SHAPE SEGMENTATION AND RETRIEVAL BASED ON THE SKELETON CUT SPACE

CONG FENG

This research was supported by my family.

Cover: Skeleton and Shape segmentation

Shape Segmentation and Retrieval Based on The Skeleton Cut Space

Cong Feng
PhD Thesis Rijksuniversiteit Groningen

ISBN 978-90-367-9905-8 (printed version)
ISBN 978-90-367-9904-1 (electronic version)

university of groningen

Shape Segmentation and Retrieval Based on The Skeleton Cut Space

PhD thesis

to obtain the degree of PhD at the
University of Groningen
on the authority of the
Rector Magnificus Prof. E. Sterken
and in accordance with
the decision by the College of Deans.

This thesis will be defended in public on

Friday 22 September 2017 at 12.45 hours

by

Cong Feng

born on 22 May 1987
in Hunan, China

Supervisor
Prof. A. C. Telea

Co-supervisor
Dr. A. C. Jalba

Assessment committee
Prof. J. L. D. Comba
Prof. R. C. Veltkamp
Prof. N. Petkov

ABSTRACT

Shape processing is a rapidly expanding and challenging field, with applications in science, engineering, medicine, and the entertainment industry. Shape segmentation and shape retrieval are two important subfields of shape processing. Segmentation is concerned with the detection of shape parts and patches that obey certain specific properties. Retrieval aims to efficiently find a set of similar shapes to a given query shape from a large collection.

Many methods have been proposed for shape segmentation and retrieval, based on so-called boundary and volumetric representations of the shape. A different representation is the medial one. For 3D shapes, medial representations use so-called curve and surface skeletons to compactly encode the shape geometry and/or topology and allow reasoning about shape properties such as part-whole structure, genus, articulation, thickness, and symmetry. While curve skeletons have been extensively used for segmentation and retrieval, surface skeletons are far less present in this context, mainly due to the difficulty in computing them. However, recent state-of-the-art methods allow efficient, accurate, and simple computation of surface skeletons, opening the way for their usage.

In this thesis, we investigate how surface skeletons can be efficiently and effectively used to support shape segmentation and retrieval. To this end, we propose several contributions. First, we define a new concept – the cut space – which allows extending existing part-based segmentation techniques to use the surface skeleton. Secondly, we propose a unified part-and-patch segmentation methods using the cut space. We show how our segmentation methods can handle both boundary and volumetric representations of 3D shapes. Thirdly, we show how descriptors can be extracted from the cut space to support efficient and high-quality shape retrieval. Finally, we propose a novel 3D multiscale skeletonization method that achieves superior quality as compared to existing state-of-the-art methods in the same class. We demonstrate all our results by proposing efficient implementations thereof and comparing our results with a wide range of competitive methods on a comprehensive family of 3D real-world shapes.

SAMENVATTING

Vormverwerking is een snel uitbreidend en uitdagend veld met toepassingen in de wetenschap, engineering, geneeskunde, en de vermaakindustrie. Vormsegmentatie en vormzoektechnieken zijn twee belangrijke componenten van vormverwerking. Segmentatie bestudeert het detecteren van vormonderdelen (*parts* en *patches*) met bepaalde eigenschappen. Vormzoektechnieken bestudeert het efficiënt vinden van vormen die gelijkenissen hebben met een gegeven vraagvorm, gegeven een grote verzameling vormen om in te zoeken.

Veel methodes bestaan voor vormsegmentatie en vormzoektechnieken, gebaseerd op zogenaamde rand- en volumerepresentaties van vorm. Een derde representatie is de mediale. Voor 3D vormen, mediale representaties gebruiken zogenaamde curve- en oppervlakskeletten om de geometrie en/of topologie van vormen te coderen. Deze representaties stelt men in staat om aspecten van vorm te analyseren zoals deel-geheelstructuur, genus, articulatie, dikte, en symmetrie. Curveskeletten zijn reeds zeer bekend in segmentatie en vormzoektechnieken. Opervlakskeletten worden veel minder gebruikt aangezien ze veel moeilijker te berekenen zijn. Niettenmin kunnen recente methodes oppervlakskeletten snel, accuraat, en eenvoudig extraheren. Dit opent nieuwe wegen voor het gebruik van oppervlakskeletten.

In dit proefschrift wordt het gebruik van oppervlakskeletten voor het efficiënt en effectief ondersteunen van vormsegmentatie en vormzoektechnieken bestudeert, met de volgende contributies. Allereerst wordt er het concept van snijruimte voorgestelt, dat bestaande curveskelet-gebaseerde partsegmentatietechnieken uitbreidt tot het oppervlakskelet. Een verenigde part-en-patchsegmentatiemethode wordt vervolgens gepresenteert op basis van de snijruimte. Deze methodes worden toegepast op beide rand- en volumerepresentaties. Ten derde worden vormdescriptoren gedefinieerd op basis van de snijruimte en ter ondersteuning van efficiënte en hoogwaardige vormzoektechnieken. Tenslotte wordt er een nieuwe 3D multischaal skeletonisatiemethode geïntroduceerd die skeletten van een hogere kwaliteit kan produceren in vergelijking met bestaande competitieve methodes in dezelfde categorie. Alle voorgenoemde resultaten worden gedemonstreerd door middel van efficiënte implementaties en vergelijkingen van hun resultaten met een breeed spectrum van competitieve methodes op een brede verzameling van realistische 3D vormen.

PUBLICATIONS

This thesis is based on material which has appeared in the publications listed below.

CONFERENCES AND JOURNALS

- C. Feng, A. Jalba, and A. Telea. Part-based segmentation by skeleton cut space analysis. In *Mathematical Morphology and Its Applications to Signal and Image Processing (Proc. ISMM)*, pages 607–618. Springer LNCS 9082, 2015
- C. Feng, A. Jalba, and A. Telea. Improved part-based segmentation of voxel shapes by skeleton cut spaces. *Mathematical Morphology – Theory and Applications*, 1:60–78, 2016
- C. Feng, A. Jalba, and A. Telea. A descriptor for voxel shapes based on the skeleton cut space. In *Proc. Eurographics Workshop on 3D Object Retrieval (3DOR)*. Eurographics, 2016

BOOK CHAPTERS

- J. Koehoorn, C. Feng, J. Kustra, A. Jalba, and A. Telea. Unified part-patch segmentation of mesh shapes using surface skeletons. In P. K. Saha, G. Borgefors, and G. S. di Baja, editors, *Skeletonization: Theory, Methods, and Applications*, chapter 2. Elsevier Limited Press, 2016
- A. X. Falcão, C. Feng, J. Kustra, and A. Telea. Multiscale 2D medial axes and 3D surface skeletons by the image foresting transform. In P. K. Saha, G. Borgefors, and G. S. di Baja, editors, *Skeletonization: Theory, Methods, and Applications*, chapter 4. Elsevier Limited Press, 2016

To my family.

CONTENTS

1 INTRODUCTION

1.1 SHAPES AND SHAPE PROCESSING

The three-dimensional world we live in consists of *shapes*. These are both natural, like landscape, plants, animals; and human made, like the wealth of objects that surround us in our houses. Besides their physical presence, shapes have entered since long the virtual universe: Three-dimensional (3D) computer games abound, displaying increasingly complex and realistic renderings of either reality-based or imaginary worlds. Computer-aided design (CAD) applications allow designers, architects, and artists to create 3D content. 3D simulation and visualization applications allow engineers and scientists to create and explore various types of data embedded in three dimensions [199]. 3D scanning technologies allow the acquisition of content from the physical world to enter into the computer universe, in the form of CT and MRI scanners [75], laser scanners, and 3D cameras [30]. Last but not least, we can convert the virtual to the real by using 3D printing technologies [204].

This wealth of digital shape information has brought with it also the need of appropriate *methods* for dealing with shapes. In this spectrum, tools and techniques for shape analysis and processing are particularly important, as follows.

Shape analysis: This class of methods is concerned with extracting quantifiable information from 3D shapes. The information can be used next either by humans, *e.g.* to assess properties of the shapes under study – such as an engineer who monitors the results of a 3D simulation to gain insight on the simulated phenomenon, or a medical doctor who examines a CT scan for diagnosis purposes. However, equally important is the use of extracted information from shapes to drive computer applications that process these shapes further. Examples hereof are simulations of deformable objects, computer games and vehicle pilot simulators where collisions have to be detected, and tools for the graphics designer which allow the selection, enhancement, or removal of specific parts of a 3D shape. For all these applications, we need to extract a range of properties of shape, such as thickness, curvature, noisiness, topology, symmetry, and duplication. Other types of shape analysis include finding our how similar two shapes are, or finding the most similar shapes to a given query shape in a large collection – a process known also as Content-Based Shape Retrieval (CBSR) [193].

Shape processing: This class of methods uses information extracted from shapes to further change these, to various ends. A well-known example is *segmentation*, where an input shape is cut (partitioned) into several smaller-scale, and usually simpler, shapes [168]. Such shapes can then be in turn analyzed, or processed, much more efficiently and/or effectively than the global input. Another equally well-known example is *cleaning*: Given any acquisition device like a scanner, the output of the acquisition is typically of limited quality, given the technological bounds of the acquisition process, but also imperfections of the physical shape being acquired. For many applications, it is essential to detect, repair, or remove such imperfections. Other types of shape processing involve simplification (removing unnecessary detail to extract the shape's essence and/or compress its digital representation), refinement (increasing ghe representation resolution to support further processing that requires this), registration (aligning two or more shapes to fit each other optimally), and editing (creating new shapes from one or more existing ones).

As both 3D acquisition devices and application requiring increasingly more complex shapes have evolved, so has the need for more advanced 3D shape analysis and processing techniques. Exploring new ways to create such techniques, and in particular shape processing techniques, is the general context of this thesis.

1.2 MEDIAL SHAPE DESCRIPTORS

Let us, for a start, define a 3D solid shape Ω as a compact volumetric[1] subset of $\mathbb{R}^3$. We next denote the boundary, or interface, separating this shape from the surrounding space by $\partial\Omega$.

To be able to analyze and process shapes on a computer, we need first and foremost a way to *represent* Ω and/or $\partial\Omega$ in a digital form. To this end, three main types of representations are known for 3D shapes [189]:

1.2.1 *Boundary representations*

Boundary representations focus on the shape's surface $\partial\Omega$. The most widespread way to represent $\partial\Omega$ in a discrete form is to use a so-called *polygon mesh* structure. Essentially, this is a piecewise-linear approximation of $\partial\Omega$ based on a set of polygons (typically, triangles), whose vertices reside on or close to $\partial\Omega$ [21].

1 By *volumetric*, we mean a subset of $\mathbb{R}^3$ whose intrinsic dimension is 3. By intrinsic dimension, we mean here the number of independent variables that are needed to describe the respective subset. As such, a curved surface (intrinsic dimension 2) and a curve (intrinsic dimension 1) embedded in $\mathbb{R}^3$ are not considered by us as volumetric shapes in the above acception.

Boundary representations, also called *b-reps*, have many advantages. First and foremost, they are relatively simple and compact to store and process on a computer – see the *compactness* property in [189]. For example, a surface having one million triangles takes, roughly, 48 megabytes of memory. Given the current memory sizes of modern computers, this is not a large amount. Secondly, boundary representations are sufficient for a wide range of applications, including the rendering (drawing) of realistic depictions of Ω, animation, and the design of virtual universes. Thirdly, this representation is simple to manipulate for both computer programs and humans (designers). Rendering and manipulating boundary representations is also supported very well on modern graphics hardware (GPUs). As such, the vast majority of shape analysis and processing applications nowadays use boundary representations.

1.2.2 *Volumetric representations*

However flexible, b-reps have several limitations. First and foremost, by definition, they represent only $\partial\Omega$, and not the interior Ω of a shape. For several types of application, we need to explicitly store and use internal information. For instance, volumetric ray tracing, used to create photorealistic renderings of half-translucent objects, may need to model various parameters that live in Ω rather than on the surface $\partial\Omega$ via techniques called physically-based rendering [133]. Certan visualization use-cases, such as examining CT and MRI scans, need to access all data stored in the volume Ω, *e.g.*, tissue density, blood flow directions, or tissue anisotropy [137].

Volume representations solve such issues by effectively sampling the entire $\Omega \subset \mathbb{R}^3$. The simplest, and most general, way to do this is to use a uniform sample grid of voxels $\mathbf{x}_i \in \mathbb{Z}^3$, by analogy to the way images are represented by pixels. Such voxel grids can effectively represent several so-called *fields*, each which can encode a different property of the shape. As the most basic level, one can encode the shape itself by using a so-called phase field $\phi : \mathbb{Z}^3 \to \mathbb{R}$, where ϕ takes *e.g.* positive values over Ω and negative values elswehere. This way, the shape itself can be defined as the threshold-set $\Omega = \{\mathbf{x} \in \mathbb{Z}^3 | \phi(\mathbf{x}) \geq 0\}$ (see *e.g.* [83]). The boundary $\partial\Omega$ is then captured by the isosurface, or level-set [111], $\partial\Omega = \mathbf{x} \in \mathbb{Z}^3 | \phi(\mathbf{x}) = 0\}$. A simplification of the above is provided by so-called *binary* volumes, which define ϕ to be 1 inside Ω, or over the volume's so-called foregound, and 0 outside of it, *i.e.* over the background.

However powerful in representing more information about a shape than b-reps, volumetric representations also have serious limitations. First, they require (much) larger amounts of memory and processing power to handle shapes at the same level of detail as compared to b-reps – for memory, roughly one to two orders of magnitude more is needed to capture the same level of detail that a b-rep stores for $\partial\Omega$,

as compared to the b-rep itself. Volumes larger than roughly 3000^3 voxels cannot be stored in the RAM of typical computers, which makes processing them slow. Secondly, the fixed-grid position of the voxel centers $\mathbf{x}_i$ makes it hard to represent data at various levels of detail (sampling resolution) over different portions of the shape, an issue that b-reps do not have (for more details, see *e.g.* [199], Chapter 3). Finally, rendering shapes as voxel sets does not yield the same quality as rendering them as b-reps, given that a raw voxel set is only a piecewise constant representation of the shape Ω, nor do level-sets extracted from them. Figure 1.1 illustrates this.

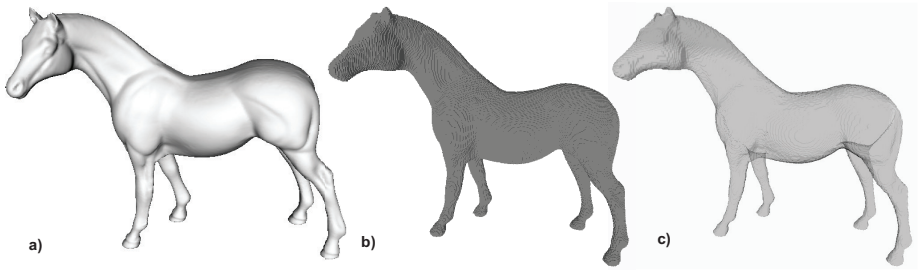


Figure 1.1: A horse model is shown as a b-rep mesh in (a), and has 48K vertices, 97K triangles. (b) The same model, represented as a binary voxel volume (resolution 512^3 voxels). (c) A level set corresponding to the volume's foreground-background interface.

1.2.3 *Medial representations*

As outlined, boundary representations can be easily extracted from voxel ones, using isosurfacing techniques (Sec. 1.2.2). Conversely, volumetrix representations of b-reps can be easily produced by a process known as *voxelization* [35]. As such, one could argue that the two above shape representations are sufficient, as they complement each other, and as we can easily convert from one to the other.

However, several types of shape analysis and manipulation are not easily supported by either b-reps or voxel models. For example, graphics designers need to deform 3D shapes to specific postures, *e.g.* to create new shapes in industrial design, or to create animations of virtual characters [183]. Doing this by using b-reps or volume models is possible, but tedious, as it requires the setting and careful manipulation of many control points. Moreover, this is often not intuitive. Separately, many applications require matching two or more shapes. In the matching process, the geometry, topology, and symmetry of the shape should be all talen into account [193]. Reasoning about topology and symmetry requires more refined shape representations than b-reps and voxel models. Thirdly, shape metrology applica-

tions need to compute certain properties of shape, such as local thickness [200] which can be expensive to perform directly on b-reps and voxel models. Last but not least, in many applications we need to abstract shapes by keeping their structural essence, such as reducing a hand to a graph having five branches for its fingers [95].

Medial representations provide a family of powerful tools for shape representation and manipulation that fills the gap between b-reps and volume models, and also covers the above use-cases. The first such representation was proposed under the name of shape *skeletons* by Blum in 1967 [20]. Given a 2D shape $\Omega \subset \mathbb{R}^2$, the Blum skeleton is the set of centers of so-called maximally inscribed disks in Ω. Skeletons have been shown next to capture important properties of a shape, such as its genus, protrusions, local thickness, and boundary geometry [172]. Separately, they are as compact as b-reps. Finally, the set of skeleton points $\mathbf{x}_i$, each annotated with its distance to the closest point on $\partial\Omega$, also called the Medial Axis Transform (MAT) [189], provides a *dual* representation of shape. That is, given a shape Ω, we can compute its MAT; and from that MAT we can reconstruct Ω.

Following their success for 2D shape processing and manipulations, skeletons have next been extended to 3D shapes. Two types of skeletons are known here – curve and surface skeletons – as follows.

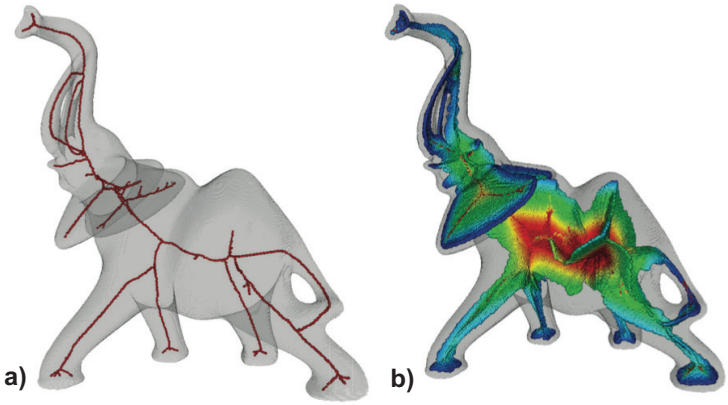


Figure 1.2: Examples of curve skeleton (a) and surface skeleton (b) of a 3D elephant shape. Images generated with the skeletonization method in [83].

Curve skeletons: The first one, and still most used, skeleton type is the curve skeleton. While a unique formal definition of the curve skeleton does not exist, it is generally accepted that this is a set of curves which are locally centered within the shape [39]. Attractive features of the curve skeleton are its ability to capture the essence of tubular articulated shapes, such as bodies of living creatures, while

being a very simple data structure. Curve skeletons reduce the dimensionality of a shape from three to one, while keeping important topology, symmetry, and (up to a certain extent) geometry information. They are also simple and fast to compute both for b-reps and volume models [178, 179]. As such, curve skeletons have been used in many applications, such as virtual path planning in CT visualization [203], virtual bronchoscopy [131], virtual colonoscopy [202], shape matching [173], shape segmentation [12], mesh quality improvement [210], shape metrology [121], shape modeling [6], and shape animation [16].

Surfce skeletons: Surface skeletons are the second skeleton type for 3D shapes. Simply put, surface skeletons use the same inscribed ball definition proposed by Blum for 2D shapes [20], but adapted to use 3D balls. They consist of a complex set of intersecting curved manifolds with boundaries. Compared to curve skeletons, surface skeletons have several technical advantages. First, they do admit a uniform, formal, definition, which can be used next to infer various properties. Secondly, they *fully* capture all aspects of a 3D shape, including geometry. As such, they allow defining a 3D MAT, and thus provide a dual representation of 3D shapes (something that curve skeletons cannot do).

However, due to their complex structure, surface skeletons are much harder to compute accurately and efficiently, as compared to curve skeletons. Also, analyzing surface skeletons is more complicated, for the same reason. Yet, recent advances in skeletonization have shown that it is possible to compute very accurate surface skeletons of complex shapes (b-reps of millions of polygons and voxel volumes of resolutions up to 1000^3) in seconds on modern computers [82, 83, 112]. Such skeletons can be then further analyzed to extract a wealth of information needed for supporting the application types outlined earlier [98]. Recent surveys in 3D skeletonization, including surface skeletons, are given in [156, 189].

Despite such recent progresses in surface skeleton computation, surface skeletons are still much less prominent in applications as compared to curve skeletons. Still, this does not mean that they do not have such potential. A few use-cases are present in the literature including shape segmentation [97, 107, 142], image-based shape reconstruction [82], shape classification [98, 144], and shape modeling [206, 207]. However, the overall number and variety of applications that surface skeletons enable is far inferior to those supported by curve skeletons.

1.3 RESEARCH QUESTIONS

Summarizing the above discussion, we find that surface skeletons have unexplored potential in supporting 3D shape processing applications, which current state-of-the-art methods allow their computation as easily, efficiently, and accurately as for the well-known, and frequently used, curve skeletons.

As such, we are now able to formulate our main research question:

How can we use surface skeletons of 3D shapes to efficiently and effectively support a range of shape processing applications?

We next refine this research question into several directions:

Scope: In our investigation, we cannot, of course, cover all possible practical applications of skeletons listed in Sec. 1.2.3. As such, we have to select a few high-potential ones. We decide to focus our work on applications in shape segmentation and shape retrieval. This choice is motivated by the large prominence of these applications in earlier skeletonization literature but also by their high practical relevance. In particular, this allows us to easily compare the advantages that our surface skeleton-based approaches will provide with respect to many existing approaches in the same areas, such as based on curve skeletons;

Representation: As outlined in Sec. 1.2, 3D shapes are mainly represented using b-reps and voxel models. As such, we choose to investigate the usage of surface skeletons in applications based on both types of representations. Hence, our work will feature both mesh-based and voxel-based skeletonization methods, and corresponding new techniques.

Theory *vs* practice: This thesis has several theoretical contributions: The introduction of the cut space for both b-rep and voxel representations, with applications in shape segmentation; the definition of shape similarity and shape retrieval based on this cut space; and the presentation of a novel surface skeletonization algorithm (for details on all above, see Sec. 1.4). However, being focused on shape processing applications, our work has a practical nature: For all presented theoretical developments, we propose algorithms to compute the respective concepts; present results of these algorithms on real-world shapes; and compare our results, both qualitatively and quantitatively with established methods.

