voxel. Associated feature voxels indicated by line segments. Shortest-path set Γ shown in magenta. Border segments of dilated path set Γ' shown as red and green.

Because the collapse measure ρ is monotonically ascending, the simplified skeletons $\mathcal{S}_\tau$ are in principle *connected* by default. No special homotopy-preserving provisions are needed to ensure this, unlike e.g. [127, 137, 36], due to the global nature of the collapse measure. Nevertheless, we observe in Figure 4.10 that the computed curve skeletons may have some insignificant parts that are not connected to the main part. These disconnections of curve-skeleton parts are caused by the way in which we detect the curve skeleton, in combination with inaccuracies in feature point and shortest path computation due to the discretization. This is illustrated in Figure 4.15. Figure 4.15(b) shows a selected voxel which has been correctly identified as a curve skeleton voxel: we count two connected components in the boundary of the dilated path set Γ'. However, in Figure 4.15(c) we selected a nearby voxel that has not been detected as a curve-skeleton voxel, and which has a slightly different configuration of feature voxels. This configuration happens to generate more shortest paths, which causes the smaller area to be filled by the dilation, resulting in only one boundary component: the voxel is incorrectly classified as a non-curve skeleton voxel, causing a disconnection. These disconnections are not problematic because they have a low importance $\rho_\mathcal{C}$: the area bounded by the red border segment in Figure 4.15(b) bounds only a small area and does not represent a significant feature. The disconnected parts can be safely removed from $\mathcal{C}$ if desired, and disappear anyway when raising the threshold τ, as can be seen in Figure 4.11.

The simplification, or pruning, is *continuous* for the vast majority of shapes. That is, small changes in τ result in small changes in $\mathcal{S}_\tau$. The continuity can be ascribed to the fact that the shortest geodesic of a point $q \in \mathcal{S}$ evolves smoothly over $\mathcal{S}$, so that $\rho_\mathcal{S}$ is continuous. The Jordan curves in general evolve smoothly over $\mathcal{C}$ except at curve-skeleton junctions, so that $\rho_\mathcal{C}$ is continuous on $\mathcal{C}$ and only contains jumps at $\mathcal{C}$-junctions. The simplified skeletons can be considered a *multiscale* shape descriptor, because ρ is monotonically ascending on $\mathcal{S} \setminus \mathcal{C}$. Surface-skeleton points whose feature points are separated by large boundary features are assigned a higher importance than points whose feature points are separated by small features. Likewise, curve-skeleton points that represent areas containing small boundary features receive a lower importance than curve-skeleton points representing areas containing large features. Our multiscale representation is *hierarchical* because ρ represents a continuous hierarchy of nested skeletons in which each $\mathcal{S}_\tau$ represents a separate level.

Our approach satisfies the two additional desirable properties we proposed in Section 2.3 for methods that compute both curve and surface skeletons. First, the curve skeleton is *included* in the surface skeleton, because the curve skeleton is considered as a special case of the surface skeleton, namely as those points having more than one shortest geodesic. Second, our importance measure treats the non-skeleton, surface and curve-skeleton points in a *uniform* manner: each object point is processed independently and receives an importance given by a geometric quantity computed on the surface boundary.

Our approach has only a *single parameter* τ; no other hacks or settings are needed. The meaning of the importance ρ which is thresholded by τ is quite simple and intuitive: for a given skeleton point p, $\rho(p)$ represents the fraction of the object's boundary which is represented by that point.

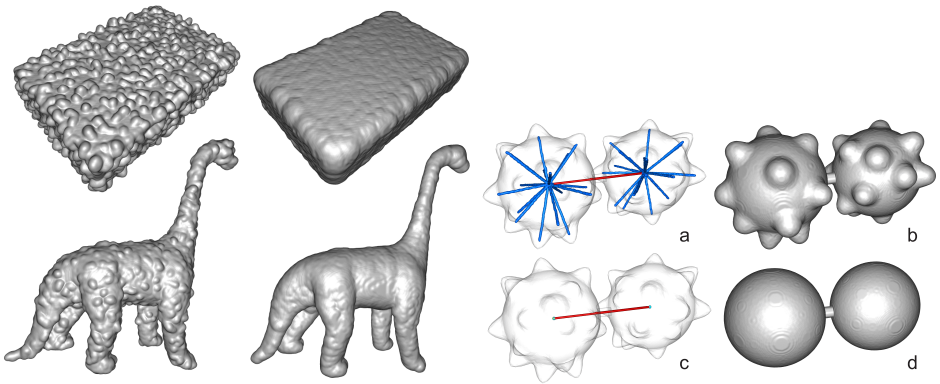
The *robustness* property requires the curve and surface skeletons of a noisy surface to be close to that of the corresponding smooth surface. We can achieve this behavior

by setting τ such that skeleton parts due to noise are pruned. Figure 4.13 shows the Dino object with and without boundary noise, obtained by randomly displacing points a distance of around 5% of the object radius in boundary-normal direction, and its skeleton for two thresholds τ. As expected, we observe that $\mathcal{S}'_\tau$ of the noisy Dino (b) is much noisier than $\mathcal{S}'_\tau$ of the Dino without noise (a) for small values of τ: $\tau < 0.05$. The skeleton of the noisy Dino can be made robust (d) by increasing τ. At $\tau = 0.1$, the structure of the simplified skeleton of the noisy Dino (d) is comparable to the skeleton of the non-noisy Dino (c). The importance measures on both are also very similar as indicated by the colors. In the above, we used the simplified equalized skeleton $\mathcal{S}'_\tau$. If we use the non-equalized skeleton $\mathcal{S}_\tau$, the spurious branches of the curve skeleton remain, until τ is so high that the surface skeleton is completely removed. Figure 4.14 illustrates robustness to a different kind of noise consisting of ripples or waves on the boundary surface, for both surface and curve skeletons.

Our definition of the collapse measure ρ provides us with a natural *skeleton-to-boundary mapping*. Surface skeleton points map to curves on the object surface, whereas curve-skeleton points map to compact areas. Figure 4.11 shows the skeleton-to-boundary mapping for several curve-skeleton points for the Hand and Horse objects. We observe that the components correspond to meaningful parts of the boundary, such as the legs or ears of the horse and fingers of the hand. The skeleton-to-boundary mapping can be used in various applications, e.g., for selection purposes in geometric modeling applications. Another application of the skeleton-to-boundary mapping is shape segmentation. The component sets associated with the curve skeleton branches coming together in a junction provide us with meaningful components: distinct, logical object parts. Combining the meaningful components that are associated with all curve skeleton junctions yields a shape segmentation having desirable properties, as follows. Because the skeleton-to-boundary mapping is based on shortest geodesics, the borders between segments are smooth, minimally twisting, and robust to noise. Two novel 3D shape-segmentation methods based on our simplified curve skeleton are presented in detail in Chapter 6.

A nice feature of our method is that curve-skeleton *junction detection* is performed during the skeletonization process itself. As our curve skeleton is not everywhere one voxel thick, detecting the junctions as a post-processing step is non-trivial. More importantly, it can be argued that junctions that are detected during skeletonization bear more significance than those extracted afterwards [23]. In the former case, they are intrinsic to the curve skeleton definition, whereas in the latter case, the junctions are merely a by-product and are as good as the curve skeleton itself.

The original object, or a simplified version thereof, can be *reconstructed* from the surface skeleton by placing at each voxel its maximally inscribed ball. In general, a faithful shape reconstruction from the curve skeleton is not possible by using only balls. One simple application of our simplified skeletons is that of surface smoothing. By reconstructing a surface from the simplified skeleton $\mathcal{S}_\tau$ for a small τ value, small-scale surface noise is replaced by sphere segments. For the purpose of surface smoothing, the simplified equalized skeleton $\mathcal{S}'_\tau$ is used, so that the spurious curve and surface skeleton parts due to the noise near the boundary are simplified together. Figure 4.16(a) shows two examples. Of course, τ cannot be too high, otherwise the reconstruction becomes too inaccurate, or "spherified". Reducing noise in this manner works best at thick object parts, such that the



(a) Surface smoothing of the Box and Dino objects. (b) Noise removal for two interconnected balls. The interconnection is not broken.

Figure 4.16: Surface smoothing by reconstruction from $\mathcal{S}'_\tau$.

inscribed balls of such parts are large in comparison to the inscribed balls in the perturbations. Indeed, noise on thin parts can be considered as object features, more so than noise on thick parts. Noise on the thin parts, namely the ridges of the box and the neck of the Dino, is less reduced than the thick parts. A nice feature of using the collapse measure is that the reconstructed object cannot become disconnected due to the skeleton simplification, because the simplified skeleton is always connected. In contrast, if we would simplify the Dino in Figure 4.16(a) using only the collapse measure as defined on $\mathcal{S} \setminus \mathcal{C}$, the neck would become disconnected. This is emphasized in Figure 4.16(b) showing a noisy Dumbbell shape, consisting of two interconnected balls. Although the interconnection and the noise are of similar scale, the noise is removed, but the interconnection is not broken.

The monotonicity of the collapse measure comes at a price. Our measure is not so efficient to compute as purely local importance measures, due to the global operations involved. Computing the feature transform takes $O(n)$ [90], where $n = |\Omega|$. Computing the set of shortest paths for an object voxel using Dijkstra (or A*) takes $O(b \log b)$ in the worst case, where $b = |\partial\Omega| \approx \log n$. The worst case is when the boundary is a sphere, as the shortest-path algorithm visits practically all boundary voxels for diametrically-opposed feature voxels. However, shortest paths are not computed between arbitrary boundary voxels, but always between associated feature-voxels, so that for objects that consist of distinct parts, the algorithm visits only a small subset of the boundary voxels. Computing the component set using our spatial subdivision scheme for an object voxel takes $O(\log b)$. In total, the worst case of our algorithm is $O(n(b \log b))$, but practical cases are far below this limit. Table 4.2 shows the relation between n, b, and the running times for several practical objects. In particular, we note that the speed of our implementation compares favorably to Dey and Sun's method [36], which is one of the most recent curve-skeletonization methods, and is similar to our method. On a comparable machine, their approach took half an hour to compute a curve skeleton for the Rockerarm. Our

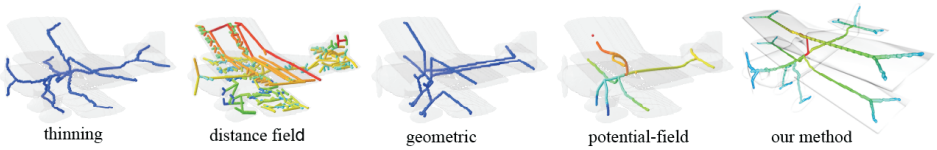


Figure 4.17: Curve skeletons of the Plane produced using four methods (images from [24]), compared with the result of our method.

approach took under 10 minutes, using a voxel resolution of 384^3 for a full curve and surface skeleton hierarchy, while observing that the complexity of the curve skeleton is comparable (compare Figure 4.11 and [36, Figure 5]). Our approach is faster than the potential-field method of Cornea et al. [25], which reportedly takes up to half an hour for voxel resolutions in the order of 200^3 on a comparable PC.

When discussing our method, we should stress the fact that our approach is, to our knowledge, the only one in existence that generalizes a global importance measure for 2D shapes, namely the boundary-distance measure, to 3D. This measure is the cornerstone of our method, as it guarantees the satisfaction of all the desirable properties considered, similar to its analogous 2D counterpart, the AFMM. Furthermore, we like to note that our method readily allows a *parallel implementation*. Each object point is treated independently (see the pseudocode in Figure 4.3). Only the feature transform that we compute initially is an inherently sequential process, and we do not need a post-processing step, such as a thinning or erosion step. In light of the current developments in microprocessor technology toward multicore systems, parallelizable algorithms are becoming more and more important.

Finally, our method is straightforward to implement, thanks to its voxel-based nature. The full pseudocode of our method is given in Figure 4.3. As can be seen, the most "complex" elements are the feature transform, Dijkstra's shortest-path algorithm, and connected-component computation. If the optimizations as discussed in Section 4.5.4 are desired, a caching scheme, a spatial-subdivision scheme, and a slight modification of Dijkstra's algorithm need to be implemented.

4.7.2 Comparison with other methods

Cornea et al. [24] made a comparison between four curve-skeletonization methods: a basic thinning, distance field, geometric, and potential-field method. In Figure 4.17, we visually compare the results of these four methods for the Plane shape (image from [24]), with the result of our method (right-most image). We observe that our method produces superior results. The price that we pay is that our method is slower than the first three methods mentioned, although it is of the same order as the potential-field method.

We would further like to compare the surface skeleton produced by our global importance measure with the skeleton resulting from a purely local measure. Consider Figure 4.18. The left two images show the surface skeleton $\mathcal{S}_\tau$ of the Cow object computed by our method. The right two images (taken from [137]) show the surface skeleton resulting from the moment-based measure from Rumpf and Telea [118], which is essentially a

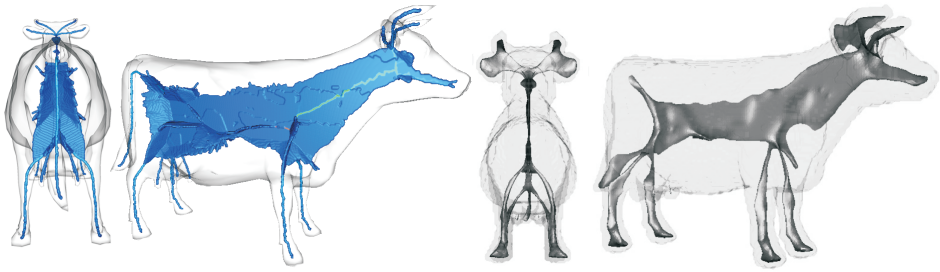


Figure 4.18: A comparison between the surface skeleton produced by using our global measure (left) and a typical local measure (right).

field-based detector for the singularities of the shape's distance transform. The threshold used for the latter images was carefully chosen such that the skeleton is both connected and noise free. We chose τ for our method such that the skeletons are most similar. Yet, we observe that our skeleton at this comparable simplification level is able to capture more details, such as the horizontal skeleton sheets and the skeleton in the udder and tail. This can be explained by the fact that local measures cannot distinguish between these fine structures and noise, and will eliminate them together, whereas global measures can. Our skeleton is connected regardless the setting of τ.

Prohaska and Hege [100] have also used the shortest-path length between feature points to detect and simplify surface skeletons, using the same idea as Costa [28], who extended the boundary-distance measure to 3D. However, as these methods do not detect the curve skeleton, they have to use a thinning method to ensure homotopy of the result. In contrast, our collapse measure ensures connectedness of the result while using the simple and intuitive thresholding strategy.

Dey and Sun [36] defined the curve skeleton as the singularities in the shortest-path length measure on the surface skeleton. There are some important differences with our method (which we developed independently from theirs). Most importantly, we produce *multiscale* curve and surface skeletons, while their approach does not. Second, their approach is more of a hybrid method, in which first the surface skeleton is computed with a Voronoi-based method. The medial geodesic function measure is then computed, after which a local flux measure [127] is used to find the curve skeleton on the derived surface skeleton structure. Because this latter measure is local, an erosion strategy is used to enforce connectedness. The sampling resolution must be high enough to accurately compute the involved divergence. In contrast, our approach computes a monotonic measure for both the curve and surface skeleton. The measure can be computed independently on a point-by-point fashion, making our algorithm simple and allowing parallel processing. Furthermore, all our computations are based on a non-derived structure, namely the object surface, whereas Dey and Sun need to compute a non-simplified surface skeleton first. This allows us to use a voxel representation, requiring simple data structures and algorithms, keeping our algorithm straightforward and efficient. A second advantage of using only non-derived structures and integral quantities, such as curve length and com-

ponent areas, is that our approach is very robust, even for coarsely sampled shapes.

Finally, let us mention that we have opted for a voxel-based approach for its ease of implementation. A disadvantage of this is that polyhedral models first need to be voxelized. The resulting skeletons are influenced by the grid's orientation, causing a slight loss of rotation invariance, and the grid's resolution, which may cause a loss of detail for objects containing small features. Methods acting directly on continuous geometrical data do not have this orientation issue (of course, they do have discretization issues). However, we would like to indicate that the definition of our global importance measure is not limited to discrete space. As long as the key ingredients of this measure are available, namely, computation of the feature points, shortest geodesics, and connected components, our approach can be adapted to other shape representations.

4.8 Conclusion

We have presented the collapse measure, a novel importance measure that enables the robust computation of multiscale curve and surface skeletons of 3D shapes. To our knowledge, this is the first truly global 3D importance measure that can be used to obtain both surface and curve skeleton hierarchies in a uniform manner. Our measure is constructed by extending the existing boundary-distance measure to 3D. The surface skeleton and curve skeleton are computed independently. On the surface skeleton the importance is expressed as curve length, whereas on the curve skeleton it is expressed as the area of a boundary component, which gives the measure an intuitive geometric meaning, and allows us to reason about it and deduce several properties of interest. We have proposed a practical algorithm that is straightforward to implement using a small set of robust operations. It does not need any post-processing steps to preserve topology. We have shown that our algorithm delivers good results on a wide range of real-world, complex objects.

Chapter 5

Segmenting Simplified Surface Skeletons

In this chapter we present a method to extract the structure of the simplified surface skeletons that we presented in the previous chapter. Our goal is to find the sheet intersection curves that partition the surface skeleton into its constituent sheets. These surface-skeleton segmentations can be used in several applications, such as shape analysis and matching tasks.

5.1 Introduction

Our goal in this chapter is to develop an algorithm for partitioning the simplified skeletons of binary voxel shapes into disjoint segments. As indicated in Section 2.2, the surface skeleton consists of a set of 2D surfaces, called *sheets*. A sheet boundary consists of 1D curves which are either part of the surface-skeleton boundary, or curves where at least three sheets intersect. Given this property, the sheet intersection curves are also called *Y-curves* [30]. Our aim is to robustly partition the surface skeleton into its constituent sheets. Being based on our simplified surface skeletons, we inherit robustness and multiscale properties of the partitioning. Figure 5.1 depicts a result of our method.

To give just one of many examples in which skeleton segmentations can be used, Siddiqi et al. [128] use them as a step in their 3D shape retrieval pipeline. They build a graph on the skeleton segments and use a graph-matching algorithm to do shape retrieval. As the authors indicate, the specific skeletonization method they use is not robust to very noisy shapes. We argue that our method can be used to resolve this issue.

It seems natural to use existing segmentation approaches for discrete surfaces based on digital topology. An example is the well-known template-based method of Malandain et al. [79]. However, we found that these methods produce undesirable results on our simplified skeletons. A naive segmentation using such local methods yields a different number of segments than it would for the non-simplified skeleton, which is undesirable: the segmentations should be noise-invariant. To correctly segment our simplified skeletons,

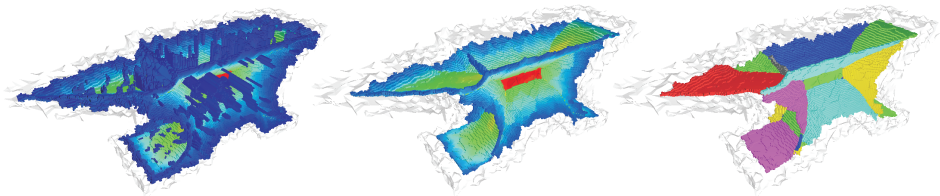


Figure 5.1: Segmenting the simplified surface skeleton of a noisy anvil. Non-simplified skeleton (left), simplified skeleton (middle), segmentation (right).

we consider both the simplified and non-simplified skeleton, denoted $\mathcal{S}_\tau$ and $\mathcal{S}_\upsilon$. Additionally, we consider not only the skeleton itself for detecting Y-curve points, but also its relation to the shape boundary by means of the feature transform and shortest geodesics. The idea is to use the feature-set cardinality to classify Y-curves. Although this works in the continuous case, it does not translate directly to discrete cases. We present a solution that uses the same ingredients as our skeletonization method from Chapter 4. This makes our approach more robust than local methods for complex skeletons. Our segmentation method can handle shapes with large amounts of noise on the boundary, without having to prune the segmentation afterwards. Besides the segmentation itself, the simplified Y-network is also a useful result of our approach, which can be used in shape analysis tasks.

The outline of this chapter is as follows. Section 5.2 details on the structure of surface skeleton and the definition of the surface-skeleton segmentation. Section 5.3 presents our method for computing the simplified Y-network Y_τ, at simplification level τ, and the decomposition of Y_τ into its constituent Y-curves. In Section 5.4, we extend Y_τ using information from a non-simplified Y-network Y_υ to correctly and robustly segment the simplified skeleton $\mathcal{S}_\tau$ at level of detail τ. Section 5.5 presents and discusses results. Section 5.6 concludes this chapter.

5.2 Skeleton structure and segment definition

Let Ω be a 3D shape with boundary $\partial\Omega$, let D be the distance transform (Eq. 2.2), let F be the feature transform (Eq. 3.1). Similar to Eq. 2.1, the surface skeleton $\mathcal{S}$ can be defined as those object points having at least two feature points:

$$\mathcal{S}(\Omega) = \left\{ p \in \Omega \;\middle|\; |F(p)| \geq 2 \right\}. \tag{5.1}$$

It is well known that the skeleton of a 3D shape consists of manifolds, called *sheets*, that meet in curves, called *Y-curves* [30]. A sheet boundary consists of Y-curves and/or skeleton boundary curves. In addition to the above sheet boundary curves, a 3D skeleton can also contain isolated curves in some degenerate cases, such as a cylinder. We assume for the time being that the skeleton contains no such degeneracies. The set of Y-curves is called the *Y-network*, denoted $Y = \{y_1, ..., y_n\}$. In the following, a *skeleton segment* is equivalent to a sheet. For example, Figure 5.2 shows the skeleton of a box, containing 12

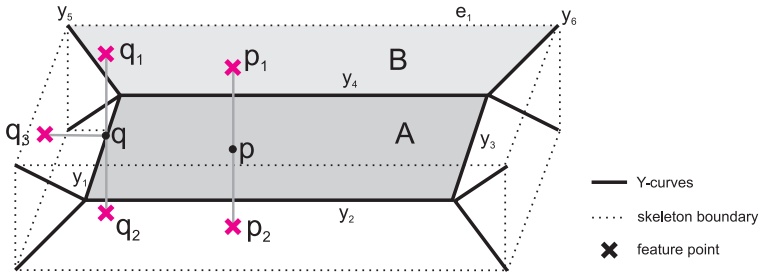


Figure 5.2: Skeleton structure of a box. Point p is a sheet point having two feature points p_1, p_2, point q is a Y-curve point having three feature points q_1, q_2, q_3.

skeleton boundary curves (dotted lines), and a Y-network containing 12 Y-curves (thick lines), yielding 13 sheets, or segments. For illustration, segment A is bounded by Y-curves $y_1 ... y_4$, whereas B is bounded by Y-curves y_4, y_5, y_6 and skeleton boundary curve e_1.

Skeleton points can be classified by the number of feature points they have [46]. A skeleton boundary-curve point has an infinite number of feature points in the continuous case, because the inscribed ball at such a point has a contiguous arc of contact points with $\partial\Omega$. A sheet point p has exactly two feature points, $|F(p)| = 2$. Finally, a Y-curve point q, where three sheets meet, has three feature points, $|F(q)| = 3$. Each sheet contributes two feature points, but each feature point is shared with one of the two other sheets. Figure 5.2 illustrates such a sheet point p and Y-curve point q and their respective feature points (crosses).

Within a sheet, the associated feature points evolve smoothly over the object boundary. That is, for a point $p + \Delta p$ close to p, the feature points $F(p + \Delta p)$ neighbor the feature points $F(p)$ of p. The reason for this is as follows. Take two interior sheet-points p and q that are at a small distance ϵ. By definition, each point has two feature points $F(p) = \{p_a, p_b\}$, $F(q) = \{q_a, q_b\}$, where p_a and q_a are corresponding, i.e., are at the same side of the sheet. Let the distance ϵ between p and q go to 0 and assume that the feature points do not evolve smoothly. Then we have that the distance δ between (at least) one of the two corresponding feature pairs, e.g., p_a and q_a, cannot go to 0. This means that we have one skeleton point $p = q$ which has three feature points p_a, q_a and $p_b = q_b$, which means that p cannot be an interior-sheet point, leading to a contradiction.

5.3 The simplified Y-network

In this section, we show how we extend our work on simplified skeletons from Chapter 4 to compute the simplified Y-network, on which our skeleton segmentation is based.

5.3.1 Computing the Y-network

In order to find the Y-network of a simplified skeleton $\mathcal{S}_\tau$ we must check if a voxel is on a Y-curve or not. In the continuous $\mathbb{R}^3$ space, a Y-curve point has (at least) three feature points: $|F| \geq 3$ (Sec. 5.2). However, as indicated in Section 4.5 we have to use the extended feature transform $\overline{F}$ in $\mathbb{Z}^3$ (Eq. 4.9). In this extended feature transform, many object voxels will have at least three feature voxels, not just the Y-curve voxels. For example, $\overline{F}(p)$ in Figure 5.3(a) contains 4 feature voxels $p_1 ... p_4$ and $\overline{F}(q)$ contains 6 voxels: $q_1 .. q_6$. If we naively use the cardinality of $\overline{F}$ to detect the Y-curves, then both p and q would be selected, which is wrong for p. To solve this problem, we group each two feature points that have a small geodesic distance (shortest-path length) on the boundary. More formally, we define an equivalence relation $a \sim b$ on $\overline{F}$:

$$a \sim b \Leftrightarrow \text{length}(\gamma(a, b)) < \tau \,, \tag{5.2}$$

where τ is the same threshold that we used to create simplified skeletons (Eq. 4.8). This relation gives rise to a number equivalence classes. We now replace $\overline{F}$ by the so-called *simplified feature transform* $\overline{F}_\tau$, defined as a set of class representatives following Eq. 5.2, that is, a subset of $\overline{F}$ containing one point from each equivalence class. Which particular point we choose in a class is not important, as we are only interested in the cardinality of $\overline{F}_\tau$. However, to be consistent we choose those points that admit the largest geodesic distances.