1.4 STRUCTURE OF THIS THESIS

The structure of this thesis is as follows.

Chapter 2 gives an overview of the fields that our work builds upon. After introducing skeletonization concepts and definitions, we review the main properties of 3D curve and surface skeletons, and discuss recent skeletonization methods, with a focus on surface skeletons. This helps deciding which such methods we can use next and, where the case arises, which limitations such methods have that may im-

pact upon our work. Next, we overview applications of 3D skeletons in the areas of shape segmentation, shape matching, and shape metrology. This allows us to detect unexplored application areas (for surface skeletons), which we will next target.

Chapter 3 presents our first contribution – the skeleton cut space. The cut space is a novel way to capture geometric information atop the surface skeleton, which extends the well-known MAT (Sec. 1.2.3). By enriching the surface skeleton with such information, we show next how part-based shape segmentation can be efficiently supported. We compare our segmentation method based on the cut space with other part-based segmentation methods based on curve skeletons. The comparison shows the added-value of using the surface skeleton for part-based segmentation.

Chapter 4 extends the cut space computation and its application to part-based shape segmentation presented in Chapter 3. First, we propose several refinements in the actual computation of the cut space, which make it more stable for complex shapes. Secondly, we show how more detailed segment information can be extracted from this cut space, yielding less oversegmentation but also a better capture of small details. Next, we show how the cut space can be used for other applications such as shape editing. Finally, we compare the computational performance of our new segmentation method with existing methods.

Chapter 5 demonstrates how the cut space introduced in Chapter 3 can be used to support a different kind of application – shape retrieval. For this, we extract a shape descriptor from the cut space histogram introduced in Chapter 3 for segmentation purposes. Next, we propose several distance metrics to compare such descriptors. Finally, we evaluate the retrieval power of our proposed method on a comprehensive collection of shapes, and compare the aggregated quality of our method with known shape retrieval methods.

Chapter 6 changes the focus to mesh-based representations. While Chapters 3-5 show how the skeleton cut space and its application can handle voxel models, we now aim to show how the same cut space can be adapted to handle mesh models. This raises a number of non-trivial technical issues, which we next solve. Next, we extend the part-based segmentation abilities of the cut space introduced in Chapter 3 to also handle patch-based segmentations, and show how we can generate hybrid part-patch segmentations – a topic, to our knowledge, not addressed by most skeleton-based segmentation methods known. We demonstrate the added-value of the new segmentation method by comparing it with existing part and patch based methods on a wide collection of real-world 3D shapes.

Chapter 7 presents a novel 3D surface skeletonization method. Based on our practical insights collected during using existing state-of-the-art skeletonization methods (Chapters 3-5, we found several limitations of such methods which adversely affect their efficient and effective use in such applications. As such, we argue that a better skeletonization method is of added value in such (and other) applications. We present here such a method, which is based on a well-known efficient framework for 2D and 3D image manipulation, the Image Foresting Transform (IFT) [58]. By leveraging precision and computational efficiency properties of the IFT, our new skeletonization method can achieve superior performance (in both above-mentioned directions) as compared to a wide range of state-of-the-art surface skeletonization methods for voxel models. An additional important contribution is that our method computes 3D multiscale skeletons, for which only a handful of methods currently exist.

Chapter 8 concludes this thesis. We reflect here on our initial goal (Sec. 1.3) and discuss how well surface skeletons can support the shape processing application types we focused on. Additionally, we discuss several potentially valuable directions for future work.

All chapters that introduce contributions in this thesis (Chapters 3 - 7) correspond to respective publications, as outlined in the preamble of the thesis. This leads, however, to some unavoidable replications in the introducing sections of the respective chapters, where related work and definitions are presented. While this is possible to remove, we have preferred to keep the text of these chapters are close as possible to the respective publications (leading thus to the aforementioned replication), to ease tracing back our work to published material. Where additional material, not present in the respective publications, exists, we include this material in the corresponding chapters.

2 RELATED WORK

As outlined in Chapter 1, our main research question regards the design of innovative 3D shape processing methods using surface skeletons. As such, we review in this chapter related work to both these topics – skeletonization (Sec. 2.1) and 3D shape processing (Sec. 2.2). We end this chapter with a conclusion (Sec. 2.3) of how the current state-of-the-art in skeletonization allows us to approach our research question.

2.1 MEDIAL DESCRIPTORS

To proceed, let us first introduce some supporting terminology. Given a shape, its skeletons, also called medial axis, or more general, medial descriptor, is a thin, low-dimensional structure centered in the shape that captures both the shape's geometry and topology. In digital image processing, skeletons have been introduced by Blum [20] for binary 2D shapes. Since then, they have been extended in several directions, such as the *dimensionality* of the shape they are computed from (from 2D to 3D shapes [65, 175]); the *representation* method used for both the shape and its skeleton (from volumetric to boundary and point-cloud representations [82, 112]); and the algorithms used to compute the skeleton (for an extensive review, see [189]). In parallel with these extensions, the range of applications of skeletons has grown from simple low-level 2D computer vision tasks to shape segmentation [146, 169], feature detection [98, 144], shape matching [13, 186], path planning [66, 170], shape metrology [135, 172], information visualization [51, 212], and image compression [211].

As such, the skeletonization area has become a wide and complex research field. To better frame and scope our research question (and work we next perform to address it), we next present an overview of the main concepts (Sec. 2.1.1 and algorithms (Sec. 2.1.2) related to skeletonization and also to our research question. For a wider review of 2D skeletonization, we refer the interested reader to [172]. For an equally wide review of 3D skeletonization, we refer to several surveys in the area [39, 156, 189].

2.1.1 *Definitions*

Consider a continuous, compact, shape $\Omega \subset \mathbb{R}^n$. In practice, $n \in \{2,3\}$, which corresponds to the typical 2D and 3D shapes used in computer graphics and related fields. Let next $\partial\Omega$ denote the boundary of the shape.

The Euclidean distance transform $DT_{\partial\Omega} : \Omega \to \mathbb{R}_+$ of a shape Ω is defined as

$$DT_{\partial\Omega}(\mathbf{x} \in \Omega) = \min_{\mathbf{y}\in\partial\Omega} \|\mathbf{x}-\mathbf{y}\|. \tag{2.1}$$

Here, $\|\cdot\|$ denotes the Euclidean distance in $\mathbb{R}^n$. As such, the distance transform is often referred to as the Euclidean Distance Transform (EDT). In this thesis, we will only consider Euclidean distances, so we will omit the qualification 'Euclidean' for brevity. Intuitively, $DT_{\partial\Omega}(\mathbf{x})$ gives the shortest distance between any point $\mathbf{x}$ of the embedding space $\mathbb{R}^n$ and the closest point of the shape's boundary. Figure 2.1 shows a simple 2D binary shape and its corresponding distance transform. As visible, the distance transform values monotonically increase from $\partial\Omega$ towards the shape's interior. The points where $DT_{\partial\Omega}$ is locally maximal, also called *ridges* of the $DT_{\partial\Omega}$ [172], are of particular importance, as they will define the skeleton of Ω.

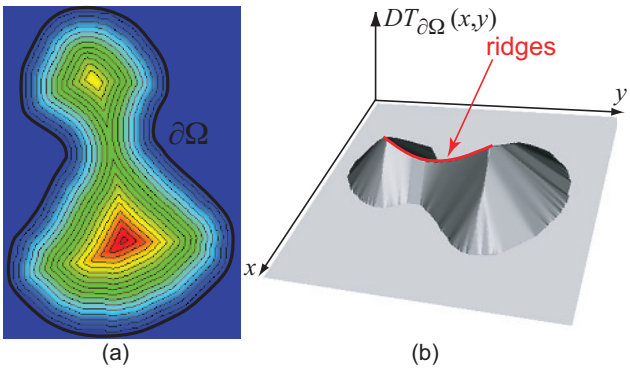


Figure 2.1: Two-dimensional shape (a) and its distance transform shown via isolines and color coding (a) and a height plot (b). Image adapted from [189].

A formal definition of the skeleton $S_{\partial\Omega}$ can be given using the distance transform as follows:

$$S_{\partial\Omega} = \{\mathbf{x} \in \Omega | \exists \{\mathbf{f}_1, \mathbf{f}_2\} \subset \partial\Omega, \mathbf{f}_1 \neq \mathbf{f}_2, \|\mathbf{x}-\mathbf{f}_1\| = \|\mathbf{x}-\mathbf{f}_2\| = DT_{\partial\Omega}(\mathbf{x})\}. \tag{2.2}$$

Since the skeleton is defined based on the Euclidean distance $\|\cdot\|$, it is also often referred to as an Euclidean skeleton. In other words, $S_{\partial\Omega}$ is the locus of points inside

the shape who admit at least two different closest-points $\mathbf{f}_1$ and $\mathbf{f}_2$ on $\partial\Omega$. These closest points are also called feature points of $\mathbf{x}$ [148]. Definition 2.2 is equivalent to Blum's original definition of the skeleton as the locus of centers of maximally inscribed disks in Ω [20]. The feature points are then the contact points of such maximally inscribed disks with the shape boundary $\partial\Omega$ [71].

The contact points define the so-called feature transform $FT_{\partial\Omega} : \Omega \rightarrow \mathscr{P}(\partial\Omega)$

$$FT_{\partial\Omega}(\mathbf{x} \in \Omega) = \underset{\mathbf{y}\in\partial\Omega}{\operatorname{argmin}} \|\mathbf{x}-\mathbf{y}\|, \tag{2.3}$$

where $\mathscr{P}$ denotes the power set.

Several other equivalent definitions to Eqn. 2.2 exist. For instance, Kimia *et al.* propose a so-called *grassfire analogy*: Imagine a fire front being started at the boundary $\partial\Omega$, which next advances inwards in Ω with isotropic speed. The locations where fire fronts coming from different parts of the boundary meet, define the skeleton [91]. This definition leads to direct, and simple, ways of computing the skeleton for 2D shapes [198], using *e.g.* the Fast Marching Method (FMM) [165] to propagate the fire front. Alternatively, the skeleton can be defined as the points in Ω where the gradient of the distance transform, $\nabla DT_{\partial\Omega}$, has a non-zero divergence [175], or looking for singular points of the higher-order moments of $DT_{\partial\Omega}$ [153]. Both above ways are essentially detectors for the aforementioned ridges of the distance transform graph. Finally, Giblin *et al.* propose to define the so-called *symmetry set* of a shape Ω, defined as the locus of centers of balls which are bi-tangent to the boundary $\partial\Omega$ [72]. The skeleton can be seen as a subset of the symmetry set, where only inscribed balls in Ω are considered [71]. This definition is suggestive to the property of skeletons of capturing the local symmetry of the object boundary by means of lower-dimensional structures. Finally, skeletons can be defined as subsets of points of the edges of the Voronoi diagram of a (typically dense) sampling of the boundary $\partial\Omega$ [3, 124]. This definition is based on the Voronoi diagram's property that places an edge at equal distance from two sites.

2.1.2 *Skeletonization methods*

Skeletonization methods are algorithms that compute typically approximate versions of the skeleton $S_{\partial\Omega}$ implied by Eqn. 2.2 or any equivalent skeleton definition, as those listed in Sec. 2.1.1. Understanding the different types of skeletonization algorithms that exist in the literature is essential for an efficient and effective application thereof to further problems such as shape processing. As such, we present next a (necessarily brief) review of the main types of skeletonization methods. For a recent state-of-the-art survey on 3D skeletonization, we refer to [189].

To achieve such a review, several axes need first to be defined to divide the search space of existing methods. We use next a taxonomy based on the following

axes: dimensionality of the embedding space $\mathbb{R}^m$, dimensionality of the skeleton, and type of sampling used to represent both the shape and its skeleton.

2.1.2.1 *2D skeletons*

Skeletons of 2D shapes can be computed by directly applying the various definitions listed in Sec. 2.1.1 to a sampled representation of Ω. Two main types of techniques exist here, based on the representation: Voronoi-based techniques represent Ω, or more precisely its boundary $\partial\Omega$, by piecewise-linear interpolation, *i.e.*, as a (closed) polyline. Next, the skeleton can be computed as a subset of the vertex-set of this polyline, specifically the Voronoi edges which are included in Ω [9, 124]. It has been shown that the thus computed 2D skeleton converges to the actual medial axis when the sampling of $\partial\Omega$ becomes infinitely dense.

A more popular representation uses a nearest-neighbor sampling of $\mathbb{R}^2$, *i.e.*, represents 2D shapes as digital binary pixel images over $\mathbb{Z}^2$, where one color denotes foreground (points in Ω) and the other color denotes background (points in $\mathbb{R}^2 \setminus \Omega$, also denoted as $\overline{\Omega}$). 2D image-based skeletonization methods are currently the most used in practice. These can be further classified into field-based methods and thinning methods. Field-based methods essentially compute a (typically local) detector function over Ω which indicates the presence of a skeletal pixel at the current location [53, 153, 175, 198]. Thinning methods simulate the grassfire evolution (Sec. 2.1.1) by iteratively removing pixels from $\partial\Omega$, typically in increasing distance order to the initial boundary [18, 172]. Removal stops when no further pixel can be pruned without disconnecting the skeleton. In general, distance-field methods produce smoother results and can handle shapes having a higher amount of boundary noise; in contrast, thinning methods are simpler to implement and, in general, faster.

Figure 2.2 shows the 2D skeleton of a leaf shape, computed by two methods: the Augmented Fast Marching Method (AFMM) [198] and the advection-based method in [83]. The color-coding of the computed skeletons outlines another very important skeletonization feature – skeleton simplification – which deserves further attention also in the context of our work. Simply put, skeletonization is not a continuous operation: Small changes in the input shape boundary $\partial\Omega$ can yield disproportionately large change in $S_{\partial\Omega}$, by creating so-called spurios *ligature* branches that correspond to small (convex) perturbations of $\partial\Omega$ [189]. Such perturbations may be side-effects of the sampling process used to create discrete (polygonal or pixel) representations of Ω, so they are typically only hindering further analyses. To eliminate such problems, one usually defines a so-called *importance* metric

$$\rho : S_{\partial\Omega} \rightarrow \mathbb{R}^+ \tag{2.4}$$

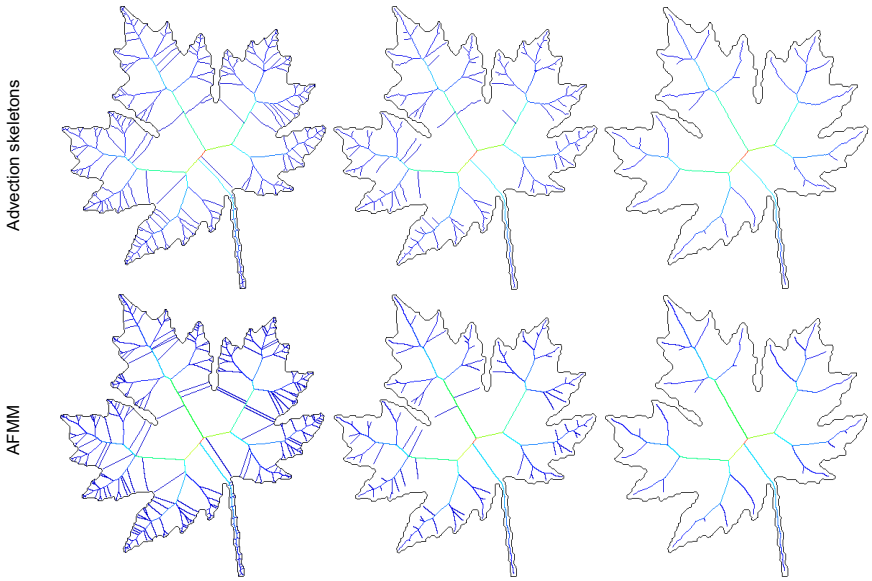


Figure 2.2: Skeletons of a 2D shape computed by two different methods (AFMM [198] and [83])). From left to right, skeletons are progressively simplified based on an importance metric. Image from [83].

so that all spurious branches receive a low importance value. The clean, also called pruned, skeleton can then be easily computed by upper-thresholding ρ – a process also known as skeleton regularization [189].

Many skeleton importance metrics have been proposed. Following Reniers *et al.*, we distinguish between local and global metrics. Local metrics $\rho(\mathbf{x})$ typically use only information situated in a small vicinity of the skeleton point $\mathbf{x}$. Such metrics are the distance-to-boundary or angle between feature vectors [3, 65, 175]. However simple and fast to compute, local metrics cannot distinguish between locally similar, but globally different, shape configurations, so their thresholding can inadvertently remove important skeleton points, which can ultimately disconnect the skeleton, changing its topology (see *e.g.* [145], Fig. 1). Global measures consider a larger vicinity of $\mathbf{x} \in S_{\partial\Omega}$ when computing $\rho(\mathbf{x})$. We are aware of essentially a single such metric, called the collapsed-boundary metric [53, 83, 124, 198]. In 2D, this metric assigns to a skeletal point $\mathbf{x}$ the length of the shortest boundary segment along $\partial\Omega$ delimited by the two feature points $\mathbf{x}_1$ and $\mathbf{x}_2$ of $\mathbf{x}$. As such, skeletal points corresponding to noise (small scale) elements on $\partial\Omega$ will have low importance, whereas points corresponding to important shape parts will have a large importance. Figure 2.2 shows two global importance metrics computed respectively by the AFMM [198] and advection-based [83] methods, by color coding (blue=low importance, red=high importance). As visible, more central skeleton points have a higher importance. Hence, by progressively upper-thresholding ρ, we can obtain progressively simplified skeletons, as the sequences of images in the same row show. An important additional property of the collapsed boundary metric is that it is monotonically increasing from the skeleton-branch tips to the skeleton center. As such, thresholding this metric always never introduces spurious disconnections in the skeleton. As we shall see in Sec. 2.1.2.2, similar local and global importance metrics can also be defined for 3D skeletons.

Given that our thesis' focus is on 3D shape processing, we conclude here our review of 2D skeletonization methods. As we shall see in the next section, several 3D skeletonization methods inherit properties and approaches from the 2D methods outlined above.

2.1.2.2 *3D skeletons*

Skeletons of 3D shapes know a slightly more complex taxonomy than 2D skeletons. We distinguish here between so-called 3D surface skeletons and 3D curve skeletons. Both skeleton types and their corresponding computation methods are overviewed next.

Surface skeletons:
Surface skeletons, also called medial surfaces, are defined following Eqn. 2.2 for

shapes $\Omega \subset \mathbb{R}^3$. They consist of a set of intersecting manifold surfaces with boundaries – analogously to 2D skeleton which consist of a set of intersecting curves of finite length. The skeletal manifolds meet, or intersect, along a set of so-called Y-intersection curves [28, 42, 102]. These are skeletal points that admit three feature points on $\partial\Omega$. For a more detailed characterization of the skeletal points based on their number and distribution of feature points, we refer to [71]. Surface skeletons are also known as 3D medial surfaces.

While the formal definition of surface skeletons is simple (Eqn. 2.2), efficiently and robustly computing such skeletons from complex, real-world, 3D shapes is still an open problem. The difficulties are due to the much more complex structure of such skeletons; the larger variability of noise types in 3D; and the larger size of sampled datasets required to accurately describe 3D shapes.

Surface skeletonization methods can be further classified based on the shape representation method used. As for 2D skeletons, we distinguish here between boundary representations (where shapes Ω are represented by a sampled description of their boundary $\partial\Omega$, typically as a 3D polygon mesh or 3D oriented point cloud); and volumetric representations (where shapes Ω are represented by a binary voxel volume in $\mathbb{Z}^3$.

Volumetric surface skeletonization methods are the oldest class. Methods in this class proceed much as their 2D image counterparts, *i.e.*, by using the distance transform analysis [49, 69, 92, 148, 153, 175]. Such methods can be extended, just as their 2D counterparts, by the computation of a global importance metric, which allows a continuous pruning of the surface skeleton [83, 145]. Other methods include 3D thinning methods [7, 17, 128, 128], which follow the idea of recursively removing voxels from $\partial\Omega$ in a typically distance-to-boundary-driven order. The main advantage of volumetric surface skeletonization methods is their relative implementation simplicity, which requires only a single type of data structure – a uniformly sampled field over $\mathbb{Z}^3$. However, such methods require considerably more memory than boundary-representation methods (discussed next) – typically $O(N^3)$ instead of $O(N^2)$ samples for a shape sampled with N samples on each Cartesian 3D axis.

Boundary-representation surface skeletonization methods, also known as mesh skeletonization methods, use piecewise-linear (polygonal) representation for both the shape boundary $\partial\Omega$ and the skeleton $S_{\partial\Omega}$. Moreover, sample positions can be located anywhere in $\mathbb{R}^3$ rather than on a fixed $\mathbb{Z}^3$ grid as volumetric methods do. This allows both a higher precision in approximating the desired shapes, and also more freedom in non-uniformly distributing sample points to achieve this approximation, *e.g.*, placing more samples where the shape has a higher curvature or spatial variation. As such, these methods are considerably more memory-efficient than volumetric methods – for the same available memory, they can approximate more complex shapes or approximate the same shapes more accurately. Finally, this rep-

resentation allows capturing the 'infinitesimally thin' nature of surface skeletons more faithfully than a voxel representation, where the surface skeleton will always have a finite thickness (equal to the voxel size).

Boundary-representation surface skeletonization methods are relatively more recent, and less numerous, than volumetric surface skeletonization methods. Notable methods in this class are the so-called ball shrinking methods [82, 112]. Such methods explicitly compute the locus of maximally inscribed balls in $\partial\Omega$, creating one ball per boundary tangent at each surface sample point $\mathbf{f}_1$, and finding the second point of tangency $\mathbf{f}_2$ by iterative optimization. Conceptually, such methods are quite simple, and admit trivial parallelization. To date, these methods can deliver the highest-resolution 3D surface skeletons. As shown in [82], such skeletons can be used to reconstruct very faithful approximations of complex 3D shapes, up to a level that is visually undistinguishable from polygon-based reconstructions. Other methods related to this class are the so-called union-of-balls methods [15, 73]. One important deficit of ball-shrinking methods is that they typically yield a *point cloud* approximation of $S_{\partial\Omega}$, which has only limited usability for further applications. Reconstructing the piecewise linear (polygonal) skeletal manifolds from such a point cloud is quite challenging, given that they can have an arbitrary large number of Y intersection curves [15, 81, 82, 96]. A recent survey of surface-skeleton reconstruction methods from skeletal point clouds, an operation also called *structuration*, outlines these difficulty in detail [43]. Another issue of boundary-representation methods is that they cannot directly store the distance transform $DT_{\partial\Omega}$ (Eqn. 2.2) for all points in Ω. Indeed, to do this, we need a volumetric sampling of Ω. This limits the types of skeleton detectors we can use – for instance, it is much harder to use distance-field-based detectors in a boundary-representation setting.