Using the simplified feature transform $\overline{F}_\tau$, we can rephrase the definition of the simplified surface skeleton $\mathcal{S}_\tau$ (compare Eq. 5.1):

$$\mathcal{S}_\tau(\Omega) = \left\{ p \in \Omega \;\middle|\; |\overline{F}_\tau(p)| \geq 2 \right\} . \tag{5.3}$$

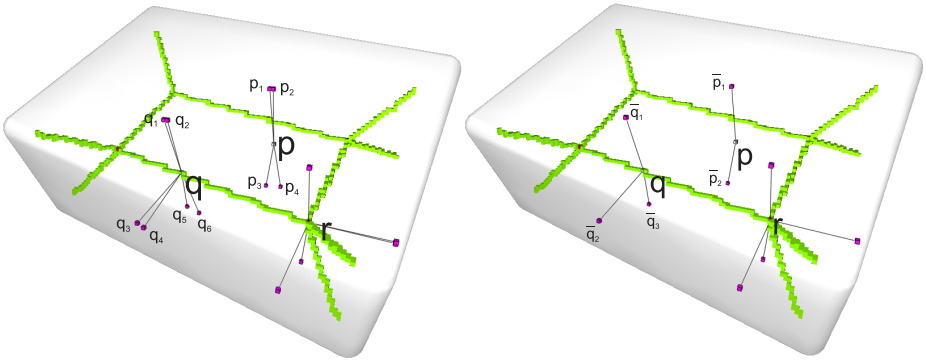
In $\mathcal{S}_\tau$ all sheet points p that have a shortest path between their two feature points that is shorter than τ, or in other words, that have a lower importance than τ, are pruned. Note that in this chapter, $\mathcal{S}_\tau$ denotes the simplified surface skeleton only, and does not include the curve skeleton as was the case in Chapter 4.

Now, the *simplified Y-network* is straightforwardly defined as:

$$Y_\tau = \left\{ p \in \Omega \;\middle|\; |\overline{F}_\tau(p)| \geq 3 \right\} , \tag{5.4}$$

which is a subset of $\mathcal{S}_\tau$. For a Y-curve point q, where three sheets meet in q's neighborhood, this means that if one of the sheets in its neighborhood is pruned because its importance is lower than τ, the point is not considered a Y-curve point in Y_τ. It is important to note that the simplified Y-network is not simply a post-processing of the simplified skeleton. Instead, the Y-network is computed *directly* out of the shape, using the integral quantity of geodesic distance on the object boundary. This is more stable than extracting the Y-curves from the voxelized skeleton using templates for instance, in analogy with the junction-detection property of the curve-skeleton (see Section 2.3.2), and also offers the same natural scale parameter as for skeleton computation.

Figure 5.3 shows a non-axis aligned box with its simplified network Y_τ, $\tau = 10$. Like in the continuous case, the discrete Y-network forms a connected structure. Three



(a) The feature transform $\overline{F}$. (b) The simplified feature transform $\overline{F}_\tau$.

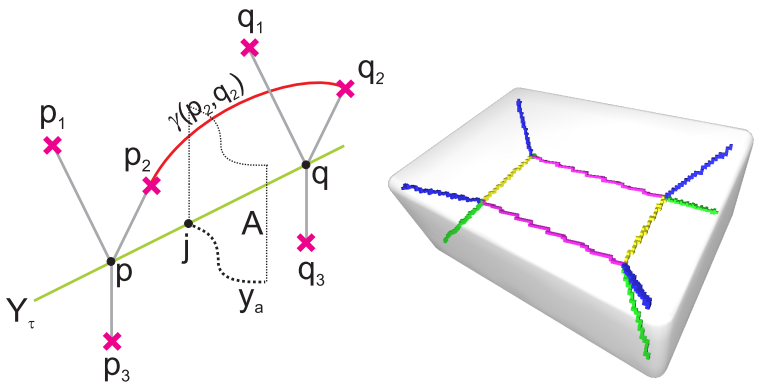
Figure 5.3: A (non-axis aligned) box with its detected network $Y_{\tau=10}$. Three selected points p, q, r: a sheet point p, a Y-curve point q, and a Y-curve intersection r.

points p, q, r are selected: a sheet point p, a Y-curve point q, and a Y-curve intersection point r. Their feature voxels are connected to the corresponding points by line segments. Figure 5.3(a) shows the non-simplified feature transform $\overline{F}$ for the selected points, while Figure 5.3(b) shows the simplified feature transform $\overline{F}_\tau$. We see the merit of $\overline{F}_\tau$: p and q can be classified as a sheet and a Y-curve voxel respectively based on the cardinality $|\overline{F}_\tau|$, but not on $|\overline{F}|$. For point q for instance, $\overline{F}_\tau$ gives us exactly three feature points $\overline{q}_1, \overline{q}_2, \overline{q}_3$, i.e., the representatives of the classes $\{q_1, q_2\}, \{q_3, q_4\}, \{q_5, q_6\}$.

5.3.2 Y-network Decomposition

Although Eq. 5.4 enables the detection of the Y-network voxels, it does not provide us with the structure of the Y-network as a collection of Y-curves. This section presents how to compute such a decomposition.

To decompose the network Y_τ into its n Y-curves $\{y_1, ..., y_n\}$, we define two points $p, q \in Y_\tau$ to be on the same Y-curve when there is no junction in the Y-network between them. For illustration, Figure 5.4(a) sketches (part) of a Y-network. Let p, q be two points on Y_τ, each one having three feature points, with the indices p_1, p_2, p_3 and q_1, q_2, q_3 chosen in such a way that they "correspond", that is, the sum of the geodesic distances $\sum_i \text{length}(\gamma(p_i, q_i))$ is minimal. Suppose now there is a junction j between p, q, due to a Y-curve y_a as shown (dotted) in the figure. Such a Y-curve y_a results from a sheet A with a local importance higher than τ, otherwise y_a would have been pruned and would not be present in Y_τ. Thus, we can detect whether there is a junction j between p and q by looking at the geodesic distance between the feature points of p and q. Because the feature points evolve smoothly within a skeletal sheet, we argue that if there is a Y-curve y_a between p, q due to sheet A, the geodesic distance between the feature points p_2 and q_2 must be larger than τ, because A's importance is at least τ. Generalizing, two points $p, q \in Y_\tau$ belong to the same Y-curve y if and only if all three geodesic distances between



(a) Two points p, q on a Y-network. (b) The detected Y-curves of a box.

Figure 5.4: Y-network decomposition into Y-curves.

each pair of feature points (p_i, q_i) are smaller than τ:

$$p, q \in y \in Y_\tau \Leftrightarrow \forall_{i \in \{1,2,3\}} \text{length}(\gamma(p_i, q_i)) \leq \tau \; . \tag{5.5}$$

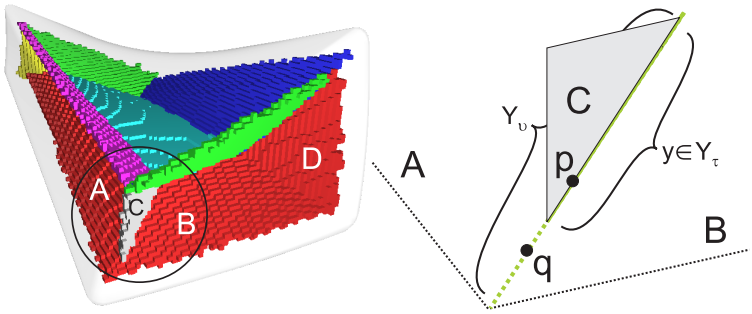
One remark is due. Voxels where several Y-curves come together have more than three feature points (see e.g. Figure 5.3(b), point r). To handle these cases, Eq. 5.5 is applied for all subsets of both p and q, having 3 feature points.

One might ask why we do not just detect the Y-curve junctions and use these to separate the Y-curves, which is more trivial. The first reason is that our approach is of sub-voxel precision: a single voxel may contain multiple Y-curves. Second, our method does not use a topological analysis of the Y-network voxels and is thus more accurate in cases where Y-curves meet under small angles. Figure 5.4(b) shows the segmentation of the Y-network of a box obtained using Eq. 5.5. The Y-curves are distinctively colored.

5.4 Skeleton segmentation

Although it seems natural to use the simplified network Y_τ to segment the skeleton $\mathcal{S}_\tau$, this can sometimes deliver undesired results. Figure 5.5(a) shows the segmentation produced for the skeleton $\mathcal{S}_\tau$ ($\tau = 10$) of a twisted box. We expect 13 segments, similar to the skeleton segmentation of a non-deformed box (Figure 5.2). Yet, for the deformed box some segments get merged inadvertently. For example, the skeleton sheets A, B and D are incorrectly merged, whereas they should be distinct segments as for a non-deformed box.

Figure 5.5(b) is a schematic close-up of the situation. The problem is that the Y-curve $y \in Y_\tau$ does not extend all the way to the skeleton boundary. The missing part is indicated by the dotted line. A narrow tunnel connects the areas A and B on the skeleton manifold, so that they end up in the same segment. The reason that y is too short is that sheet C is simplified for lower values of τ than A and B at the dotted line segment. The



(a) Sheets A, B and D are wrongly in the same segment.

(b) Close-up of (a).

Figure 5.5: Segmented simplified skeleton of the Deformed Box using Y_τ.

cardinality of the simplified feature transform for these points is 2, so that point q, for example, is not detected as a Y-curve point by Eq. 5.4. In other words, only two sheets are found to come together at q in the *simplified* skeleton, namely A and B. For the same reason, other surface segmentation approaches based on local topology, e.g. [79], would fail segmenting this simplified skeleton. A second reason why the connected component approach would fail is that our skeletons are up two voxels thick, if we refrain from any post-processing like thinning.

We solve the issue by noticing that curve y in Figure 5.5(b) would be longer for less simplified skeletons. For non-simplified skeletons, we would not have the problem at all. Hence, to segment a skeleton $\mathcal{S}_\tau$ we use a less simplified Y-network Y_υ, $\upsilon < \tau$, which contains longer, extended, versions of the Y-curves which ended prematurely in Y_τ. However, we must be careful to only extend Y-curves from Y_τ, and not to incorporate *any* Y-curves that only occur in Y_υ. Hence, we only consider those Y-curves $y \in Y_\upsilon$ for which there is at least one point $p \in y$ in Y_τ. We call this set the extension of Y_τ using Y_υ, denoted as $Y_{\tau,\upsilon}$:

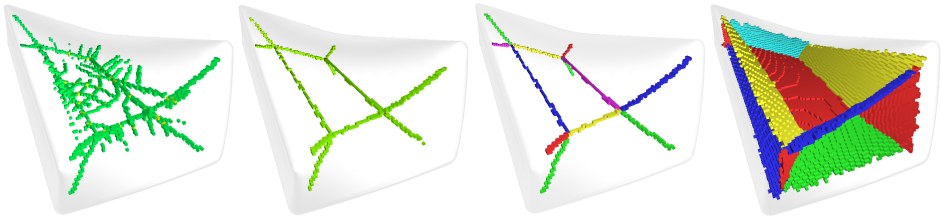
$$Y_{\tau,\upsilon} = \left\{ y \in Y_\upsilon \;\middle|\; \exists_{p \in y}\, p \in Y_\tau \right\}. \tag{5.6}$$

Finally, we compute the decomposition of $Y_{\tau,\upsilon}$ into its respective Y-curves by taking both scales τ and υ into account and adapting Eq. 5.5 accordingly:

$$p, q \in y \in Y_{\tau,\upsilon} \Leftrightarrow \forall_{i \in \{1,2,3\}} \text{length}(\gamma(p_i, q_i)) \leq \begin{cases} \tau & \text{if } p \in Y_\tau \wedge q \in Y_\tau \\ \upsilon & \text{otherwise.} \end{cases} \tag{5.7}$$

As the value of υ we use 5 voxel lengths, which represents the non-simplified skeleton as explained in Section 4.5.1.

After computing $Y_{\tau,\upsilon}$ we can segment $\mathcal{S}_\tau$ as follows. We consider the skeleton $\mathcal{S}_\tau$ as a 26-connected graph of voxels, from which we remove the voxels occupied by $Y_{\tau,\upsilon}$, and then determine the connected components in the remaining skeleton graph using a flood-fill for instance. As $\mathcal{S}_\tau$ can be up to two voxels thick and the Y-network is only one voxel thick, we first dilate the Y-network by 1 voxel in each 26-direction, before removing them



(a) $Y_\upsilon, \upsilon = 5$ (b) $Y_\tau, \tau = 10$ (c) $Y_{\tau,\upsilon}$ (d) segmentation

Figure 5.6: Correctly segmenting the simplified skeleton of a deformed box. (a) Non-simplified Y-network Y_υ. (b) Simplified Y-network Y_τ. (c) Extension of Y_τ using Y_υ. (d) Correct segmentation.

from the skeleton graph. Hereafter we erode the dilated Y-network so that these voxels are also part of a segment. Clearly, many other alternative implementations for finding the segments are possible once we have a robust and complete Y-network.

Figure 5.6 shows Y_υ, Y_τ, $Y_{\tau,\upsilon}$ (decomposed in its Y-curves), and the segmentation based on $Y_{\tau,\upsilon}$ respectively, which is a correct segmentation of the deformed box skeleton as opposed to Figure 5.5(a).

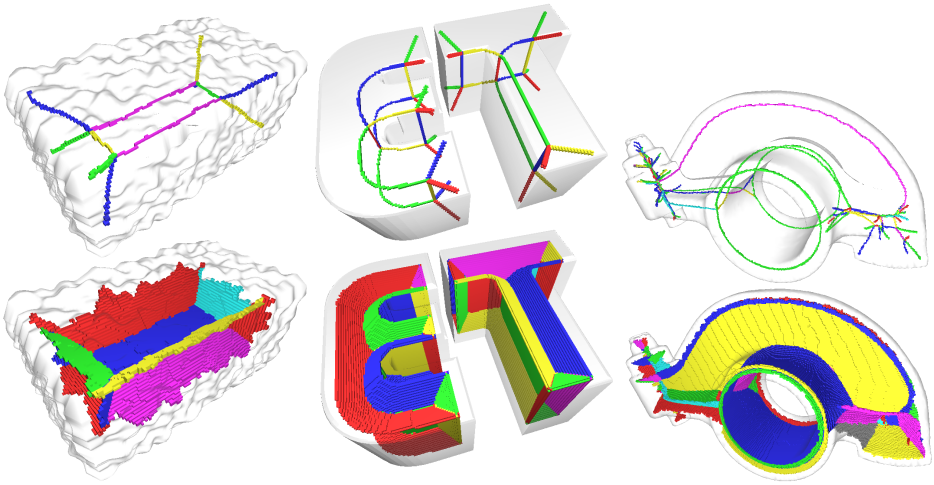
5.5 Results and discussion

We have tested our method on shapes of varying complexity and amount of boundary noise. As input objects we used 3D triangle meshes, voxelized using binvox [84] in various resolutions ranging up to two million object voxels. For all images, the original object mesh representation is rendered instead of its voxel representation for nicer display.

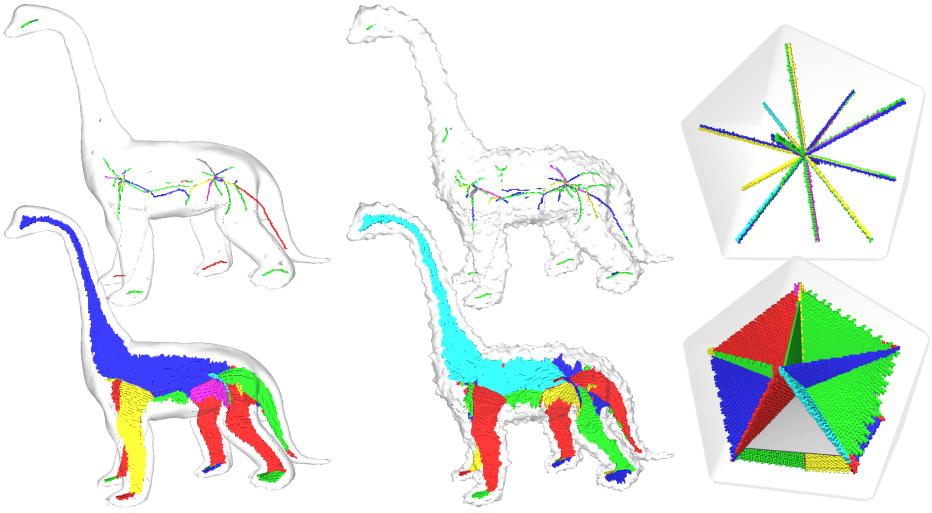
Figure 5.7 shows a selection of the results. For each object, τ is chosen based on the noise level of the object, and we show both the extended simplified network $Y_{\tau,\upsilon}$, decomposed into its Y-curves (top image), and the segmentation of the simplified skeleton (bottom image) using $Y_{\tau,\upsilon}$. In Figures 5.7(a) and 5.7(e), we added 5-10% boundary noise to show the robustness of our approach.

We highlight some of the results. For the Noisybox, we see that our method correctly detects 13 segments and 12 Y-curves. For the Dinosaur and Noisydino, the four legs and feet are all assigned different segments. Our method correctly segments the skeleton of the Dodecahedron (Figure 5.7(f)). This is a difficult skeleton to segment as it contains degeneracies, in the sense that each Y-curve actually separates five sheets instead of the usual three. Finally, Figure 5.8 shows both the Y-network and the segmentation for a human brain (the skeletons are shown in Figure 4.12). The images illustrate that our method produces accurate results for such complex surfaces and corresponding skeletons.

Table 5.1 shows timing measurements performed on a Pentium IV 3 GHz, with 1 GB of RAM for all objects in this chapter. Columns "dim", "$|\Omega|$", and $|\partial\Omega|$ denote the grid resolution, and the number of object and boundary voxels respectively. Columns "$\mathcal{S}$ time" and "segm. time" denote the wall-clock time for computing the skeleton and



(a) Noisybox, $\tau = 20$ (b) ET, $\tau = 10$ (c) Rockerarm, $\tau = 10$



(d) Dino, $\tau = 16$ (e) Noisydino, τ=16 (f) Dodecahedron, τ=10

Figure 5.7: Y-network decomposition (top), $\mathcal{S}_\tau$ segmentation (bottom).

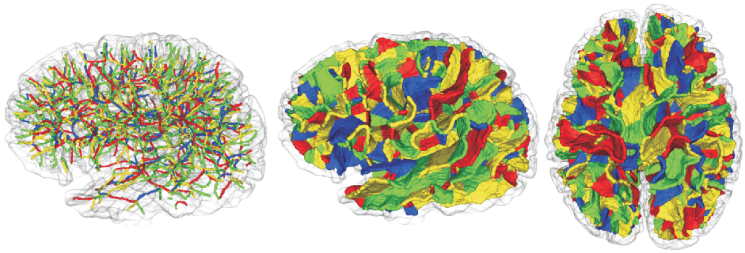


Figure 5.8: Y-network and $\mathcal{S}_\tau$-segmentation for a human brain.

Table 5.1: Timing measurements

Object	dim	$\lvert\Omega\rvert$	$\lvert\partial\Omega\rvert$	$\mathcal{S}$ time (s)	segm. time (s)
Deformed Box	65x64x124	247k	24k	21	11
Dodecahedron	124x124x124	945k	39k	48	17
ET	125x78x173	1.045k	93k	304	37
Noisydino	99x325x365	1.128k	100k	62	33
Rockerarm	366x188x112	2.000k	151k	470	62

segmentation respectively. Computation of the segmentation is non-optimized, as the skeleton computation times are dominating anyhow.

5.6 Conclusion

We have presented a voxel-based approach for robustly segmenting simplified skeletons of 3D shapes, based on the simplified feature transform and simplified Y-network. The Y-network is also decomposed into its respective Y-curves. It could be argued that the simplified Y-network is more useful in certain shape analysis or retrieval tasks than the segmentation itself, because the Y-curves change more continuously under shape deformations than the segmentation does.

Our entire method relies upon the choice of a single parameter τ, which controls the simplification of the skeleton, but *not* the segmentation itself, thereby yielding a fully autonomous segmentation method. One limitation of our current implementation is that cylindrical object-parts having degenerate curve skeletons may locally result in over-segmentation, because the feature points do not evolve smoothly there. Second, the two-voxel thickness of the skeletons might yield undesirable topological changes in the skeleton for too complex objects, when compared to thin skeletons. Again, this is not a major problem, as standard thinning methods can be used to get a one-voxel thin skeleton.

We have considered using our robust skeleton segmentations to produce robust patch-type segmentations. The idea was to project the segmentation of the skeleton to the shape surface by means of the feature points. The feature points of each skeleton segment then produces two segments on the surface: one on each side of the segment. A straightforward application of this idea clearly results in over-segmentation. For example, a box has 13 skeleton segments, yielding 26 surface segments, necessitating some merging step afterwards. We did not explore this further. Instead, in Chapter 7 we present a more straightforward patch-type segmentation method based on simplified surface skeletons that is not based on the partitioning presented here.

In future work, it would be interesting to investigate whether the skeleton partitioning can be used to "unprune" the skeleton. Although simplified skeletons are robust to noise when setting τ appropriately, the simplification not only prunes the noisy skeleton parts, but also removes parts of the skeleton representing features of the shape. In fact, this is why in Figure 4.16(a) the reconstructed box is not only rid from noise, but is also smoothed on its edges. A skeleton partitioning of the box's simplified skeleton contains only the essential segments. It seems interesting to try to unprune the simplified skeleton

by adding voxels from the non-simplified skeleton that are in the extension of these essential segments. Reconstructing the box from such an unpruned skeleton would enable edge-preserving denoising.

Chapter 6

Part-type Shape Segmentation

In this chapter and the next one we deal with the problem of 3D shape segmentation. Shape segmentation (also called partitioning or decomposition) is the task of dividing the shape surface into disjoint, compact subsets, called segments. We base the segmentation solely on the geometry of the shape and not on any secondary attributes such as texture and color. Segmentation applications include shape analysis, shape matching, medical imaging, collision detection, and other geometric processing methods employing divide-and-conquer strategies. One can distinguish between patch-type and part-type segmentations [123], both having their own merits. The former type is a geometry-oriented abstraction of the shape, whereas the latter is a high-level, semantically-oriented abstraction that agrees more with how a human would understand the part structure of a shape. The difference is illustrated in Figure 6.1. Whereas a typical part-type method produces two segments, one for each box-part of the shape, a patch-type method typically finds quasi-flat patches that are separated by high-curvature creases, in this case the faces of the boxes. In this chapter we focus on part-type methods, whereas in Chapter 7 we focus on the problem of patch-type segmentation.

6.1 Introduction

Part-type segmentation is the more loosely defined task of decomposing a 3D shape into its meaningful components, namely those that a human being would intuitively perceive as the distinct, logical parts of the shape. These methods often rely on cognition principles and the quality of their results is inherently subjective [8]. Part-type segmentations are most suitable for articulated shapes, that is, consisting of clear, distinct parts, such as animals, humans, plants, and other natural shapes, and less suitable for faceted or blob-like shapes. In this chapter, we present a part-type segmentation framework that uses the curve skeleton as computed in Chapter 4.

Our part-type segmentation method aims to assign a segment to each meaningful part of a shape. We consider shapes to be the union of several simpler shapes: the *parts* of the shape. Although it is rather subjective what exactly constitutes a meaningful part, we can

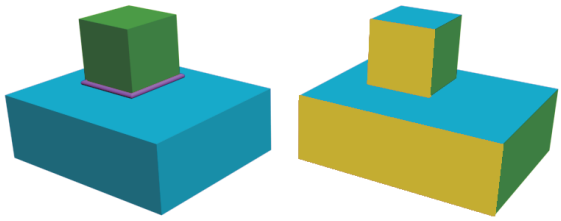


Figure 6.1: Part-type (left) versus patch-type segmentation (right) of a shape consisting of two boxes.

formulate the following requirements on a segmentation:

- Each segment should be a connected boundary component.
- The shape should not be under-segmented. This implies that each segment should be *simple*: it should not contain any bifurcations.
- The shape should not be over-segmented. This implies that each part should be assigned at most one segment.
- Segment borders should be tight. Each segment should contain the whole part and nothing more.
- Segment borders should be as smooth as the shape allows.

Segmentation methods commonly employ the *minima rule* [53], which is the notion that when two separate shapes interpenetrate, they do so along negative minima of principal curvatures, that is, in a concave crease. When such a crease forms a closed loop, it can be considered a *part cut*, separating the two parts. When it does not form a closed loop, one has to decide whether it should be closed to form a part cut, and if so, how. It holds further that the more concave the crease is, the more salient the part cut is [54]. In case of our example in Figure 6.1, the part cut formed by the border between the two interpenetrating boxes is indeed formed by a concave contour, indicated by the magenta curve. Methods can put the minima rule to use by explicitly placing part cuts at these concave creases (e.g. [69]), or in an implicit way, by clustering surface elements based on geodesic and angular distances, which essentially models the minima rule (e.g. [61]).

Other segmentation methods use the curve skeleton to identify parts. As indicated in Section 2.3, a well-known property of the curve skeleton is that it "allows for component-wise differentiation". In other words, it can be used to infer the shape's part structure. Indeed, segments should be simple, they should not contain bifurcations, and curve skeletons represent bifurcations explicitly in the form of junctions. We therefore base our segmentation framework on the following notion: when two (simple) shapes interpenetrate, their respective curve skeletons join, resulting in a new junction in the curve skeleton located near the intersection. We call this the *junction rule*, in analogy with the minima rule. Figure 6.2 illustrates both the minima rule and the junction rule for two interpenetrating

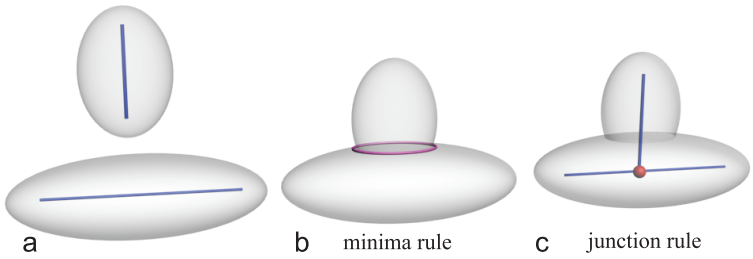


Figure 6.2: (a) An interpenetration of two ellipsoids. (b) The minima rule finds the concave crease (magenta curve). (c) The junction rule finds the curve-skeleton junction.

ellipsoids. Whereas the minima rule is boundary-based, as it uses surface curvature, the junction rule is interior-based, as it uses information about both the shape boundary and its interior by means of the curve skeleton. By basically using more information, the junction rule can lead to more robust algorithms, especially for noisy and low-resolution shapes, for which curvature information is problematic to compute robustly. The junction rule is also more suitable for voxel shapes, for which the boundary normal is not readily available. Another advantage of the junction rule is that it is not ambiguous: a junction in the curve skeleton can be clearly defined and detected. In the minima rule, on the other hand, a concave crease can be anywhere between shallow and deep, varying along the crease, and the curvature might even vanish for certain shapes. This happens for example when the smaller box in Figure 6.1 is slightly moved to the left so that its left-hand face aligns with the left-hand face of the larger box. Deciding which creases are salient enough and how they should be closed is a subtle process, which we avoid by using the junction rule.

An important difference between both rules is that whereas the minima rule provides the part cut explicitly as the concave crease, the junction rule only signals the event of part interpenetration, but does not say anything about the location of the part cut. However, recall that our particular curve skeleton definition (Eq. 4.5) associates with each curve-skeleton point a simple closed curve, i.e., a Jordan curve, on the shape surface. Each Jordan curve (locally) divides the surface into two connected components and thus presents a candidate part cut. In curve-skeleton junctions, where multiple branches come together, the Jordan curves of the different branches are combined, dividing the boundary into more than two components.