A different way of computing surface skeletons from boundary (mesh) representations uses Voronoi diagrams. This essentially extends the ideas shown earlier for the computation of 2D skeletons, *e.g.* in [9, 124]. The key observation is that, for 3D shapes, Voronoi diagram vertices can be filtered into the more restrictive Voronoi *poles*, which yield a sampling of $S_{\partial\Omega}$ [3, 46]. However, implementing such methods, including their underlying 3D Voronoi diagram algorithms, so that they robustly and efficiently handle *any* kind of non-uniformly-sampled 3D surface, is highly challenging. Even refined implementations of Voronoi diagrams, such as those provided in [3, 27], have difficulties to handle such 3D surfaces. Yet, the advantage of these methods is that they directly produce high-quality polygonal representations of the medial surface, without any of the aforementioned difficulties of the ball-shrinking methods.

Figure 2.3 shows several examples of 3D surface skeletons computed with various methods, both volumetric and mesh-based. As visible, surface skeletons for complex shapes have an intricate structure. Moreover, we see the large amount of small-scale details on the boundary of surface skeletons created by small-scale

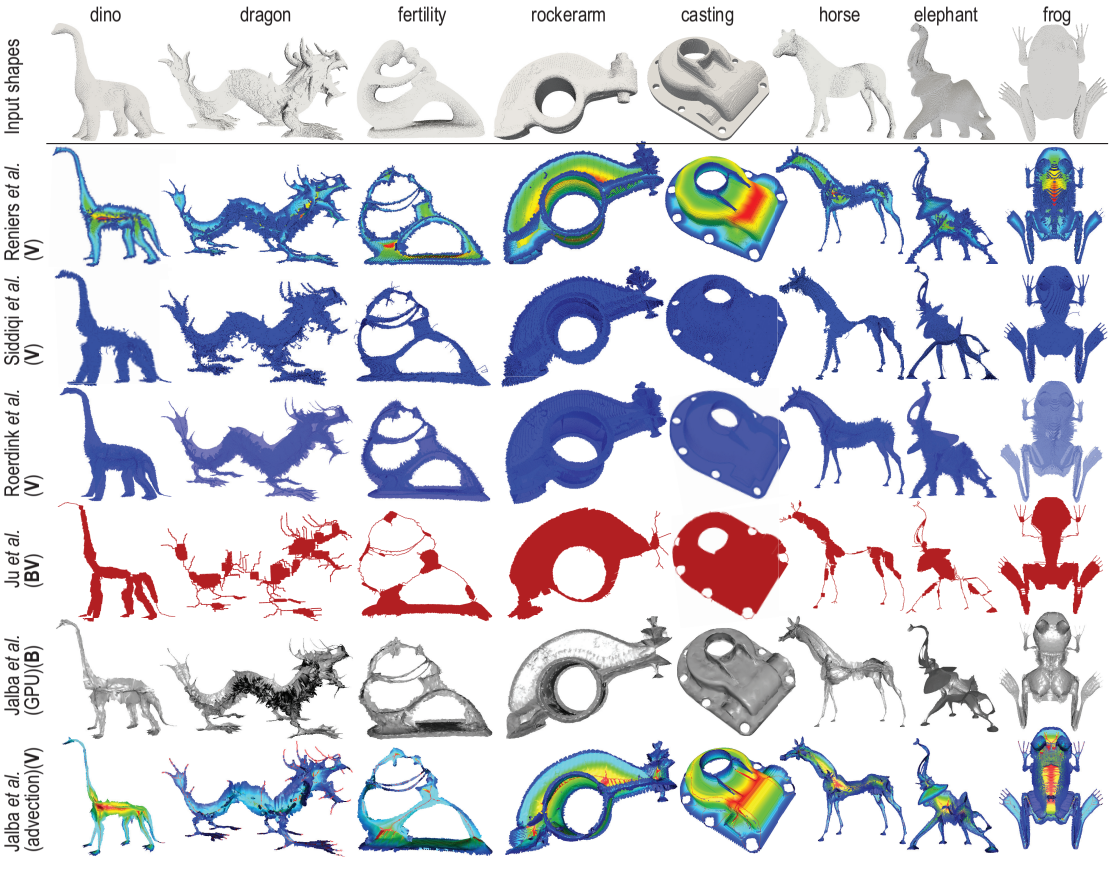


Figure 2.3: Examples of 3D surface skeletonization methods: Reniers *et al.* [145]; Siddiqi *et al.* [175]; Roerdink *et al.* [148]; Ju *et al.* [86]; Jalba *et al.* [82]; and Jalba *et al.* [84]. Methods are volumetric (**V**), boundary-based (**B**), or mixed volumetric and boundary-based (**BV**). Images adapted from [189].

details on the shape boundary.

Curve skeletons:
Besides surface skeletons, 3D shapes admit a second type of skeleton: curve skeletons. We denote the curve skeleton of a shape Ω next by $CS_{\partial\Omega}$. Curve skeletons are one-dimensional (curvilinear) structures embedded in $\mathbb{R}^3$, and locally centered within $\partial\Omega$. Numerous curve skeletonization methods have been proposed in the literature, using both volumetric techniques [179] and boundary-representation techniques [178], their number being considerably higher than the number of surface skeletonization methods in existence. This is explained by two factors. First, computing 3D curve skeletons is considerably simpler than computing 3D surface skeletons, due to the much simpler structure and topology of the former. Secondly, mapping the properties of curve skeletons to application tasks [39], such as segmentation, shape matching, and surface processing is considerably easier than mapping properties of the more complex surface skeletons to these or other applications.

However, curve skeletons also have important limitations, as follows.

First and foremost, there is still no universally accepted, formal, definition of curve skeletons – many ways of defining a 3D curve which is 'locally centered' within an arbitrary 3D shape exist [189]. Only few papers propose a formal definition. For instance, several authors define $CS_{\partial\Omega}$ as the locus of points $\mathbf{x} \in S_{\partial\Omega}$ which admit two different, equal-length, shortest paths γ_1 and γ_2 on $\partial\Omega$ between the feature points $\mathbf{f}_1$ and $\mathbf{f}_2$ of $\mathbf{x}$ [45, 82, 145]. However, it is not formally shown what this definition would imply in terms of 'local centrality' of $CS_{\partial\Omega}$. Another approach is to define the curve skeleton as the locus of points $\mathbf{x} \in S_{\partial\Omega}$ situated at equal distance from the boundary of the surface skeleton [81]. This boundary can be defined as

$$\partial S_{\partial\Omega} = \{\mathbf{x} \in S_{\partial\Omega} | type(\mathbf{x}) = A_3\}. \tag{2.5}$$

Here, $type()$ denotes Giblin's [71] classification of skeletal points into A_2^1 points (located inside the manifolds of $S_{\partial\Omega}$); A_k^1 points on intersection curves of these manifolds, also called Y-intersection curves [28, 42, 102]; and A_3 points, located on the skeleton boundary $S_{\partial\Omega}$, respectively. Giblin's classification is based on the so-called order of contact of a maximally inscribed ball centered at $\mathbf{x}$ with $\partial\Omega$: A_2^1 points have exactly two contact points; A_k^1 points have exactly k contact points; and A_3 points have an infinity of contact points – more precisely, the contact set is a circle sector or spherical segment. However mathematically crisp, computing this skeletal point classification is delicate for sampled (mesh or voxel) shapes, due to the sparsity of sampling of $\partial\Omega$ and the fact that a voxel model cannot capture the exact Euclidean distances between $\partial\Omega$ and $S_{\partial\Omega}$. While several heuristics

are proposed to this end [42, 98, 102], computing the skeletal point classification, and thus accurately detecting the skeletal boundary, is still very delicate. We will discuss several ways to alleviate such issues in Chapter 3.

The lack of a formal definition of curve skeletons makes it hard, if not conceptually impossible, to verify that a given method computes the 'right' curve skeleton (or a suitable approximation thereof), since we do not have a ground truth version of this curve skeleton. This issue is solved in practice by assessing the computed curve skeletons by means of structural properties which can be formally assessed, such as the fact they have to be composed of a set of 1D curves, the fact that they have to match the topology of the input shape Ω, and the fact that they have to be included in the surface skeleton $S_{\partial\Omega}$ (the last inclusion property can only be used when we have a way to compute the surface skeleton). The ill defined local centeredness property is assessed either visually (in a qualitative manner) [178], or by a voting process, *i.e.* computing the Haussdorff distance between the curve skeleton to assess and a set of curve skeletons computed by generally accepted methods (in a quantitative manner) [84, 179].

Secondly, curve skeletons capture less information of a 3D shape than the counterpart surface skeletons. Indeed, as noted earlier, the surface skeleton encodes the full definition of a 3D shape. More formally, the pair $(S_{\partial\Omega}, DT_{\partial\Omega}|_{S_{\partial\Omega}})$, also called the Medial Axis Transform ($MAT_{\partial\Omega}$), is a dual representation of $\partial\Omega$ [189]. The pair $(\mathbf{x} \in S_{\partial\Omega}, DT_{\partial\Omega}(DT_{\partial\Omega}))$ is also called a *medial atom*. Indeed, we can construct the MAT from $\partial\Omega$, as described above; and we can reconstruct Ω from the MAT as the union of balls of centers $\mathbf{x} \in S_{\partial\Omega}$ having as radii $DT_{\partial\Omega}(\mathbf{x})$ – the so-called union-of-balls method [26, 73]. Curve skeletons do not permit such a reconstruction, except in the case of objects having everywhere a perfectly circular cross-section (tubular shapes created by extruding a set of curves). As such, curve skeletons are mainly used to capture the topology of tubular 3D shapes (*e.g.*, their number of tunnels and protuberations). As we shall see in Sec. 2.2, this supports applications such as shape matching and shape segmentation for mainly rounded, organic, articulated shapes, but does not allow a wider treatment of either more general shapes or more general shape-processing tasks.

Figure 2.4 shows several examples of 3D curve skeletons computed with various methods, both volumetric and mesh-based. Compared to the surface skeletons of the corresponding shapes (Fig. 2.3, we see indeed that the curve skeletons have a much simpler structure. However, we also see that the variability of curve skeletons for the *same* shape, computed by different methods, is much higher than the variability of the corresponding surface skeletons. This is a direct consequence of the variability in curve-skeleton definitions which has been outlined earlier. Finally, as for surface skeletons, we see that curve skeletons also have a large amount of spurious branches, created by small-scale perturbations of the shape boundary $\partial\Omega$.

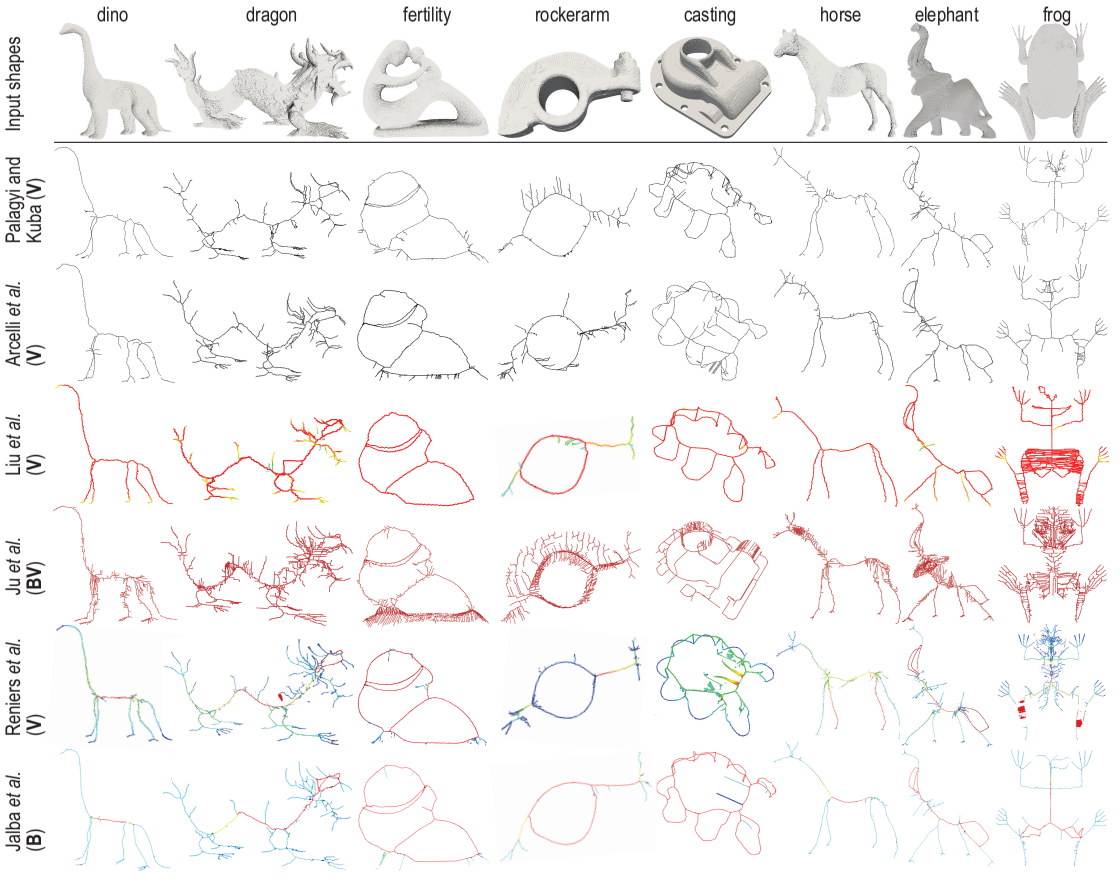


Figure 2.4: Examples of 3D curve skeletonization methods: Palagyi and Kuba [128]; Arcelli *et al.* [7]; Liu *et al.* [106]; Ju *et al.* [86]; Reniers *et al.* [145]; and Jalba *et al.* [82]. Methods are volumetric (**V**), boundary-based (**B**), or mixed volumetric and boundary-based (**BV**). Images adapted from [189].

2.1.2.3 *Skeletonization challenges*

As outlined in Chapter 1, our goal is to exploit surface skeletonization methods for applications in 3D shape processing. Clearly, the success of this endeavor depends on two factors:

- the suitability of surface skeletons for the types of applications considered;
- the ease of computing high-quality surface skeletons themselves.

The first factor is a conceptual one, which forms the essence of the work described next in this thesis. The second factor, however, is not, strictly speaking, part of our quest: Indeed, to be able to see how surface skeletons can support shape processing, we need in the first place to avail of methods that allow us to easily and efficiently compute good-quality surface skeletons. If such methods are not appropriately chosen, then it is hard to test our hypothesis (that surface skeletons can indeed support shape processing), since the skeletons we work with are suboptimal. We overview next several quality properties of surface skeletons and skeletonization methods that are relevant for this question.

The quality of skeletons and skeletonization methods are subtly interrelated aspects. Indeed, the quality of a computed skeleton $S_{\partial\Omega}$ depends, first and foremost, on the exact definition used for it. As mentioned earlier in Sec. 2.1.1, several largely equivalent definitions of surface skeletons exist. However, depending on how such definitions are exactly transposed into algorithms, differences in the resulting skeletons can follow. Separately, the quality of a computed skeleton depends on the various approximations used in the respective algorithms, starting with the representation model used for 3D shapes (volumetric or boundary-based).

Several papers treat the topics of skeleton(ization) quality, most notably being [39, 156, 178, 179, 189]. Overall, the following quality aspects are largely agreed upon as being relevant for surface skeletons (most such aspects are also relevant for curve skeletons; however, we exclude these from our discussion next, since we are interested in surface skeletons).

1. **Homotopy:** Following their definition (Eqn. 2.2), surface skeletons should have the same homotopy as the shapes they come from. This implies that $S_{\partial\Omega}$ has to have the same number of connected components, voids (cavities), and tunnels (loops) as Ω. This property is important for applications where we use the skeleton to reason about the shape's topology. Homotopy preserving can be problematic for both voxel and boundary-based representations when a too low sampling rate is used.

2. **Invariance:** Skeletons should be invariant to isometric transforms T applied to the input shape Ω. More precisely, the transform and the skeletonization

operation should be commutative, *i.e.*, $MAT_{T(\partial\Omega)} = T(MAT_{\partial\Omega})$. This property is important for virtually all applications using skeletons, as it allows one to flexibly define a shape with respect to its scale, translation, and rotation in the embedding space $\mathbb{R}^3$. In particular, this is important for shape matching applications, where one typically does not care about such transformations [186]. Boundary-representation skeletons are in general invariant with respect to such transformations, since they use a representation based on points $\mathbf{x} \in \mathbb{R}^3$, which is accurate up to floating-point precision. Volumetric skeletons are in general affected by such transformations, since they use a fixed-grid representation, typically aligned with the $\mathbb{R}^3$ axes. Indeed, it is in general impossible to have a binary (voxel) shape stay identical with respect to *e.g.* rotations. As such, the best volumetric representations can do is to minimize the effects of transformations to small ranges, typically one voxel.

3. **Thinness:** By definition, surface skeletons are infinitesimally thin objects, as they are manifolds (surfaces) embedded in $\mathbb{R}^3$. As such, skeletonization methods should try to compute skeletons which are as thin as the representation method used allows. For boundary-based representations, this is easy, since surface skeletons are here polygonal meshes [15, 73, 82, 112]. Volumetric representations have several issues here. Indeed, the best such representations can achieve is to construct a one-voxel-thin skeleton. This creates problems in areas where several skeletal manifolds meet, where it is hard to precisely say what one-voxel-thickness means. The problem is solved typically by applying various types of morphological (thinning) filters as postprocessing tools on the computed surface skeletons, to remove possibly spurious voxels from the skeleton [7]. Such filters propose different heuristics on how to locally define one-voxel-thickness.

4. **Centeredness:** By definition, every point $\mathbf{x}$ of a surface skeleton has to be the center of a maximally inscribed ball in $\partial\Omega$ (Eqn. 2.2). However, enforcing this property is not trivial, given that we use a sampled representation of $\partial\Omega$. Boundary-based methods fare relatively well here: They usually enforce centeredness by testing the distance of skeletal points $\mathbf{x}$ to all sample vertices of a point-cloud representation of $\partial\Omega$ [82, 112]. Formally, this allows a very accurate positioning of $\mathbf{x}$ with respect to a *subset* of $\partial\Omega$. Indeed, one does not usually consider here the piecewise-linear reconstruction of $\partial\Omega$, *i.e.*, one does not verify that the skeletal balls are inscribed with respect to the *polygons* that form $\partial\Omega$. This considerably simplifies the implementation of the skeletonization, but creates MATs whose radii can be larger than the true ones. A simple solution for this issue is to use a very dense (and uniform) sampling of $\partial\Omega$. However, this increases computational costs, as

such skeletons can then easily have hundreds of thousands or even millions of sample points, even for relatively simple input shapes [82].

The centeredness problem is even more complicated for volumetric representations. Indeed, the fixed voxel grid does not usually allow finding voxels which are *exactly* at equal distances from two different voxels on the boundary of a digital shape. The problem is illustrated in Fig. 2.5 for a simple axis-aligned 2D rectangular shape represented on a pixel grid (the same issues occur for 3D voxel shapes). If the rectangle is of even height, there are no pixels on the grid at equal distance from both the top and bottom edges. As such, the correct skeleton (shown in th e left image) cannot be approximated well by the digital skeleton (shown in the right image): The digital skeleton will either have a two-pixel-thickness along its central branch (gray pixels in Fig. 2.5), or fully miss all pixels on this branch.

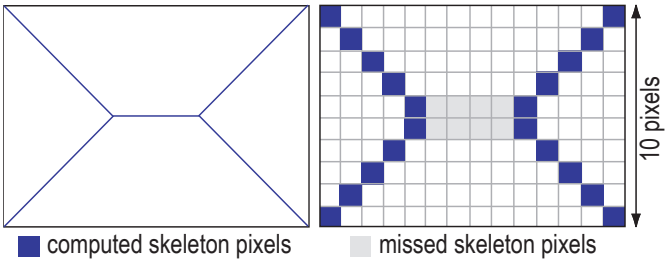


Figure 2.5: Centeredness problems digital skeletons. Image taken from [199].

The problem translates further in the fact that we cannot satisfy the homotopy, thinness, and centeredness properties on a volumetric grid. As such, the existing solutions propose various trade-offs of the above properties. For instance, one can compute an approximate thick skeleton, by relaxing the strict distance-equality condition $\|\mathbf{x}-\mathbf{f}_1\| = \|\mathbf{x}-\mathbf{f}_2\|$ in Eqn. 2.2, *e.g.* introducing a small tolerance of around the size of one voxel [145]. This method has been originally proposed for 2D shapes and their distance and feature transforms [139]. This creates a thick, but homotopic and centered skeleton. Next, this skeleton can be pruned to become one-voxel-thick and still connected, but necessarily not perfectly centered [7].

Centeredness issues exist also for the computation of the feature transform $FT_{\partial\Omega}$ (Eqn. 2.3). If we apply this definition on a voxel grid representation, we will only find a *subset* of points of the true $FT_{\partial\Omega}$. In turn, this can create problems when we need the actual number (and/or position) of these feature points to reason about the skeleton, such as when we want to compute a global importance metric (Eqn. 2.4). A solution to this problem follows the

same tolerance idea mentioned earlier – we compute a so-called Extended Feature Transform (EFT) by relaxing the strict 'closest point' condition in Eqn. 2.3 by a small tolerance of around one voxel. However, the impact of this tolerance is significantly higher for feature transforms than for computing skeletons, as it can create a very large number of irrelevant feature points. Various heuristics are proposed to balance between the number of captured feature points and the used tolerance [145]. We will investigate this issue further in Chapter 3, where we will propose our own EFT.