In this chapter we present a framework for segmenting articulated voxel shapes based on the junction rule. Figure 6.3 illustrates our framework, which consists of three stages. In the first stage the curve skeleton is computed as described in Chapter 4, using the optimization that we present in Section 6.3.1. In the second stage, cut points on the curve skeleton are selected. *Cut points* are cut-generating points: points whose associated Jordan curves make for suitable part cuts. We developed two schemes to automatically select cut points. The first and most straightforward one is to select the curve-skeleton junctions directly, called the *junction-cut* scheme. However, whereas this method yields good results for some shapes, it leads to unexpected results for others. The reason is that a curve-

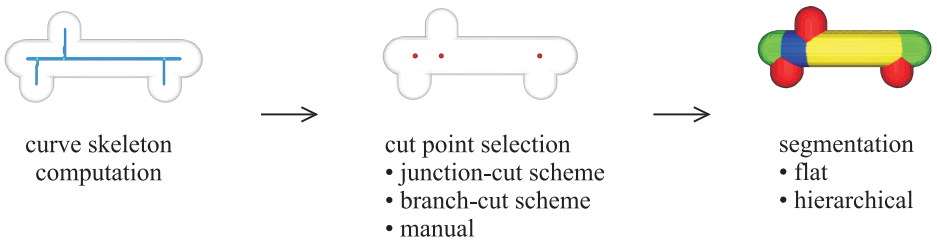


Figure 6.3: An overview of our framework.

skeleton junction merely signals the interpenetration of parts, but the junction itself does not give the optimal placement for the part cuts, nor does it indicate the correct number of part cuts that should be generated. In the second scheme, called the *branch-cut* scheme, we search, starting from the junctions, along the curve-skeleton branches for more suitable locations than the junctions themselves, based on a so-called "geodesicness" measure of the Jordan curve. As a third option, the cut points can be chosen manually if one desires to manually fine-tune the segmentation. In the third and last stage of our framework, a segmentation is produced from the selected cut points. Here, we have developed two options that lead to different segmentation types: we produce either a hierarchical or a "flat" segmentation.

Our framework satisfies all the above-mentioned segmentation requirements. In particular, a strong feature of our framework is that it produces smooth part cuts, because they are piecewise geodesic by construction. Furthermore, the saliency of the parts is guaranteed by the curve-skeletonization stage, in which the curve skeleton is simplified using our intuitive collapse measure ρ_C (Eq. 4.4), so that only significant curve-skeleton points are retained. This makes our method robust to boundary noise. Finally, the segmentations are pose-invariant, that is, the segmentations do not change (significantly) under different poses of the shape.

The outline of this chapter is as follows. In the next section we discuss related work on part-type segmentation. In Section 6.3 we present an optimization of the curve skeleton computation, when compared to the algorithm from Chapter 4, and detail on the robust detection of junctions, a necessary component of our method. Section 6.4 details on the cut point selection. In Section 6.5 we use the cut points to create a segmentation. Section 6.6 presents results and a discussion. We evaluate the method according to criteria identified in literature and we compare our method to the relevant state-of-the-art methods. Section 6.7 concludes this chapter.

6.2 Related work

Part-type segmentation methods can be classified in various ways. One is by the type of information that they use: we distinguish between boundary-based and interior-based methods. Our approach is interior-based. In an overview of the major shape segmentation techniques, Shamir [123] identifies the following approaches: region growing, hierarchi-

cal clustering, iterative clustering, spectral clustering, and other approaches. This classification shows the current emphasis on boundary-based methods, and the relation of shape segmentation to clustering.

Boundary-based methods typically use the minima rule. Katz and Tal [61] define a distance matrix on the mesh faces, and then produce a fuzzy clustering of the faces. In a fuzzy clustering, a single face can belong to multiple clusters, to each with a certain probability, together summing to 1. The distance function used is a combination of geodesic and angular distances and essentially models the minima rule. After clustering, the fuzzy part cuts are refined by a graph min-cut algorithm, to ensure that they are smooth and non-jaggy. Liu and Zhang [75] use the same distance matrix to construct an affinity matrix on which spectral clustering is performed. The eigenvectors of the affinity matrix are used to aid K-means clustering. The algorithm is simpler than Katz and Tal's [61], but is reported to have problems when the intended part cut passes a featureless region. Katz et al. [60] first extract prominent points of the shape, based on their geodesic distances, to ensure pose-invariance of the segmentation. Here, prominent points are typically the tips of parts, such as the fingers of a hand. Each segment is made to represent at least one feature point. Additionally to these segments, a core segment is extracted by a spherical mirroring operation. Lee et. al. [69] employ the minima rule explicitly to find the concave creases. As these might not be closed, they are closed to form a loop. Several measures are defined to guide the closing, as simply taking the shortest path between the endpoints does usually not give satisfying results. The most salient loops are then chosen, and a snake-based approach moves the loops to more suitable locations, forming part cuts. It is interesting to note that when closing the partial creases, the authors use a "centricity" measure to ensure that the loops are perpendicular to the curve skeleton, something that users are found to be interested in. This suggests that the boundary-based minima rule alone does not lead to the best results, but that interior-based skeletal information is essential.

Manual segmentation methods also exist. Funkhouser et al. [41] for example provide a manual scissoring method, that is made "intelligent" by guiding the user. The user paints a stroke on the mesh, after which the stroke is closed automatically by computing the shortest paths between endpoints. After this, the part cut can be manually refined by over-drawing strokes.

Interior-based methods consider the shape volume in addition to its surface. Mortara et al. [88] propose a multi-resolution approach, that blows a spherical bubble at each mesh vertex and studies how the intersection of that bubble with the surface evolves. They analyze both the curvature of a vertex over variable-size neighborhoods, and the topology of the surface in that neighborhood. Vertices are classified into specific feature types, such as tips, pits, and joints. Mortara et al. [89] specialize this approach to partition a shape into generic body and tubular parts. Dey et al. [35] perform a segmentation by determining the stable manifolds of the maxima in a flow function that extends the distance-to-boundary function. Because this inevitably causes over-segmentation, as there might be many maxima due to noise, the segments are merged in a subsequent step.

Another way to consider the shape's interior is by means of a skeletal structure, such as the curve skeleton. Our approach belongs to this category. The characterizing components of these approaches are the particular curve skeleton definition that they use, how

events on the curve skeleton are detected, and how these events are mapped to meaningful part cuts. Li et al. [72] compute a curve skeleton using edge contraction. As the curve skeleton computed in this manner is not connected by default, it is made connected by inserting branches. Then, a plane is swept along the branches, starting from the leaf vertices. The cross sections are analyzed to signal events based on the sudden changes in cross section perimeter and/or a change in its topology. These events are then mapped to the boundary by the same planar cross sections. Cornea et al. [25] use force-following of boundary particles in a potential-field inside the shape to compute the curve skeleton. As a possible application of their skeletonization technique, they present a shape segmentation, by projecting skeleton segments that are delimited by junctions back to the boundary using the particle trajectories. Lien et at. [73] observe that segmentation and curve-skeletonization share similar properties and propose a method that does both simultaneously. First, a simple approximate convex decomposition is made, in which each component has a maximum concavity that is less than a tunable threshold. Then, a simple skeleton is extracted for each component. If for a component the quality of the skeleton is not sufficient, the algorithm recursively decomposes and skeletonizes the component. Brunner [19] presents a method that voxelizes a triangle mesh, and its curve skeleton is computed by a thinning approach. Since the association of curve-skeleton voxels with the boundary is lost during thinning, it is reconstructed by associating each triangle with the closest curve-skeleton voxel. The individual curve-skeleton branches, delimited by junctions, are mapped to the boundary as final segments.

Instead of the curve skeleton, one can also use Reeb graphs [51], produced from the iso-contours of a suitable mapping function on the surface. Tierny et al. [139] define the mapping function as the curvature-constrained geodesic-distance to the closest feature points, detected as the extrema of another geodesic-based function. A subset of the contours is selected based on their topological and geometric evolution, and are used as part cuts.

A comparative study of five recent part-type methods is given by Attene et al. [8]. In Section 6.6.2 we compare the results of our method with those of several of the above-mentioned methods.

6.3 Preliminaries

In Section 6.3.1 we present an optimization in our curve skeleton computation by using the fact that we do not need to compute the surface skeleton to do part-type segmentation, resulting in a considerable speed-up. In Section 6.3.2 we detail on the robust detection of curve-skeleton junctions, a necessary step in order to apply the junction rule.

6.3.1 Skeleton computation optimization

In Chapter 4 we have established that the curve skeleton $\mathcal{C}$ (Eq. 4.5) that we compute is homotopic to the shape. Assuming the shape consists of a single connected component, the curve skeleton is connected. The connectivity of the curve skeleton allows us to make the following optimization to the algorithm if only the curve skeleton is needed

instead of the complete surface skeleton, as is the case here. Instead of processing all object voxels, we stop when we have detected the first curve-skeleton voxel p, called the seed voxel. From there, we consider the 26-adjacent voxels of p, and iteratively continue the curve skeleton detection only for those neighbors classified as curve-skeleton voxels. Because there may be some small parts of the curve skeleton that are disconnected from the main part, due to discretization inaccuracies (see Section 4.7.1), we search for a new seed voxel if the number of curve-skeleton voxels detected so far is smaller than a threshold, which we set to an empirically determined value of 1% of the object volume (evaluated as voxel count) for all shapes shown in this chapter. This simple modification of the algorithm means that we no longer have to visit all object voxels, but only a 1D subset. This improves the worst-case time complexity of the algorithm from $O(n(b \log b))$ to $O(\sqrt[3]{n}(b \log b))$, giving speed-ups up to a factor 10 as compared to the full skeleton computation presented in Chapter 4. For example, the time needed for computing the curve skeletons for the Horse384 and Armadillo256 objects from Figure 4.10 goes down from 200 and 86 seconds to 25 and 9 seconds respectively (timings without initialization).

6.3.2 Robust junction detection

In Section 4.3 we explained how the shortest-path set $\Gamma(p)$ associated with each object voxel p is used to detect curve-skeleton voxels. Recall that we dilated the voxels in the path set slightly so that we obtain a thick set of voxels Γ' on the object surface (Section 4.5). We determine the border voxels of the set, and count the number of connected border segments. In case we count two segments, we detect a regular curve-skeleton point, as Γ represents a Jordan curve. In case of three or more segments, we detect a curve-skeleton junction. Basically, we use the genus of Γ' to classify the curve-skeleton points as either junction or regular.

However, the genus of $\Gamma'(p)$ is a conservative criterion for detecting junctions. The computed genus may be higher than in the original object, due to boundary noise or discretization artifacts. This is problematic if we want to use the detected junctions for segmentation. In spirit of the collapse measure, we resolve this by discarding junctions that have small components in their component set C, as they likely result from noise. One could simply filter out small components in $C(p)$ and count the remaining components: if at least three components remain, point p is a junction. However, junctions on $\mathcal{C}$-loops cannot be detected in this manner, as these junctions generate only two components.

The solution is to associate each connected border segment of the dilated path-set $\Gamma'(p)$ with the component in $C(p)$ it bounds, and then filter and count the border segments instead of the components. Concisely put, a point p is a junction if and only if its dilated path-set Γ' has at least three connected border segments, that bound large enough components. We denote the set of robust junctions J_τ:

$$J_\tau = \Big\{ p \in J_0 \;\Big|\; |\Gamma'_\tau(p)| \geq 3 \Big\}, \tag{6.1}$$

where J_0 is the set of conservatively detected junctions from Section 4.5, and $\Gamma'_\tau(p)$ are

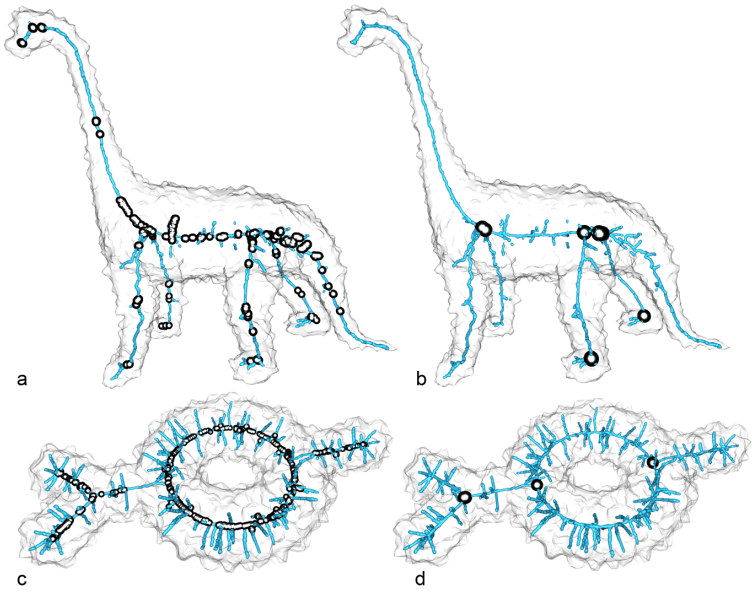


Figure 6.4: Conservative junctions J_0 (a,c). Robust junctions J_τ (b,d). Junctions represented by black balls.

the "pruned" border segments in the dilated path-set $\Gamma'(p)$:

$$\Gamma'_\tau(p) = \left\{ b \in \Gamma'(p) \;\middle|\; c \in C(p) \wedge b \subseteq \partial c \Rightarrow \frac{\text{area}(c)}{\text{area}(\partial\Omega)} > \tau \right\}, \tag{6.2}$$

where b is a connected border segment from $\Gamma'(p)$ and ∂c is the border of component c on $\partial\Omega$. Although related to the computation of $\rho_{\mathcal{C}}$, we do not have to compute $\rho_{\mathcal{C}}$ to compute J_τ: we only have to compute the component sets for the conservative junctions J_0. Equivalent to the computation of $\mathcal{C}_\tau$, the simplification level τ is a user parameter used to distinguish between small-scale noise and signal. The parameter must be set experimentally. We use a fixed value of $\tau = 0.001$, that is, 0.1% of the total surface area, for all the non-noisy shapes shown in this chapter. For the noisy shapes we used a fixed value of $\tau = 5\%$.

Figure 6.4 shows the effect of using Eq. 6.1. Figures 6.4(a) and (c) show the noisy Dino and Genus1 objects with conservatively detected junctions J_0 depicted as black balls. Figures 6.4(b) and (d) show the robust junctions computed by Eq. 6.1.

6.4 Cut point selection

After the simplified curve skeleton has been computed, one can use multiple ways to select cut points on the curve skeleton. One option is to let users select cut points manually. Because the curve skeleton is a 1D structure, and every curve-skeleton point has a valid

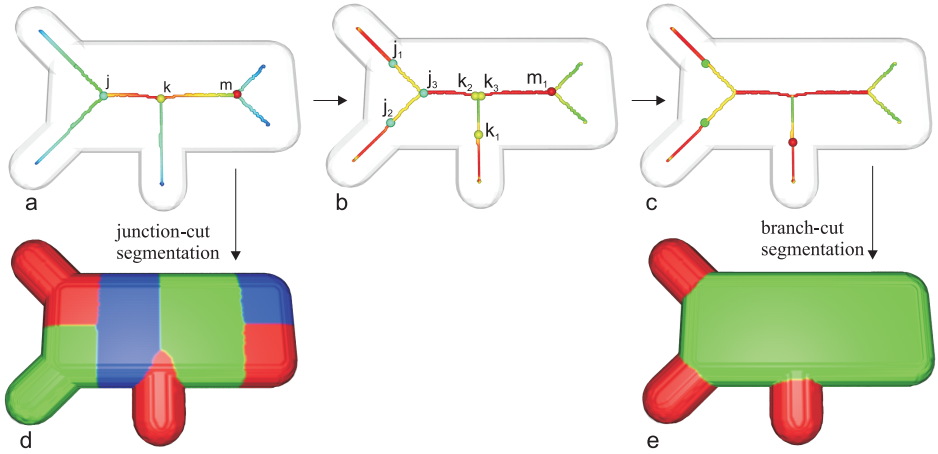


Figure 6.5: (a) The curve skeleton of an artificial shape consisting of a chamfered box and three attached tubes. The curve skeleton has a 2-junction j, a 1-junction k, and a 0-junction m. (b) Candidate cut points based on σ (encoded by color map). (c) Final branch cut points after junction-type detection. (d) Segmentation using junction cut points yields undesirable results. (e) Segmentation using the branch cut points yield correct results.

part cut associated with it by means of its Jordan curve, manual selection is an easy and intuitive process. Nevertheless, fully automatic segmentation is preferred for most applications. We have developed two automatic schemes, both based on the junction rule. The first scheme is called the *junction-cut* scheme. This method simply chooses the curve-skeleton junctions directly as the cut points. The shortest-path sets Γ of the junctions are then used to create either a flat or hierarchical segmentation (see Section 6.5). Several junction-cut segmentations obtained using this approach are shown in Figure 6.16.

Although the junction-cut scheme produces satisfying results for some shapes, it delivers non-intuitive results for some others. Consider Figure 6.5(d). We observe the following problems:

- Over-segmentation. We constructed the shape as consisting of a chamfered box with three tubes. We thus expect to find four parts, but we find seven instead. In other words, some part cuts are *invalid*.
- Non-tight part cuts. Although the bottom-left tube is given its own segment, the part cut does not split off the tube tightly.
- Non-smooth part cuts. Some of the part cuts are not smooth. The part cut of the bottom-left tube for example has a 90 degrees corner on top of the box. Non-tight and/or non-smooth part cuts are *inaccurate*.

We next explain why the junctions are suitable for signaling the interpenetration of parts, but why they are not optimal as cut-generating points. For the sake of discussion, we assume the generic case of junctions having exactly three emanating branches. In

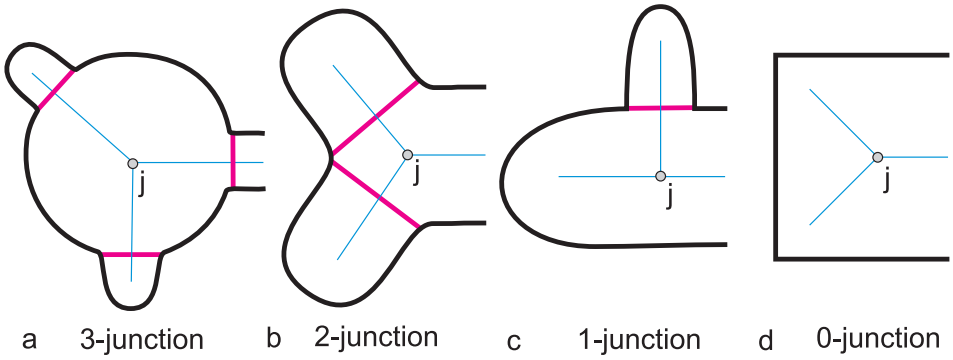


Figure 6.6: Schematic overview of the four junction types and their desirable part cuts (magenta line-segments). An n-junction ($n = \{0, 1, 2, 3\}$) generates $n + 1$ parts using n part cuts. Curve skeletons are blue.

non-generic cases junctions may have more branches. Our algorithm, presented in the following sections, can also handle these cases.

First, by using junctions as cut points, three part cuts per junction are generated, because three branches come together in a junction. However, on actual shapes we may find that junctions either indicate the intersection of 4, 3, or 2 parts, or do not indicate a part cut at all. Let an n-junction be a junction for which n part cuts should be generated, that is, it represents the intersection of $n + 1$ parts. Figure 6.6 schematically shows how one would intuitively place part cuts for four different junction types. Each of the four shapes gives rise to a different number of part cuts. We have observed that 1-junctions occur most frequently in real-world shapes, whereas 3-junctions are uncommon (one example is the junction in the body of the Crab shape in Figure 6.19). Figure 6.6(d) shows a 0-junction: it does not represent a valid part cut at all. The junction results from the fact that the skeleton extends into the two corners of the rectangle. Although the shape could be interpreted as consisting of three interpenetrating parts, namely as the two corners and the rectangle's rump, this is non-intuitive and no part cuts should be generated.

Considering that a continuous deformation exists between each two of these shapes, it is clear that the junction type is inherently subjective. For example, if the disc of Figure 6.6(a) becomes smaller and smaller, the situation transforms, suddenly, at some point, into that of Figure 6.6(b). As another example, when we increase the width of the smaller part in Figure 6.6(c), the situation gradually changes into that of Figure 6.6(b). Note that we do not consider here the continuity, or visual conductance [59], of branches to distinguish parts, and only their relative sizes. The two examples indicate that part-type shape segmentation can be considered a subjective task [8]. To summarize, to segment using the junction rule, we should be able to differentiate between the different junction types.

The second issue with using junctions as cut points is that the path sets of the junctions are ill suited as part cuts as they do not split off parts tightly and smoothly. Figure 6.5(b) illustrates this using a chamfered box with three attached tubes. The path set of junction j (Figure 6.5(a)) does not split off the two tubes accurately: each tube contains a

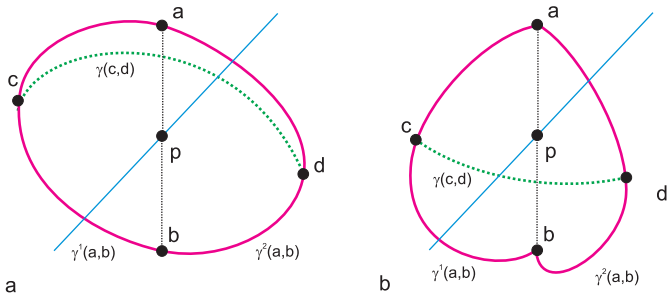


Figure 6.7: The geodesicness measure computed for a curve-skeleton point p with feature points a, b and Jordan curve $\Gamma(p) = \{\gamma^1(a,b), \gamma^2(a,b)\}$. (a) High geodesicness ($\sigma \approx 1$). (b) Medium geodesicness ($\sigma \approx 0.5$).

large part of the chamfered box, and the segment borders exhibit a number of curvature discontinuities.

We propose the *branch-cut* scheme to resolve these issues. An example branch-cut segmentation is shown in Figure 6.5(e), which correctly finds the box and the three tubes, tightly split off. The branch-cut scheme first finds the set of accurate (tight and smooth) part cuts for each junction, and then uses junction-type detection to select the subset of valid part cuts. In Section 6.4.1, we define the geodesicness measure, which guides the selection of accurate part cuts, which is explained in Section 6.4.2. In Section 6.4.3 we describe the selection of a valid subset of the cut points by means of junction-type detection.

6.4.1 Geodesicness measure

Recall from Section 4.3 that the Jordan curve $\Gamma(p)$ that is associated with each regular (non-junction) curve-skeleton point p consists of two shortest geodesics $\Gamma(p) = \{\gamma_1(a,b), \gamma_2(a,b)\}$ between p's feature points a, b. Because the piecewise geodesic Jordan curves are used as part cuts, the borders are smooth for the larger part, a desirable property for any segmentation method. However, this property does not need to hold at the two feature points $F(p)$ where the two geodesics might meet under an angle. We next introduce the *geodesicness measure* $\sigma : \mathcal{C} \rightarrow [0..1]$ on the curve skeleton which measures the degree to which the total Jordan curve $\Gamma(p)$ is geodesic.

A geodesic is the curved-space generalization of a straight line. Thus, our measure σ should be 1.0 if and only if the two geodesics meet at an angle of 180 degrees in both feature points. In that case the whole Jordan curve is a geodesic. Using differential geometry, we can project $\Gamma(p)$ in a small neighborhood of the feature point a onto the surface tangent plane at a, to determine the angle that the two geodesics make, and then do the same for b. However, for low-resolution voxel shapes and shapes that are noisy, a relatively large neighborhood must be taken, in which case the projection may not be accurate.

We take a more robust approach instead. Let c be the midpoint of one geodesic

$\gamma^1(a,b)$, and d of the other $\gamma^2(a,b)$ (see Figure. 6.7(a)). We compute the shortest geodesic $\gamma(c,d)$. If $\gamma(c,d)$ is a subset of Γ, it follows that Γ is a shortest geodesic between c,d. Furthermore, it can be seen that the more $\gamma(c,d)$ deviates from Γ, the more Γ deviates from a geodesic at the feature points. This deviation can be measured by the surface area spanned by the geodesic triangle acd, or, in terms of difference in length, which we choose because it is more easily computed:

$$\sigma(p \in \mathcal{C}) = \frac{\text{length}(\gamma(c,d))}{\text{length}(\gamma(c,a)) + \text{length}(\gamma(a,d))} \,. \tag{6.3}$$

This is a dimensionless measure attaining values between 0 and 1, the latter indicating a geodesic Jordan curve. The case of junctions, in which Γ consists of multiple Jordan curves, is handled by computing the minimum σ among Jordan curves.

We have assumed that curve-skeleton points have two feature points. However, when the curve skeleton coincides with a sheet intersection curve, a point $p \in \mathcal{C}$ has (at least) three feature points, resulting in more than two shortest geodesics. To handle these cases, we compute the measure σ by taking the minimum value of σ computed between each pair of geodesics.

6.4.2 Selecting candidate cut points

We use the robust junctions J_τ as computed in Section 6.3 to search for cut points whose Jordan curves make for accurate part cuts, based on the geodesicness measure σ as defined in Section 6.4.1. Higher values of σ indicate a smooth Jordan curve and thus a good candidate for cut point placement. Figure 6.5(c) depicts σ on $\mathcal{C}$. Looking at the branches emanating from the junctions into the tubes, we see that σ is low at the junction itself and reaches a maximum inside the tubes. We thus proceed as follows. Starting from a junction $j \in J_\tau$, we search along the $\mathcal{C}$-branches emanating from j for the first point q which has a sufficiently high value of σ. The search for q on a branch stops when another junction or the end of the branch is reached. In practice, σ does not reach the maximum of 1.0 inside each protrusion that can be considered a part, so we tolerate lower values: $T_\sigma < \sigma$. Extensive experimental study suggests that setting the threshold T_σ to 0.8 gives good results. For some branches, the very first (regular) point of the branch might have high enough σ to be marked as a cut point (e.g., point m_1 in Figure 6.5(b)). For other branches, we might find no such point: no point generates a suitable part cut (see, e.g., the other branches emanating from m).