5. **Smoothness:** The manifolds with boundaries that constitute a surface skeleton are known to be piecewise second-order continuous $\mathscr{C}^2$ [134, 172]. Hence, the skeletons computed from approximate (volumetric or boundary-based) representations should obey this property as well as the representation allows it. Smoothness is, in theory, important for applications which assess skeletal manifold differential properties, such as curvature. However, we are not aware of such applications. More practically, smoothness is important for getting a good insight when visualizing the extracted skeletons. For boundary-based representations, visual smoothness can be achieved relatively easily by *e.g.* using a fine sampling of the skeleton, followed by classical normal interpolation and Gouraud or Phong shading [15, 81]. For volumetric representations, this can be done by using splat-based techniques to render the extracted voxel skeleton [84]. We will use the latter type of technique in our applications in the following chapters.

6. **Detail resolution and regularization:** The detail resolution property is related to the fact that both shapes and their skeletons are discrete (sampled) objects. As such, they cannot capture all details of the original continuous shapes and skeletons they aim to represent. A good-detail resolution method is defined as a method that manages to capture details (of the shape and skeleton) in a *controlled* manner; that is, the user should be able to specify the scale and/or type of details that should be captured, and the method should provide parameters that can be set for this to be achieved. This property is also known as *regularization*, as it is often used to eliminate small-scale noise from the skeleton. While detail resolution and regularization are sometimes seen as distinct properties [189], we prefer to consider them jointly, since it is hard to formally distinguish between noise and small-scale details.

   As explained earlier for boundary-based methods, using a denser sampling of $\partial\Omega$ typically creates a denser surface skeleton; within certain conditions, this sampled skeleton converges to the true continuous skeleton as the sampling density increases [3]. However, obtaining a dense sampling on the so-called *ligature* branches, that correspond to small convex bumps on $\partial\Omega$, is

technically challenging [82]. This issue manifests itself also for volumetric skeletons: While the entire space in Ω is densely and regularly sampled, this does not mean that increasing the sampling rate (resolution) will automatically make the computed skeleton converge to the true continuous skeleton. Indeed, consider the issue of the axis-aligned 2D rectangle shape in Fig. 2.5 discussed earlier: We can increase the grid resolution as much as desired, this will not capture the central skeletal branch if the shape's height is still an even number of pixels.

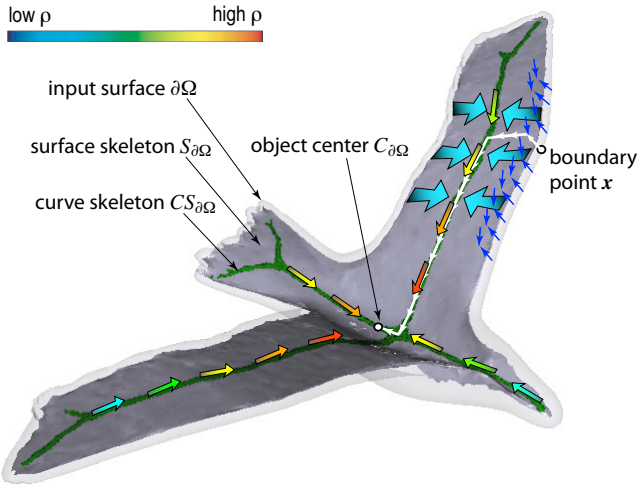


Figure 2.6: Defining 3D skeleton importance following a boundary-collapse principle. Image adapted from [84].

As mentioned, regularization is a subset property of detail-resolution, which is concerned with providing means to allow one to control how the produced skeleton is influenced by small-scale details present on the input boundary $\partial\Omega$. The by far most important aspect of regularization is eliminating spurious skeleton branches that are caused by small-scale details on $\partial\Omega$. This serves both eliminating branches which are caused by sampling noise, and also eliminating branches which are caused by 'true' shape details, which are however irrelevant for the application at hand. Regularization methods proposed for 2D skeletons (Sec. 2.1.2.1) have also been extended to 3D skeletons. Local regularization methods are largely identical to their 2D counterparts, *i.e.*, use local properties such as the distance-to-boundary, angle of feature vectors, or divergence of the distance-transform gradient to eliminate spurious skeleton points [65, 148, 153, 175, 184]. These methods have the same limitations as their 2D counterparts, *i.e.*, they cannot distin-

guish between a locally identical but globally different skeletal configuration. Global regularization methods largely solve this issue. As in 2D, all such metrics that we are aware of use a 'collapsed boundary' metric to assess how important a skeleton point is. Intuitively, this is equivalent to imagining a transport (advection) field that moves boundary points $\mathbf{x} \in \partial\Omega$ into the surface skeleton following some mass-conservation principle. Skeletal points which 'collect' more boundary points are deemed to be more important.

Figure 2.6 illustrates the idea: Imagine a transport vector field $\tilde{\tilde{>}} : \Omega \to \mathbb{R}^3$. Points $\mathbf{x}$ on the shape surface $\partial\Omega$ are carried by this field until they hit the surface skeleton $S_{\partial\Omega}$. Next, the vector field $\mathbf{x}$ carries these points towards the curve skeleton $CS_{\partial\Omega}$ which, as outlined in Sec. 2.1.2.2, is typically seen as a part of the surface skeleton. Finally, after the points reach $CS_{\partial\Omega}$, they are carried along the curve skeleton until they all collapse in a single point, denoted as the center of the object $C_{\partial\Omega}$ (this last collapse step makes sense only for genus 0 object whose surface and curve skeletons have a tree structure, *i.e.*, no loops). Given this advection process, the importance $\rho(\mathbf{x})$ of a (curve or suface) skeleton point can be defined as the amount of boundary points that 'flow' through $\mathbf{x}$.

Several methods exist that follow (more or less closely) the above boundary collapse principle to define global skeletal importance metrics. The first we know of was proposed by Reniers *et al.*, who define $\rho(\mathbf{x} \in S_{\partial\Omega}$ to be equal to the length of the shortest geodesic path $\gamma \subset \partial\Omega$ that connects the two feature points $\mathbf{x}_1$ and $\mathbf{x}_2$ of $\mathbf{x}$ [145]. Note that such geodesic paths were also used earlier by Dey and Sun to define curve skeletons, but not to compute importance metrics [45]. Both Reniers *et al.* and Dey and Sun use volumetric representations to compute such geodesics, by means of shortest-path algorithms such as Dijkstra's method [47], whose lengths are accurately computed using various digital estimators [93]. The metric of Reniers *et al.* has been recently extended to boundary-representation (mesh) skeletons [82]. For this, one needs to design and implement efficient and accurate shortest-path tracing algorithms on 3D mesh surfaces, which makes such methods quite complex. A final result in this area was proposed by Jalba *et al.* [84] for volumetric skeletons, who compute ρ and $\mathbf{v}$ by solving a system of partial differential equations (PDEs) that simulate a Navier-Stokes like advection of mass from $\partial\Omega$ in the field $\nabla DT_{\partial\Omega}$. This method elegantly generalizes the 2D AFMM importance metric to both 3D surfaces and curve skeletons, and is also much faster than *reniers08*, as no expensive shortest-path computations on $\partial\Omega$ are needed. All above boundary-collapse methods monotonically increase from the skeleton boundary $\partial\S_{\partial\Omega}$ to its center. In turn, this means that thresholding them by increasing values delivers increasingly simple, but connected, skeletons. As such, skeletons produced by such methods are sometimes also

called *multiscale* skeletons – a term first used in [53]. In Chapter 7, we present a novel method to compute such geodesic paths in an efficient way.

Overall, having a good regularization method is essential to obtaining clean surface skeletons, which can be next used in applications. In our following work, we shall use both the volumetric-based boundary collapse metric [84] and the fast mesh-based shortest-path metric [82] to regularize 3D surface skeletons. Moreover, in Chapter 7, we shall propose a new way to compute such a boundary collapse regularization metric, which combines simplicity and computational efficiency.

7. **Reconstruction:** As outlined in Sec. 2.1.2.2, the MAT of a 3D shape is a dual representation, *i.e.*, it allows one to reconstruct the shape Ω. This is important for the following reason: It formally tells us that no information is lost when using the MAT to analyze a given shape, as compared to analyzing the shape directly. Hence, if designing shape analysis or shape processing operations using the MAT is easier than doing these operations on the shape itself, this approach should not have limitations as compared to a direct shape analysis.

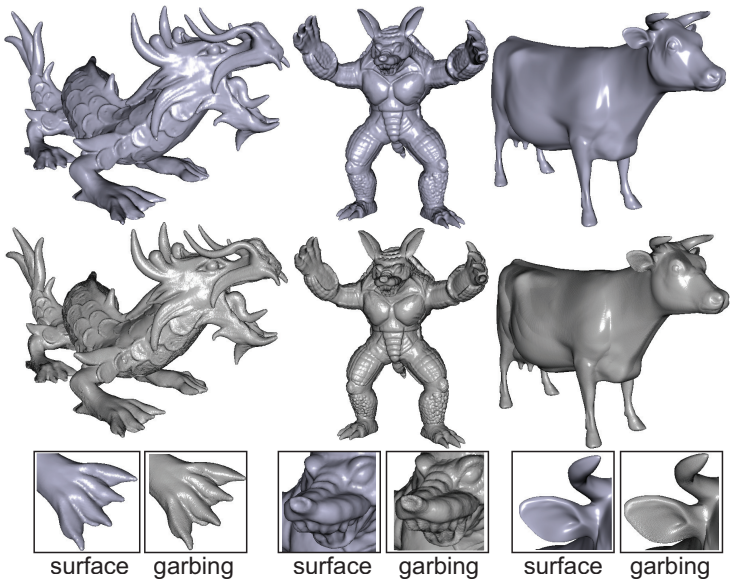


Figure 2.7: Reconstructing a shape from its surface skeleton using texture splatting. Top row shows a classical surface (polygonal) rendering. Middle row shows the splatting-based reconstruction from the surface skeleton. Bottom row shows details. Image taken from [82].

One well-known such an MAT-based shape processing operation is reconstructing the shape from its MAT, an operation also known under the name of *garbing* [82]. This can be done by simply computing the union-of-balls of centers $\mathbf{x} \in S_{\partial\Omega}$ having as radii $DT_{\partial\Omega}(\mathbf{x})$. Doing this by brute force 'drawing' of 3D balls can be expensive for both volumetric and boundary-based representations. As such, various methods have been proposed to accelerate the reconstruction. In volumetric space, one can compute the distance transform DT_S of the skeleton points $\mathbf{x} \in S_{\partial\Omega}$, by using *e.g.* the Fast Marching Method [165] to propagate an inflating front from $S_{\partial\Omega}$ outwards, and stop the propagation of each point $\mathbf{y}$ of this front when $DT_S(\mathbf{y}) = DT_{\partial\Omega}(\mathbf{x})$ [153]. The cost of this method is $O(N)$ for a shape Ω of N voxels. For boundary-based methods, a view of the shape Ω can be reconstructed from its point-cloud surface skeleton by splatting a radial depth billboard texture centered at every skeletal point $\mathbf{x}$, scaled to $DT_{\partial\Omega}(\mathbf{x})$. Given that the texture drawing is highly parallelized by the graphics card, this method runs in near-real-time even for skeletons having hundreds of thousands of points [82]. Figure 2.7 illustrates this method. As visible, splatting-based reconstruction can generate views of the input shape which are practically indistinguishable from a classical polygonal rendering of the surface $\partial\Omega$. However, a disavantage of this method as compared to volumetric reconstruction is that it only creates *views* of the 3D shape.

Reconstruction from the skeleton is strongly connected to skeleton regularization. Indeed, by using a (slightly) regularized skeleton in the reconstruction, one can obtain a shape on whose surface small-scale noise (and details) have been removed. More formally, all such details are replaced in the reconstruction by spherical segments. This offers an effective way of smoothing, or filtering, noise on 3D surfaces. We detail this topic further in Sec. 2.2.

8. **Scalability and ease of use:** The final properties – scalability and ease of use – that we discuss here relate to skeletonization methods rather than to skeletons themselves. Scalability relates to the computational cost required to compute surface skeletons of 3D shapes. This cost is a function of the shape's sampling resolution, and can be expressed in memory requirements, time needed for the computation, or a mix of both.

   Concerning memory, a first salient observation is that boundary-based methods are substantially more cost-efficient than volumetric methods. Indeed, for a given shape, having roughly N samples along any spatial axis, one needs about $O(N^2)$ samples (vertices) to represent $\partial\Omega$ as a mesh, and about $O(N^3)$ samples (voxels) to represent Ω as a binary volume. For large values of N, the costs of volumetric representation can become very high. For example, a volumetrically sampled binary shape Ω at resolution 1024^3, en-

coded with one byte per voxel, requires 1 GB RAM. An equivalent mesh describing $\partial\Omega$, which has roughly 1M triangles, would need about 24 MB RAM.

Concerning speed, however, the conclusions are more complex. Indeed, the speed of a skeletonization algorithm depends mainly on the type of algorithm, and much less on whether it uses a volumetric or mesh representation. For instance, the regularization methods presented in Reniers *et al.* [145] and Jalba *et al.* [82] are quite similar. However, the main reason why the latter method is faster than the former one is not the fact it uses a boundary representation, but the high parallelization achieved on the GPU. Another example is given by comparing the methods of Reniers *et al.* [145] and Jalba *et al.* [84]. While both methods are volumetric, the latter is considerably faster (over one order of magnitude) than the former. The main reason for this is the different approach taken for computing the boundary-collapse regularization metric.

Besides raw performance, however, one has to consider also the ease of use of the underlying methods. Apart from method-specific implementation details, volumetric methods are in general much easier to use (and extend) than mesh-based methods. As already explained, this is due to the much simpler data structures volumetric methods use – essentially just a number of voxel volumes. In contrast, mesh-based methods typically use a wide set of data structures such as point clouds, meshes, and spatial partitioning schemes for enabling efficient search [15, 73, 82].

Concluding the above analysis, we cannot find overwhelmingly strong advantages for volumetric or mesh-based methods. Each class of methods has its own advantages and limitations, and briefly outlined above. This is also reflected in the literature, with the same authors supporting both volumetric methods [84, 179] and alternatively boundary-based methods [82, 98]. Hence, we will study using both types of skeletons in our work next. Moreover, in Chapter 7, we will propose a new 2D and 3D multiscale skeletonization method.

2.2 SHAPE ANALYSIS AND PROCESSING

In computer graphics and computer imaging, *shape analysis* is (loosely) defined as the field concerned with detecting and quantifying various properties of shapes. Such properties are next useful by themselves, *e.g.*, to assess a collection of shapes from a given perspective; and also to support the construction of further *shape processing* methods that modify a given shape. In this context, 3D shape processing is defined as the set of operations that alter the properties of a 3D shape representation, as captured by a computer model. As such, processing can refer to mod-

ification of the shape's intrinsic attributes (geometry, topology, color, texture) or extrinsich attributes (position, orientation, size in the embedding $\mathbb{R}^3$ space)[130]. In the context of our work, and more generally in computer graphics, shape processing refers mainly to the modification of intrinsic parameters only.

Shape analysis and processing is an extremely wide field, which we do not aim to survey here. The interested reader can further consult various recent books on the topic [24, 40, 64]. Rather, given the context of our work, we present next a brief survey of 3D shape analysis and processing applications where 3D (surface) skeletons play an important role.

2.2.1 *Shape metrology*

Shape metrology is an important subclass of shape analysis. Given a shape $\Omega \subset \mathbb{R}^3$, metrology proposes a set of measurement operators $M(\Omega)$ that aim to quantify various metric properties of the shape. Such operators can be further classified into global and local ones. *Global* operators take as input the shape Ω only, *i.e.*, assess properties of the entire shape. Examples hereof are measuring the volume $|\Omega|$ and surface area $|\partial\Omega|$. *Local* operators take as input the shape Ω, but also several local positions $\mathbf{x}_i \in \Omega$ that describe the places of interest where measuring should occur. Examples hereof are geodesic distance, curvature, and local thickness. Local operators are typically more complex (and interesting) than global ones for shape processing purposes – our field of interest. Hence, we detail these below.

CURVATURE: Curvature of 3D surfaces $\partial\Omega$ can be defined in several ways. The simplest idea is to reuse the notion of curvature of a 2D planar curve $\gamma \subset \mathbb{R}^2$. For a point $\mathbf{x} \in \gamma$, the curvature $\kappa(\mathbf{x})$ can be defined as $1/R$, where R is the radius of the largest circle that is tangent to γ at $\mathbf{x}$ and approximates γ around $\mathbf{x}$ best (the latter condition is needed since there would be typically two such circles, with centers located on both sides of γ). Another way to define κ is by locally approximating γ by somr parametric arc-length representation $\gamma(s)$. Then, the curvature at s is defined as $\gamma''(s)$.

Given next a smooth surface $\partial\Omega$, a point $\mathbf{x} \in \partial\Omega$, the normal $\mathbf{n}$ of $\partial\Omega$ at $\mathbf{x}$, we can construct a so-called normal section plane $\pi(\mathbf{x})$ that contains $\mathbf{x}$ and is tangent to $\mathbf{n}$. Following this,

- The *normal curvature* of $\partial\Omega$ at $\mathbf{x}$ is then the curvature of the planar curve $\partial\Omega \cap \pi$ evaluated at $\mathbf{x}$;
- For a point $\mathbf{x}$, there exists an infinity of planes $\pi(\mathbf{x})$, which can be thought to 'rotate' along the axis $\mathbf{n}$. The maximal, respectively minimal, normal curvature values computed over this family of planes are called the *principal curvatures* κ_1 and κ_2. The orientations of the plane π for which κ_1 and κ_2 are

realized are also called principal curvature directions, $\mathbf{e}_1$ and $\mathbf{e}_2$ respectively, and are tangent to $\partial\Omega$ at $\mathbf{x}$. The values of κ_i and $\mathbf{e}_i$ can be computed using principal component analysis (PCA) of the so-called curvature tensor of $\partial\Omega$, and they can be further used to classify $\mathbf{x}$ as a planar, convex, concave, or saddle point[48];

- The *Gaussian curvature* is the product $\kappa_1\kappa_2$. It offers a simpler way to characterize the surface at a given point: Positive values indicate so-called locally convex (or locally concave, depending on the direction we look from) surfaces, *e.g.*, a sphere; negative values indicate so-called saddle points; and zero values indicate surfaces such as a plane or a cylinder;

- The *mean curvature* is simply the mean of the principal curvatures, *i.e.*, $(\kappa_1 + \kappa_2)/2$. Mean curvature best describes, intuitively, how convex or concave a shape is. For instance, a cylinder surface has a positive mean curvature, but a zero Gaussian one.

FEATURES: Curvature analysis is very important for shape processing, as it allows the detection of *features* on such surfaces. These are local surface elements which exhibit properties that distinguish them from the remainder of the surface, and which allow one to reason about and/or manipulate the shape. For example, high (mean) curvature regions indicate features such as edges, creases, or cusps, which represent salient details, and further allow *segmenting* the shape into relevant components (see Sec. 2.2.2 next), or matching various shapes[209] (see Sec. 2.2.3).

Computing curvature information on digital shapes is delicate, as the operation involves, by definition, computing second derivatives. As such, discretization resolution and noise filtering issues become very important, analogously to those that affect 3D skeleton computation (see Sec. 2.1.2.3, Regularization). Curvature is typically estimated by finite differences, which involve considering a few sample points (mesh vertices or voxels) around the point $\mathbf{x}$ of interest. To reduce noise, *i.e.* regularize curvature estimation with respect to the sampling resolution, many methods essentially apply low-pass filtering with a filter ϕ of finite-support radius R. Given curvature as a point function $\kappa : \partial\Omega \to \mathbb{R}^+$, we thus evaluate the regularized

$$\kappa_{reg}(\mathbf{x} \in \partial\Omega) = (\phi * \kappa)(\mathbf{x}), \tag{2.6}$$

where $*$ denotes signal convolution. Several such regularization schemens exist, see [117, 120, 195]. Anisotropic diffusion schemes can also be used to enhance detection of curve-like features, such as edges. For this, a diffusion (low-pass filtering) process is used, with a filter ϕ which is essentially stronger

(has more response) in the principal direction $\mathbf{e}_1$ (along which κ is maximum) than along the orthogonal direction $\mathbf{e}_2$ [32, 34]. For a review of the above, we recommend [196]. However, tuning regularization parameters is delicate: Too much regularization eliminates discretization issues, but filters out small-scale features, such as weak(er) and/or dense edges; too little regularization captures small-cale features, but also noise. We discuss such issues in the context of mesh segmentation in Chapter 6.

Besides curvature, other mechanisms can be used to detect surface features. Most importantly in our context, surface skeletons can do that. For 2D shapes, it is known that the tips (endpoints) of the skeleton $S_{\partial\Omega}$ (Eqn. 2.2) correspond, via the feature transform of the skeleton $FT_{S_{\partial\Omega}}$ (Eqn. 2.3), to convex points on $\partial\Omega$; similarly, concave points on $\partial\Omega$ correspond to tips of the branches of the skeleton of the complement shape $\overline{\Omega}$, *i.e.* the shape where foreground and background are inverted as compared to Ω [172]. This is extrapolated to 3D where the boundary curves $\partial S_{\partial\Omega}$ correspond, also via $FT_{S_{\partial\Omega}}$, to convex ridges, or edges, of $\partial\Omega$ [172]. Hence, if we can robustly detect the boundary of the surface skeleton, and we can 'backproject' these to the shape surface, we can find convex ridges of the latter.

Doing the above in practice is not entirely straightforward, since (a) detecting the boundary of a surface skeleton is not trivial, both for voxel-based representations [143] and for mesh-based representations [98]; and (b) the skeleton's feature transform $FT_{S_{\partial\Omega}}$ is a not a one-to-one mapping. Concerning point (b), indeed, for a sphere Ω, the skeleton $S_{\partial\Omega}$ is a single point $\mathbf{x}$, and $FT_{S_{\partial\Omega}}(\mathbf{x}) = \partial\Omega$. Several heuristics have been proposed to simplify the set of points delivered by $FT_{S_{\partial\Omega}}(\mathbf{x})$, so as to obtain one-dimensional features, such as the locations of edges on $\partial\Omega$ should be, for both voxel shapes [142, 144] and mesh shapes [98]. The main added-value of these detectors is their ability to extract *sharp* locations of even shallow (rounded) edges, without having the regularization problems posed by curvature estimation. Figure 2.8 illustrates this for several such detectors. We further illustrate and refine such detectors in Chapter 6.

Feature detection is an important part of the more involved shape segmentation operations. Simply put, segments are either specific types of features, or they are delimited by features. Shape segmentation is discussed next in Sec. 2.2.2.

GEODESIC DISTANCE: Given two points $\mathbf{x}_1 \in \partial\Omega$ and $\mathbf{x}_2 \in \partial\Omega$, the geodesic distance is defined as the length $\|\gamma\|$ of the shortest curve $\gamma(\mathbf{x}_1, \mathbf{x}_2) \subset \partial\Omega$ whose endpoints are $\mathbf{x}_1$ and $\mathbf{x}_2$. As such, this type of curve is also called the *shortest path* between $\mathbf{x}_1$ and $\mathbf{x}_2$. Note that, for two such points, there may be more than one shortest path γ – for example, consider the case of two diametrically opposed points on a sphere surface, when there is an infinity of shortest paths between them, each equal to a half grand circle. However, the length $\|\gamma(\mathbf{x}_1, \mathbf{x}_2)\|$ is a unique value. Computing shortest paths between points on a 3D shape sur-

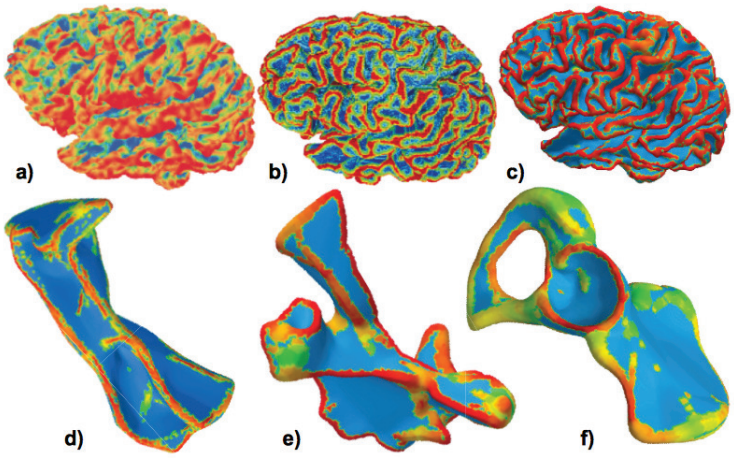


Figure 2.8: Edge-like feature detection on voxel surfaces with the methods in [195](a,d), [144](b,e), and [98] (c,f). Red indicates edges; blue indicates quasi-flat areas.

face is important for several applications, *e.g.*, mesh construction, texture coordinate generation, and data visualization [79, 136, 187]. Closer to our interest, such geodesics are the key ingredient of computing a global regularization metric for 3D skeletons (Sec. 2.1.2.3, Regularization), and also for defining 3D curve skeletons (Sec. 2.1.2.2, Curve skeletons). We will use such mesh-based shortest-path computations when presenting our method to compute mesh shape segmentations in Chapter 6.

Computing (the length of) shortest paths on discretized versions of $\partial\Omega$ specializes next according to the discretization type. For piecewise-linear representations (polygonal mesh surfaces), several such methods are proposed, *e.g.* tracing streamlines from $\mathbf{x}_1$ in the gradient of the distance-transform $DT_{\mathbf{x}_2} : \partial\Omega \rightarrow \mathbb{R}^+$ (computed using Fast Marching Methods) until they reach $\mathbf{x}_2$ [132], further enhanced with heuristics for speed-*vs*-accuracy trade-offs[187] and also implemented on the GPU [82]. Such methods are however quite complex, given the fact that geodesics need some technical adaptations to be defined on polyhedral surfaces [79, 136]. We used the method in [82] for our mesh shape segmentation in Chapter 6.

For voxel shapes, the method of choice proceeds as follows. First, one represents the voxelized surface as an adjacency graph G whose nodes are the voxel centers and edges are the (typically 27-neighbor) adjacency relations. Each edge has a weight representing its Euclidean length. Next, the Dijkstra algorithm [47] is used to find the shortest path in G between the two given vertices $\mathbf{x}_1$ and $\mathbf{x}_2$. The speed search can be enhanced by using A^* heuristics [145], where the heuris-

tic search cost is the length $\|\mathbf{x}_1 - \mathbf{x}_2\|$. In that paper, Reniers *et al.* also propose a caching scheme that is meant to further accelerate the computation of many shortest paths on a surface $\partial\Omega$, by reusing parts of the earlier computed paths rather than tracing from scratch. However, the scheme is not further detailed, and thus it is hard to replicate. To ensure accurate length computations on digital voxel surfaces, the graph edges can be further weighted by coefficients, following Kiryati and Szekely's scheme [93]. In detail, edges of length 1 are weighted to become 0.9016; edges of length $\sqrt{2}$ become 1.289; and edges of length $\sqrt{3}$ become 1.615, respectively. We have used this scheme in our shortest-path length estimations in Chapters 3 and 4, and also provided additional speed-ups for the path tracing.

THICKNESS: Measuring the thickness of a shape is important for many applications such as 3D metrology, where one needs to automatically determine if manufactured parts fit certain dimension constraints. For instance, in [200], this is used to determine if shapes created by 3D printing are locally thick enough to resist the manufacturing process. Estimating shape thickness is also used to compute so-called *shape descriptors* for the process of 3D shape matching [160], as discussed separately in Sec. 2.2.3.

Shape thickness is, by definition, a local property, as there is no single thickness value that can characterize an entire shape, in general. Given a shape Ω and a point $\mathbf{x} \in \Omega$, one well-known definition of the thickness $t(\mathbf{x})$ is

$$t(\mathbf{x}) = 2 \cdot \max(\{R | \forall \mathbf{x} \in \Omega : \mathbf{x} \in B(\mathbf{p}, R) \subset \Omega\}) \tag{2.7}$$

that is, the diameter of the largest inscribed ball in Ω that contains $\mathbf{x}$ [50]. Evaluating Eqn. 2.7 can be done by computing the distance transform $DT_{S_{\partial\Omega}}$ of the shape, marking all points $\mathbf{y} \in \partial\Omega$ with the value $\varepsilon(\mathbf{y} =) DT_{S_{\partial\Omega}}(\mathbf{s})$ of the closest skeleton point s to $\mathbf{y}$, and next setting $t(\mathbf{x})$ to the value $\varepsilon(\mathbf{y})$ of the closest surface point $\mathbf{y}$ to any interior point $\mathbf{x}$. Thys yields a thickness estimation for both surface points and interior points. For this, we however need a very accurate (centered) computation of the surface skeleton $S_{\partial\Omega}$, which, as explained in Sec. 2.1.2.3, can be difficult. A simpler method uses morphological opening and closing operations to find parts of Ω which are thinner than a user-given value τ[200]. However, this evaluates the so-called threshold sets $\{\mathbf{x} \in \Omega | t(\mathbf{x}) \geq \varepsilon\}$ rather than the full thickness signal. Concluding, surface skeletons are also useful for thickness evaluation. However, given our limited available time, we did not investigate this issue further.

2.2.2 *Shape segmentation*

Segmenting a shape Ω means producing a subset of *segments* $C_i \subset \Omega$ that describe parts of the shape which are relevant for specific applications. At the highest level,

thus, shape segmentation enables easier treatment of the shape information: Indeed, by definition, a segment is smaller than the shape it belongs to. Thus, a segment is arguably also simpler than the entire shape. As such, analyzing the segment should be easier (and/or faster) than analyzing the entire shape it comes from. Shape segmentation can be further classified according to several axes:

BOUNDARY OR INTERIOR: A segmentation can partition either the boundary $\partial\Omega$ or the interior (volume) Ω of a shape. In general, volume segmentations imply also boundary segmentations, as we can create boundary segments $C_i^{\partial\Omega} = \Omega \cap C_i^{\Omega}$ by considering boundaries of volume segments C_i^{Ω} which correspond to the shape boundary. Skeletons an be used to create both types of segmentations. Interestingly, surface skeletons (of dimension 2) create surface segmentations (segments have dimension 2) [142] and curve skeletons (of dimension 1) usually create volume segmentations (segments have dimension 1)[12, 141]. That is, the dimensionality of the segments plus the dimensionality of the skeleton is one larger than that of the embedding space ($\mathbb{R}^3$). Given that, as we explained, volume segmentations can be converted to surface segmentations; that the latter are more common in 3D shape processing; and that our focus is on *surface* skeletons, we focus on surfacce segmentations in this thesis.

REPRESENTATION: As shapes can be represented either as voxels or meshes, segmentation methods naturally split into two classes. Typically, volume segmentations require voxel representations; and boundary segmentations require (only) surface representations. Since we are interested in both types of shape representations, we explore next both voxel (Chapters 3 and 4) and mesh-based (Chapter 6) segmentations.

PARTITIONING: Non-partitioning segmentations deliver a set of segments which may either overlap, *i.e.*, $\exists i \neq j | C_i \cap C_j \neq \varnothing$, or not fully cover the entire shape, *i.e.*, $\cup_i C_i \neq \Omega$. An example of non-partitioning segmentation is given by hierarchical basis functions described in [33]. Although in the respective paper such overlapping segments are further processed to create non-overlapping ones, the core of the method is non-partitioning. In ontrast, partitioning segmentations deliver a set of segments which are non-overlapping, *i.e.*, $\forall i \neq j, C_i \cap C_j = \varnothing$, and fully cover the shape, *i.e.*, $\cup_i C_i = \Omega$. Partitioning segmentations are, in general, easier to use and cover more use-cases. As such, we focus only on partitioning segmentations in this thesis;

SCALE: Single-scale segmentations deliver a single partitioning $\{C_i\}$ of a shape. Multiscale segmentations deliver a set of such partitionings, where each partition corresponds to a so-called *scale*. Fine-scale partitionings capture smaller,

but more detailed, shape parts, and typically have more segments. Coarse-scale partitionings capture larger, less detailed parts, but have fewer segments. The scale is typically controlled by user-specified parameters of the process. Multiscale segmentations are more powerful, as they allow one to control the level-of-detail to work on. Regularized skeletons are multiscale (see Sec. 2.1.2.3, Multiscale); as such, they can produce multiscale segmentations[141, 142]. Given this, we focus next on multiscale segmentations;

PART OR PATCH: Segmentation methods can also be classified according to the nature of the segments they produce. *Part-based* methods aim to separate a shape into its 'natural' parts (segments), as perceived *e.g.* by a human [23]. As such, these methods are used mainly with organic, natural, shapes such as scans of body parts, silhouettes, or other articulated shapes. Usually, parts also correspond to volumes – if we think *e.g.* of partitioning a human hand, we think of detecting its fingers and palm and not the *surfaces* thereof. *Patch-based* methods aim to separate parts of a surface that show little internal variation but large external variation, such as quasi-flat regions separated by creases. As such, these methods are used mainly with non-articulated shapes. The emphasis put on flatness (curvature) also makes such methods applicable to faceted, synthetic, shapes such as engine parts or other man-made objects. Given that parts are defined based on surface properties, part-based segmentations fall largely in the class of surface partitioning – if we think *e.g.* of partitioning a cube, we think of separating its six faces, and not cutting it into smaller volumes. Both part-based and patch-based segmentation methods can be supported by skeletons, see *e.g.*[141, 142].

We choose next to discuss segmentation methods based on the part *vs* patch criterion, as follows.

2.2.2.1 *Part-based segmentation*

As outlined, parts are typically regions of a shape that a human would distinguish as different fron their surroundings. Following computer vision work, it has been seen that a part is a region that 'sticks out' of the rump of a shape, representing a bump, or protrusion. More formally, a part is a high-curvature region of the shape which is separated, boundary-wise, by a band of low curvature – see the *minima rule* in [23, 77, 177]. Many methods exist in this class, as follows (see also Figs. 2.9, 2.10, and 2.11 for several examples).

Li *et al.* [103] sweep the curve skeleton of a shape by a series of planes π_i locally orthogonal to the skeleton's tangent vector at different skeleton points $\mathbf{x}_i$, and compute a series of *cuts* $\Gamma_i \pi_i \cap \partial\Omega$. When two consecutive cuts have highly different lengths $\|Gamma_i\|$ and $\|Gamma_{i+1}\|$, a transition in the structure of the

shape is thought to take place, so a segment border is identified. However, this method constrains the segment borders ∂C_i to be planar curves.

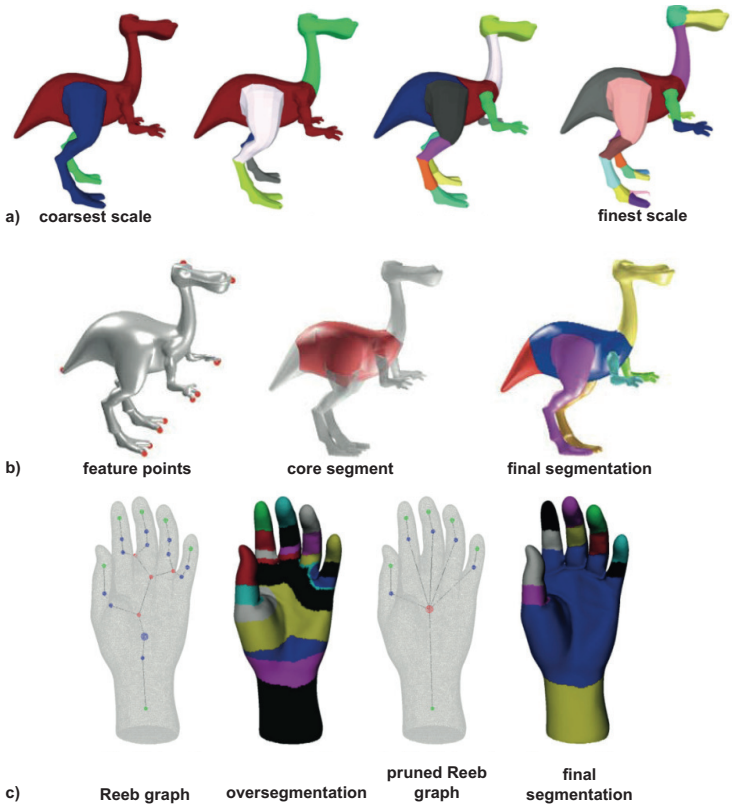


Figure 2.9: Part-based segmentation examples. (a) Fuzzy clustering [87]; (b) Feature points and core extraction [88]; (c) Reeb graphs [201]. See Sec. 2.2.2.1.

Katz *et al.* [87] segment a surface by bottom-up hierarchical clustering of small quasi-flat parts. Clustering is driven by a part similarity based on curvature properties. As such, this delivers parts which are relatively flat and separated by high-curvature creases. Given the hierarchical process, the result is a multiscale segmentation. They next refine the method to use feature points and core extraction in [88]. Key to this method is a pre-detection of so-called feature points, *i.e.*, salient points of $\partial\Omega$ which should determine different segments, if possible, such as tips of protrusions (which roughly correspond to the endpoints of a 3D curve skeleton of the shape). The method can also create a multiscale segmentation.

Tierny *et al.* [201] use Reeb graphs to produce multiscale segmentations [201]. A Reeb graph is essentially a skeleton which, in contrast to the Euclidean skele-

tons implied by Eqn. 2.2, capture shape topology (branching parts) but not shape geometry. Next, shape parts are computed to correspond to the Reeb graph edges.

Shapira *et al* [169] proposed a shape-diameter function (SDF). The SDF is a local estimation of the shape's diameter at each boundary point. Note that this is strongly related to the notion of shape thickness discussed in Sec. 2.2.1. Next, the boundary $\partial\Omega$ is clustered based on similar SDF values. Thus, segments are implicitly defined as similar-thickness regions of $\partial\Omega$. Since thickness is a volumetric property, this method can be seen as a part-based segmentation method.

Golovinskyi *et al.* [74] propose randomized cuts for segmentation. Key to this method is the observation that earlier methods are quite sensitive in terms of the delivered segmentation, in function of the input shape (or small changes thereof) and algorithm parameters. As such, they propose to compute a large set of segment borders, using cuts generated following a variation of [87], or by k-means clustering. Next, segments are found from this large set of 'randomized cuts' by maximizing the probability that segment borders share the same mesh edges in $\partial\Omega$. However, the brute-force search approach of the method (over the segmentation space) makes it quite expensive.

A separate class of methods use Euclidean curve skeletons (Sec. 2.1.2.2). The intuition here is simple: Following the Reeb graph approach [201], one detect the junction points of a curve skeleton, and uses these to place separating cuts to create segment borders. For this, Au *et al.* use an efficient mesh-based method to compute the curve skeleton [12]; next, junctions are detected and minima rules, like in [87], are used to create the separating cuts. Reniers *et al.* [146] use a similar method, but a different way to compute voxel-based curve skeletons, following [145]. However, this method suffers from oversegmentation. They alleviate the problem by proposing a so-called 'junction rule', by analogy with the minima rule discussed above. The junction rule filters out curve-skeleton junctions which do not encode part-rump boundaries, and reduces oversegmentation signifiantly [141]. Both approaches of Reniers *et al.* use geodesic curves as cuts (segment borders). As such, borders are smooth and tight by construction, even for noisy shapes. We will exploit this idea into our own segmentation work in Chapters 3 and 4. Finally, Serino *et al.* refine [141] by further classifying curve-skeleton points into single points, simple curves, and complex sets. This further reduces oversegmentation and makes the segmentation also robust to discretization noise [162, 163]. Finally, Kustra *et al.* segment 3D mesh shapes by detecting the manifolds of the surface skeleton and back projecting groups of manifolds that are deemed to belong to the same segment to the input surfacec $\partial\Omega$ [98]. However, this method is quite complex to implement and contains a significant set of heuristics.

A particular challenge of part-based segmentation methods resides in the ambiguous definition of what a 'part' is. Even for relatively simple objects, such as the horse in Fig. 2.11, different methods yield quite different results. Moreover, we

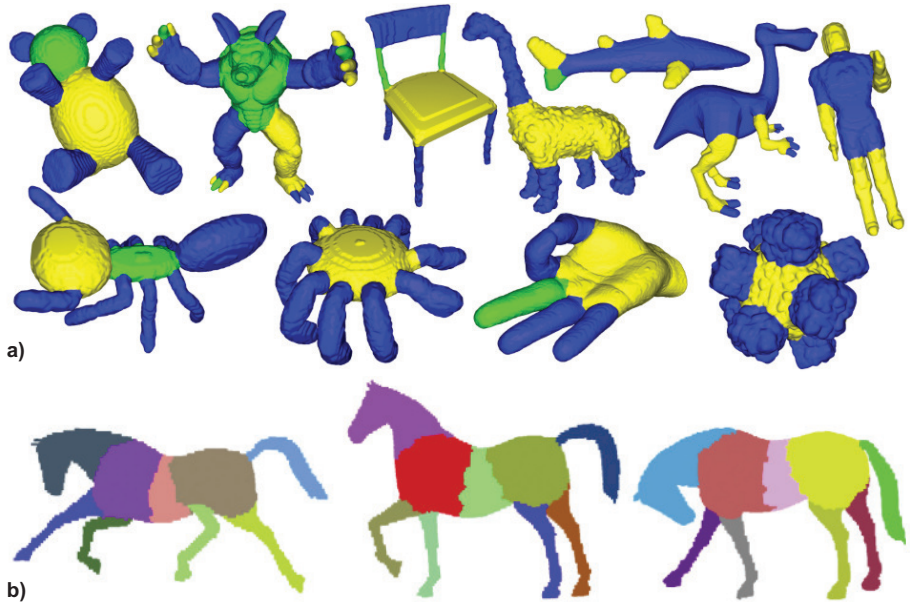


Figure 2.10: Additional part-based segmentation examples. (a) Junction rule [141]; (b) Method of Serino*et al.*[163]. See Sec. 2.2.2.1.

see that skeleton-based methods (Fig. 2.11b-f) yield arguably more natural results. The result in Fig. 2.11f is produced by our own surface-skeleton-based method, which we will describe in Chapter 3. Given such results, from an application perspective, we can state that part-based segmentation is a still active field of research, and that skeleton-based segmentations are a competitive approach.

Given our focus on surface skeletons, we see that, although many part-based segmentation methods use curve skeletons and Reeb graphs, a single method exists which uses surface skeletons [98]. However, this method does not produce patch-based segmentations, and only handles mesh-based representations. As explained in Sec. 2.1.2.3 (Reconstruction), only surface skeletons fully encode a shape's geometry. As such, using such skeletons is interesting for part-based segmentation. Additionally, it is interesting to be able to handle voxel shapes too. We show how this can be done in Chapters 3 and 4.

2.2.2.2 *Patch-based shape segmentation*

An early method in this class explicitly searches for fitting simple 3D primitive shapes to the input shape Ω to segment[10, 11]. The key idea is that the structure of such simple primitives will show up in the final segmentation. For instance,

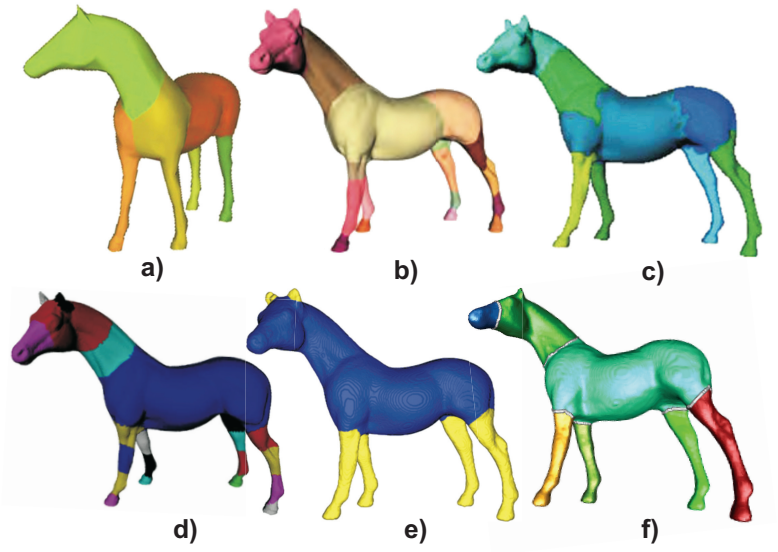


Figure 2.11: Comparison of six part-based segmentation methods for a horse shape. (a) [107]. (b) [105]. (c) [10]. (d) [201]. (e) [141]. (f) Our own method described in Chapter 3. Method (a) does not use skeletons. Methods (b-e) use curve skeletons. Method (f) uses surface skeletons.

if the shape Ω has sharp edges and admits fitting a cube, then the cube's sharp edges will show up as aligned with the actual shape edges in the final segmentation. The method uses an iterative clustering technique that collapses mesh edges so as to merge its faces, much like in mesh decimation approaches [68]. The clustering process is driven by a cost function which measures how well we can fit a primitive (plane, sphere, cylinder) on the existing mesh faces which are neighbors. Hence, quasi-flat parts of the input shapes are quickly fitted to one of the available primitives. As illustrated in Fig. 2.12a, the method works well for synthetic shapes (which admit a decomposition into the set of proposed primitives), but far less well for natural, irregular, shapes.