To summarize, we obtain for each junction j a set of candidate cut points Q, whose cardinality is at most the number of branches at j. For branches whose measure σ stays low until the end we find no cut points, so that $|Q|$ is lower than the number of branches. Figure 6.8 schematically shows the detected candidate cut points $\{q_i\}$ that we would detect in this manner for the four different junction types. We observe that although the detected part cuts are tight and smooth (i.e., straight lines in this 2D projection), not all cut points make for valid part cuts, and some cut points should be removed from Q. In Figure 6.8(a), all cut points are valid: no points should be removed from Q. In Figure 6.8(b), point q_2 should be removed, in Figure 6.8(c), points q_2 and q_3 should be removed. In Figure 6.8(d) the only candidate cut point q_1 that was found should be

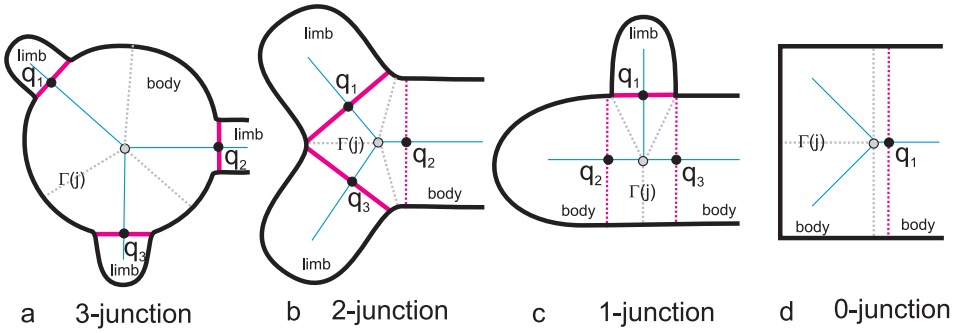


Figure 6.8: Four junction types (a,b,c,d). The Jordan curves $\Gamma(q_i)$ for the candidate cut points q_i are shown as magenta line-segments. Valid part cuts shown as thick magenta lines, invalid ones are dashed. Shortest-path sets of the junctions are shown using gray, dashed line-segments $\Gamma(j)$.

removed. To remove these invalid cut points, we have to perform junction-type detection, as explained in the next section.

6.4.3 Junction-type detection

After finding accurate cut points, we perform junction-type detection to select the valid cut points among them. Hereto, we make the assumption that each intersection of parts involves one *body* part, which is the "largest" intersecting part, and several *limb* parts, which are the "smaller" parts. In Figure 6.8(a) for example, there is one disc-shaped body and three limbs. In Figure 6.8(c), there is one body and one limb, for which q_1 is the part cut. We thus want to distinguish the body part from the limb parts by their sizes. We can express the size of a part in at least two ways: by means of the part's surface area, or by means of its circumference at the point of intersection. The problem with the former is that it is a non-local property, possibly giving non-intuitive results. For example, the limb due to q_2 in Figure 6.8(a) can have a smaller or a larger surface area than the disc-like part depending on what the hidden right-hand part of the shape looks like. The part that is considered the body would thus depend on what the hidden part looks like, which is clearly undesirable. A better, more local criterion is the circumference of the limb at the intersection.

We further notice that the junction can be considered to lie inside the body, and never in one of the limbs: the body is represented by two branches emanating from the junction, while the limb has one (consider Figure 6.2(c) for example). Hence, our idea is to compare the circumference of the limb at the part cut, which we measure by the length of the Jordan curve $\Gamma(q_i)$ of cut point q_i, with the circumference of the body, given by the maximum length among the Jordan curves in $\Gamma(j)$ of the junction j. Note that although typically $\text{length}(\Gamma(q_i)) = 2\rho_{\mathcal{S}}$, this does not hold for non-generic cases in which q has three feature points. Let ϑ_j denote the "part-cut similarity" of a point $q \in \mathcal{C}$ with respect to the junction

compute curve skeleton $\mathcal{C}$
compute robust junctions J_τ
merge all junctions $j, k \in J_\tau$ for which $\|j - k\| < D(j) + D(k)$
compute geodesicness σ on $\mathcal{C}$
cut points $P \leftarrow \emptyset$
for each junction $j \in J_\tau$ **do**
  {Find candidate cut points $Q(j)$:}
  for each branch b emanating from j **do**
    find first $p \in b$: $\sigma(p) > T_\sigma \wedge \vartheta_j(p) < T_\vartheta$, abort when $p \in J_\tau$
    follow b backward from p while $\sigma(p) > T_\sigma$
    if point p found **then**
      $Q \leftarrow Q \cup \{p\}$
    else
      find first $p \in b$: $\sigma(p) > T_\sigma$, abort when $p \in J_\tau$
      if point p found **then**
        $Q \leftarrow Q \cup \{p\}$
      end if
    end if
  end for
  {Remove invalid cut points from $Q(j)$:}
  sort points in $Q = \{q_1..q_n\}$ by descending similarity ϑ
  if $\vartheta(q_1) < T_\vartheta$ **then** {j is a 3-junction}
    {all cut points are valid}
  else if $\vartheta(q_n) > T_\vartheta$ **then** {j is a 2-junction}
    $Q = Q \setminus \arg\max_{q \in Q} \rho_{\mathcal{C}}(q)$
  else
    $Q = Q \setminus \{q_1\}$
    if $\vartheta(q_2) - \vartheta(q_3) > 0.2$ **then**
      $Q = Q \setminus \{q_2\}$
    end if
  end if
  $P = P \cup Q$
end for

Figure 6.9: Pseudocode of the branch-cut scheme.

$j \in J_\tau$ that generated q:

$$\vartheta_j(q) = 1.0 - \frac{\text{length}(\Gamma(j)) - \text{length}(\Gamma(q))}{\text{length}(\Gamma(j))} \,. \tag{6.4}$$

We clamp values of ϑ to 1.0. The basic idea is that if the similarity $\vartheta(q)$ is high enough, that is, higher than the threshold T_ϑ, candidate q should be removed. We have fixed T_ϑ to 0.6 for all shapes shown in this chapter.

By removing invalid cut points, we essentially merge those candidate limbs with the

body that are considered part of the body. In doing so, we have to make sure that we do not merge more than two limbs with the body: no part should contain a bifurcation, otherwise it is considered to consist of multiple parts. Indeed, using the ϑ-criterion we would remove all three cut points in Figure 6.8(b), because all three intersecting parts have a similar circumference. A second constraint is that limbs to be merged should really stand out from the other limbs in terms of the part cuts. Otherwise, a small perturbation of the shape might result in different limbs being merged, making the segmentation unstable.

We incorporate these considerations in our algorithm, whose pseudocode is shown in Figure 6.9. The invalid cut point removal starts at line 20. The cut points are first ordered on descending values of ϑ. Note that it is possible that junctions generate only one or two candidate cut points (e.g., junction m in Figure 6.5). To have a minimum of three ϑ-values per junction, we repeat the lowest value. Now, based on the ϑ-values, we remove zero, one, or two cut points, as follows.

First, we check whether we are dealing with a 3-junction, by seeing whether all ϑ-values are below the threshold T_ϑ (line 22). If this is the case, all cut points are valid. If this is not the case, we check if we are dealing with an obvious 2-junction, by seeing whether all ϑ-values are above the threshold T_ϑ. If this is the case, we have to remove one cut point. However, as all ϑ-values are above the threshold, they can all be considered similar and removing the point with the highest ϑ can be considered an arbitrary choice. As we cannot distinguish between cut points by means of their ϑ-values, we remove the cut point that admits the largest surface area $\rho_\mathcal{C}$ (line 25). Finally, if some of the ϑ-values are above and some are below the threshold, we might be dealing with a 2, 1, or a 0-junction and proceed as follows. In any case, the cut point that has the highest ϑ is considered invalid and should be removed (line 27). Then, we also remove the second cut point (q_2) if it stands out from the next one (q_3) by requiring that the difference between their ϑ-values is large enough (line 28). In this manner either one or two cut points are removed. Note that the case of 0-junctions are partly handled by the candidate cut point selection: a 0-junction would obtain less than three candidate cut points ($|Q| < 3$). In this manner, removing up to two cut points can result in no cut points being left, correctly indicating a 0-junction.

Figure 6.10 shows for the Hand and Noisy Dino shapes, the curve skeletons $\mathcal{C}$, junctions J_τ, the geodesicness measure σ encoded as a color map on $\mathcal{C}$, the candidate cut points, and the final part cuts.

6.4.4 Higher-order junctions

We restricted our discussion to junctions having exactly three emanating branches, but our approach also works for junctions having more than three branches, called *high-order* junctions. For these junctions of order 4 or more, the assumption of multiple limbs intersecting with a single body is still valid. However, we have to take into account the instability of higher-order junctions. Consider Figure 6.11, which shows a 4-branch junction that is not stable: an insignificant change in the shape (the non-alignment of the two vertical limbs) causes the 4-branch junction (a) to transform into two 3-branch junctions (b). Depending on the specific values of ϑ, this might affect the junction-type detection and result in a different number of final cut points, because our algorithm considers the

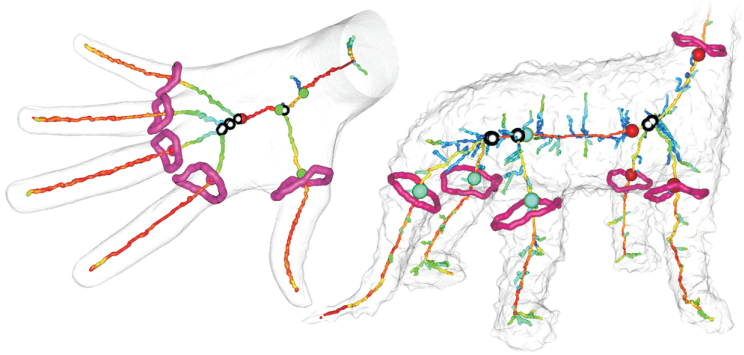


Figure 6.10: Part cuts (magenta loops) for the Hand (a) and Noisy Dino shape (b). Curve skeletons $\mathcal{C}$ with rainbow color-map encoding σ (red=1, blue=0), junctions J_τ (black balls), candidate cut points (colored balls).

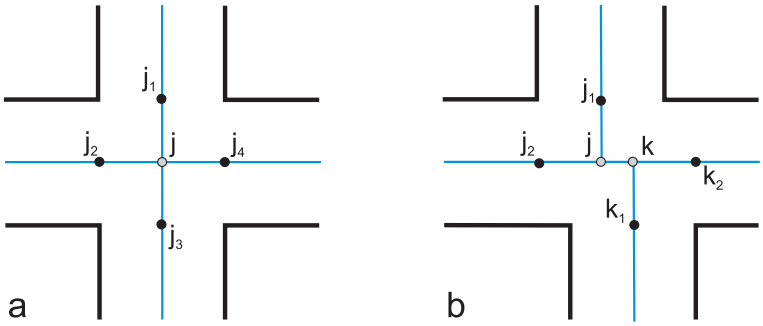


Figure 6.11: (a) A 4-branch junction j that generates four candidate cut points. (b) A similar shape produces two 3-branch junctions j and k, each generating two cut points.

merging of up to two limbs per junction with the body.

We resolve the issue by "stabilizing" the junctions before searching for candidate cut points. We do this by merging each two junctions $j, k \in J_\tau$ that are within a distance of $D(j) + D(k)$ from each other (where D is the radius of the inscribed ball). From then on, the two junctions j and k are considered as one when iterating over J_τ. Furthermore, we make sure that the $\mathcal{C}$-voxels between j and k cannot be marked as cut points. This solves the instability of the junctions with respect to small change in body/limb configurations. It also makes the occurrence of higher-order junctions much more common. An example of the result of this merging step can be seen in Figure 6.13(a), in which all the junctions in the hand are merged.

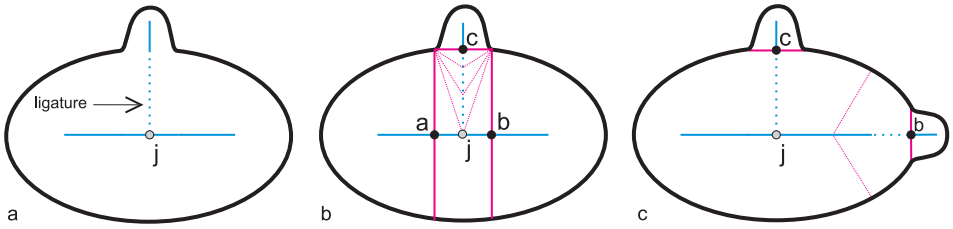


Figure 6.12: (a) Growth of a limb causes a skeleton ligature. (b) Detecting cut points by skipping ligatures. (c) Limitation of the junction rule. All figures can be seen as 2D shapes or cross sections of 3D shapes.

6.4.5 The branch-cut scheme and its relation to ligatures

In this section we try to understand the branch-cut scheme in relation to curve-skeleton ligatures [15]. In doing so, we give a limitation of our approach as we presented it thus far, and present a refinement to overcome it.

For 2D shapes, the junction rule and minima rule are related, as follows. It is well known that the minima of boundary curvature, i.e., concave corners, correspond to skeleton ligatures. Ligatures are formed by the many-to-one relations in which a small boundary segment maps to a large skeleton segment (see also Figure 2.2). August et al. [9] have related ligatures to the growing of limbs from a main body, and consider them the unstable parts of the skeleton. We observed that if ligatures arise through such a growing of a limb, they are usually connected to the skeleton of the main body by a junction, which resulted in the formulation of the junction rule. It is important to note however that not all ligatures generate a junction.

Our branch-cut scheme uses the idea that the growing of a limb causes the formation of a ligature that is connected to the rest of the skeleton by a junction. By skipping points on the curve skeleton with low geodesicness, we basically skip the ligatures. This is illustrated in Figure 6.12(a), which can be seen as a 2D shape consisting of an ellipse body with a limb that has grown from it. The ligature is indicated by the dashed line segment. In 2D, when tracing the ligature from the junction j, the ligature ends when the feature points make an angle that is approaching 180 degrees. Indeed, the feature-angle measure used in many existing pruning methods (see Section 2.6) is based on the observation that the stable, non-ligature, parts of the skeleton are formed by these near 180 degree angles. Using this criterion we find candidate cut points a, b, and c (see Figure 6.12(b)). However, point a and b do not present the growing of limbs. In 2D, we can remove these invalid cut points by the fact that their inscribed disc is of similar size to the junction's disc, whereas the disc of point c is much smaller than the other discs.

Our branch-cut scheme is analogous to the above-mentioned idea for 2D shapes. We now view the shapes in Figure 6.12 as the cross sections of their respective 3D shapes. In 3D, the Blum skeleton is replaced by the curve skeleton, and the feature points are replaced by Jordan curves. Our branch-cut scheme skips the ligatures by skipping curve-skeleton points whose Jordan curves have low geodesicness. Indeed, looking at the feature angle does not work in 3D: we have to consider all points that the part cut comprises. An

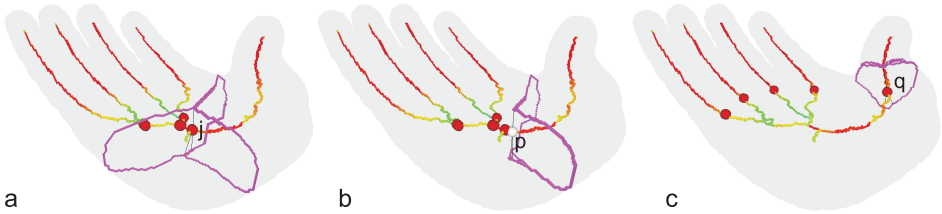


Figure 6.13: Finding the correct cut point q for the thumb. (a) Junction j and its shortest-path set. (b) First accurate cut point p for which $\sigma(p) > T_\sigma$ can be considered invalid. (c) Modification of algorithm finds first point q for which $\sigma(p) > T_\sigma \wedge \vartheta(p) < T_\vartheta$, and does make for a valid and accurate part cut (cut points for fingers also shown).

example for which it would not work is for the ligature going to the thumb of the hand shape in Figure 6.10(a): all feature points on the ligature admit a large angle. After finding the candidate cut points, we removed the invalid cut points by comparing the part cut length of the candidate with its corresponding junction, which is similar to the comparing of inscribed disc radii in 2D.

In Figure 6.12(c) we see a limitation of the junction rule. In uncommon situations, limbs may grow in such a way that the ligature lies in the extension of the skeleton. In this case, no new junction is formed, and we do not detect this part. One way to resolve this issue is to detect ligatures directly without using the junction rule. However, we do not want to do this by looking at boundary curvature, as was done by August et al. [9], because this is unstable for noisy and coarse voxel shapes. Furthermore, we face the problem of determining of how many points of the Jordan curve need to lie on a concavity for a curve-skeleton point to be flagged as ligature. We leave the idea of detecting ligatures of the curve skeleton explicitly to place cut points as future work.

Nevertheless, to alleviate this (uncommon) problem sketched in Figure 6.12(c), we introduce the following modification to our algorithm (see Figure 6.9). First, we find on a branch b the first cut point p for which $\sigma(p) > T_\sigma$ *and* for which $\vartheta(p) < T_\vartheta$ (line 9). Then, we walk back along b while $\sigma(p) > T_\sigma$ (line 10). If we find no point for which $\sigma(p) > T_\sigma$ and $\vartheta(p) < T_\vartheta$, then we resort to the old method of taking the first point for which $\sigma(p) > T_\sigma$ (line 14).

The above modification ensures for example the correct segmentation of the Hand shape in Figure 6.13. When looking at the branch associated with the thumb emanating from junction j, we see that the very first point p satisfies the high geodesicness criterion (see Figure 6.13(b)). As this point has a high similarity ϑ_j, the candidate cut point is considered invalid and will be removed subsequently, incorrectly merging the thumb with the palm. Using the presented modification, when searching for the first point that satisfies both $\sigma > T_\sigma$ and $\vartheta < T_\vartheta$, we find cut point q, which correctly splits off the thumb (Figure 6.13(c)).

6.4.6 Algorithm details

We now discuss some non-essential details of the algorithm.

As explained in the previous sections, we search along branches emanating from junctions to look for candidate cut points. We stated in Section 4.7.1 that the curve skeleton can be up to two voxels thick. Walking along the thick curve skeleton is not straightforward as each voxel has more than two adjacent voxels. Although it is possible to apply a thinning operation first, we proceed differently. When we walk along the branches in $\mathcal{C}_\tau$ emanating from a junction $j \in J_\tau$, we perform a Dijkstra distance propagation on the curve-skeleton voxels until we reach another junction or the endpoint of a branch. For each $\mathcal{C}$-endpoint or other junction, we trace back to j so that we obtain a path on each branch that emanates from j.

After determining valid cut points, it might be that two cut points, although they come from different junctions, basically represent the same part cut. In a segmentation, these two cut points might create a thin "sliver" segment. We detect this situation by checking for each two cut points that come from different junctions whether their shortest-path sets Γ touch. If they do, even if by one voxel, we remove one of the two cut points. Without this modification, a sliver segment would for example be generated between the ant's head and body in Figure 6.19.

6.5 Segmentation

In the previous section we discussed the selection of cut points. This section details on how we obtain either a flat or hierarchical shape-segmentation from the cut points, selected using the junction-cut scheme, the branch-cut scheme, or manually.

The most straightforward segmentation approach is the *flat segmentation*. The shortest-path sets Γ (Eq. 4.3) of the cut points are used directly as part cuts, to divide the shape boundary into connected components. The segmentation is called flat because it is non-hierarchical: it can be considered the finest level in the hierarchy. In order to produce a *hierarchical segmentation*, we consider the component sets C of the cut points. We can distinguish between foreground and background components. In a component set C consisting of k components $C_{1..k}$, the largest component C_k is called the background component, the remaining ones are called foreground components. The foreground components are those that one would consider meaningful and intuitively associate with the cut point. The background component is merely the remaining surface. In Figure 4.5(f) for example, the background components are the blue components. Furthermore, we can also consider the combined foreground components as a meaningful component. We denote this compound component by $C'(p) = \cup_{1 \leq i < k} C_i(p)$.

Let FG be the set of foreground components of all cut points combined. The segmentation should be based on FG, but the components in FG are not disjoint. We now present an algorithm for creating a flat segmentation from FG consisting of disjoint segments, at a certain scale τ. By creating segmentations for all the available scales, a hierarchy of segmentations is obtained. The pseudocode of the hierarchical segmentation is shown in Figure 6.14.

To create a segmentation at scale τ we consider all components $f \in FG$ in ascending order of their area (line 5), but only those components that are larger than the specified scale τ (line 6). The potential segment s is computed as the set difference between f

```
1: FG ← {C_i(p) | p is a cut point ∧ 1 ≤ i < k} ∪ {C'(p) | p is a cut point}
2: FG ← FG ∪ {∂Ω}
3: procedure computeSegmentationAtScale(τ)
4: S ← ∅
5: for each f ∈ FG in ascending order of area(f) do
6:    if 1/area(∂Ω) area(f) ≥ τ then
7:       s ← f \ S
8:       if f ∩ S ≠ ∅ ⇒ area(s) > 0.1 · area(f ∩ S) then
9:          S ← S ∪ s
10:         label s as a segment at hierarchy level τ
11:      end if
12:   end if
13: end for
14: end procedure
15: procedure computeHierarchy()
16: for each f_i, f_{i+1} in FG in ascending order of area(f_i) do
17:   if area(f_{i+1}) − area(f_i) > 0.1 · area(f_i) then
18:      computeS(1/area(∂Ω) area(f_i))
19:   end if
20: end for
21: end procedure
```

Figure 6.14: Pseudocode for computing an hierarchical segmentation.

and the voxels that the existing segments occupy, denoted S: $s = f \setminus S$ (line 7). Before labeling the potential segment s, we check whether f overlaps any existing segments. If not, s is added without restriction. If f overlaps, we only add s if it contributes enough to the segmentation, that is, if it adds at least 10% of the area that it overlaps (line 8). This is to prevent tiny segments due to different junctions having similar components among their component sets. This occurs for example when using the junction-cut approach, due to the fact that junctions computed by the algorithm in Section 4.5 may consist of multiple voxels, having almost the same component sets. After processing all components in FG, the object surface might not be fully covered, because the background components have been left out in the segmentation. Therefore, we add to FG the whole boundary as the largest component (line 2), which ends up as a single segment filling up the remaining part.

In order to compute a hierarchical segmentation consisting of fine to coarse levels (line 15), we simply consider all components from FG in ascending order of area, and produce a segmentation for each of those areas. To limit the number of generated hierarchy levels, in our implementation we only compute a hierarchy if two consecutive areas differ by at least 10% from the smaller area f_i (line 17). The different segmentations produced at the various scales actually form a hierarchy, because every segment is included in a segment from a coarser scale, due to the inclusiveness of the component sets (Eq. 4.6).

It is important to note that the flat segmentation may differ from the finest level in

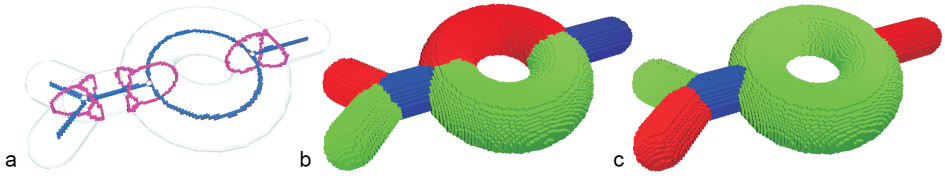


Figure 6.15: (a) Junction cut points and their shortest path-sets. (b) Flat segmentation. (c) Finest level of the hierarchical segmentation.

the hierarchical segmentation. Conceptually, the difference between the two approaches is that within the flat segmentation, the connected components are determined for all the part cuts at once, whereas for the hierarchical segmentation, the connected components are computed per cut point, and are later combined. This might result in different segmentations in the case of shapes with tunnels. This is illustrated in Figure 6.15. Whereas the flat segmentation splits the tunnel into two, the hierarchical segmentation leaves the tunnel intact.

6.6 Results and discussion

We implemented our framework in C++ and ran it on a Pentium 4, 3GHZ with 1GB of RAM. As input we used several polygonal meshes voxelized using binvox [84], for various resolutions ranging up to 384^3 voxels. We used shapes from the McGill Shape Benchmark [145], the AIM@SHAPE Database [2], and the Princeton Shape Benchmark [124]. We used both organic (having smooth edges) and geometric shapes (having sharp edges). All segmentations are shown with a minimum coloring scheme applied and smooth iso-surfaces have been computed from the voxel output and are rendered using a raytracer [99] for clearer display.

Figures 6.16, 6.17, and 6.18 all show junction-cut segmentations. Figure 6.16 shows for four objects the simplified curve skeleton $\mathcal{C}_\tau$ and robust junctions J_τ, both with $\tau = 0.01$, and three selected levels from the hierarchical segmentation. Figure 6.17 shows the segmentations of the Tree object for all levels. Figure 6.18 shows for several objects the segmentation at the finest scale.

Results of the branch-cut scheme are shown in Figure 6.19 using a flat segmentation. Most segmentations here can be colored using only two colors, indicating that part cuts usually do not touch (and never intersect).

6.6.1 Evaluation

We evaluate our framework using the criteria proposed by Attene et al. [8].

The *correctness* of part-type segmentations is inherently subjective [8]. Our method does not target a particular application, but instead claims that it finds logical parts. We assess the correctness of our method visually. As can be seen from the examples in Figures 6.18 and 6.19, our framework handles organic, geometric, noisy shapes, and shapes

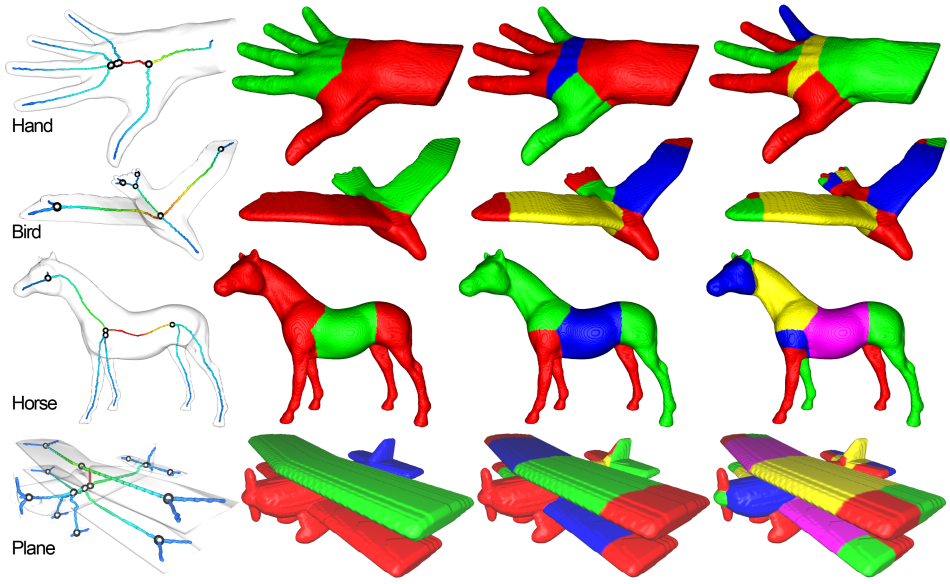


Figure 6.16: Simplified curve skeletons $\mathcal{C}_\tau$ with detected junctions (first column), and three levels of the junction-cut hierarchical segmentation (other columns).