Skeletons have also been used for patch-based segmentation. As already mentioned, Reniers *et al.* [142] compute surface skeletons of voxel shapes using the method in [145] and back-project these to the input surface $\partial\Omega$ to find locations of convex edges. These are then linked to create a partitioning of $\partial\Omega$ into patches. To handle concave edges (dents), the skeleton of the complement shape $\overline{\Omega}$ is computed and processed similarly, and finally the two sets of patches are merged. The method shows good results, but is quite complex. Moreover, computing the skeleton of the complement is quite expensive, since one needs to skeletonize a voxeliza-

tion of a bounding box encompassing the entire shape Ω. Figure 2.12 illustrates both [10] and [142].

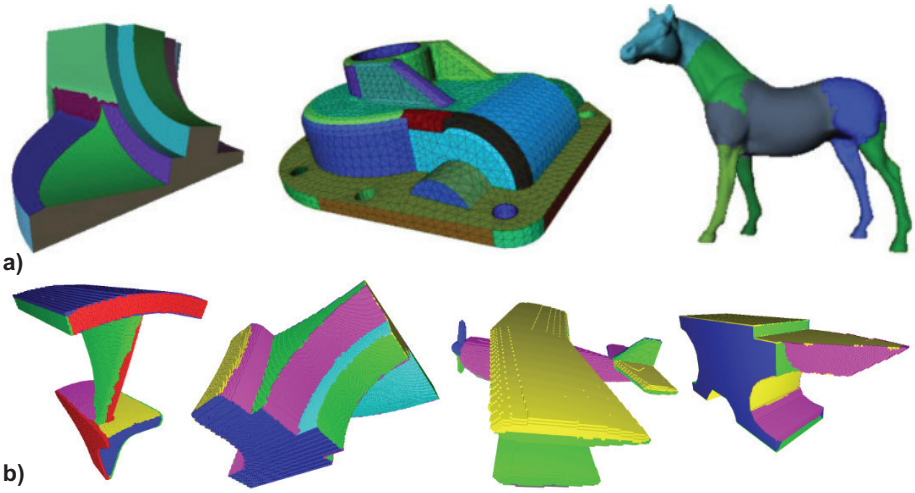


Figure 2.12: Patch-based segmentation examples. (a) Primitive fitting [10]. (b) Surface-skeleton method [142]. See Sec. 2.2.2.2.

2.2.3 *Shape matching and retrieval*

Shape matching is concerned with evaluating the similarity of two shapes. More formally, given two 3D shapes Ω_1 and Ω_2 in the space $\mathscr{O}$ of all possible 3D shapes, matching aims at computing a function $\sigma : \mathscr{O} \times \mathscr{O} \rightarrow \mathbb{R}^+$ so that $\sigma(\Omega_1, \Omega_2)$ reflects the *perceived* similarity of Ω_1 and Ω_2. That is, when a human would say that two shapes are very similar, σ is high; whereas when one would say that two shapes are very different, σ is low. Note that the problem can also be formulated by defining a distance, rather than a similarity, function. In this case, the distance is the reciprocal of the similarity function. Given the similarity σ, we say that two shapes Ω_1 and Ω_2 match well when $\sigma(\Omega_1, \Omega_2)$ is high.

One of the most important applications of shape matching is to support content-based shape retrieval (CBSR), which can be formulated as follows: Given a collection of shapes $C = \{\Omega_i\}$, and a so-called query shape Ω, find the most similar shape to Ω in C. This can be generalized to finding the n most similar shapes to Ω. CBSR aims at providing a search mechanism for 3D shapes similar to Google's search mechanism for text and images.

When C is large, and there are many queries, it becomes impractical to evaluate σ over all pairs $(\Omega, \Omega_i) | \Omega_i \in C$. This is not only due to the collection size, but also

the typically high cost of computing σ given two complex, high-resolution, shapes. An important aspect that makes this cost high is that we are not looking only for exact matches: For instance, if Ω would be the horse model in Fig. 2.12a (right), all three horses in Fig. 2.10b should yield high similarity scores. More formally, we are looking for a similarity metric σ which is not sensitive to transformations that do not (largely) affect the perceived nature of the shape. Such transformations include rigid ones (scaling, translation, rotation), but also different sampling resolutions, and pose (articulation).

Given the above, a key query for CBSR is the development of efficient and effective metrics σ. This is typically approached by computing a so-called shape *descriptor*, which is usually a high-dimensional value, or vector, $d(\Omega) \in \mathbb{R}^d$. Having such a descriptor reduces σ to computing the distance between two d-dimensional vectors $d(\Omega_1)$ and $d(\Omega_2)$. This can be done very efficiently using various distance metrics, such as *e.g.* using Euclidean, cosine, Helinger [160, 160], or Earth Mover (EMD) [150, 152] distances. As such, the quest is reduced to finding a good descriptor function d.

Shape descriptors can use different types of information, as follows:

Geometry: A first important one is the geometry of the shape, *i.e.*, the set of sample points $\{\mathbf{x}_i\}$ that describe $\partial\Omega$. For instance, Osada *et al.* [125] proposed a distribution that captures properties such as angle and distance determined by random pairs of points on $\partial\Omega$. For keeping rotation invariance, Kazhdan *et al.* [89] propose to use spherical harmonics [89], and the further improve this to include reflective and rotational (axial) symmetry [90]. Thickness measurements are also used as a descriptor [157].

Histogram descriptors refine the above by capturing the distribution of values of a geometric metric over a shape. For example, Liu *et al* [108] propose a directional histogram that is similar to [157]. This was reently extended by Schmidt *et al.* [160] to capture both thickness and so-called depth complexity (number of intersections of a ray with the shape) as a 2D histogram. This captures, up to a certain extent, both geometry (thickness) and topology (concavities).

Appearance: When available, information such as lighting, colors, and texture on $\partial\Omega$ can be effectively used. For instance, the DB-VLAT descriptor in [194] combines dense SIFT descriptors taken from several 2D shape views. Since SIFT descriptors can capture corners and edges quite well, and such details are a function of texture and color gradients, DB-VLAT can effectively integrate such additional sources of information.

Skeletons: As explained, skeletons capture both geometry and topology; if regularized, they are not sensitive to small-scale shape details; and they are by definition

invariant to affine transformations. As such, they have been used since long for CBSR. For 2D shapes, Sebastian *et al.* [161] extend the shock graphs proposed by Siddiqi *et al.* [173] and corresponding shock grammar [171] for 2D shape matching. This method can capture geometry well, but has shortcomings in capturing topology. For 3D shapes, Sundar *et al.* [186] pioneered (to our knowledge) the use of use 3D curve skeletons to match shapes. For this, they extract individual skeletal branches to construct a graph, and next use graph-matching algorithms to compare such shape graphs. The method is next refined in [37] to use the EMD metric to compare 3D curve skeletons to match shapes. For 2D shapes, Bai *et al.* [13] propose an alternative to [186] by first pruning 2D medial axes to remove irrelevant branches (which are expensive for matching) using the DCE method [14], and and next using paths between skeletal endpoints to construct a graph. While faster than [186], this method cannot effectively handle shapes having few salient skeletal endpoints – that is, shapes that are not well described by curve skeeltons. Moreover, this method has not been extended to 3D.

Summarizing the above, we see that, on the one hand, skeletons have been effective for constructing descriptors for CBSR. On the other hand, there are only a few methods that extend to 3D shapes; and, for these, only the simpler curve skeleton is used. In Chapter 5, we show how we extend the above ideas to use the joint geometry-and-topology information present in the surface skeleton to construct effective descriptors for CBSR.

2.3 CONCLUSION

In this chapter, we have reviewed existing results in the area of using 3D skeletons to support shape processing. From the reviewed material, two conclusions emerge:

1. From a *technical* perspective, recent state-of-the-art methods have made both 3D curve and 3D surface skeletons attractive tools for shape analysis and processing. While older methods had limitations with respect to computational scalability, maximal size of the processed datasets, accuracy, robustness to noise, and representation type (voxel of mesh), recent methods have largely removed these issues. As such, we believe that both surface and curve skeletons are ready to be used in shape processing applications;

2. From an application perspective, however, the prominence of 3D skeletons is still relatively limited, as compared to other existing techniques. However, we see that all the application classes we considered – feature extraction, shape segmentation, and shape matching and retrieval – show a few promising results based on the usage of skeletons. Separately, we see that *surface* skeletons, despite their higher power of capturing shape geometry

and topolology, are far less frequently used in such applications than the simpler *curve* skeletons.

The overall conclusion of the above two observations is that there exists a strong, and unexplored, case for the added-value of using surface skeletons in supporting 3D shape processing applications. The rest of this thesis is dedicated to exploring this case in several directions.

3 PART-BASED SEGMENTATION BY SKELETON CUT SPACE ANALYSIS

As outlined in Chapter 2, Sec. 2.2.2, surface skeletons have been rarely used so far for supporting 3D shape segmentation. In this chapter, we take a first step to explore this gap. For this, we present a new method for part-based segmentation of voxel shapes that uses medial surfaces to define a segmenting cut at each medial voxel. The cut has several desirable properties – smoothness, tightness, and orientation with respect to the shape's local symmetry axis, making it a good segmentation tool. We next analyze the space of all cuts created for a given shape and detect cuts which are good segment borders. Our method is robust to noise, pose invariant, independent on the shape geometry and genus, and is simple to implement. We demonstrate our method on a wide selection of 3D shapes.[1]

3.1 INTRODUCTION

Shape segmentation aims to decompose a 3D shape into a set of parts that obey certain application-related properties, and is used in many contexts such as image analysis, registration, and 3D modeling [167]. *Patch-based* segmentation detects quasi-flat segments whose borders follow local curvature maxima on the shape surface, and is most used for faceted shapes [142]. *Part-based* segmentation follows a semantics-oriented approach, aiming to find shape parts that one would intuitively perceive as being logically distinct, and is used for natural shapes [140].

For a shape $\Omega \subset \mathbb{R}^3$, part-based segmentations (PBS) using *partitioning cuts* create a set of cuts $c \subset \partial\Omega$ that divide the shape boundary $\partial\Omega$ into disjoint parts. Desirable PBS properties, *e.g.* smoothness, orientation, tightness, and position of cuts that create segments, can be stated in terms of the cut-set $\mathscr{B} = \{c\}$. Finding a good segmentation is thus mapped to finding a cut-set $\mathscr{B}$ having such properties, a hard problem due to the high dimensionality of the cut space.

We present a new way to produce PBS of 3D voxel shapes by *skeleton cuts*. First, we construct, at any shape point, a cut that is locally and globally smooth, tightly wraps around the surface, is self-intersection free, and is locally orthogonal to the shape's local symmetry axis. For this, we use the shape's medial surface.

1 The text of this chapter is based on the paper: Part-based Segmentation by Skeleton Cut Space Analysis (C. Feng, A. Jalba, A. Telea), *Mathematical Morphology and Its Applications to Signal and Image Processing (Proc. ISMM)*, eds. J. A. Benediktsson and J. Chanussot and L. Najman and H. Talbot, Springer LNCS 9082, 2015, pp. 607-618.

Next, we construct the cut space $\mathscr{S} \subset \partial\Omega$ that contains all such cuts for a given shape. We extract the cut-set $\mathscr{B} \subset \mathscr{S}$ yielding our PBS by analyzing the global distribution of cut properties over $\mathscr{S}$. We demonstrate our method on a variety of 3D shapes and compare our results with eight existing PBS methods.

Section 3.2 reviews related work. Section 3.3 presents our method. Section 3.4 illustrates our method on a wide variety of 3D shapes and also compares it with related methods. Section 3.5 discusses our method. Section 3.6 concludes the chapter.

3.2 RELATED WORK

As outlined in Sec. 2.2.2, two main shape segmentation types exist [1, 11, 168]: *Patch-based* methods segment a shape's surface into quasi-flat patches bounded by sharp surface creases, and are suitable for synthetic shapes. *Part-based* segmentation (PBS), our focus, cuts a shape's surface into its logical components, useful for shapes formed of articulated parts, *e.g.* human bodies. plants, and other natural structures.

Most PBS methods find segments along what a human would see as logical shape parts, in two steps: (a) find *where* to cut a shape to isolate a part; and (b) find *how* to build a cut, once its location is set. These steps are addressed in different ways. As the topology of the shape skeleton or medial axis matches the part-whole shape structure [172], many methods use medial axes to place cuts. Au *et al.* use curve skeletons [12], where each skeleton branch maps to a part. Cuts are built by optimizing for cut concavity and length via minimal cuts [88]. Golovinskiy *et al.* create a large randomized cut-set and find part borders as the cuts on which most surface edges lie [74]. Shapira *et al.* note that skeletonization and segmentation are related, and compute a scalar shape-diameter function (SDF) on the shape surface to segments as surface faces with similar SDF values [169]. Tierny *et al.* segment shapes hierarchically by topological and geometrical analysis of their Reeb graphs, which are similar to curve skeletons [201]. Chang *et al.* compute shape medial surfaces, separate their manifolds, and back project each manifold on the shape surface to find a segment [28]. Dey and Sun extract curve skeletons as the maxima of the medial geodesic function (MGF) which encodes the length of the shortest path between feature points of points in the shape [44], and segment tubular parts as those which minimize the eccentricity of such paths. Reniers *et al.* construct a part for each branch of a shape's curve skeleton [140]. Part borders correspond to curve-skeleton junction points, and are created by the shortest paths in [44]. However, curve skeletons can contain many spurious junctions which change widely when the shape is slightly perturbed. Reniers *et al.* alleviate this by heuristics that shift cut-points along the curve skeleton to optimize for cut stability and planarity [141]. Yet, this method cannot segment shapes of large geometric, but little topological,

variability, like a pawn chess piece: Its curve skeleton has no junction points, so [141] cannot separate the pawn's head, body, and base, although these have different thicknesses.

Summarizing, the two elements of a good PBS (*where* to cut, and *how* to cut) are targeted in complement by different methods: Skeleton-based methods construct good partitioning cuts efficiently, *e.g.* by shortest-paths [44, 141]. Yet, curve skeletons do not encode enough of the shape geometry. Global search methods that analyze a wide set of shape cuts offer good ways to select where to partition [74, 169]. Yet, they do not offer explicit constraints for the cut shapes, and exhaustive cut space search is expensive. Our method combines the advantages of the two above classes of methods, while minimizing their limitations.

3.3 METHOD

Our method has a simple intuition: Say we want to cut the shape in Fig. 3.1 a close to points $A \dots E$. Which properties should these cuts have to yield a 'natural' PBS? In other words: How would a human draw such cuts? Figure 3.1 a shows five *undesirable* cuts: A is noisy, although it crosses a perfectly smooth surface zone; B is self-intersecting; C and D are too loose (long); and E is unnaturally slanted – a human asked to cut the shape at that point would arguably do it so across the finger's symmetry axis. Figure 3.1 b shows five cuts for the same points, computed with the method we proposed here. We argue that these cuts are more suitable for PBS than those in Fig. 3.1 a, as they are (1) tight, (2) locally smooth, (3) self-intersection free, (4) and locally orthogonal to the shape's symmetry axis. An additional property that cuts should satisfy is (5) being closed curves, so that they divide the shape's surface into different parts. We construct such cuts as follows: First, we compute a simplified medial surface of the input shape (Sec. 3.3.1). For each medial point, we next construct a cut having the above properties (Sec. 3.3.2). This answers the question "how to cut". By analyzing the resulting cut space, we next select a small cut-set that gives us the borders of salient shape-parts (Sec. 3.3.3). This answers the question "where to cut".

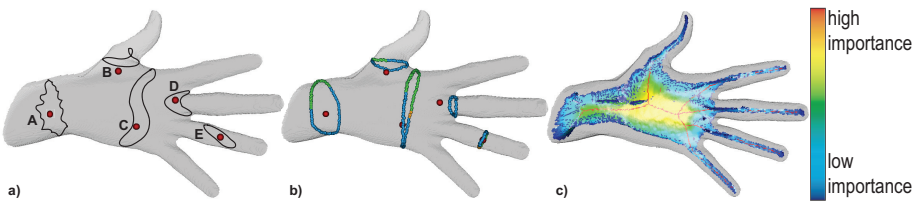


Figure 3.1: Possible cuts for part-based segmentation. Suboptimal cuts (a). Cuts created by our method (b). Medial surface colored by its importance metric (c).

3.3.1 *Skeletonization*

The Euclidean distance transform $DT_{\partial\Omega} : \Omega \to \mathbb{R}_+$ of a shape $\Omega \subset \mathbb{Z}^3$ with boundary $\partial\Omega$ is

$$DT_{\partial\Omega}(\mathbf{x} \in \Omega) = \min_{\mathbf{y}\in\partial\Omega} \|\mathbf{x}-\mathbf{y}\|. \tag{3.1}$$

The medial surface, or surface skeleton, of $\partial\Omega$ is defined as

$$S_{\partial\Omega} = \{\mathbf{x} \in \Omega | \exists\{\mathbf{f}_1,\mathbf{f}_2\} \subset \partial\Omega, \mathbf{f}_1 \neq \mathbf{f}_2, \|\mathbf{x}-\mathbf{f}_1\| = \|\mathbf{x}-\mathbf{f}_2\| = DT_{\partial\Omega}(\mathbf{x})\} \tag{3.2}$$

where $\mathbf{f}_1$ and $\mathbf{f}_2$ are the contact (or feature) points with $\partial\Omega$ of the maximally inscribed ball in Ω centered at $\mathbf{x}$ [71, 148]. These define the feature transform $FT_{\partial\Omega} : \Omega \to \mathscr{P}(\partial\Omega)$

$$FT_{\partial\Omega}(\mathbf{x} \in \Omega) = \operatorname*{argmin}_{\mathbf{y}\in\partial\Omega} \|\mathbf{x}-\mathbf{y}\|. \tag{3.3}$$

Medial surfaces are sensitive to small-scale noise on Ω, especially when using voxel-based models. To alleviate this, they can be *regularized* by a computing a metric $\rho : S_{\partial\Omega} \to \mathbb{R}_+$ such as the medial geodesic function (MGF) which sets $\rho(\mathbf{x})$ to the length of the shortest path on $\partial\Omega$ between the two feature points of $\mathbf{x}$ [44]. As the MGF monotonically increases from the medial surface boundary to its center, upper thresholding it yields connected and noise-free simplified medial surfaces (though tunnel preservation requires additional work) [145]. Figure 3.1 c shows a regularized medial surface using the MGF method in [145].

3.3.2 *Cut model*

The first step of our PBS is to compute a rich set of cuts, or cut space $\mathscr{S}$, which all satisfy properties (1-5) listed in Sec. 3.3. To build a cut $c \in \mathscr{S}$, consider a point $\mathbf{x} \in S_{\partial\Omega}$. As we shall next see in Sec. 3.3.3, we will create cuts from all skeletal points $\mathbf{x}$ located on the (simplified) surface skeleton $S_{\partial\Omega}$, and we will subsequently discard some of these cuts. However, for now, we concentrate on explaining how such a single cut $c(\mathbf{x})$ is built, given a skeleton point $\mathbf{x}$.

By definition, $\mathbf{x}$ has at least two feature points $\mathbf{f}_1$ and $\mathbf{f}_2$ on $\partial\Omega$ (Eqn. 3.2). Consider, for now, that there are precisely two such points. We first trace the shortest path $\gamma_1 \subset \partial\Omega$ between $\mathbf{f}_1$ and $\mathbf{f}_2$ (Fig. 3.2 a), whose length is the MGF value for $\mathbf{x}$ (Sec. 3.3.1). Next, we find the midpoint $\mathbf{m}$ of γ_1, *i.e.* the voxel of γ_1 furthest in arc-length distance from both $\mathbf{f}_1$ and $\mathbf{f}_2$. We then trace a ray through $\mathbf{x}$ and oriented in the direction $\mathbf{x}-\mathbf{m}$, and find the point $\mathbf{o}$ where this ray 'exits' Ω (Fig. 3.2 b). Intuitively, $\mathbf{o}$ is on the 'other side' of $S_{\partial\Omega}$ as opposed to $\mathbf{m}$. Finally, we construct the

two shortest paths on $\partial\Omega$ connecting $(\mathbf{f}_1, \mathbf{o})$ and $(\mathbf{f}_2, \mathbf{o})$ respectively (Fig. 3.2 c,d). Our final cut c for point $\mathbf{x}$ is given by $\gamma_1 \cup \gamma_2 \cup \gamma_3$.

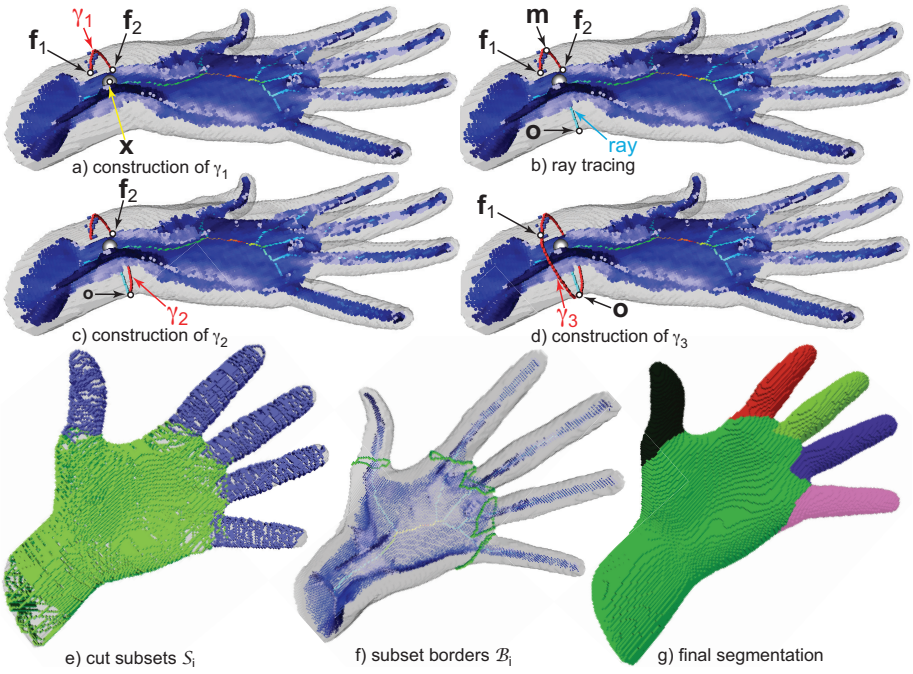


Figure 3.2: Cut construction (a-d) and cut space analysis (e-g) for part-based segmentation.

While c is piecewise geodesic (so locally smooth), it can be non-smooth at the three endpoints $\mathbf{f}_1, \mathbf{f}_2$ and $\mathbf{o}$ of γ_i. Also, our construction does not make c as tight as possible globally. To fix both issues, we perform 5 iterations of a constrained Laplacian smoothing pass over c, with a kernel size of 10 voxels. We prevent c leaving the surface, by reprojecting its voxels to their closest points on $\partial\Omega$ after each iteration. This smooths out possible 'kinks' at $\mathbf{f}_1, \mathbf{f}_2$ and $\mathbf{o}$, thus making c globally smooth and tight. If such kinks are very small or inexistent, smoothing has no effect, as c is globally geodesic. In that case, Laplacian smoothing shifts c's points along the surface normal, since c's acceleration c'' is normal to the surface, so reprojection moves the smoothed points back to their original location.

3.3.2.1 *Cut properties:*

Our cuts meet the desired properties we require for PBS:

1. *tight:* Cut parts γ_i are piecewise-geodesic, thus shortest curves on $\partial\Omega$. Also, the constrained Laplacian smoothing shortens potential kinks present at the geodesic endpoints, thus making the entire c wrap tightly around the shape;

2. *smooth:* Guaranteed by the same properties as for tightness – piecewise geodesicness and constrained Laplacian smoothing;

3. *self-intersection free:* c is a geodesic triangle (three geodesics linking three *different* points on $\partial\Omega$) whose edges do not intersect except at endpoints;

4. *locally orthogonal* to the symmetry axis: The cut $c(\mathbf{x})$ surrounds the medial surface $S_{\partial\Omega}$ around the skeletal point $\mathbf{x}$, by construction. Hence, it also surrounds the so-called *curve skeleton* of $\partial\Omega$, which is a 1D structure locally centered within $S_{\partial\Omega}$ with respect to its boundary $\partial S_{\partial\Omega}$. While we do not have a formal proof of local orthogonality, we observed in practice that our construction always creates cuts that are visually orthogonal to the curve skeleton;

5. *closed:* The cut c is a closed (Jordan) curve by construction.

3.3.2.2 *Implementation:*

As explained in Sec. 3.3.1, we use a simplified (regularized) skeleton as a basis to compute our cuts. In practice, we found that the skeletonization method in [83] gave good results, in terms of computational speed and ability to regularize a skeleton. However, for lower voxel resolutions (typically, under roughly 200^3 voxels), this method can occasionally produce disconnected skeletons. This can be fixed as follows. First, we detect all connected components of the simplified skeleton, and select the largest one (in terms of voxel count). Next, we connect all other components found to the largest component by adding digital voxel lines (computed with Bresenham's algorithm in 3D) between their respective closest voxels, if the length of such line segments is lower than roughly 4 voxels. This is based on our observation that the disconnections created by the skeletonization method in [83] are typically small. This prevents reconnecting spurious components which are located far away from each other in the regularized skeletons. All such remaining spurious components are then discarded. We verified that this heuristic strikes a good balance between providing a rich *and* connected skeleton and using various simplification levels to regularize the skeleton.

To build γ_1, we need two feature points $\mathbf{f}_1$ and $\mathbf{f}_2$. Two issues exist here: (1) Computing the feature transform $FT(\mathbf{x})$ on digital shapes cannot be done via Eqn. 3.3, given the finite voxel grid resolution [139, 145]. To fix this, we compute the so-called extended feature transform $EFT(\mathbf{x})$ which finds all closest-points on $\partial\Omega$

to all 26 neighbors of $\mathbf{x}$, and which is a superset of $FT(\mathbf{x})$ [145]. From this superset, we select exactly two feature points that best represent the symmetric embedding of $S_{\partial\Omega}$ in Ω. For this, we select the two feature points $\{\mathbf{f}_1, \mathbf{f}_2\} \subset EFT(\mathbf{x})$ that maximize the angle $\widehat{\mathbf{f}_1\mathbf{x}\mathbf{f}_2}$. We trace the ray used to find $\mathbf{o}$ by Bresenham's 3D line-tracing algorithm on the voxel shape. We compute geodesics by Dijkstra's shortest-path algorithm on the connectivity graph of voxels of $\partial\Omega$, using A^* heuristics to speed the search, and using edge weights that approximate neighbor-voxel distances by Kiryati's scheme [93] for better path-length accuracy. Finally, we reproject Laplacian-smoothed points on the shape surface by using the fast ANN library for finding nearest-neighbors [122].

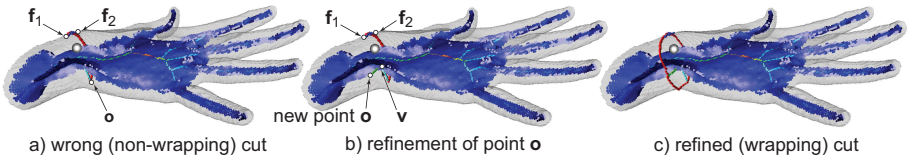


Figure 3.3: Refinement of cut construction.

In a few cases, point $\mathbf{o}$ found as above lies in such a location with respect to $\mathbf{f}_1$ and $\mathbf{f}_2$ that the three geodesics γ_1, γ_2, and γ_3 do not form a closed cut wrapping around the surface skeleton, as intended. Figure 3.3 a shows such a situation. Here, the direction of the geodesic from $\mathbf{f}_1$ to $\mathbf{o}$ is the same with the direction of the geodesic from $\mathbf{f}_1$ to $\mathbf{f}_2$, or, in other words, the two geodesics share a common portion. In contrast, the geodesics between $\mathbf{f}_2$ to $\mathbf{o}$ and between $\mathbf{f}_1$ and $\mathbf{f}_2$ are disjoint, which is the intended configuration.

We solve this problem by an iterative ray-tracing procedure as follows. When the situation is encountered (detected by finding of two of the three geodesics overlap for more than one voxel), we create a new ray passing through the midpoint $\mathbf{v}$ of the segment $(\mathbf{x}, \mathbf{o})$ and having the direction $\mathbf{f}_1 - \mathbf{f}_2$, if the geodesic from $\mathbf{f}_1$ to $\mathbf{o}$ is wrong; if the geodesic from $\mathbf{f}_2$ to $\mathbf{o}$ is wrong, we do the same, but use the ray direction $\mathbf{f}_2 - \mathbf{f}_1$. The new ray tracing yields a new location for the point $\mathbf{o}$ (see Fig. 3.3 b) which is tested as described above. We repeat the process until we obtain a wrapping cut (Fig. 3.3 c). The process typically converges after a few iterations.

We also experimented with a different way to compute cuts, based on planar slicing. For a skeleton point $\mathbf{x}$, having feature points $\mathbf{f}_1$ and $\mathbf{f}_2$, consider the case when $\mathbf{x}$, $\mathbf{f}_1$, and $\mathbf{f}_2$ are not colinear – that is, the two feature vectors of $\mathbf{x}$ form an angle different from 180 degrees. In such case, the above-mentioned three points define a unique plane π. A cut $\gamma = \partial\Omega \cup \pi$ can then be defined as the intersection of the shape's surface with the plane. In discrete voxel space, computing γ should be done carefully, so as to avoid disconnections, non-smooth regions, or regions

where the curve is thicker than one voxel. This can be done by considering two parallel planes π_+ and π_- separated by a distance of $\sqrt{3}$, located in both half-spaces determined by π. The intersection of π_+ and π_- with $\partial\Omega$ thus creates a band B_π of voxels of local thickness $\sqrt{3}$ – or, in other words, the intersection of π_+ and π_- with the volume Ω creates a thick slice. Figure 3.4a shows (in cyan) an example of such a thick slice. The thick band may be, at places, thicker than one voxel, but is guaranteed to be connected. We can next extract our desired cut γ from B_π as follows. We trace a shortest path γ_1 between $\mathbf{f}_1$ and $\mathbf{f}_2$ using Dijkstra's algorithm over the voxels in B_π, and mark all voxels in B_π which are visited during the shortest-path tracing process as blocked. Next, we trace a second shortest path γ_2 over the non-blocked voxels remaining in B_π. Given the blocking, γ_2 will connect $\mathbf{f}_1$ and $\mathbf{f}_2$ so as to form a closed curve, when taken together with γ_1. The final cut is thus $\gamma = \gamma_1 \cup \gamma_2$. Figure 3.4b shows an example of such a cut.

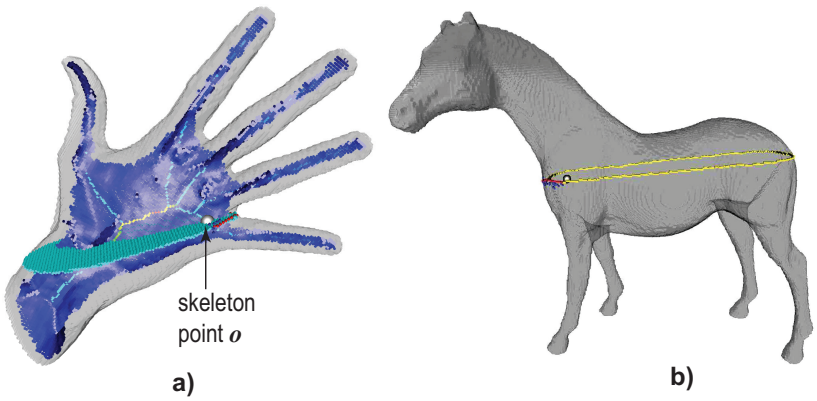


Figure 3.4: a) Thick slice (in cyan) created for a skeleton point. b) Planar cut (in yellow) created by our slicing algorithm.

The slice-based cut method is faster than the geodesic-tracing method described earlier, as the Dijkstra search for the shortest path must now consider only the thin band B_π rather than the entire surface $\partial\Omega$. Also, it generates globally straighter cuts – the deviation from a globally geodesic curve being equal to the negligible thickness of the band band B_π. Such a global geodesic property cannot be guaranteed by our earlier geodesic-tracing method, which, as explained in Sec. 3.3.2.1, generates only *piecewise* geodesic cuts. However, the slice-based method does not work when the points $\mathbf{f}_1$, $\mathbf{f}_2$, and $\mathbf{x}$ are colinear. Such cases do occur, consider *e.g.* the skeleton of a box shape. A good way to combine the two cut methods is to test, for each skeleton point $\mathbf{x}$, for the colinearity condition. If this does not hold, we can use the faster slice-based cut construction; if it does, we revert to the slower, but more general, three-piece geodesic cut construction.

3.3.3 *Cut space analysis*

We can create a cut $c(\mathbf{x})$ for any voxel $\mathbf{x}$ of a shape's medial surface $S_{\partial\Omega}$, which has good properties for PBS. Intuitively, $c(\mathbf{x})$ is a good way to cut the shape at point $\mathbf{x}$, *if* we want a cut there. We now must decide *where* we want to cut to get a PBS with desired global properties. Let $\mathscr{S} = \{c(\mathbf{x}) | \mathbf{x} \in S_{\partial\Omega}\}$ be the space of all cuts created from $S_{\partial\Omega}$. Given our cut properties, cuts on the same shape-part share similar properties *e.g.* orientation and length. Cuts for different parts have different properties. Consider our hand model: Finger cuts are short; wrist cuts have average length; and palm cuts are longest. For a shape consisting of a rump and protruding parts, cuts for parts are shorter than cuts for the rump.

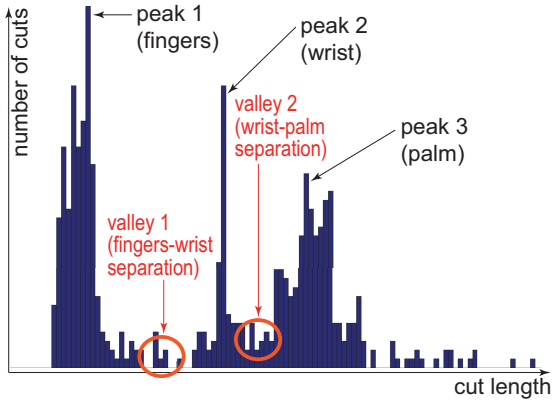


Figure 3.5: Cut-length histogram for the hand model (Sec. 3.3.3).

We use these insights to partition $\mathscr{S}$ in subsets $\mathscr{S}_i$ so that $\cup_i \mathscr{S}_i = \mathscr{S}$ and $\mathscr{S}_i \cup \mathscr{S}_{j\neq i} = \varnothing$ by using the histogram of cut lengths over $\mathscr{S}$. Histogram peaks show large similar-length cuts, so partitioning it by thresholds in the valleys between peaks gives our desired subsets $\mathscr{S}_i$. To find such thresholds robustly, we filter the histogram by mean shift [36] to 'sharpen' the cut-distribution and separate peaks from valleys more clearly. We define a peak as a histogram value ex'ceeding λ times the cut count $\|\mathscr{S}\|$, and a valley as a value less than $\mu = \lambda/3$. Setting $\lambda = 0.01$ gave good results for all shapes we tested our method on. Figure 3.5 shows the cut-length histogram for the hand model. Its three main peaks describe cuts on the fingers, wrist, and palm; the two valleys give the two thresholds needed to separate fingers from the palm and the palm from the wrist.

We initially used the average Hausdorff distance between a cut and its neighbor cuts to classify cuts. However, we observed that this distance does not capture the length similarity of cuts well enough. We also experimented using the maximum length ratio between a cut and its neighbor cuts in order to classify cuts. How-

ever, the results were also not positive. We also tried a third variant to partition the histogram using Otsu's method [126]. However, in its original form, this method only works for histograms having two peaks. Overall, the histogram partitioning scheme given above produced significantly better results than these two early experiments.

Subsets $\mathscr{S}_i$ do not (yet) coincide with our desired segments. Indeed, an $\mathscr{S}_i$ can contain logically disjoint cuts of similar lengths – *e.g.* all cuts on the fingers (blue in Fig. 3.2 e) are in the same subset. Also, $\mathscr{S}$ does not fully cover $\partial\Omega$, since we compute it from the simplified medial surface. This is shown by the gaps between cuts in Fig. 3.2 d. To fix this, we first define a cut $c(\mathbf{x})$ as being a border $\mathscr{B}_i$ of subset $\mathscr{S}_i$ if $c(\mathbf{x})$ belongs to a different subset than any of the cuts $c(\mathbf{y})$, where $\mathbf{y}$ are the 26-neighbors of $\mathbf{x}$ on $S_{\partial\Omega}$. Using this definition, we find the set of cuts $\{\mathscr{B}_i\}$ that represent the borders of our final segments (Fig. 3.2 f). Note that, if a cut is marked as border, at least one of its neighbor cuts will be in a different cut subset, by definition. Hence, that neighbor cut will also be a border, so more than one border will be produced from a 3^2 voxel neighborhood. To remove such duplicates, we keep, for each such neighborhood, the shortest border.

We compute our final segments by finding the connected components of $\partial\Omega$ separated by borders, via a flood-fill algorithm on $\partial\Omega$ (Fig. 3.2 g). To this end, we dilate the border cuts a few voxels, to make sure that the flood-fill will not 'leak' between the connected components they aim to separate (see Fig. 3.6a). As seeds for the flood fill, we use neighbor voxels of the dilated border voxels (see Fig. 3.6b). Finally, we assign the voxels of the dilated cuts to the resulting segment in a nearest-neighbor fashion. This way, we obtain an exact partitioning of $\partial\Omega$, where every voxel is assigned to one and only one segment.

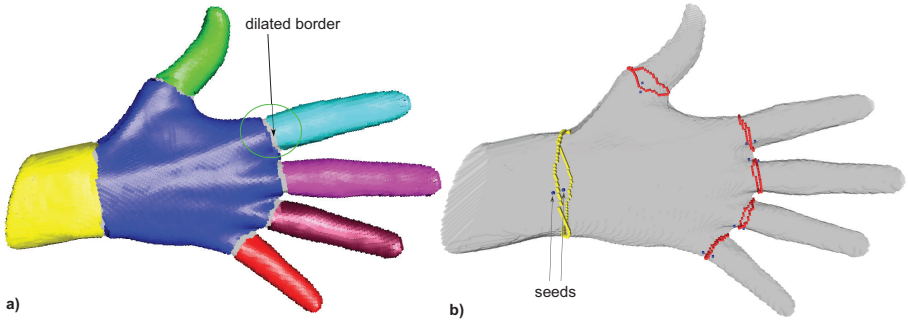


Figure 3.6: Flood fill for segment detection. a) Dilated border curves, in gray. b) Seed points close to borders, in blue.

To visualize the segmentation, we can render the voxel surface $\partial\Omega$ color-coded with segment IDs. However, this would yield a poor quality image, especially at low voxel resolutions. Moreover, comparing such images with mesh segmentations would artificially make it look worse, due to the low rendering quality. As such, we visualize the segmented surface using a *splatting* technique: For each voxel center $\mathbf{x}$, we define a round splat, implemented as an OpenGL point primitive with radius equal to about the size of a voxel in screen space. Next, we estimate the normal $\mathbf{n}(\mathbf{x})$ that the original surface $\partial\Omega$ would have at point $\mathbf{x}$. For this, let $B(\mathbf{x},R)$ be a ball of radius R centered at $\mathbf{x}$, and let $X(\mathbf{x})$ be the set of voxel centers on $\partial\Omega$ which fall within $B(\mathbf{x},R)$. Let $\mathbf{e}_i$, $1 \leq i \leq 3$, be the eigenvectors computed by PCA analysis on the point set $X(mathbfx)$, in increasing order of their eigenvalues λ_i. Then, the normal $\mathbf{n}(\mathbf{x})$ corresponds to the normalized value $\mathbf{e}_1/\|\mathbf{e}_1\|$. In cases where the PCA decomposition delivers $\lambda_1 = \lambda_2$, *i.e.* the structure of the point set $X(\mathbf{x})$ is not locally well approximated by a surface, we simply set $\mathbf{n}(\mathbf{x})$ to the value of the normal of the nearest voxel where the PCA analysis yielded a consistent normal. Finally, having the normal information, we render the OpenGL points using classical Phong shading. Figure 3.7a shows the result of this rendering method. As we can see, voxelization artifacts are barely visible. We will use this rendering method to show our segmentation results next in Sec. 3.4.

As a side observation, we found an interesting connection between the the cut space and the curve skeleton of a shape. Figure 3.7 shows this. Here, the blue points represent the centers of all cuts in the cut space of the hand shape, defined as the average positions of all voxels on each cut. Yellow points show the curve skeleton of the same shape, as computed by the method in [83]. As visible, along *tubular* regions of the shape, such as the fingers and wrist, the two sets of points coincide quite well. This indicates, implicitly, that our cuts will produce a similar segmentation to methods that use the curve skeleton to place geodesic-like partitioning borders between segments, *e.g*, [141].

3.4 RESULTS AND COMPARISON

We have tested our method on several shapes provided as 3D polygon meshes, voxelized by *binvox* [123] at resolutions up to 400^3 voxels. Figure 3.8 compares our results with [141], the best medial-descriptor voxel PBS method we know. We get very similar results, but find more fine-grained segments than [141] – see finger and ear details of the animal models, pig tail, dragon spikes, and microscope lens. Segment borders are smooth and locally orthogonal to the shape's symmetry axis, *i.e.*, similar to how a human would cut the shape at the respective places. Our method finds segments of various sizes, ranging from details (dragon's tail, hound's ears), to large parts (limbs of various models). Figure 3.9 a-k compares our method with eight PBS methods on two shapes [10, 100, 103, 105, 107, 140, 141, 201]. Here,

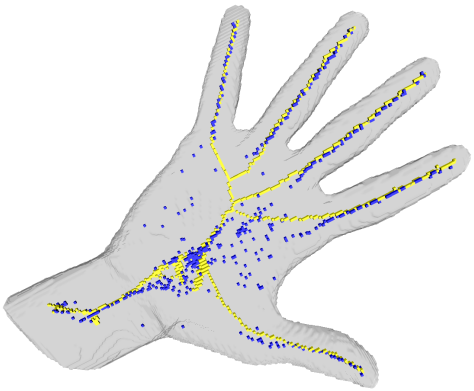


Figure 3.7: Relation of the curve skeleton (yellow) of a shape with the centers of the cuts in its cut space (blue).

Reniers *et al.* (1) denotes [140], and Reniers *et al.* (2) denotes [141]. These methods span from voxel-based to mesh-based, and use various segmentation heuristics (skeleton, curvature, salience, and topology-based). We argue that our method creates equally or, in some cases, more plausible PBSs. Since both our method and [141] use medial descriptors, computed by the same underlying method [145], a relevant question is how the two methods differ. We use (a) medial *surfaces*, while [141] uses *curve* skeletons; and (b) we find segment borders by analyzing *all* possible cuts, while [141] places such borders around the curve-skeleton branch junctions. Fig. 3.9 l-p shows five examples where the public implementation of [141] fails to segment at all. We find two causes for this: The shape parts in Fig. 3.9 l cannot be well described by curve-skeleton branches, as they are nearly rotationally symmetric. As few (if any) such junctions exist, [141] fails. The shape in Fig. 3.9 n is described by a mix of medial surfaces (base plate) and curve skeletons (tubular parts). As [141] only uses curve skeletons, data on the base plate is incomplete or missing. For the shapes in Fig. 3.9 m-p, the many heuristics in [141] to select cuts centered on the curve-skeleton fail, as they imply that such cuts should be nearly planar. This does not happen for the above shapes.

We can also produce a *multiscale* PBS: For this, we simply change the values of λ and μ used to partition the cut space via its length histogram (Sec. 3.3.3). High λ values and low μ values yield fewer and more differentiated segments (in terms of local thickness); closer values of *lambda* and μ yield a finer-grained segmentation. Figure 3.9 r shows three such scales for the armadillo shape.