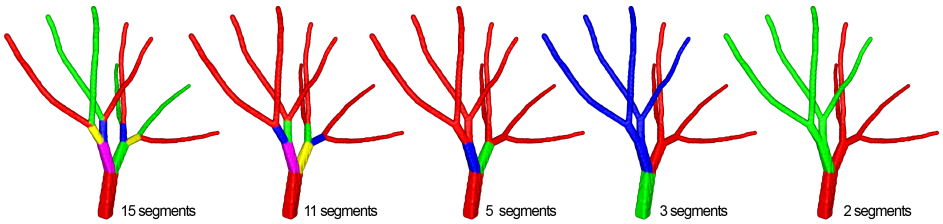


Figure 6.17: All levels of the junction-cut hierarchical segmentation for a tree shape.

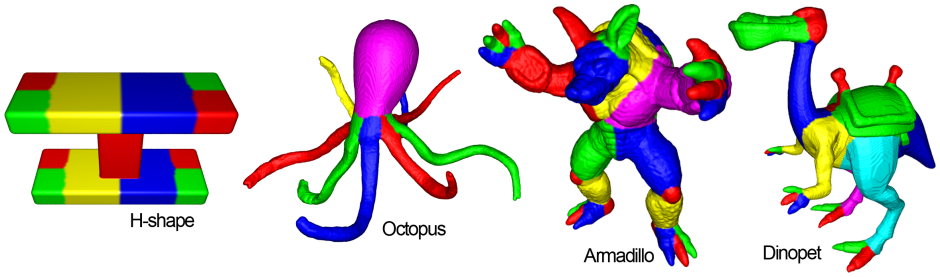


Figure 6.18: Junction-cut segmentations.

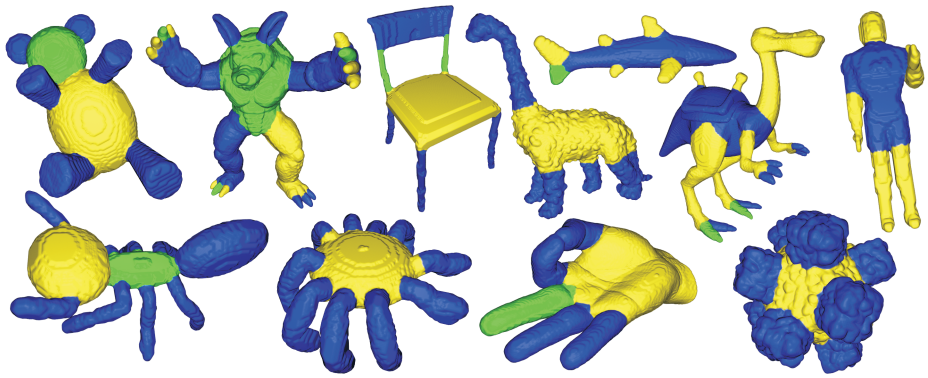


Figure 6.19: Branch-cut segmentations.

with tunnels. We further observe that our segmentation framework is able to extract fine details, such as the toes and fingers of the Armadillo and Dinopet objects in Figure 6.19, provided the grid resolution is high enough and the curve skeleton has not been simplified too much by the user.

As expected, the branch-cut scheme produces better results than the junction-cut one. The latter produces exactly three parts for each junction, whereas the former produces between 0 to 4 segments per junction, distinguishing between junction types. Also, some junctions do not signal the interpenetration of parts at all. Note that this does not invalidate the junction rule: the junction rule says that an interpenetration implies a junction, not vice versa. The saliency of the part cuts is guaranteed, because the simplified curve skeleton $\mathcal{C}_\tau$ contains only significant branches, guaranteed by the scale parameter τ.

As can be seen in Figure 6.19, the branch-cut segmentations all agree with intuition and do not suffer from over-segmentation. They do not suffer from severe under-segmentation in the sense that there are no parts that contain bifurcations. However, some shapes could be considered under-segmented, such as the dinopet, for which the head and neck are both assigned to the same segment.

The *part cuts* that our method produces have several desirable properties. They are found directly as the Jordan curves associated with the cut points. They are *closed* by default (unlike e.g. [69]), and no refinement of part cuts is necessary (unlike e.g. [61]), keeping the algorithm straightforward to implement and intuitive to use. Furthermore, the part cuts are inherently *smooth*, as the Jordan curves are piecewise geodesic by construction. The branch-cut scheme produces even smoother cuts, as the cut points are placed at points with a high total geodesicness, measured by our geodesicness measure σ.

The framework is able to produce *hierarchical* segmentations, in which each segment is fully included in another segment at a coarser scale, as shown in Figure 6.16. The framework can also be considered *multiscale* through the use of the user parameter τ [115], which controls the simplification of the curve skeleton. The user parameter is intuitive as it thresholds the importance measure $\rho_\mathcal{C}$ which has a geometric meaning: it assigns to a point the smaller surface area induced by the Jordan curve at the point.

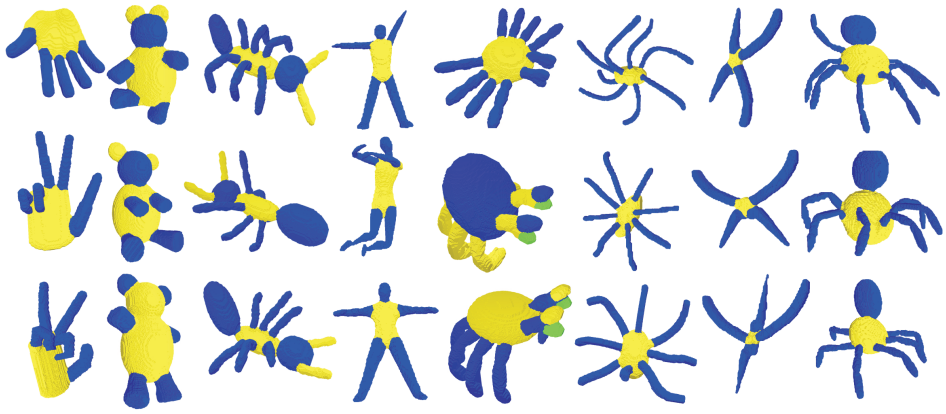


Figure 6.20: Pose-invariance of the branch-cut segmentations.

Following from the multiscale property is the *robustness to noise*. This robustness is guaranteed by the robust junction detection, and by the robustness of the geodesicness measure σ. Figures 6.10(b) and 6.19 show that the Dinosaur shape is correctly segmented under presence of boundary noise. Another example is given by the noisy Sea mine in the bottom right in Figure 6.19. Segmentation methods using surface curvature, e.g., those based on the minima rule, might have problems here, as the curvatures change considerably under these boundary perturbations.

The segmentations are *pose-invariant*. That is, the number and location of segments do not change under different poses of the same shape. The reason is that the structure of the underlying curve skeleton does not change significantly under deformations of the shape. The branch-cut scheme is more pose-invariant than the junction-cut scheme: even if a junction is introduced by a deformation, it will not immediately result in extra part cuts because the geodesicness measure σ on this new $\mathcal{C}$-branch has a low value. Only if the new limb part really protrudes, σ reaches a high enough value.

We evaluated the pose-invariance of our approach on the McGill Shape Benchmark [145]. The shapes in this database are divided into two groups. The first group contains 10 classes of articulated shapes: ants, crabs, hands, humans, octopuses, pliers, snakes, spectacles, spiders, and teddy bears. Each class consists of up to 30 poses of the same shape. In contrast, the second group contains classes containing different shapes and may vary in their number of parts. To test the pose-invariance, we ran our method on the first group. Figure 6.20 shows three selected shapes from each of the 8 interesting classes (the snakes and spectacles all consist of one part). We observe that pose-invariance is slightly violated for the teddy and crab classes. For some of the teddy shapes, the head and body are merged, because the neck is too thick in comparison with the head, and does not form a contraction. For the crab class, some of the scissors are split into three segments, whereas others are not, depending on how protruding they are. Nevertheless, the pose-invariance can be considered satisfying overall.

The *asymptotic complexity* of our method is dominated by the curve skeleton com-

Table 6.1: Table with measurements. Times in seconds. See the text for details.

object	dim	$\|\Omega\|$	$\|\partial\Omega\|$	init	$\mathcal{C}$	hier.	branch	ρ
Armadillo	188x245x207	905k	80k	23	8.6	41	83	7.5
Dinopet	334x366x180	1,810k	136k	49	15	54	90	40
Hand	366x154x257	1,300k	94k	36	30	21	40	5.6
Horse	366x316x171	2,038k	119k	48	25	34	63	10
Noisydino	125x346x365	1,421k	114k	41	16	19	69	14
Plane	217x304x98	545k	110k	20	53	21	58	39
Octopus	366x259x335	1,860k	154k	53	14	9.5	40	11

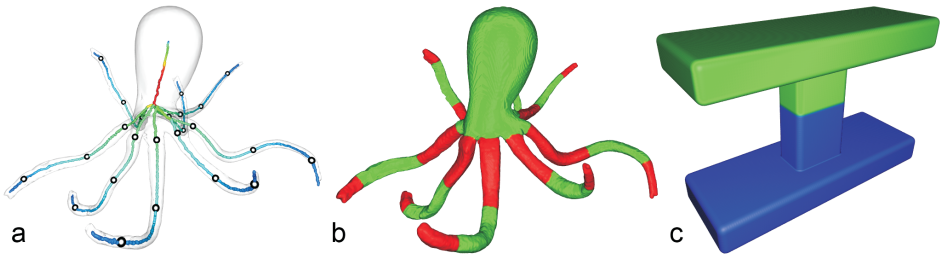


Figure 6.21: (a) Manually selected cut points, and (b) resulting segmentation. (c) Segmentation in two halves.

putation. Due to the optimization presented in Section 6.3.1 we do not have to process all object voxels, as we did in Chapter 4 in order to compute the full surface skeleton. Table 6.1 shows performance measurements on our framework for several objects that are shown in this chapter. Columns "object", "dim", "$|\Omega|$", "$|\partial\Omega|$" denote object name, voxel-grid dimensions, number of object voxels, and number of boundary voxels respectively. Column "init" denotes the time needed for initialization, including loading the object and computing the spatial-subdivision datastructure needed for efficient computation of component sets (Section 4.5.4). Column "$\mathcal{C}$" denotes the time in seconds to compute the non-simplified curve skeleton $\mathcal{C}$ using the speed-up from Section 6.3.1. Column "hier." denotes the time for computing all levels in the hierarchical junction-cut segmentation. This time strongly depends on the number of levels generated, and is non-optimized. Column "branch" denotes the time for computing a branch-cut segmentation. Column "ρ" denotes the time required to compute ρ on $\mathcal{C}$, that is, to obtain simplified curve skeletons. Note that ρ is not needed for computing the segmentation: only the curve skeleton $\mathcal{C}$ and robust junctions J_τ are needed. The time needed to compute the robust junctions J_τ is not shown as it is negligible: up to 5 seconds for the considered objects.

Only a single *control parameter* is needed: the curve-skeleton simplification threshold τ, which we discussed in Chapter 4. The branch-cut point placement has two internal parameters: T_σ for the minimum geodesicness of a cut point and T_ϑ for distinguishing between body and limbs. We have experimentally determined appropriate settings for these parameters and kept these fixed for the shapes shown in this chapter.

As mentioned, our segmentation framework also supports manual selection of cut

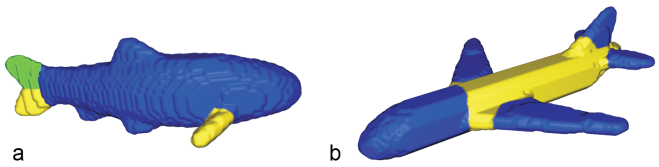


Figure 6.22: Some limitations of the branch-cut segmentation. (a) Some of the fins are not segmented because they are too tapered. (b) Part cuts are not as smooth as they could be for the wings.

points. In Figure 6.21(a), we manually selected three such points on each tentacle of the Octopus. The resulting segmentation containing three segments per tentacle is shown in Figure 6.21(b). Another interesting possibility is to pick the root of the curve skeleton as a cut point. The root R is defined as that curve-skeleton point for which ρ reaches its maximum (Section 4.3). In case R is a non-junction point, $\rho(R)$ is half the object-surface area. A segmentation based on the root thus divides the object in two in a natural manner, as exemplified in Figure 6.21(c) for the H-shape. In future work, this special case could be easily extended to segmenting the object in n equally sized segments, where n is a user parameter.

Finally, we discuss limitations of our approach. By using the curve skeleton and associated Jordan curves, each presenting a potential part cut, we have essentially reduced the problem of segmentation from 2D to 1D. However, this also causes some limitations. In case a part is strongly tapered, we may obtain a low σ all along the respective $\mathcal{C}$-branch, and we might not find a cut point. In practice, if a part is indeed so strongly tapered that σ does not become higher than our fixed threshold for the *whole* $\mathcal{C}$-branch, then it must be a small part, and failing to segment it is not a serious problem. This is illustrated in Figure 6.22(a): the less protruding fins of the fish are not segmented. A second limitation is that we have to choose part cuts from a limited set such that they are smooth, but beyond that we cannot enforce complete smoothness. As a result, the part cuts for the wings of the plane in Figure 6.22(b) are sub-optimal. Another limitation is demonstrated by the plane: its rump is divided into two parts, which is undesirable from a semantic point of view. It might be possible to resolve this issue by including additional criteria, such as visual conductance, which we leave for future work.

Although our method works on voxel shapes, we argue that it can be adapted to polyhedral shapes as well. Computation of the curve skeleton and associated Jordan curves is available for meshes [36]. However, because the implementation is non-trivial and not available to us, we have chosen for our own voxel-based implementation. The other two steps, computing the geodesicness measure and placing the part cuts, are also not inherently voxel-based.

6.6.2 Comparison

We now briefly compare the results of our method to several other state-of-the-art shape segmentation methods by visual inspection of the results. Figure 6.23 shows the Homer

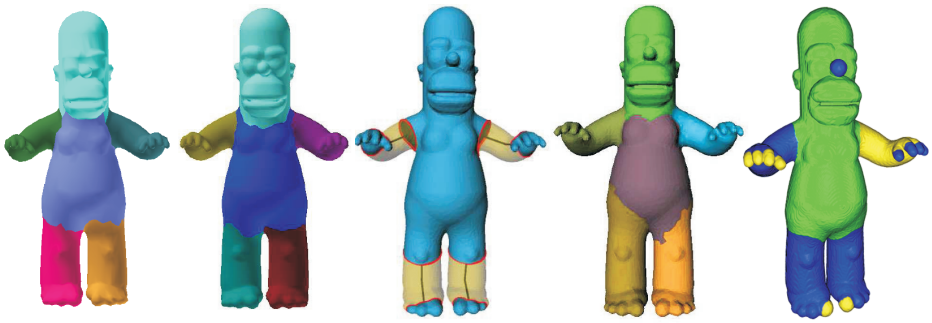


Figure 6.23: Visual comparison of segmentations for the Homer shape. Methods from left to right: [61], [60], [89], [7], and our branch-cut segmentation.

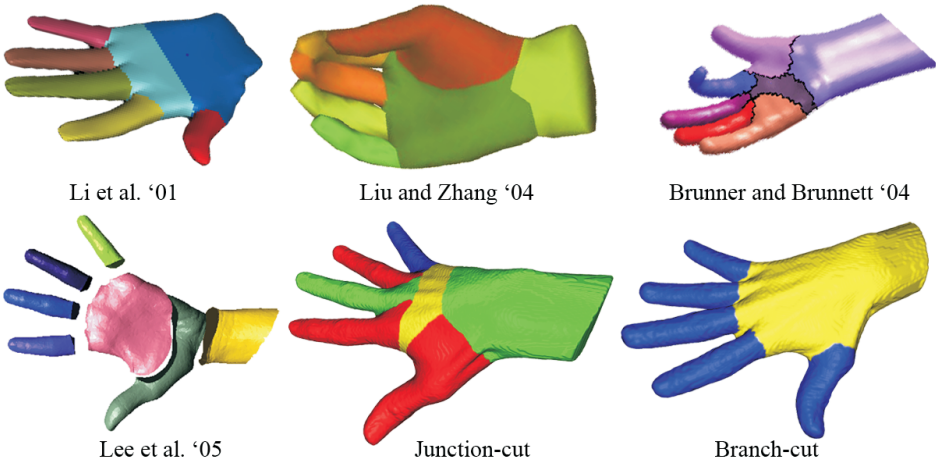


Figure 6.24: Visual comparison of segmentations for the Hand shape. Methods from left to right: [72], [75], [19], [69], our junction-cut segmentation, and our branch-cut segmentation.

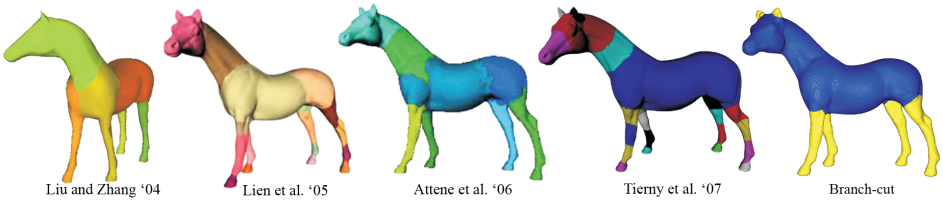


Figure 6.25: Visual comparison of segmentations for the Horse shape. Methods from left to right: [75], [73], [7], [139], and our branch-cut segmentation.

shape segmented using four existing methods, named Katz and Tal '03 [61], Katz et al. '05 [60], Plumber [89], and HFP [7] (images from [8]) and our own branch-cut approach as the rightmost image. We notice that our method captures fine features as the fingers and the nose, whereas the other methods do not. Our method segments only the two largest toes, as the other toes are not separated from each other by the exterior volume. We further observe that our method, like [89], does not label the head as a separate segment. The reason for this is that the neck does not have a (significantly) smaller circumference than the body. Hence, our method considers the body and head belonging to the same part. Finally, our segment borders for the legs are tight and smooth, whereas the other methods can be seen to have some problems for these areas.

In Figure 6.24 we compare our junction-cut and branch-cut segmentations for the Hand shape with yet four other methods: Li et al. [72], Liu and Zhang [75], Brunner and Brunnett [19], and Lee et al. [69] (images from the respective papers). Two of these methods [72, 19] also use the curve skeleton. We observe that our branch-cut segmentation tightly splits off the fingers and thumb, whereas our junction-cut method and the other two skeleton-based methods do not. The reason for this is that these methods incorrectly assume that each junction indicates three intersecting parts. The other skeleton-based methods and the minima-rule based method of Lee et al. [69] all generate a part cut on the palm itself, whereas our branch-cut approach keeps the entire palm within one segment.

Finally, Figure 6.25 compares results on the Horse shape.

6.7 Conclusion

In this chapter, we presented a part-type segmentation framework based on our simplified curve skeletons. Hereto we introduced the junction rule, which is the notion that the curve skeleton can be used to infer the shape's part structure by means of its junctions. We argue that the interior-based junction rule has as advantage over the boundary-based minima-rule that it leads to more robust results, especially for shapes with noisy boundaries and for voxel shapes. Due to the simplification of the curve skeleton, only salient parts are found. Furthermore, the junction rule avoids some of the subtleties that the minima rule has.

Part cuts are found by detecting cut points on the curve skeleton according to the junction rule. We have presented two automatic cut point selection schemes. The junction-cut scheme is most straightforward as it selects the junctions of the curve skeleton directly. The branch-cut scheme remedies over-segmentation by differentiating between junction types, and improves the tightness and smoothness of the part cuts by using our novel geodesicness measure. Final part cuts are selected by considering the body/limb relationships of the intersecting parts. Our framework is conceptually straightforward, and the resulting segmentations have several desirable properties, such as smooth part cuts and pose-invariance.

Chapter 7

Patch-type Shape Segmentation

In the previous chapter we used the simplified curve skeleton to extract a shape's high-level part structure (Figure 6.1(a)). In this chapter, we use the simplified surface skeleton to create patch-type segmentations. Patches are quasi-flat areas separated from each other by relatively high-curvature creases. We first define a skeleton-based surface-classifier on the shape boundary that separates convex creases from quasi-flat areas. By using this classifier on both the skeleton of the shape interior and the skeleton of the shape exterior, we find convex and concave creases that combined divide the shape into patch-type segments (Figure 6.1(b)).

7.1 Introduction

Detecting features such as ridges and valleys in datasets such as 2D grayscale images and 3D CT and MRI volumetric scans, is an important and active area of research. Features such as edges and corners must be classified in a robust way in order to enable further analyses on such datasets, like edge-preserving denoising or robust partitioning of areas bounded by such edges. The so-called local classification of surfaces, in particular the separation of highly curved ridges from low curvature areas, is also an important prerequisite in numerous surface processing applications such as surface matching and feature-preserving simplification.

Traditionally, most surface classifiers used in practice employ one or another variation of ridge detection based on higher-order surface derivatives, such as gradients, curvature, or moments. Although a wealth of such methods exist, curvature estimation on noisy voxel surfaces is an inherently delicate process. Many such methods trade off the precision of ridge detection for stability, by using different types of filtering over (small) neighborhoods.

As indicated in Chapter 2, curvature is intimately connected to skeletons. This relation is explicitly expressed in the symmetry-curvature duality theorem [71], which states that skeleton branches terminate at convex curvature extrema, and can also be seen in the correspondence of skeleton ligatures to concave corners [15, 9]. The duality has not often

been used for ridge detection or surface classification on 3D shapes, which we believe is due to the infamously unstable nature of 3D skeletons, the difficulty of computing them, and the lack of an appropriate scale notion.

This chapter consists of two parts. First, we use our simplified surface skeletons to define a multiscale surface classifier that separates convex ridges or edges from smooth regions. We illustrate our method on the classification of anatomical surfaces, with a focus on cortical surfaces, that is, surfaces of brains, and compare our method with a classical curvature-based classifier. Second, we use our classifier to detect both convex and concave edges, to come to a patch-type segmentation of the shape. By computing the surface skeleton of the shape exterior, concave edges can also be detected. The convex and concave edges combined partition the boundary into disjoint segments, which present a patch-type segmentation of the shape.

This chapter is structured as follows. In Section 7.2 we overview related work in the area of surface classification and patch-type segmentation. In Section 7.3 we present our skeleton-based classifier and its results. In Section 7.4 we present the related patch-type segmentation method. In Section 7.5 we present and discuss the segmentation results. Section 7.6 concludes the chapter.

7.2 Related work

We first discuss related work on surface classifiers, with an emphasis on cortical classification. Then, we discuss existing work in patch-type segmentation.

7.2.1 Cortical surface classification

The canonical quantity for edge detection on surfaces is the curvature tensor. Different methods exist for its evaluation on discrete surfaces, such as shown by Moreton and Séquin [87], Clarenz et al. [21], and Desbrun et al. [34]. Besides curvature, surface classifiers can be based on related integral quantities, such as moments [22]. Globally speaking, all such methods use a local surface classification, and thereby trade edge detection accuracy for stability via some built-in smoothing.

Extracting cortical surface features, such as sulci and gyri, from MR brain volumes is focus of extensive work. Such features are used in studies of inter-subject gyral and sulcal variability [47] or to identify structural and functional patterns in Alzheimer patients [138].

Many methods for sulci extraction use the surface's (mean) curvature. Sulcal fundi are defined as crest lines of extremal curvature. Similar approaches can be used for gyral structures. Such methods are semi-automatic, requiring the user to define two or more points on a sulcus, which are then connected by optimizing a curvature-based cost. Several such approaches exist, using weighted geodesics [11], dynamic programming [62], fast marching [83] and 3D curve-tracking [105]. Evaluating curvature extrema involves higher-order derivatives, so these methods can be quite unstable on highly convoluted cortical surfaces coming as limited resolution voxel scans. Cachia et al. [20] alleviate such

problems using a scale-space of the underlying curvature signal, thereby trading precision for stability.

To overcome stability problems, other methods find sulcal fundi by locally maximizing the distance from the cortical surface to a bounding hull around it. The methods of Goualher et al. [47] and Lohmann [76] find fundi as the deepest boundaries of surfaces obtained by subtracting the white and gray matter from the bounding hull. Combining curvature and distance-based criteria leads to more stable, but significantly more complex to implement, methods [57, 58, 134, 133].

7.2.2 Patch-type segmentation

We now briefly overview some of the patch-type segmentation methods that relate to our approach. Garland et al. [44] perform a hierarchical clustering of the mesh faces to produce a patch-type segmentation consisting of planar segments. Clarenz et al. [22] perform a fuzzy multiscale segmentation of a 3D shape, based on surface curvature. However, this method often generates noisy edges in low-curvature regions. Mangan and Whitaker [81] partition a surface into patches of similar curvature using a watershed algorithm that uses curvature as its height function. Zuckerberger et al. [147] give an improved watershed method and give numerous segmentation applications. Provot et al. [101] segment voxel shapes by detecting discrete planes with variable width.

Although uncommon, using skeletons to detect features is not new. Hisada et al. [52] use the skeleton in combination with denoising and filtering techniques to detect salient shape features. However, their technique works on polyhedral shapes and they do not produce patch-type segmentations. Related to our approach is the *local feature size* [3], which is defined as the Euclidean distance from a boundary point to the skeleton. In contrast, our classifier (Section 7.3) is quasi-global as it integrates information from a large part of the boundary, by using shortest paths on the boundary, and as such can distinguish between locally similar, but globally different boundary configurations.

All in all, existing patch-type segmentation methods for voxel shapes cannot handle both boundary noise and soft edges, something that we address in this chapter.

7.3 Skeleton-based surface classifier

To define the surface classifier we only consider the surface skeleton $\mathcal{S}$ and do not consider the curve skeleton $\mathcal{C}$. Recall that we compute the simplified surface skeleton, which we denote $\mathcal{S}_\tau$, by applying a simple thresholding strategy to the importance measure $\rho_{\mathcal{S}}$. This measure is computed on each skeleton point by taking the maximum length of the shortest paths between the feature points. In the rest of this chapter, the word "skeleton" refers to the surface skeleton.

Key to our approach is an extension of the symmetry-curvature duality theorem [71] to 3D, which states that the skeleton terminates at maxima of convex curvature. In 2D, this means that the skeleton terminates at convex corners, while in 3D, this means that skeleton sheets terminate at convex edges. Whereas for sharp edges the skeleton terminates exactly at the edges, for smooth edges the skeleton terminates at some distance from the edge (see

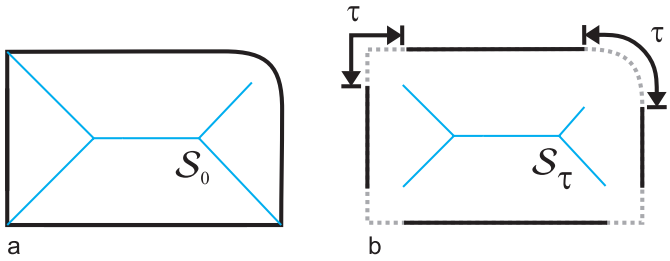


Figure 7.1: (a) Skeleton endpoints terminate at curvature maxima, i.e., corners. For smooth corners, they terminate at some distance from the boundary. (b) Simplified skeletons generate gaps (dashed curves) of width τ in the feature collection (thick curves).

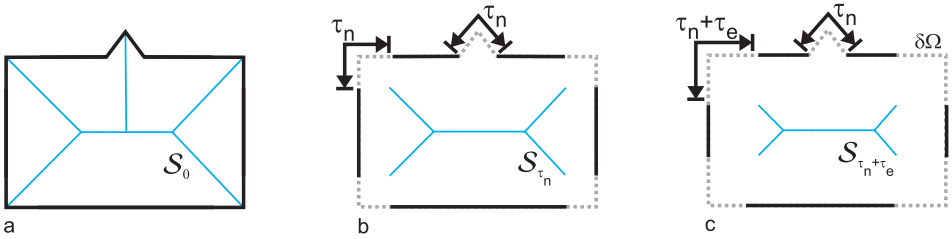


Figure 7.2: (a) Non-simplified skeleton. (b) Simplified skeleton at scale τ_n. (c) Simplified skeleton at scale $\tau_n + \tau_e$. Thick lines are feature collections V.

Figure 7.1(a)). Normally, it is problematic to use this duality to detect edges as skeletons are unstable due to boundary noise and as such may contain spurious parts that terminate at insignificant edges. We address this issue by using our simplified surface skeletons.

The key idea of our approach is simple: by increasing the simplification level τ, we prune the skeleton $\mathcal{S}_\tau$ near its rims first, because $\rho_\mathcal{S}$ is lowest near the rims. Because the skeleton reaches into the edges of the surface, we remove the skeleton parts whose feature points lie on and near these edges. In this fashion, we can detect edges by the absence of feature points. By using the skeleton and associated feature points, suitably pruned to remove noise effects, we can robustly detect such edges, without any discrete curvature computation.

Let V be the set of feature points corresponding to the simplified skeleton $\mathcal{S}_\tau$. We call this set the *feature collection* of $\mathcal{S}_\tau$:

$$V(\mathcal{S}_\tau) = \bigcup_{p \in \mathcal{S}_\tau} \overline{F}_\tau(p) \; . \tag{7.1}$$

We use here the simplified feature set $\overline{F}_\tau$ (see Section 5.3), so that we always obtain precisely two feature points in $\overline{F}_\tau$, even for skeleton rim points that have a contiguous set of feature points. This is important, as we would otherwise not detect a gap in the feature collection for these rim points.

By increasing the threshold τ on the importance measure $\rho_\mathcal{S}$, gaps will appear in the

feature collection V on and near convex edges of the shape surface. We detect such gaps and use them to detect the edges. However, one complication is that the parameter τ is also used to prune spurious skeleton parts that are due to boundary noise or discretization artifacts. Setting τ to the noise level τ_n opens V on the edges, but also on noisy parts, which we do not want to detect as edges. Therefore, we have to increase τ further to $\tau_n + \tau_e$: the feature collection V is opened further on edges, but not on boundary noise. This is illustrated in Figure 7.2, for the sake of clarity in 2D. In Figure 7.2(a), the non-simplified skeleton $\mathcal{S}_0$ of a box with a small-scale noise bump is shown. The feature collection (thick lines) covers the whole boundary. When τ is set to the noise level τ_n (Figure 7.2(b)), the openings in V on the convex bump and near the non-noisy convex corners have the same size, so that we cannot differentiate between the two situations. By further increasing τ to $\tau_n + \tau_e$ (Figure 7.2(c)), V is further opened on the corners, but not on the bump. The term τ_e controls the minimum detected edge width and should be chosen as small as possible, to ensure thin edges, but at the same time large enough to account for the small inaccuracies in the feature point locations caused by the discretization. We verified that a conservative setting of $\tau_e = 4$ gives good results on a wide range of objects, including 3D brain surfaces but also several other 3D synthetic and organic shapes as shown in Section 7.5.

Thus, our classifier $\mathcal{D}$ on the boundary surface is defined as the geodesic distance to the nearest point in the feature collection V:

$$\mathcal{D}(p \in \partial\Omega) = \min_{v \in V} \text{length}(\gamma(p, v)) \,, \tag{7.2}$$

where $\gamma(a, b)$ is the shortest geodesic between a, b on the object boundary $\partial\Omega$ computed as in Chapter 4, by using Dijkstra's algorithm on the boundary graph. Points at a distance of at least $\tau_n/2$ from a feature point in V are considered edge points E:

$$E(\Omega) = \Big\{ q \in \partial\Omega \;\Big|\; \mathcal{D}(q) > \tau_n/2 \Big\}. \tag{7.3}$$

The edge-width parameter τ_e controls the minimum width of the detected edges, but not the maximum. In case of smooth, round parts of the shape, the openings in V and thus the edges might become thicker than τ_e. The importance measure $\rho_{\mathcal{S}}$ varies quickly for the skeleton representing the round part, so that in the discrete case, V may be opened at the round part by a slight increase of τ. This is as expected, since it is not possible to specify the exact location of an edge over smooth parts having no curvature variation. Note that in the continuous case, there would not be a gap in the feature collection at all on the curved part, because the skeleton endpoint has the half-circle as feature points. However, as we use the simplified feature set $\overline{F}_\tau$, we do not have this problem in the discrete case, as only the two feature points are kept that are furthest apart.

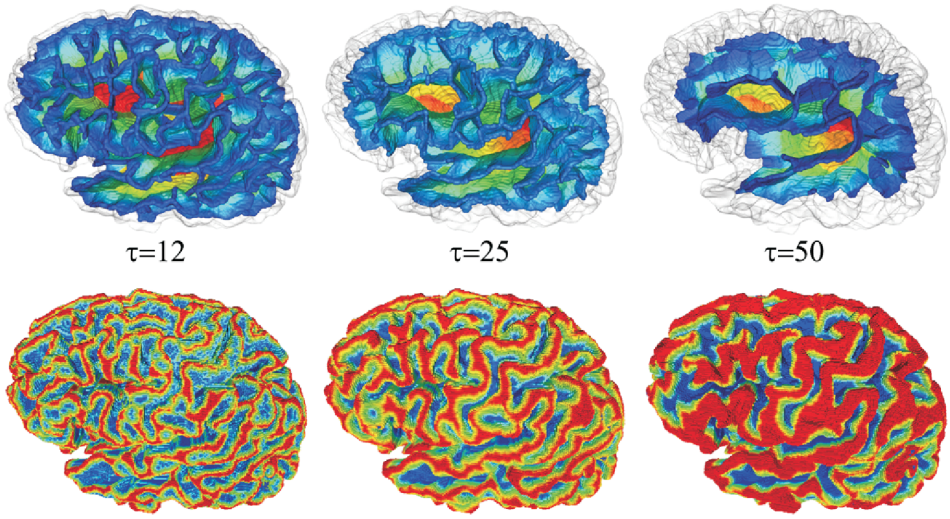


Figure 7.3: Brain surface classification for different skeleton simplification levels τ. Top: simplified skeletons. Bottom: the corresponding surface classifier $\mathcal{D}$.

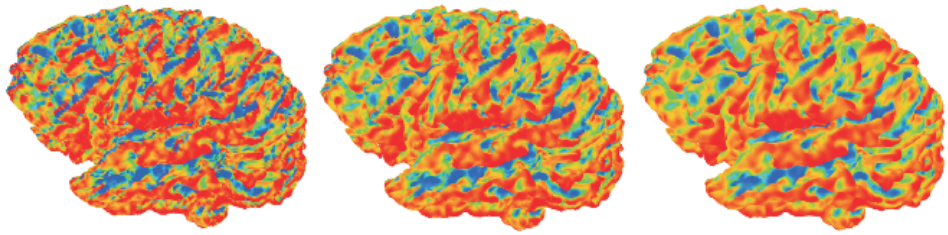


Figure 7.4: Brain surface classification using a smoothed curvature estimator (blue=low curvature, red=high curvature), with increasing diffusion times.

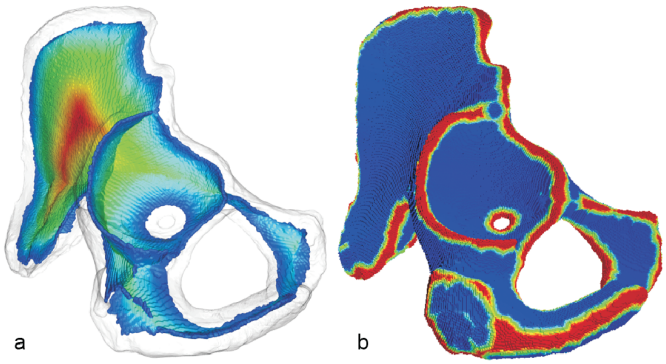


Figure 7.5: Surface classification of the hip dataset

7.3.1 Results

We illustrate our classifier $\mathcal{D}$ on a brain cortex surface, computed from an MRI scan of 256^3 voxels resolution. Figure 7.4 shows the curvature-based classifier proposed by Taubin [135] applied to the brain surface. To reduce small-scale noise artifacts we apply the diffusion (heat) equation geodesically to the cortical surface. To take into account the fact that the underlying grid is irregular, we use the approach of Desbrun et al. [33], equivalent to anisotropic diffusion on regular grids. The three different images in Figure 7.4 correspond to increasingly longer diffusion times t, which are equivalent to increasingly larger Gaussian filters [22].

Figure 7.3 shows our skeleton-based classifier, computed using three progressively simplified skeletons. The simplification levels τ are chosen so that they match the sizes of the Gaussian filters that describe the curvature smoothing in Figure 7.4. We notice several things. First, the simplification level τ for the skeleton-based classifier has a very similar effect as the Gaussian filtering, or smoothing time, for the curvature classifier: small values yield sharper (but potentially noisier) edges, larger values yield smoother, but thicker, edges. However, we also see that the skeleton-based classifier separates the convex gyral edges (curvature maxima) from the quasi-flat and concave regions quite sharply, even at low simplification levels, whereas the curvature-based classifier produces results where the separation is less clear. The skeleton-based classifier is also able to produce noise-free results directly in voxel space (Figure 7.3 top), whereas the curvature classifier used here (and other similar ones) need to construct a local polygonal, or local tangent-plane, approximation of the voxel data.

Figure 7.5 illustrates our 3D skeleton and surface classifier on a hip dataset, to demonstrate the applicability of our method for different datasets besides cortical surfaces, including shapes with tunnels. Just as for the brain dataset, we see the clear separation of convex edges from non-convex regions.

As already mentioned, there is a strong connection between skeletons and convex shape features. Hence, our skeleton-based surface classifier shares several properties with curvature-based classifiers: sharp edges are detected more precisely than soft edges; there is an analogy between Gaussian filtering of the curvature signal and geodesic-distance-based simplification of the skeleton, whereby the Gaussian filter size and the skeleton importance threshold both act as scale parameters.

However, there are also important differences. Our skeleton-based classifier uses only integral computations, and is hence inherently more robust than derivative-based curvature methods. Also, the skeleton classifier can be seen as a quasi-global operator, since our importance measure gathers information that may come from a large part of the surface. In contrast, curvature estimators are strongly local, as they only analyze a small neighborhood at each surface point. Together, these facts explain the difference in robustness of the considered classifiers.

7.4 Segmentation method

In the previous section we detailed on the computation of our skeleton-based classifier. In this section, we use the classifier to create a patch-type segmentation. An overview of the

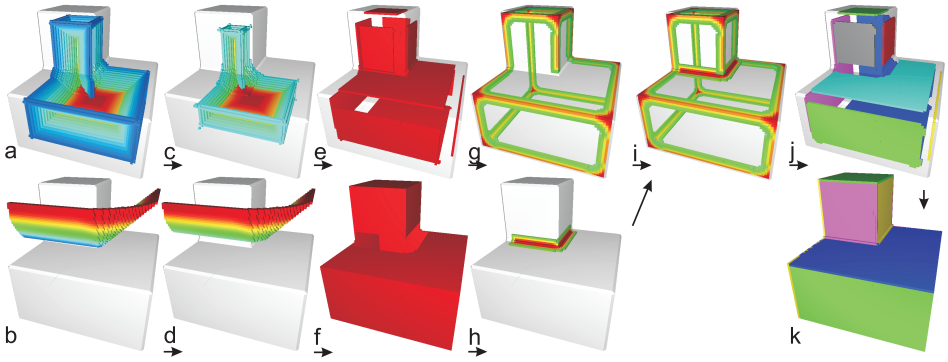


Figure 7.6: Overview of our approach. (a) Interior and (b) exterior skeletons, rainbow color-map encodes importance measure. (c) Simplified interior and (d) exterior skeletons. (e,f) Gaps in feature points. (g) Convex edges. (h) Concave edges. (i) Combined edges. (j) Connected components. (k) Final segmentation.

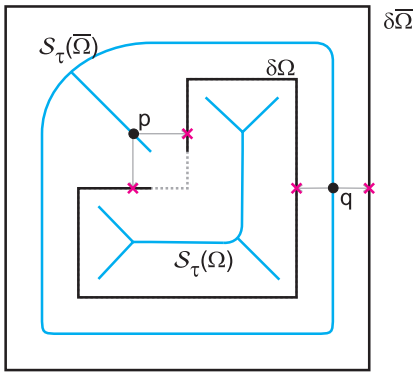


Figure 7.7: Exterior skeleton $\mathcal{S}_\tau(\bar{\Omega})$ of exterior volume bounded by $\partial\bar{\Omega}$, two skeleton points $p, q \in \mathcal{S}_\tau(\bar{\Omega})$, and their feature points $F(p)$, $F(q)$.

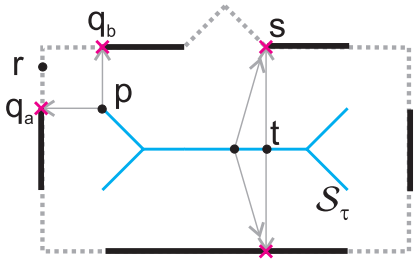


Figure 7.8: Normal computation

approach is shown in Figure 7.6.

7.4.1 Exterior skeleton

As explained, we use the simplified skeleton of the shape interior Ω, denoted $\mathcal{S}_\tau(\Omega)$, to detect convex shape edges $E(\Omega)$, and the skeleton of the shape exterior $\bar{\Omega}$, denoted $\mathcal{S}_\tau(\bar{\Omega})$, to find the concave edges $E(\bar{\Omega})$.

Computation of the exterior skeleton $\mathcal{S}_\tau(\bar{\Omega})$ is slightly different from the computation of $\mathcal{S}_\tau(\Omega)$, as follows. The exterior volume $\bar{\Omega}$ is enclosed in a bounding box around the entire shape. The voxels on the surface of this box form an additional boundary $\partial\bar{\Omega}$

which is disjoint from the object boundary $\partial\Omega$. Exterior voxels have feature voxels on both $\partial\Omega$ and $\partial\bar{\Omega}$. Because these two boundaries are not connected, no shortest path can be computed between two voxels on either boundary. We assign to these voxels an infinite importance ($\rho_{\mathcal{S}}$=∞), so that they will never be simplified regardless the setting of τ. This is the desired behavior: the exterior skeleton should only be simplified for concave edges, not for any other parts. Note that in Figure 7.6(d), the exterior skeleton is not shown for the sake of clarity. A cross-section of the exterior skeleton of that object is shown in Figure 7.7. Voxels in $\bar{\Omega}$ that have all their feature voxels on $\partial\Omega$ are processed in the same manner as interior voxels. Point p is a point having both feature points on $\partial\Omega$, whereas point q has one feature point on $\partial\Omega$ and one on $\partial\bar{\Omega}$. Point q will never be simplified, regardless the setting of τ.

7.4.2 Normal-sensitive edge erosion

After detecting the convex and concave edges, the combined edge E divide the boundary into connected components (Figure 7.6(j)). These components have gaps between them of at least width τ_e. To fill these gaps, we dilate the components onto the gaps, thereby eroding the edges E. We do not perform an erosion ordered on geodesic distance on the boundary, as this may not always place the final segment borders exactly at the high-curvature creases, even if these are available. Instead, we do a so-called *normal-sensitive erosion* of the edges. For this, we need to compute a normal on every boundary point first. Estimating the normals of a noisy voxel surface using standard approaches typically smooths normals everywhere, not only on the noisy parts. These smoothed normals affect the erosion quality, so more sophisticated normal estimators are needed [85]. Fortunately, the simplified surface skeleton, which we have already computed, can effectively be used to compute non-smoothed boundary normals.

It is well known that the *feature vectors*, which point from a skeleton point $p \in \mathcal{S}$ to its feature points $q \in F(p)$ are normal to the boundary $\partial\Omega$ [97]. Hence, we use $\frac{q-p}{\|q-p\|}$ as our normals. This is shown in Figure 7.8 in 2D for illustrative purposes. By using our robust simplified skeleton $\mathcal{S}_\tau$, which does not contain any spurious parts due to noise, the resulting normals are also robust to noise. Normals may be ambiguous at a few boundary points in the case of skeleton ligatures, e.g. point s in Figure 7.8: multiple skeleton points have point s as feature point. In such cases, we use the normal of the skeleton point that has the largest angle between its two feature vectors, which is point t in case of s.

However, the feature collection of the simplified surface skeleton does not completely cover the shape surface, and hence the normals are not available everywhere. For each boundary point r for which the normal is not availabe, we get the nearest known point q and get its associated skeleton point p. The normal of r is then the feature vector at p whose dot product with $r - p$ is maximal, which is $\frac{q_a-p}{\|q_a-p\|}$ in case of Figure 7.8. The erosion of the edges is then performed in the order of most similar normals, that is, each voxel e in the set of edge voxels E is assigned to the component of the non-edge voxel p that admits the minimum-cost path between p and e in the boundary graph, where the cost between two adjacent voxels is given by their normal difference.

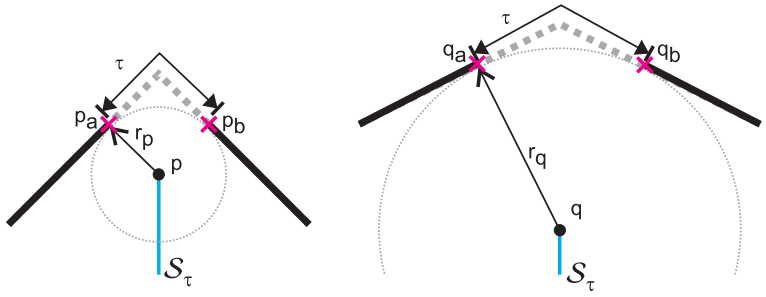


Figure 7.9: Cross-section of a sharp (left) and blunt edge (right). Inscribed ball centered at a point p has feature points p_a and p_b and radius r_p. Feature pairs p_a, p_b and q_a, q_b both are at geodesic distance τ from each other. The sharp edge has a smaller inscribed ball than the blunt edge ($r_p < r_q$).

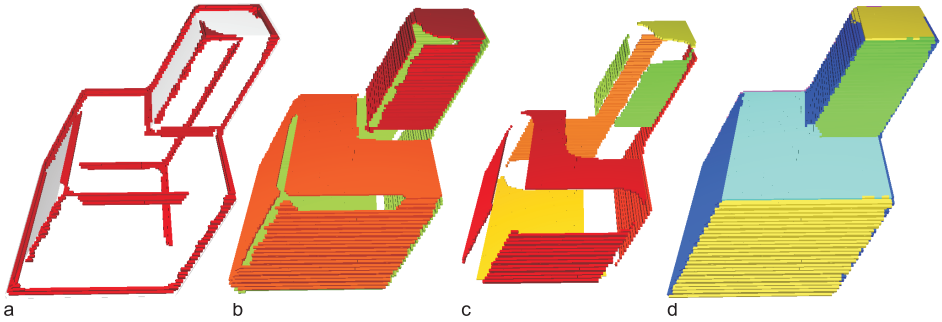


Figure 7.10: Handling corners. The detected edges E (a). Connected components have thin connections (b). Splitting components (c). Final segmentation after edge erosion (d).

7.4.3 Handling corners

Although Eq. 7.3 is well suited to detect the shape edges, problems occur at corners, where edges of different strength come together. The reason is that for the same setting of τ, blunt edges have larger inscribed balls than sharp edges (see Figure 7.9). At corners where a sharp and blunt edge come together, only the smallest ball fits inside the shape. Hence, the feature collection will contain feature points on both sides of the sharp edge, making the blunt edge undetected near the corner.

Figure 7.10 illustrates this for the two stacked boxes from Figure 7.6, but now skewed. Figure 7.10(a) shows the set of detected edge voxels E, which has disconnections near some of the corners. The resulting segmentation in Figure 7.10(b) shows thin connections between components. Clearly, this is an under-segmentation. A straightforward solution is to dilate the edges E so that these connections between components disappear. The edge erosion step (Section 7.4.2) will then remove the dilated edges. Let H_ϵ be the set of points at a geodesic distance of at least ϵ of the detected edge voxels E, so that $H_0 = \partial\Omega \setminus E$. For high enough ϵ there will be no thin connections left in the connected component

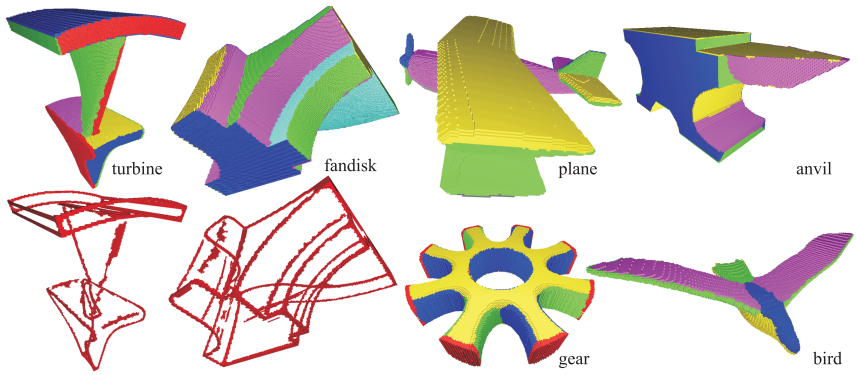


Figure 7.11: Segmentations of several shapes. The detected edges are shown for the Turbine and Fandisk shapes.

of H_ϵ. However, for high values of ϵ some of the smaller connected components in H_ϵ may be completely removed, or some of the components in H_0 may inadvertently be split into multiple components. We proceed as follows. Starting with $\epsilon = 0$, we iteratively increase ϵ with small increments, and consider the connected components in H_ϵ. We check whether a component in a level H_i is split into multiple components in level H_j ($i < j$). In that case, we let the component split only if the resulting components are not too small and have a different orientation (using the normals as computed in Section 7.4.2). The first constraint is to prevent small components from completely disappearing in the final segmentation. The second is to prevent components with thin parts from being split up into multiple segments. Figure 7.10(c) shows the result after splitting of components, whereas Figure 7.10(d) shows the final segmentation after edge erosion. Note that the presented solution for handling corners is not necessarily the only and best one, but gives satisfactory results in practice.

7.5 Results and discussion

We have implemented our patch-type segmentation algorithm in C++ and ran it on an Intel Core 2 Extreme 3 GHz (using 1 CPU), with 2 GB of RAM.

We now present results of the patch-type segmentation method. We used shapes from the INRIA Gamma database [56]. The shapes have resolutions ranging up to 300^3 voxels. Figure 7.11 shows the resulting segmentations for several shapes. Most segments are placed as expected and no over-segmentations are produced. The segments are quasi-flat: smooth areas separated by relatively high curvature creases. Note that on the tips of the wing no segments are placed because the resolution in the tips is too limited for the skeleton to reach into. We further observe that sharp and straight borders are produced for soft edges, such as in the Gear and Bird shapes.

Our segmentation approach has several desirable properties, as follows. First, we can detect very soft and vanishing edges, because we detect edges as gaps in the feature collection. These gaps arise when we threshold the skeleton's importance measure $\rho_{\mathcal{S}}$, which

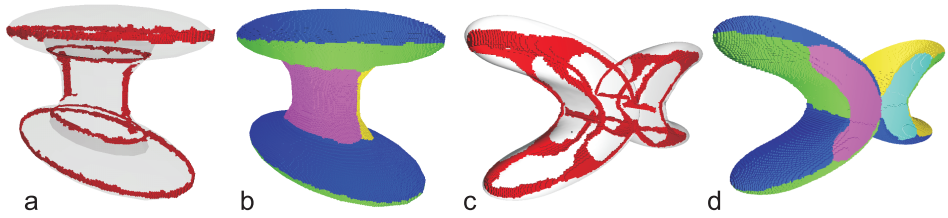


Figure 7.12: (a,c) Detected edges and (b,d) segmentations of a smooth H-shape and smooth X-shape respectively.

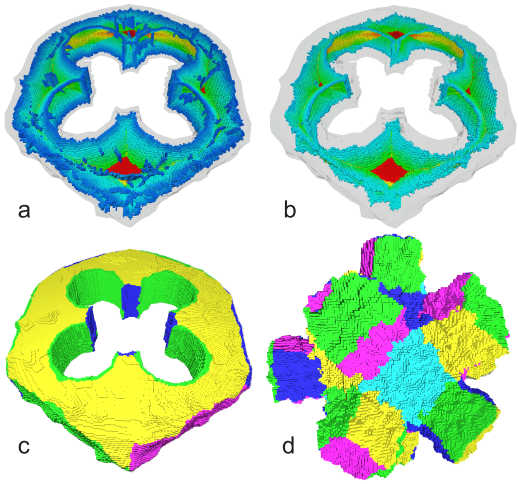


Figure 7.13: Segmentation of two noisy shapes. (a) Non-simplified skeleton $\mathcal{S}_0(\Omega)$. (b) Simplified skeleton $\mathcal{S}_{\tau=15}(\Omega)$. (c) Resulting segmentation. (d) Segmentation of noisy Sea mine.

represents geodesic distance between feature points. For both smooth and sharp edges, setting a threshold of τ ensures gaps of at least width τ. Figure 7.12 shows the detected edges and the resulting segmentations of a smooth H-shape consisting of ellipsoids and a smooth X-shape consisting of two bent boxes with vanishing edges. We observe no spurious segments for both cases. For the H-shape, each ellipsoid is split in two. The vanishing edges of the X-shape are detected well, and sharp, straight, segment borders are generated for them by the edge erosion step.

Second, our method is capable of handling shapes with boundary noise, because it uses the simplified skeleton. We assume the noise to be uniform, so that the scale parameter τ can be set to such a value that the skeleton does not contain any spurious parts due to noise. Figure 7.13(c,d) shows the segmentation of the noisy Tap threads and Sea mine shapes respectively. For the Tap threads shape, both the non-simplified (Figure 7.13(a)) skeleton, and the simplified skeleton (Figure 7.13(b)) that is used in the segmentation are shown. These noisy shapes would be difficult to handle using traditional curvature-based segmentation approaches (see Section 7.5.1). Nevertheless, we should state that

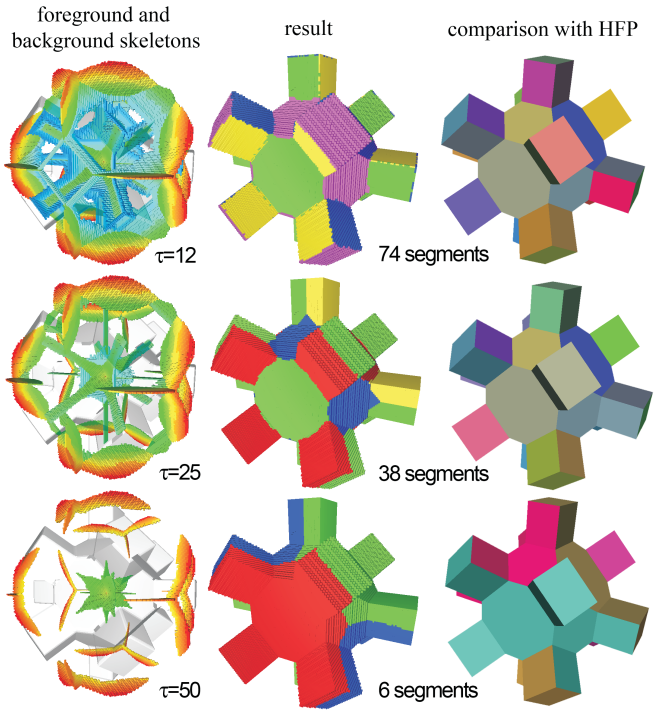


Figure 7.14: Multiscale segmentations of the Sea mine. Left column: interior and exterior skeletons. Middle column: corresponding increasingly coarse segmentations. Right column: comparison to HFP segmentations [7].

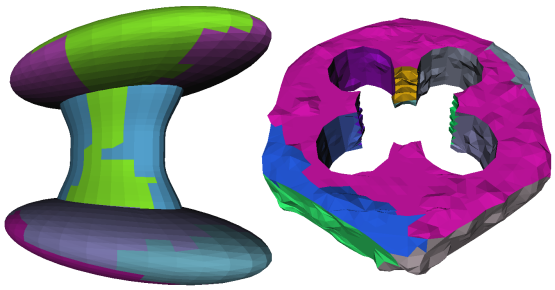


Figure 7.15: Results of HFP (compare with Figure 7.12(b) and Figure 7.13(c)).

for very noisy objects the feature collections become too sparse, potentially resulting in over-segmentation.

Third, multiscale segmentations can be created by increasing the scale parameter τ beyond the noise level. In Figure 7.14, a Sea mine shape, consisting of a polyhedron with

Table 7.1: Timing measurements. Columns show in order: shape name, its dimensions, time to compute interior and exterior skeleton in seconds, time for segmentation in seconds, and memory usage in megabytes.

shape	dim	int $\mathcal{S}$ t	ext $\mathcal{S}$ t	segm t	mem
Fandisk	295x177x276	187	233	30	1080
Sea mine	203x204x176	21	115	29	640
X-shape	119x174x296	35	101	16	450

attached boxes, is simplified at three different simplification levels (τ=12,25,50). Interior and exterior skeletons are shown in the left column (a rainbow color-map encodes the importance measure). The resulting segmentations are shown in the right column. At level τ=12, all patches of the Sea mine are identified. At τ=25, only the interior skeleton center sheets remain inside each box, producing two segments for each attached box. As we can see here, a property of our approach is that the borders between coarse-scale segments not necessarily lie on curvature creases of the surface. Indeed, when we would reconstruct the original object from this simplified skeleton by placing the inscribed balls on the skeleton points, the reconstructed shape would be a smoothed version of the original shape. Each box becomes similar to an ellipsoid, of which the segmentation consists of two halves as we have seen in the H-shape example (Figure 7.12(b)).

Table 7.1 show indicative running times for a few of the shapes shown in this chapter. Computation of the skeletons takes the larger part of the time due to the large number of shortest path computations involved, needed for computing the importance measure $\rho_{\mathcal{S}}$. Computing the exterior skeleton takes the longest, as the number of exterior voxels is typically much larger than the number of interior voxels. The time needed for detecting the edges and performing the segmentation is comparatively small, as it has time complexity order $O(b \log b)$ for computing the geodesic distances used by our classifier.

Despite our method's advantages, a few limitations exist. First, patches should be completely bordered by convex and/or concave creases in order to be identified as segments, so our method is more suitable for geometric than for organic shapes, for which a part-type segmentation would be better choice. Second, for parts of the object that are thin, we might not detect blunt edges. As indicated in Section 7.4.3, inscribed balls for weak edges require larger radii than for sharp edges. Unfortunately, thin parts of the object do not allow inscribed balls with a high radius, and thus we might not detect edges in such cases.

7.5.1 Comparison

We compare results of our method with those of Hierarchical Fitting Primitives, a state-of-the-art segmentation method of Attene et al. [7] for which the software is available on the internet. HFP clusters faces hierarchically in a greedy manner by fitting plane, sphere, and cylinder primitives. A binary tree of clusters is built in 10 seconds, after which the desired number of segments can be selected by the user. For each shape, we selected the number of clusters in HFP such that it is equal to the number of segments our method has pro-

duced. In our method the number of segments cannot be controlled directly, but follows from the chosen skeleton simplification level. For shapes that only have strong creases, HFP delivers equivalent results at the finer scales. However, Figure 7.15 shows HFP's clustering for the H-shape, which has soft edges, and the Tap threads, which has boundary noise. For the H-shape, our method splits each of the ellipsoids into two symmetrical halves corresponding to the skeleton sheet's top and bottom sides (Figure 7.12(b)). HFP's clusters on the other hand do not reflect the symmetry of the shape at all. For the Tap threads, HFP produces an erroneous segment border on the top due to the boundary noise. Our method suffers less from the noise (Figure 7.13(c)) as the simplified skeleton is unaffected by it. These examples indicate that using skeletal information can be beneficial in patch-type shape segmentation.

We have also compared our multiscale segmentations with those of HFP (see Figure 7.14, rightmost column). One difference is that our multiscale segmentations are not hierarchical: a segment at a fine scale need not lie completely in a segment at a coarser scale. HFP on the other hand always produces hierarchical segmentations. We further observe that whereas HFP always places segment borders at high-curvature creases, our method does not necessarily do that for higher simplification levels, as we explained in the previous section.

7.6 Conclusion

In this chapter we argued that 3D skeletons, which are often seen as unreliable and unstable, can be used to perform basic image-processing operations. As an example of such an operation, we showed that robust, multi-scale skeletonization of the cortical surface provides a robust and effective means for the classification of voxel-based surfaces into highly convex (ridge) regions and non-convex areas. We showed that skeleton-based classifiers are less sensitive to discretization noise, due to their integral-based construction, as compared to curvature-based classifiers which use derivatives. We illustrate our method on different voxel datasets obtained from medical imaging scans.

Furthermore, we have presented a new way of robustly detecting the edges of 3D voxel shapes and creating patch-type segmentations. We use the simplified skeleton for detecting the edges, instead of using curvature information explicitly, which makes our method robust for voxelized, noisy, and low-resolution shapes. A single user-parameter is used to allow the user to specify the amount of boundary noise. This parameter is intuitive, as it measures geodesic length, and can also be used to create multiscale segmentations. In these simplified segmentations, segment borders are not necessarily placed at shape creases. Finally, sharp segment borders are generated for soft and vanishing edges.

Future work involves the actual utilization of our classifier for concrete medical imaging applications, such as the extraction of sulcal fundi curves, as well as a more rigorous mathematical analysis of the connection between our classifier and curvature metrics.

Chapter 8

Conclusions

In the preceding chapters we have presented novel methods for skeletonization and shape segmentation. In this chapter, we sum the contributions of our work and give recommendations for future work.

8.1 Contributions

The research question posed in Chapter 1 was comprised of two parts. The first part asked how we can efficiently compute curve and surface skeletons of binary voxel shapes so that they can be used effectively as shape descriptors in various applications. By investigating the literature, we identified the properties that skeletons should fulfill to make them effective as shape descriptors. We found that robustness to noise and ability for a multiscale representation of the shape in particular are essential. One way to obtain these properties is to use a pruning method that combine a global importance measure with a simple pruning strategy. In Chapter 3, we analyzed and improved such an approach that combines the boundary-distance measure with a simple thresholding strategy for 2D shapes. We also presented and compared four algorithms for computing Tolerance-based Feature Transforms. In Chapter 4, we contributed a skeletonization method for binary voxel shapes that computes curve and surface skeletons that can be made robust by setting a single, intuitive user parameter to specify the amount of boundary noise. Furthermore, this parameter provides a multiscale representation. It does not need to be tweaked, as the simplified skeletons are connected by default. Underpinning our skeletons is a novel global importance measure, called the collapse measure, which basically extends the boundary-distance measure to 3D shapes. It is a combination of a quasi-global measure defined on the surface skeleton and a global measure on the curve skeleton. In Chapter 5, we contributed a method to partition our simplified surface skeletons into their respective sheets. We believe that these partitionings will prove to be beneficial in various applications, such as shape analysis and retrieval tasks.

The second part of our research question was how we can use skeletons to create different types of shape segmentations that benefit from the desirable skeleton properties. We

presented two novel skeleton-based methods for the segmentation of binary voxel shapes. Using our simplified curve skeletons, we contributed a novel semantically-oriented part-type segmentation method in Chapter 6. Using our simplified surface skeletons, we contributed a novel geometrically-oriented patch-type segmentation in Chapter 7. Both segmentation types benefit from the robustness of our simplified skeletons, but also from other desirable skeleton properties.

Our part-type segmentation method uses the curve skeleton to extract the part structure of the shape. Hereto, we formulated the junction rule. In combination with junction-type detection, we are able to identify the simple intersecting parts of which a complex shape is comprised. Intrinsic to our curve skeleton definition is the skeleton-to-boundary mapping: each curve-skeleton point has an associated Jordan curve on the shape surface which presents a potential part cut. Part-type segmentation thus boils down to selecting points on the curve skeleton that generate accurate part cuts. The resulting segmentations consist by construction of parts that are simple, i.e., they do not contain bifurcations, and parts that have tight and smooth borders. Finally, pose-invariance follows from the use of curve skeletons, and noise robustness follows from the use of simplified curve skeletons.

Our patch-type segmentation method uses the simplified surface skeleton to define a surface classifier. We use a generalization of the symmetry-curvature duality to 3D, in combination with the observation that our skeletons are simplified near the rims first, to detect curvature maxima of the shape surface. By combining the classifier of both interior and exterior skeleton, we detect convex and concave edges, that combined divide the shape surface into quasi-flat patches. Due to the use of simplified skeletons, our method produces clear segment borders for soft and vanishing edges, is robust to noise, and can produce multiscale segmentations.

All of our algorithms work on binary voxel shapes, which bring their own problems. When obtained using a scanning device, they are typically of low to medium resolution because of scanning hardware restrictions. Moreover, voxel shapes can be noisy due to scanning inaccuracies. Finally, the surface normal is not naturally available in voxel shapes. Computing the normal is difficult to do accurately for low resolution and noisy shapes, and effectively leads to a further reduction in voxel-shape resolution due to the large filter sizes involved. The unreliability of normal information under boundary noisy does not only hold for voxel shapes, but for all shape representations. Hence, we chose to use integral quantities in both our skeletonization and segmentation methods, namely distances, shortest-path lengths, and areas, rather than differential quantities, such as distance-field gradients and surface curvature.

In our skeletonization method, voxel shapes proved to be convenient from an implementation point of view, as they provide a regular, volumetric sampling of the shape. The curve skeleton, surface skeleton, and object are nested structures of different dimensions, that we can all three represent uniformly as voxel sets. It is important to note that most elements in our algorithms are not limited to voxel shapes. Shortest paths and areas can also be computed on meshes. In any case, our shape segmentation methods can easily be used to segment meshes by mapping the voxel result back onto the original mesh.

8.2 Future work

Our work opens up various possibilities for further research.

In this work, we focused on 2D and 3D shapes. In future work, it would be interesting to see if our skeletonization approach can be extended to 4D or even higher dimensional shapes. In addition to the curve and surface skeleton, a new "volumetric skeleton" would have to be defined for 4D shapes. The importance measure on the curve skeleton would then measure a volume, on the surface skeleton it would measure an area, and on the volumetric skeleton it would measure curve length. These higher-dimensional simplified skeletons could for example be used to analyze higher-dimensional scientific data, such as time-dependent data.

By increasing the simplification level, our simplified skeletons are pruned for the noisy features, which is desired, but are also pruned partly for the non-noisy features. In fact, this property lies at the basis of our patch-type segmentation method. However, for some applications this behavior might not be desired. In the case of skeleton-based surface-smoothing, which we did in Section 4.7.1, it is desirable to fully preserve the skeleton parts that represent feature edges. In this manner, the edges of the reconstructed shape are not smoothed in the process, effectively yielding edge-preserving denoising. To achieve this, one possibility might be to modify our importance measure in such a way that sharp non-noisy edges gain more importance. Another promising direction is to "unprune" the skeleton by using our skeleton partitioning, as we suggested in Section 5.6.

We typically see shapes as a volume with a clear tangible surface, and in this work we made no exception by considering *binary* voxel shapes only. In medical imaging, however, one often deals with gray-scale 3D images. To compute a skeleton using our method, we first have to extract a binary voxel shape by an image segmentation process, as we did for the brain shape (Figure 4.12). However, manually choosing the right segmentation level can be a subtle and time consuming process. It might be interesting to compute our simplified skeletons for all iso-surfaces in the image, to produce a multiscale representation having two dimensions: the simplification level and the iso-value. Reconstruction of the gray-scale image from this bi-dimensional representation might lead to novel image segmentation, smoothing, and compression techniques.

Finally, an interesting direction for further research is shape retrieval. We have contributed three robust geometric shape-descriptors that could all be used for this purpose: the simplified skeleton and its partitioning, the part-type shape segmentation, and the patch-type shape segmentation. It would be interesting to investigate whether one of them, or a combination, can contribute to the existing work on shape matching and retrieval.

Bibliography

[1] AHUJA, N., AND CHUANG, J. Shape representation using a generalized potential field model. *IEEE Transactions on Pattern Analysis and Machine Intelligence 19*, 2 (1997), 169–176.

[2] AIM@SHAPE repository. http://shapes.aim-at-shape.net.

[3] AMENTA, N., BERN, M., AND KAMVYSSELIS, M. A new voronoi-based surface reconstruction algorithm. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)* (1998), pp. 415–421.

[4] AMENTA, N., CHOI, S., AND KOLLURI, R. K. The power crust. In *Proceedings of the sixth ACM symposium on Solid Modeling and Applications (SMA)* (2001), pp. 249–266.

[5] ARCELLI, C., AND SANNITI DI BAJA, G. Euclidean skeleton via centre-of-maximal-disc extraction. *Image and Vision Computing 11*, 3 (1993), 163–173.

[6] ATTALI, D., AND MONTANVERT, A. Computing and simplifying 2D and 3D continuous skeletons. *Computer Vision and Image Understanding 67*, 3 (1997), 261–273.

[7] ATTENE, M., FALCIDIENO, B., AND SPAGNUOLO, M. Hierarchical mesh segmentation based on fitting primitives. *The Visual Computer 22*, 3 (2006), 181–193.

[8] ATTENE, M., KATZ, S., MORTARA, M., PATANÉ, G., SPAGNUOLO, M., AND TAL, A. Mesh segmentation - a comparative study. In *Proceedings of the IEEE International Conference on Shape Modeling and Applications (SMI)* (2006), pp. 7–19.

[9] AUGUST, J., SIDDIQI, K., AND ZUCKER, S. W. Ligature instabilities in the perceptual organization of shape. *Computer Vision and Image Understanding 76*, 3 (1999), 231–243.

[10] AUGUST, J., TANNENBAUM, A., AND ZUCKER, S. W. On the evolution of the skeleton. In *Proceedings of the Seventh International Conference on Computer Vision (ICCV)* (1999), pp. 315–322.

[11] BARTESAGHI, A., AND SAPIRO, G. A system for the generation of curves on 3D brain images. *Human Brain Mapping 14*, 1 (2001), 1–15.

[12] BITTER, I., SATO, M., BENDER, M., MCDONNELL, K. T., KAUFMAN, A., AND WAN, M. Ceasar: a smooth, accurate and robust centerline extraction algorithm. In *Proceedings of the conference on Visualization (VIS)* (2000), pp. 45–52.

[13] BLUM, H. A transformation for extracting new descriptors of shape. In *Models for the Perception of Speech and Visual Form* (1967), W. Wathen-Dunn, Ed., MIT Press, pp. 362–380.

[14] BLUM, H. Biological shape and visual science. *Journal of Theoretical Biology 38* (1973), 205–287.

[15] BLUM, H., AND NAGEL, R. N. Shape description using weighted symmetric axis features. *Pattern Recognition 10*, 3 (1978), 167–180.

[16] BORGEFORS, G., SANNITI DI BAJA, G., AND SVENSSON, S. Decomposing digital 3D shapes using a multiresolution structure. In *Proceedings of the 14th conference on Discrete Geometry for Computer Imagery (DGCI'99)* (1999), vol. 1568 of *Lecture Notes on Computer Science*, Springer Berlin / Heidelberg, pp. 19–30.

[17] BOUIX, S., AND SIDDIQI, K. Divergence-based medial surfaces. In *Proceedings of the 6th European Conference on Computer Vision - Part I (ECCV)* (2000), Springer-Verlag, pp. 603–618.

[18] BOUIX, S., SIDDIQI, K., AND TANNENBAUM, A. Flux driven automatic centerline extraction. *Medical Image Analysis 9*, 3 (2005), 209–221.

[19] BRUNNER, D., AND BRUNNETT, G. Mesh segmentation using the object skeleton graph. In *Proceedings of the 7th IASTED International Conference on Computer Graphics and Imaging* (2004), ACTA Press, pp. 48–55.

[20] CACHIA, A., MANGIN, J.-F., RIVIÈRE, D., KHERIF, F., BODDAERT, N., ANDRADE, A., PAPADOPOULOS-ORFANOS, D., POLINE, J.-B., BLOCH, I., ZILBOVICIUS, M., SONIGO, P., BRUNELLE, F., AND RGIS, J. A primal sketch of the cortex mean curvature: a morphogenesis based approach to study the variability of the folding patterns. *IEEE Transactions on Medical Imaging 22*, 6 (2003), 754–765.

[21] CLARENZ, U., DIEWALD, U., AND RUMPF, M. Anisotropic diffusion in surface processing. In *Proceedings of IEEE Visualization* (2000), pp. 397–405.

[22] CLARENZ, U., GRIEBEL, M., SCHWEITZER, M. A., AND TELEA, A. Feature sensitive multiscale editing on surfaces. *The Visual Computer 20*, 5 (2004), 329–343.

[23] CORNEA, N. D. *Curve-skeletons: properties, computation and applications*. PhD thesis, Rutgers, The State University of New Jersey, 2007.

[24] CORNEA, N. D., SILVER, D., AND MIN, P. Curve-skeleton properties, applications and algorithms. *IEEE Transactions on Visualization and Computer Graphics 13*, 3 (2007), 530–548.

[25] CORNEA, N. D., SILVER, D., YUAN, X., AND BALASUBRAMANIAN, R. Computing hierarchical curve-skeletons of 3D objects. *The Visual Computer 21*, 11 (2005), 945–955.

[26] CUISENAIRE, O. *Distance transformations: fast algorithms and applications to medical image processing*. PhD thesis, Université catholique de Louvain, Belgium, 1999.

[27] CULVER, T., KEYSER, J., AND MANOCHA, D. Exact computation of the medial axis of a polyhedron. *Computer Aided Geom. Des. 21*, 1 (2004), 65–98.

[28] DA COSTA, L. F. Multidimensional scale space shape analysis. In *Proceedings of the International Workshop on SyntheticNatural Hybrid Coding and Three Dimensional Imaging* (Santorini, Greece, 1999), pp. 214–217.

[29] DA COSTA, L. F., AND CESAR, JR, R. M. *Shape analysis and classification*. CRC Press, 2001.

[30] DAMON, J. Global medial structure of regions in R^3. *Geometry and Topology 10* (2006), 2385–2429.

[31] DANIELSSON, P.-E. Euclidean distance mapping. In *Computer Graphics and Image Processing* (1980), vol. 14, pp. 227–248.

[32] DAVIES, E., AND PLUMMER, A. Thinning algorithms: A critique and a new methodology. *Pattern Recognition 14* (1981), 53–56.

[33] DESBRUN, M., MEYER, M., SCHRÖDER, P., AND BARR, A. H. Implicit fairing of irregular meshes using diffusion and curvature flow. In *Proceedings ACM SIGGRAPH* (1999), ACM Press/Addison-Wesley Publishing Co., pp. 317–324.

[34] DESBRUN, M., MEYER, M., SCHROEDER, P., AND BARR, A. Anisotropic feature-preserving denoising of height fields and bivariate data. In *Proceedings of Graphics Interface* (2000), pp. 145–152.

[35] DEY, T. K., GIESEN, J., AND GOSWAMI, S. Shape segmentation and matching with flow discretization. In *Proceedings Workshop on Algorithms and Data Structures* (2003), vol. 2748 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 25–36.

[36] DEY, T. K., AND SUN, J. Defining and computing curve-skeletons with medial geodesic function. In *Proceedings of Eurographics Symposium on Geometry Processing* (2006), pp. 143–152.

[37] DEY, T. K., AND ZHAO, W. Approximate medial axis as a voronoi subcomplex. In *Proceedings of the seventh ACM symposium on Solid Modeling and Applications (SMA)* (Saarbrücken, Germany, 2002), pp. 356–366.

[38] DIJKSTRA, E. W. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1 (1959), 269–271.

[39] DIMITROV, P., DAMON, J. N., AND SIDDIQI, K. Flux invariants for shape. In *Proc. of Computer Vision and Pattern Recognition (CVPR)* (2003), pp. 835–841.

[40] FOSKEY, M., LIN, M., AND MANOCHA, D. Efficient computation of a simplified medial axis. In *Proceedings of the 8th Symposium on Solid Modeling and Applications (SMA)* (2003), ACM Press, pp. 96–107.

[41] FUNKHOUSER, T., KAZHDAN, M., SHILANE, P., MIN, P., KIEFER, W., TAL, A., RUSINKIEWICZ, S., AND DOBKIN, D. Modeling by example. *ACM Transactions on Graphics 23*, 3 (2004), 652–663.

[42] GAGVANI, N., KENCHAMMANA-HOSEKOTE, D., AND SILVER, D. Volume animation using the skeleton tree. In *Proceedings of the IEEE Symposium on Volume Visualization* (1998), ACM, pp. 47–53.

[43] GAGVANI, N., AND SILVER, D. Parameter-controlled volume thinning. *CVGIP: Graphical Models and Image Processing 61*, 3 (1999), 149–164.

[44] GARLAND, M., WILLMOTT, A., AND HECKBERT, P. Hierarchical face clustering on polygonal surfaces. In *Proceedings of the ACM Symposium on Interactive 3D Graphics* (2001), pp. 49–58.

[45] GE, Y., AND FITZPATRICK, J. M. On the generation of skeletons from discrete euclidean distance maps. *IEEE Transactions on Pattern Analysis and Machine Intelligence 18*, 11 (1996), 1055–1066.

[46] GIBLIN, P., AND KIMIA, B. B. A formal classification of 3D medial axis points and their local geometry. *IEEE Transactions on Pattern Analysis and Machine Intelligence 26*, 2 (2004), 238–251.

[47] GOUALHER, G., PROCYK, E., COLLINS, D., VENUGOPAL, R., BARILLOT, C., AND EVANS, A. Automated extraction and variability analysis of sulcal neuroanatomy. *IEEE Transactions on Medical Imaging 18*, 3 (1999), 206–217.

[48] HART, P. E., NILSSON, N. J., AND RAPHAEL, B. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics 4*, 2 (1968), 100–107.

[49] HASSOUNA, M. S., AND FARAG, A. A. Robust centerline extraction framework using level sets. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)* (2005), IEEE Computer Society, pp. 458–465.

[50] HESSELINK, W. H., VISSER, M., AND ROERDINK, J. B. T. M. Euclidean skeletons of 3D data sets in linear time by the integer medial axis transform. In *Proceedings of the 7th international symposium on Mathematical Morphology* (2005), Springer, pp. 259–268.

[51] HILAGA, M., SHINAGAWA, Y., KOHMURA, T., AND KUNII, T. L. Topology matching for fully automatic similarity estimation of 3D shapes. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques* (2001), ACM Press, pp. 203–212.

[52] HISADA, M., BELYAEV, A. G., AND KUNII, T. L. A skeleton-based approach for detection of perceptually salient features on polygonal surfaces. *Computer Graphics Forum 21*, 4 (2001), 689–700.

[53] HOFFMAN, D. D., RICHARDS, W., PENTL, A., RUBIN, J., AND SCHEUHAMMER, J. Parts of recognition. *Cognition 18* (1984), 65–96.

[54] HOFFMAN, D. D., AND SINGH, M. Salience of visual parts. *Cognition 63*, 1 (1997), 29–78.

[55] HONG, L., MURAKI, S., KAUFMAN, A., BARTZ, D., AND HE, T. Virtual voyage: interactive navigation in the human colon. In *Proceedings of the 24th annual conference on Computer Graphics and Interactive Techniques (SIGGRAPH)* (1997), pp. 27–34.

[56] INRIA Gamma 3D meshes research database. http://www-c.inria.fr/gamma/download/download.php.

[57] KAO, C. Y., HOFER, M., SAPIRO, G., STERN, J., REHM, K., AND ROTTENBERG, D. A. A geometric method for automatic extraction of sulcal fundi. *IEEE Transactions on Medical Imaging 26*, 4 (2007), 530–540.

[58] KAO, C.-Y., HOFER, M., SAPIRO, G., STERN, J., AND ROTTENBERG, D. A. A geometric method for automatic extraction of sulcal fundi. *IEEE Transactions on Medical Imaging 26*, 4 (2006), 530–540.

[59] KATZ, R. A., AND PIZER, S. M. Untangling the blum medial axis transform. *International Journal of Computer Vision 55*, 2-3 (2003), 139–153.

[60] KATZ, S., LEIFMAN, G., AND TAL, A. Mesh segmentation using feature point and core extraction. *The Visual Computer 21* (2005), 649–658.

[61] KATZ, S., AND TAL, A. Hierarchical mesh decomposition using fuzzy clustering and cuts. *ACM Transactions on Graphics 22*, 3 (2003), 954–961.

[62] KHANEJA, N., MILLER, M., AND GRENANDER, U. Dynamic programming generation of curves on brain surfaces. *Pattern Analysis and Machine Intelligence 20*, 11 (1998), 1260–1265.

[63] KIMMEL, R., AMIR, A., AND BRUCKSTEIN, A. M. Finding shortest paths on surfaces using level sets propagation. *IEEE Transactions on Pattern Analysis and Machine Intelligence 17*, 6 (1995), 635–640.

[64] KIMMEL, R., SHAKED, D., KIRYATI, N., AND BRUCKSTEIN, A. M. Skeletonization via distance maps and level sets. *Computer Vision and Image Understanding 62*, 3 (1995), 382–391.

[65] KIRYATI, N., AND SZÉKELY, G. Estimating shortest paths and minimal distances on digitized three-dimensional surfaces. *Pattern Recognition 26* (1993), 1623–1637.

[66] KONG, T. Y., AND ROSENFELD, A. Digital topology: Introduction and survey. *Computer Vision, Graphics, and Image Processing 48* (1989), 357–393.

[67] KOVÁCS, I., FEHÉR, A., AND JULESZ, B. Medial-point description of shape: a representation for action coding and its psychophysical correlates. *Vision Research 38* (1998), 2323–2333.

[68] LAM, L., LEE, S.-W., AND SUEN, C. Y. Thinning methodologies-a comprehensive survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence 14*, 9 (1992), 869–885.

[69] LEE, Y., LEE, S., SHAMIR, A., COHEN-OR, D., AND SEIDEL, H. P. Mesh scissoring with minima rule and part salience. *Computer Aided Geometric Design 22* (2005), 444–465.

[70] LEYMARIE, F. F. *3D Shape Representation via Shock Flows*. PhD thesis, Brown University, 2002.

[71] LEYTON, M. Symmetry-curvature duality. *Computer Vision, Graphics, and Image Processing 38* (1987), 327–341.

[72] LI, X., WOON, T. W., TAN, T. S., AND HUANG, Z. Decomposing polygon meshes for interactive applications. In *Proceedings of the 2001 Symposium on Interactive 3D Graphics* (2001), pp. 35–42.

[73] LIEN, J.-M., KEYSER, J., AND AMATO, N. M. Simultaneous shape decomposition and skeletonization. In *Proceedings of the ACM Symposium on Solid and Physical Modeling (SPM)* (2005), pp. 219–228.

[74] LIEUTIER, A. Any open bounded subset of $\mathbb{R}^n$ has the same homotopy type as its medial axis. *Computer-Aided Design 36*, 11 (2004), 1029–1046.

[75] LIU, R., AND ZHANG, H. Segmentation of 3D meshes through spectral clustering. In *Proceedings of the 12th Pacific Conference on Computer Graphics and Applications (Pacific Graphics)* (Washington, DC, USA, 2004), IEEE Computer Society, pp. 298–305.

[76] LOHMANN, G. Extracting line representations of sulcal and gyral patterns in MR images of the human brain. *IEEE Transactions on Medical Imaging 17*, 6 (1998), 1040–1048.

[77] LOTUFO, T. A., FALCAO, A. A., AND ZAMPIROLLI, F. A. Fast euclidean distance transform using a graph-search algorithm. In *Proceedings of the 13th Brazilian Symposium on Computer Graphics and Image Processing* (2000), pp. 269–275.

[78] MA, W.-C., WU, F.-C., AND OUHYOUNG, M. Skeleton extraction of 3D objects with radial basis functions. In *Proceedings of the Shape Modeling International (SMI)* (2003), p. 207.

[79] MALANDAIN, G., BERTRAND, G., AND AYACHE, N. Topological segmentation of discrete surfaces. *International Journal of Computer Vision 10*, 2 (1993), 183–197.

[80] MALANDAIN, G., AND FERNÁNDEZ-VIDAL, S. Euclidean skeletons. *Image and Vision Computing 16*, 5 (1998), 317–327.

[81] MANGAN, A. P., AND WHITAKER, R. T. Partitioning 3D surface meshes using watershed segmentation. *IEEE Transactions on Visualization and Computer Graphics 5*, 4 (1999), 308–321.

[82] MEIJSTER, A., ROERDINK, J. B. T. M., AND HESSELINK, W. H. A general algorithm for computing distance transforms in linear time. In *Mathematical Morphology and its Applications to Image and Signal Processing* (2000), J. Goutsias, L. Vincent, and D. Bloomberg, Eds., Kluwer, pp. 331–340.

[83] MÉMOLI, F., SAPIRO, G., AND THOMPSON, P. Implicit brain imaging. *Neuroimage 23*, 1 (2004), 179–188.

[84] MIN, P. Binvox: a 3D mesh voxelizer, http://www.google.com/search?q=binvox.

[85] MOLLER, T., MACHIRAJU, R., MUELLER, K., AND YAGEL, R. A comparison of normal estimation schemes. In *IEEE Visualization* (1997), pp. 19–26.

[86] MONTANARI, U. A method for obtaining skeletons using a quasi-euclidean distance. *Journal of the ACM 15*, 4 (1968), 600–624.

[87] MORETON, H., AND SÉQUIN, C. Functional optimization for fair surface design. In *Proceedings of the Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)* (1992), ACM, pp. 167–176.

[88] MORTARA, M., PATANÉ, G., SPAGNUOLO, M., FALCIDIENO, B., AND ROSSIGNAC, J. Blowing bubbles for multi-scale analysis and decomposition of triangle meshes. *Algorithmica 38*, 1 (2003), 227–248.

[89] MORTARA, M., PATANÉ, G., SPAGNUOLO, M., FALCIDIENO, B., AND ROSSIGNAC, J. Plumber: a multiscale decomposition of 3D shapes into tubular primitives and bodies. In *Proceedings of the ACM symposium on Solid Modeling and Applications (SMA)* (2004), pp. 339–344.

[90] MULLIKIN, J. C. The vector distance transform in two and three dimensions. *Graphical Models and Image Processing 54*, 6 (1992), 526–535.

[91] MUSSER, D. R., AND SAINI, S. *STL tutorial and reference guide: C++ programming with the standard template library*. Addison-Wesley Professional Computing Series, 1996.

[92] NIBLACK, C. W., GIBBONS, P. B., AND CAPSON, D. W. Generating skeletons and centerlines from the distance transform. *Graphical Models and Image Processing 54*, 5 (1992), 420–437.

[93] OGNIEWICZ, R. L. Skeleton-space: a multiscale shape description combining region and boundary information. In *Proceedings of Computer Vision and Pattern Recognition (CVPR)* (1994), p. 746751.

[94] OGNIEWICZ, R. L., AND ILG, M. Voronoi skeletons: theory and applications. In *Proc. of Computer Vision and Pattern Recognition* (1992), pp. 63–69.

[95] OGNIEWICZ, R. L., AND KÜBLER, O. Hierarchic voronoi skeletons. *Pattern Recognition 28*, 3 (1995), 343–359.

[96] PIZER, S. M., EBERLY, D., AND FRITSCH, D. S. Zoom-invariant vision of figural shape: the mathematics of cores. *Computer Vision and Image Understanding 69*, 1 (1998), 55–71.

[97] PIZER, S. M., SIDDIQI, K., SZÉKELY, G., DAMON, J. N., AND ZUCKER, S. W. Multiscale medial loci and their properties. *International Journal of Computer Vision 55*, 2-3 (2003), 155–179.

[98] PODOLAK, J., SHILANE, P., GOLOVINSKIY, A., RUSINKIEWICZ, S., AND FUNKHOUSER, T. A planar-reflective symmetry transform for 3D shapes. *ACM Transactions on Graphics (Proceedings of SIGGRAPH) 25*, 3 (2006).

[99] Persistence of Vision Raytracer, http://www.povray.org.

[100] PROHASKA, S., AND HEGE, H.-C. Fast visualization of plane-like structures in voxel data. In *Proceedings of IEEE Visualization* (2002), pp. 29–36.

[101] PROVOT, L., AND DEBLED-RENNESSON, I. Segmentation of noisy discrete surfaces. In *Proceedings of the 12th International Workshop on Combinatorial Workshop (IWCIA)* (2008), vol. 4958 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 160–171.

[102] PUDNEY, C. Distance-ordered homotopic thinning: A skeletonization algoritm for 3D digital images. *Computer Vision and Image Understanding 72*, 3 (1998), 404–413.

[103] RAGNEMALM, I. Neighborhoods for distance transformations using ordered propagation. *CVGIP: Image Understanding 56*, 3 (1992), 399–409.

[104] REINDERS, F. *Feature-based Visualization of Time-dependent Data*. PhD thesis, Technische Universiteit Delft, 2001.

[105] RENAULT, C., DESVIGNES, M., AND REVENU, M. 3D curves tracking and its application to cortical sulci detection. In *Proceedings of the International Conference on Image Processing* (2000), pp. 491–494.

[106] RENIERS, D. Personal website, http://www.win.tue.nl/~dreniers.

[107] RENIERS, D., JALBA, A., AND TELEA, A. Robust classification and analysis of anatomical surfaces using 3D skeletons. In *Proceedings of the first Eurographics Workshop on Visual Computing for Biomedicine (VCBM)* (Delft, The Netherlands, 2008), pp. 61–68.

[108] RENIERS, D., AND TELEA, A. Skeleton-based hierarchical shape segmentation. In *Proceedings of the IEEE International Conference on Shape Modeling and Applications (SMI)* (Lyon, France, 2007), pp. 179–188.

[109] RENIERS, D., AND TELEA, A. Hierarchical part-type segmentation using voxel-based curve skeletons. *The Visual Computer 24*, 6 (2008), 383–395.

[110] RENIERS, D., AND TELEA, A. Part-type segmentation of articulated voxel-shapes using the junction rule. *Computer Graphics Forum (Proceedings of Pacific Graphics 2008) 27*, 7 (2008), 1845–1852.

[111] RENIERS, D., AND TELEA, A. Patch-type segmentation of voxel shapes using simplified surface skeletons. *Computer Graphics Forum (Proceedings of Pacific Graphics 2008) 27*, 7 (2008), 1837–1844.

[112] RENIERS, D., AND TELEA, A. Segmenting simplified surface skeletons. In *Proceedings of conference on Discrete Geometry for Computer Imagery (DGCI)* (Lyon, France, 2008), vol. 4992 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 262–274.

[113] RENIERS, D., AND TELEA, A. Tolerance-based feature transforms. In *Advances in Computer Graphics and Computer Vision* (2008), vol. 4, Springer Berlin/Heidelberg, pp. 187–200.

[114] RENIERS, D., AND TELEA, A. C. Quantitative comparison of tolerance-based feature transforms. In *Proceedings of the First International Conference on Computer Vision Theory and Applications (VISAPP)* (Setúbal, Portugal, 2006), INSTICC Press, pp. 107–114.

[115] RENIERS, D., VAN WIJK, J. J., AND TELEA, A. Computing multiscale curve and surface skeletons of genus 0 shapes using a global importance measure. *IEEE Transactions on Visualization and Computer Graphics 14*, 2 (2008), 355–368.

[116] ROSENFELD, A. Connectivity in digital pictures. *Journal of the ACM 17*, 1 (1970), 146–160.

[117] ROSENFELD, A., AND KAK, A. C. *Digital Picture Processing*. Academic Press, New York, USA, 1976.

[118] RUMPF, M., AND TELEA, A. A continuous skeletonization method based on level sets. In *Proceedings of the symposium on Data Visualisation* (2002), Eurographics Association, pp. 151–158.

[119] SCHIRMACHER, H., ZÖCKLER, M., STALLING, D., AND HEGE, H.-C. Boundary surface shrinking - a continuous approach to 3D center line extraction. In *Proceedings of conference on Image and Multidimensional Digital Signal Processing* (1998), B. Girod, H. Niemann, and H.-P. Seidel, Eds., pp. 25–28.

[120] SETHIAN, J. A. *Level set methods and fast marching methods*, 2nd ed. Cambridge University Press, 1999.

[121] SHAKED, D., AND BRUCKSTEIN, A. M. The curve axis. *Computer Vision and Image Understanding 63*, 2 (1996), 367–379.

[122] SHAKED, D., AND BRUCKSTEIN, A. M. Pruning medial axes. *Computer Vision and Image Understanding 69*, 2 (1998), 156–169.

[123] SHAMIR, A. A formulation of boundary mesh segmentation. In *Proceedings of the 3D Data Processing, Visualization, and Transmission, 2nd International Symposium (3DPVT)* (2004), pp. 82–89.

[124] SHILANE, P., MIN, P., KAZHDAN, M., AND FUNKHOUSER, T. The Princeton shape benchmark. In *Proceedings of the IEEE International Conference on Shape Modeling and Applications (SMI)* (2004), pp. 167–178.

[125] SHINAGAWA, Y., KUNII, T., AND KERGOSIEN, Y.-L. Surface coding based on Morse theory. *Computer Graphics and Applications 11*, 5 (1991), 66–78.

[126] SIDDIQI, K., BOUIX, S., TANNENBAUM, A., AND ZUCKER, S. W. The hamilton-jacobi skeleton. In *Proceedings of the International Conference on Computer Vision (ICCV)* (1999), pp. 828–834.

[127] SIDDIQI, K., BOUIX, S., TANNENBAUM, A., AND ZUCKER, S. W. Hamilton-jacobi skeletons. *International Journal of Computer Vision 48*, 3 (2002), 215–231.

[128] SIDDIQI, K., ZHANG, J., MACRINI, D., SHOKOUFANDEH, A., BOUIX, S., AND DICKINSON, S. Retrieving articulated 3D models using medial surfaces. *Machine Vision and Applications 19*, 4 (2008), 261–275.

[129] STRZODKA, R., AND TELEA, A. Generalized distance transforms and skeletons in graphics hardware. In *Proceedings of EG/IEEE TCVG Symposium on Visualization (VisSym)* (2004), pp. 221–230.

[130] SUD, A., FOSKEY, M., AND MANOCHA, D. Homotopy-preserving medial axis simplification. In *Proceedings of the ACM Symposium on Solid and Physical modeling (SPM)* (2005), pp. 39–50.

[131] SVENSSON, S., AND SANNITI DI BAJA, G. Simplifying curve skeletons in volume images. *Computer Vision and Image Understanding 90*, 3 (2003), 242–257.

[132] TANGELDER, J. W. H., AND VELTKAMP, R. C. A survey of content based 3d shape retrieval methods. In *Proceedings of the International Conference on Shape Modeling and Applications (SMI)* (2004), pp. 145–156.

[133] TAO, X., HAN, X., RETTMANN, M. E., PRINCE, J. L., AND DAVATZIKOS, C. Statistical study on cortical sulci of human brains. In *Proceedings of the 17th International Conference on Information Processing in Medical Imaging (IPMI)* (London, UK, 2001), Springer-Verlag, pp. 475–487.

[134] TAO, X., PRINCE, J., AND DAVATZIKOS, C. Using a statistical shape model to extract sulcal curves on the outer cortex of the human brain. *IEEE Transactions on Medical Imaging 21*, 5 (2002), 513–524.

[135] TAUBIN, G. Estimating the tensor of curvature of a surface from a polyhedral approximation. In *Proceedings of the Fifth International Conference on Computer Vision (ICCV)* (1995), IEEE Computer Society, pp. 902–9907.

[136] TELEA, A., AND VAN WIJK, J. J. An augmented fast marching method for computing skeletons and centerlines. In *Proceedings of the Symposium on Data Visualisation (VisSym)* (2002), pp. 251–259.

[137] TELEA, A., AND VILANOVA, A. A robust level-set algorithm for centerline extraction. In *Proceedings of the Symposium on Data Visualisation (VisSym)* (2003), pp. 185–194.

[138] THOMPSON, P. M., HAYASHI, K. M., SOWELL, E. R., GOGTAY, N., GIEDD, J. N., RAPOPORT, J. L., DE ZUBICARAY, G. I., JANKE, A. L., ROSE, S. E., SEMPLE, J., DODDRELL, D. M., WANG, Y., VAN ERP, T. G., CANNON, T. D., AND TOGA, A. W. Mapping cortical change in alzheimer's disease, brain development, and schizophrenia. *Neuroimage 23 Suppl 1* (2004), 2–18.

[139] TIERNY, J., VANDEBORRE, J.-P., AND DAOUDI, M. Topology driven 3D mesh hierarchical segmentation. In *Proceedings of the IEEE International Conference on Shape Modeling and Applications (SMI)* (2007), pp. 215–220.

[140] VAN EEDE, M., MACRINI, D., TELEA, A., SMINCHISESCU, C., AND DICKINSON, S. S. Canonical skeletons for shape matching. In *Proceedings of the 18th International Conference on Pattern Recognition (ICPR)* (2006), IEEE Computer Society, pp. 64–69.

[141] VERMEER, P. J. *Medial axis transform to boundary representation conversion*. PhD thesis, Purdue University, West Lafayette, IN, USA, 1994.

[142] VERWER, B. J. H., VERBEEK, P. W., AND DEKKER, S. T. An efficient uniform cost algorithm applied to distance transforms. *IEEE Transactions on Pattern Analysis and Machine Intelligence 11*, 4 (2003), 425–429.

[143] WAND, S., ROSENFELD, A., AND WANG, A. Y. Medial axis transformation for grayscale pictures. *IEEE Transactions on Pattern Analysis and Machine Intelligence 4*, 4 (1982), 419–422.

[144] WRIGHT, M. W., AND FALLSIDE, F. Skeletonisation as model-based feature detection. *Communications, Speech and Vision, IEE Proceedings I 140*, 1 (1993), 7–11.

[145] ZHANG, J., SIDDIQI, K., MACRINI, D., SHOKOUFANDEH, A., AND DICKINSON, S. Retrieving articulated 3-D models using medial surfaces and their graph spectra. In *Proceedings of the International Workshop On Energy Minimization Methods in Computer Vision and Pattern Recognition* (2005), pp. 285–300.

[146] ZHOU, Y., AND TOGA, A. Efficient skeletonization of volumetric objects. *IEEE Transactions on Visualization and Computer Graphics 5*, 3 (1999), 196–209.

[147] ZUCKERBERGER, E., TAL, A., AND SHLAFMAN, S. Polyhedral surface decomposition with applications. *Computers & Graphics 26*, 5 (2002), 733–743.

Summary

Skeletons are geometrical shape-descriptors that are of a lower dimensionality than the shape they describe. They are centered within the shape, capture the shape's symmetry, and describe the topology and articulation of the shape in a compact manner. These qualities make skeletonization a desirable pre-processing step in a variety of applications ranging from shape segmentation and retrieval, to pose estimation and motion planning. The skeleton was formally defined by Blum in 1967. For 2D shapes, the skeleton is a set of curves. Extending the definition to 3D shapes, the skeleton is a set of surface components and curves, and is called surface skeleton. Because a 1D structure is more compact and more suitable for some applications such as motion planning, alternative skeleton definitions have been proposed that yield 1D structures, known as curve skeletons.

There are still a number of unsolved problems for the computation of both curve and surface skeletons, as follows. First, skeletons are expensive to compute accurately, especially in 3D. Second, skeletons are sensitive to discretization artifacts and noise on the shape boundary, resulting in many spurious skeleton parts. Skeletonization methods need to ensure that the produced skeletons are robust to noise, so that they can be effectively used in subsequent applications.

In the first part of this thesis, we focus on efficiently computing skeletons that are robust to noise. We first investigate the computation of Tolerance-based Feature Transforms, that help in solving a number of discretization problems and enable the robust computation of skeletons of 2D shapes. We then present a new method for computing simplified curve skeletons and surface skeletons for 3D shapes. We define a global importance measure, which yields simplified skeletons when thresholding this measure using a single, intuitive user parameter. Finally, we extract the simplified skeleton structure by identifying its constituent surface components.

In the second part of the thesis, we use our simplified skeletons to develop two new 3D shape segmentation methods. First, we present a method for creating semantically-oriented segmentations of articulated shapes using the curve skeleton. These segments capture the logical parts of the shape, such as the fingers of a hand. The segmentations are robust to boundary noise, are pose-invariant, and have smooth segment borders. Second, we present a geometrically-oriented segmentation method, using a surface-skeleton-based surface classifier. These segmentations are most suitable for faceted shapes.

Although the methods presented in this thesis work on binary voxel shapes, many of the ideas and heuristics apply to other shape representations as well.

Samenvatting

Skeletten zijn geometrische objectbeschrijvingen, die van een lagere dimensionaliteit zijn dan het object dat zij beschrijven. Ze zijn gecentreerd in het object, vangen de symmetrie van het object, en beschrijven de topologie en de articulatie van het object op een compacte manier. Deze kwaliteiten maken skeletonizatie een gewenste stap in een groot aantal applicaties, variërend van object segmentatie en object *retrieval*, tot posebepaling en bewegingsplanning. Het skelet werd formeel gedefinieerd door Blum in 1967. Het skelet van een twee-dimensionaal (2D) object bestaat uit curves. Wanneer de definitie wordt uitgebreid naar 3D objecten, is het skelet een 2D structuur bestaande uit gekromde vlakken en curves, en wordt *surface-skelet* genoemd. Omdat een 1D structuur compacter is en meer geschikt voor bepaalde applicaties zoals bewegingsplanning, zijn er alternatieve skeletdefinities voorgesteld die resulteren in 1D structuren genaamd *curve-skeletten*.

Er zijn nog steeds een aantal onopgeloste problemen bij het berekenen van curve- en surface-skeletten. Ten eerste kost het veel tijd om skeletten accuraat te berekenen, met name voor 3D objecten. Verder zijn skeletten erg gevoelig voor discretizatiefouten en ruis in het object, wat resulteert in overbodige delen in het skelet. Skeletonizatiemethodes dienen te zorgen dat de skeletten robuust zijn onder ruis, zodat ze effectief gebruikt kunnen worden in latere applicaties.

In het eerste deel van dit proefschrift leggen we ons toe op het efficiënt berekenen van skeletten die robuust zijn onder ruis. We onderzoeken tolerantiegebaseerde *feature transforms*, die een aantal discretizatieproblemen oplost en het mogelijk maakt om skeletten van 2D objecten robuust te berekenen. Vervolgens presenteren we een nieuwe methode om gesimplificeerde curve- en surface-skeletten van 3D objecten te berekenen. We definiëren een globale belangmaat welke resulteert in gesimplificeerde skeletten door middel van een enkele en intuïtieve gebruikersparameter. Ten slotte extraheren we de structuur van het gesimplificeerde skelet door zijn afzonderlijke gekromde vlakken te identificeren.

In het tweede deel van dit proefschrift gebruiken we onze gesimplificeerde skeletten om twee nieuwe 3D object segmentaties te ontwikkelen. Eerst presenteren we een methode om semantiekgeoriëntieerde segmentaties van gearticuleerde objecten te berekenen met behulp van het curve-skelet. Deze segmenten representeren de logische delen van het object, zoals de vingers van een hand. De segmentaties zijn robuust onder ruis en invariant onder verschillende poses van het object, en de randen tussen segmenten zijn glad. Daarna presenteren we een geometriegeoriënteerde segmentatie methode, die gebruik maakt van een oppervlakteclassificatie gebaseerd op het surface-skelet. Deze segmentaties zijn het

meest geschikt voor gefaceteerde objecten.

Alhoewel de methodes die we presenteren in dit proefschrift op voxelobjecten werken, kunnen de ideeën en heuristieken ook gebruikt worden voor andere objectrepresentaties.

List of publications

The following publications are related to this work:

- RENIERS, D., AND TELEA, A. Quantitative comparison of tolerance-based feature transforms. In *Proceedings of the first International Conference on Computer Vision Theory and Applications (VISAPP)*, pp. 107-114, 2006. February 25-28, Setúbal, Portugal. INSTICC Press. (Chapter 3)

- RENIERS, D., AND TELEA, A. Tolerance-based feature transforms, *Advances in Computer Graphics and Computer Vision,* Volume 4, 187-200, 2008. Springer. (Chapter 3)

- RENIERS, D., AND VAN WIJK, J.J., AND TELEA, A. Computing Multiscale Curve and Surface Skeletons of Genus 0 Shapes Using a Global Importance Measure, *IEEE Transactions on Visualization and Computer Graphics,* 14(2), 355-368, 2008. (Chapter 4)

- RENIERS, D., AND TELEA, A. Segmenting Simplified Surface Skeletons. *Lecture Notes in Computer Science*, vol. 4992, 262-274, 2008. Springer. (Proceedings of the *Conference on Discrete Geometry for Computer Imagery (DGCI)*, April 16-18, Lyon, France, 2008). (Chapter 5)

- RENIERS, D., AND TELEA, A. Skeleton-based Hierarchical Shape Segmentation. In *Proceedings of the IEEE International Conference on Shape Modeling and Applications (SMI)*, pp. 179-188, 2007. June 13-15, Lyon, France. IEEE Computer Society Press. (Chapter 6)

- RENIERS, D., AND TELEA, A. Hierarchical part-type segmentation using voxel-based curve skeletons, *The Visual Computer* 24(6), 383-395, 2008. Springer Berlin/Heidelberg. (Chapter 6)

- RENIERS, D., AND TELEA, A. Part-type segmentation of articulated voxel-shapes using the junction rule. *Computer Graphics Forum* 27(7), 1845-1852, 2008. Blackwell publishing. (Proceedings of the *Pacific Conference on Computer Graphics and Applications*, October 8-10, Tokyo, Japan). (Chapter 6)

- RENIERS, D., AND TELEA, A. Patch-type segmentation of voxel shapes using simplified surface skeletons. *Computer Graphics Forum* 27(7), 1837-1844, 2008. Blackwell publishing. (Proceedings of the *Pacific Conference on Computer Graphics and Applications*, October 8-10, Tokyo, Japan). (Chapter 7)

- RENIERS, D., JALBA, A., AND TELEA, A. Robust classification and analysis of anatomical surfaces using 3D skeletons. In *Proceedings of the first Eurographics Workshop on Visual Computing for Biomedicine (VCBM)*, pp. 61-68, October 6-7, Delft, The Netherlands, 2008. (Chapter 7)

Curriculum Vitae

Dennie Reniers was born on the 5th of December 1980 in Breda, The Netherlands. He completed his secondary education at the Mencia de Mendoza Lyceum in Breda. From 1999, he studied computer science at the Department of Mathematics and Computer Science at the Technische Universiteit Eindhoven. He received his Master's degree in 2004, with honors. His Master's thesis was titled "Multi-scale extreme simplification and rendering of point sets".

In January 2005, he started on an NWO funded PhD project at the same university under the supervision of Alexandru Telea. For this research, he developed new skeletonization and shape segmentation techniques. This led to a number of articles in international conference proceedings and journals, and his PhD dissertation in December 2008.

As of February 2009, he will start working at Solid Source, a small start-up in software quality assessment based in Eindhoven.

Index