Our method is *pose invariant*, as illustrated in Fig. 3.9 s. Indeed, our cut space histogram captures local shape thickness, which does not depend on pose.

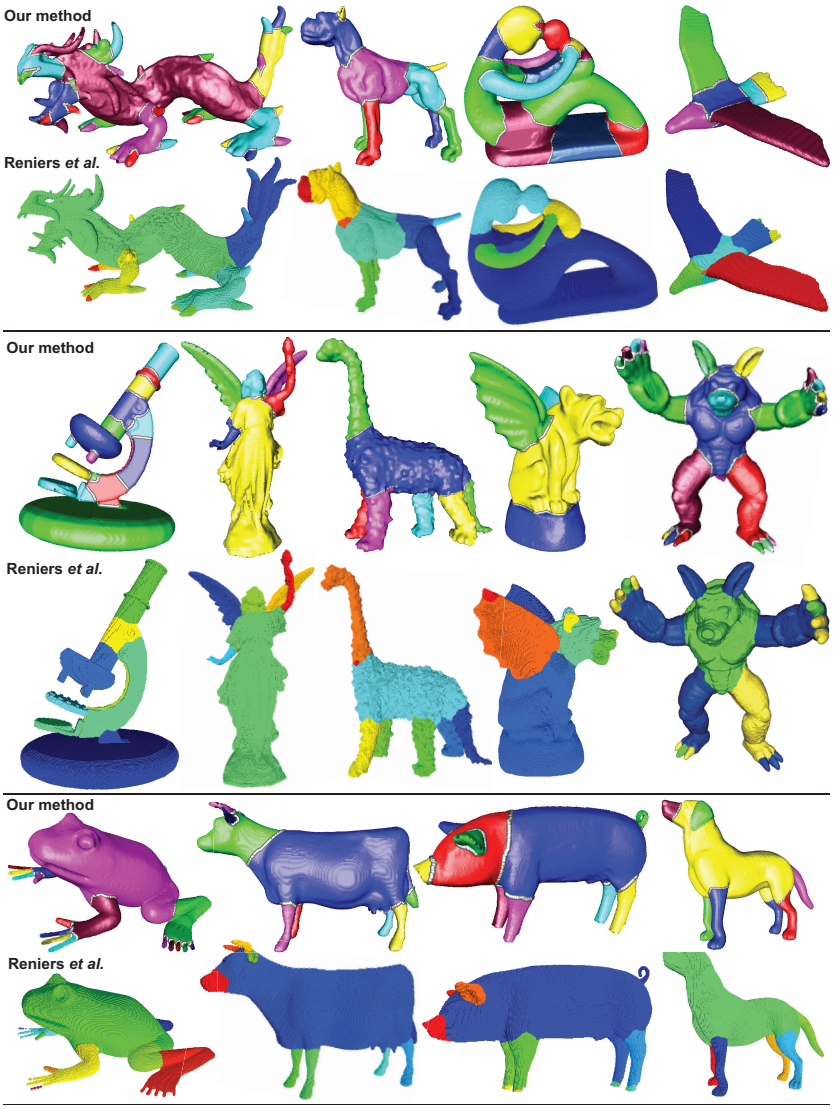


Figure 3.8: Part-based segmentations of our method *vs* Reniers *et al.* [141] (Sec. 3.4).

Table 1 shows the time for creating cuts (t_{cuts}), medial surfaces (t_{skel}), cut space analysis (t_{space}), our total time (t_{total}), and total time for [141] ($t_{Reniers}$), for our method coded in C++ on an 8-core 3.5 GHz PC. Empty cells ($t_{Reniers}$) show shapes where [141] failed. As cuts are computed independently, we parallelized our method by *pthreads*, getting a speed boost factor of 7, close to the optimal

value of 8 for our hardware. Compared to [141], on the same hardware, we are slightly faster in most cases, and can successfully segment all tested shapes.

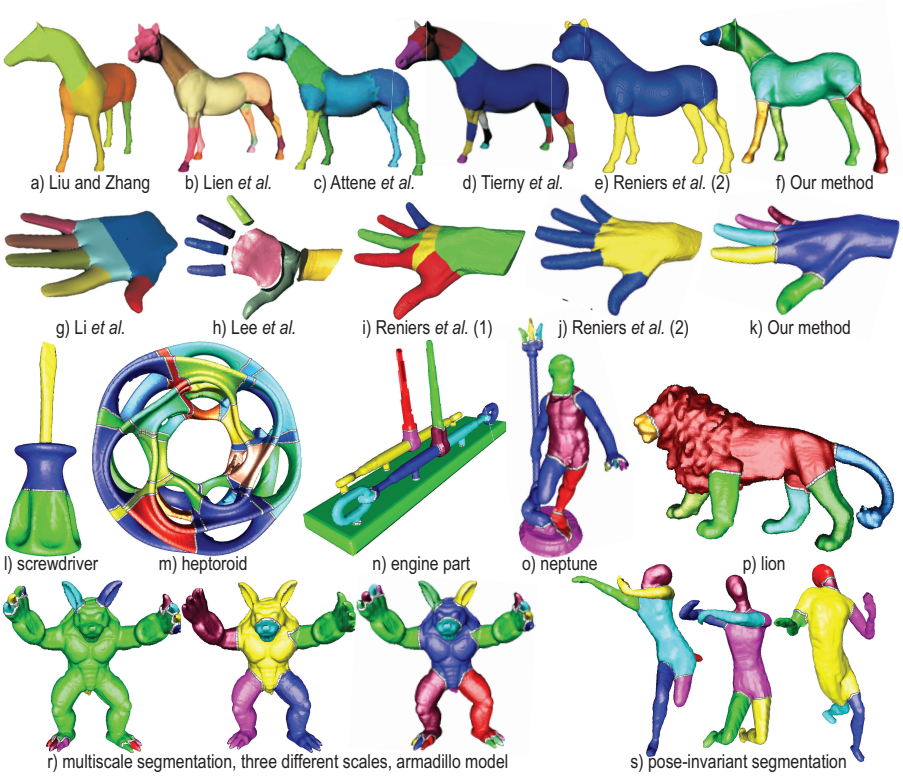


Figure 3.9: Comparison of our method with eight PBS methods (a-k). Our results for shapes where Reniers *et al.* fails (l-p). Multiscale (r) and pose-invariant (s) segmentations.

3.5 DISCUSSION

We next discuss several aspects of our proposed part-based segmentation method.

Global search: We create a PBS by finding all part-inducing cuts from the medial surface, and selecting a cut-subset by globally optimizing for part-similarity as captured by cut lengths. In contrast to purely topological PBS methods [140, 141], we search a much wider space of possible partitionings; yet, our search space is much smaller than that of other methods which look for cuts of any possible

Shapes	**cuts** $\|\mathscr{S}\|$	**voxels** $\|\Omega\|$	**voxel volume**	t_{cuts}	t_{skel}	t_{part}	t_{total}	$t_{Reniers}$
Dragon	2789	283238	400*400*400	50.8	1.90	0.03	52.73	40.26
Hound	1530	245759	300*300*300	23.24	1.51	0.01	24.76	25.1
Hyptoroid	4873	651478	400*400*400	400.5	3.36	0.04	403.90	-
Fertility	1354	199581	300*300*300	20.85	2.02	0.01	22.88	22.89
Gargoyle	488	129420	300*300*300	12.62	3.26	0.005	15.885	69.89
Microscope	1397	307863	300*300*300	44.14	1.58	0.01	45.73	198.02
Lucy	6201	1.04×10^6	300*300*300	68.01	0.63	0.09	68.73	12.7
Engine part	1501	135416	300*300*300	15.55	0.27	0.01	15.83	-
Frog	41450	1.20×10^7	300*300*300	808.2	2.48	2.16	812.8	36.93
Screwdriver	1372	306480	300*300*300	13.14	0.60	0.01	13.75	-
Noisydino	1375	194117	300*300*300	14.79	1.19	0.015	16.00	20.2
Cow	1009	143938	256*256*256	8.15	0.96	0.01	9.12	14.34
Neptune	1908	211723	420*185*251	34.7	1.22	0.02	35.94	-
Airplane	741	76700	300*300*300	6.00	0.28	0.08	6.37	-
Bird	476	45638	300*300*300	2.28	0.18	0.003	2.47	7.98
Hand	584	58071	200*84*140	2.15	0.22	0.004	2.37	-
Lion	2181	381968	300*300*300	23.16	1.08	0.02	24.27	-
Horse	884	109555	142*300*251	9.58	1.24	0.008	10.83	-
Pig	959	145215	300*300*300	10.97	1.51	0.01	12.50	22.26
Dog	1241	184805	300*300*300	15.65	1.29	0.02	16.97	18.87
Hippo	838	166932	300*300*300	12.13	2.41	0.01	14.55	25.18
Rhino	1746	403399	300*300*300	25.20	2.15	0.03	27.39	-
Armadillo	2242	436933	300*229*252	47.55	2.67	0.03	50.26	-

Table 1: Shape sizes and segmentation times by our method and Reniers *et al.* [141].

orientation [74], thereby achieving a good flexibility-performance balance.

Simplicity: In our approach, we can use *any* medial surface skeletonization method, *e.g.* [7, 148, 175], as long as it outputs regularized skeletons. This makes our method applicable to mesh-based shapes (and their medial surfaces) [82].

Regularization: We use regularized medial surfaces (Sec. 3.3.1) having voxels with large MGF values, which have far-apart feature points $\mathbf{f}_1$ and $\mathbf{f}_2$. This ensures that the ray casting used to compute cuts robustly finds cuts that wrap around the medial surface (Sec. 3.3.2).

Multiscale: Multiscale PBS occurs at two levels: (1) Simplified medial surfaces yield cuts only for important shape parts; (2) The cut histogram analysis parameters λ and μ select the level-of-detail where we search for cut-length differences.

Pose invariance: Our method is pose-invariant [141, 176], as shown by the model in Fig. 3.9 s (which is also used in [176] to show pose invariance).

Robustness: We robustly segment noisy or detail-rich surfaces, *e.g. dragon* and *dino* (Fig. 3.8) or *lion* (Fig. 3.9). Segment borders are smooth by construction (Sec. 3.3.2). Since our segmentation uses a subset of these cuts, and only considers *integral* cut properties (length) rather than differential ones (*e.g.* curvature), noise and/or small-scale details are robustly handled.

Limitations: Our method's cost is $O(\|S_{\partial\Omega}\|\|\partial\Omega\|\log\|\partial\Omega\|)$. As our method parallelizes easily (Sec. 3.4), its practical cost is similar to other skeleton-based PBS method [140, 141] or cut-based methods [74]. For space constraints, we compare with only eight related methods. More PBS methods exist, and quantitative metrics can be further used to measure segmentation quality [110]. Yet, even without such extra insights, we argue that our goal of showing that *surface* skeletons have added value for PBS as opposed to *curve* skeletons is well defended.

3.6 CONCLUSIONS

In this chapter, we have presented a new method for part-based segmentation of 3D voxel shapes by analyzing the entire space of potential partitioning cuts constructed by using the shape's medial surface. To our knowledge, our approach is the first which uses medial surfaces for part-based segmentation, and thereby shows the added-value of medial surfaces for segmentation, as opposed to the well-known use of curve skeletons for the same task. We demonstrate our method on a wide variety of 3D shapes, and compare it with eight related segmentation methods.

As an extension of this work, different ways to partition the cut space can be easily tried, *e.g.* bottom-up hierarchical clustering or cut similarities based on *e.g.* curvature, eccentricity, and orientation. This would lead to an entire family of PBS methods in a single simple implementation. Our cut-length histogram could be an effective shape descriptor for retrieval and matching. Finally, implementing our method for mesh-based shapes on the GPU should lead to large scalability increases, in terms of the resolution of the shapes we can treat, and/or in terms of computational speed. We discuss the realization of such a mesh-based segmentation method in Chapter 6.

4 IMPROVED PART-BASED SEGMENTATION OF VOXEL SHAPES BY THE SKELETON CUT SPACE

In Chapter 3, we have introduced the cut space of a 3D shape as a descriptor that can be used to produce part-based segmentations of that shape. We have seen there that the cut space can yield segmentations of articulated shapes of similar, or even higher, quality as compared to other existing part-based segmentation methods. As such, the cut space is a promising tool to support part-based segmentation. However, several aspects of the cut space construction, and the usage of the cut space in shape segmentation, have not been fully explored. For instance, the actual robustness of the method in terms of setting its various parameters is still unclear. Secondly, we have described several heuristics to handle corner-cases in the computation of the cut space. However, such heuristics may not deliver the expected results in all cases (for all shapes).

In this chapter we present a refinement of the cut space computation method (and its usage for shape segmentation) introduced in Chapter 3, with the aim of improving upon the above limitations. Specifically, we replace the histogram segmentation presented there by a more robust, and easier to use, clustering-based segmentation. This allows us also to produce multiscale segmentations of a shape. Secondly, we present a detailed analysis of the parameter space of the method, showing how its behavior can be influenced by different parameter values to obtain different results. Thirdly, we present ways to improve the computational performance, and compare our results with additional methods and on additional shapes. Finally, we present a new application of the cut space for interactive shape segmentation and editing.[1]

4.1 INTRODUCTION

Shape segmentation aims to decompose a 3D shape into a set of parts that obey certain application-related properties, and is used in many contexts such as image analysis, registration, content-based retrieval, and 3D modeling [167]. *Patch-based* segmentation detects quasi-flat segments whose borders follow local curvature maxima on the shape surface, and is most used for faceted shapes [142]. *Part-based* segmentation follows a semantics-oriented approach, aiming to find

1 The text of this chapter is based on the paper: Improved Part-Based Segmentation of Voxel Shapes by Skeleton Cut Spaces, *Mathematical Morphology – Theory and Applications*, vol. 1, De Gruyter, 2016, pp. 60-78.

shape parts that one would intuitively perceive as being logically distinct, and is used for natural shapes [140].

For a shape $\Omega \subset \mathbb{R}^3$, part-based segmentations (PBS) using *partitioning cuts* create a set of cuts $c \subset \partial\Omega$ that divide the shape boundary $\partial\Omega$ into disjoint parts. Desirable PBS properties, *e.g.* smoothness, orientation, tightness, and position of the cuts that create segments, can be stated in terms of the cut-set $\mathscr{B} = \{c\}$. Finding a good segmentation is thus mapped to finding a cut-set $\mathscr{B}$ having such properties, a hard problem due to the high dimensionality of the cut space.

We present a new way to produce PBS of 3D voxel shapes by *skeleton cuts*. First, we construct, at any shape point, a cut that is locally and globally smooth, tightly wraps around the surface, is self-intersection free, and is locally orthogonal to the shape's local symmetry axis. For this, we use the shape's medial surface. Next, we construct the cut space $\mathscr{S} \subset \partial\Omega$ that contains all such cuts for a given shape. We extract the cut-set $\mathscr{B} \subset \mathscr{S}$ yielding our PBS by analyzing the global distribution of cut properties over $\mathscr{S}$. We demonstrate our method on a variety of 3D shapes and compare our results with eight existing PBS methods. Our proposal shows that medial surfaces can be efficiently and effectively used to construct PBS segmentations of 3D shapes. This makes this type of skeletal descriptors, which are so far rarely used in shape-processing applications, more interesting for practical purposes.

In this chapter, we extend the related segmentation framework proposed in Chapter 3, and published in [60], with the following main contributions:

- We present a *clustering-based* segmentation technique using the shape cut space, which works more robustly, and is easier to use, than the earlier histogram-based technique in [60];
- We present a detailed analysis of the *parameter space* of the entire pipeline, which allows us to find good preset values for the method's free parameters, and also gives detailed insight in the method's behavior;
- We present a new *application* of our cut space segmentation technique for the interactive, user-driven, segmentation of 3D shapes.

Besides these main contributions, we also present a number of technical improvements in terms of computational performance and robustness of the cut construction that make our method competitive in speed and quality with related state-of-the-art methods.

The structure of this chapter is as follows. Section 4.2 reviews related work. Section 4.3 presents the basic method and our proposed enhancements. Section 4.4 shows how we can use our method for interactive part-based segmentation of 3D shapes. Section 4.5 presents the parameter space analysis. Section 4.6 illustrates

our method on a wide variety of 3D shapes and also compares it with related methods. Section 4.7 discusses our method. Section 4.8 concludes the chapter.

4.2 RELATED WORK

Two main segmentation approaches for 3D shapes exist [1, 11, 168]: *Patch-based* methods segment a shape's surface into quasi-flat patches bounded by sharp surface creases, and are suitable for synthetic shapes such as polyhedral models created by CAD-CAM applications. *Part-based* segmentation (PBS), our focus, cuts a shape's surface into its logical components. Such methods are suitable for shapes formed of articulated parts, *e.g.* human bodies, plants, and other natural structures that exhibit a part-whole hierarchical structure.

Most PBS methods find segments along what a human would see as logical shape parts, in two steps: (a) find *where* to cut a shape to isolate a part; and (b) find *how* to build a cut, once its location is set. These steps are addressed in different ways, as follows. Attene *et al.* segment a shape by fitting primitives from a predefined library to the shape's polygonal surface in a minimal-cost way. This approach works best when reverse-engineering shapes produced by CAD-like modeling, but less well for organic shapes [10]. Lee *et al.* construct partitioning cuts on a surface mesh by analyzing local mesh features such as curvature and centricity, using snakes to optimize for cut smoothness [99, 100]. Liu *et al.* encode the local similarity of faces in a mesh into an affinity matrix which they next decompose by spectral clustering to yield a segmentation [107]. Many similar clustering methods exist, such as based on algebraic multigrid clustering of the surface curvature matrix [34], or the fuzzy clustering approach in [87]. In a related way, mesh models can be segmented by watershed approaches applied to their surface curvature [114, 127]. An important issue of all clustering methods is that it is very hard to explicitly enforce global properties on the resulting cluster borders, which ultimately define the segmentation result.

As the topology of the shape skeleton or medial axis matches the part-whole shape structure [172], many methods use medial axes to place cuts. Au *et al.* use curve skeletons [12], where each skeleton branch maps to a part. Cuts are built by optimizing for cut concavity and length via minimal cuts [88]. Golovinskiy *et al.* create a large randomized cut-set and find part borders as the cuts on which most surface edges lie [74]. Shapira *et al.* note that skeletonization and segmentation are related, and compute a scalar shape-diameter function (SDF) on the shape surface to segments as surface faces with similar SDF values [169]. Their SDF function is related to approaches which compute histograms of shape thickness for shape retrieval tasks [109, 160]. Conversely, Lien *et al.* use shape decomposition to compute progressively refined curve skeletons [105]. Tierny *et al.* segment shapes hierarchically by topological and geometrical analysis of their Reeb graphs, which are

similar to curve skeletons [201]. Chang *et al.* compute shape medial surfaces, separate their manifolds, and back project each manifold on the shape surface to find a segment [28]. A similar segmentation method, using high-resolution point-cloud skeletons computed on the GPU [82] along the method of Reniers *et al.* [145], is proposed in [98]. Similar high-resolution surface and curve skeletons can be computed in voxel space using an advection model [83]. However, such skeletons have not yet been used for segmentation purposes. Dey and Sun extract curve skeletons as the maxima of the medial geodesic function (MGF) which encodes the length of the shortest path between feature points of points in the shape [44], and segment tubular parts as those which minimize the eccentricity of such paths. Reniers *et al.* generalize the MGF metric to extract simplified surface and curve skeletons [145], and next construct a part for each branch of a shape's curve skeleton [140]. Part borders correspond to curve-skeleton junction points, and are created by shortest paths traced on the shape surface around these junctions [44]. However, curve skeletons can contain many spurious junctions which change widely when the shape is slightly perturbed. Reniers *et al.* alleviate this by heuristics that shift cut-points along the curve skeleton to optimize for cut stability and planarity [141]. Yet, this method cannot segment shapes of large geometric, but little topological, variability, like a pawn chess piece: Its curve skeleton has no junction points, so [141] cannot separate the pawn's head, body, and base, although these have different thicknesses.

Summarizing, the two elements of a good PBS (*where* to cut, and *how* to cut) are targeted in complement by different methods: Skeleton-based methods construct good partitioning-cuts efficiently, *e.g.* by shortest-paths [44, 141]. Yet, curve skeletons do not encode enough of the shape geometry. Global search methods that analyze a wide set of shape cuts offer good ways to select where to partition [74, 169]. Yet, they do not offer explicit constraints for the cut shapes, and exhaustive cut space search is expensive. Our method combines the advantages of the two above classes of methods, while minimizing their limitations.

4.3 METHOD

Our method has a simple intuition: Say we want to cut the shape in Fig. 4.1a close to points $A \dots E$. Which properties should these cuts have to yield a 'natural' PBS? In other words: How would a human draw such cuts? Figure 4.1 a shows five *undesirable* cuts: A is noisy, although it crosses a perfectly smooth surface zone; B is self-intersecting; C and D are too loose (long); and E is unnaturally slanted – a human asked to cut the shape at that point would arguably do it so across the finger's symmetry axis. Figure 4.1 b shows five cuts for the same points, computed with the method in this chapter. We argue that these cuts are more suitable for PBS than those in Fig. 4.1 a, as they are (1) tight, (2) locally smooth, (3) self-intersection

free, (4) and locally orthogonal to the shape's symmetry axis. An additional property that cuts should satisfy is (5) being closed curves, so that they divide the shape's surface into different parts. Note that these properties follow well-known perceptual principles that model how humans understand shape and its parts, such as the minima and short-cut rules [23, 77, 177].

We construct such cuts as follows: First, we compute a simplified medial surface of the input shape (Sec. 4.3.1). For each medial point, we next construct a cut having the above properties (Sec. 4.3.2). This answers the question "how to cut". By analyzing the resulting cut space, we next select a cut-set that gives us the borders of salient shape-parts (Sec. 4.3.3). This answers the question "where to cut".

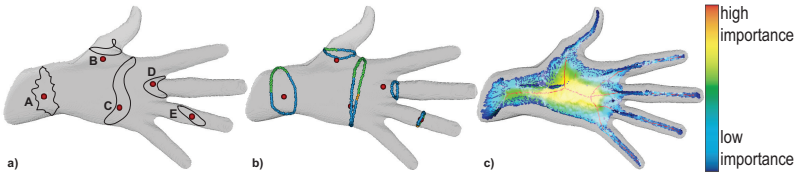


Figure 4.1: Possible cuts for part-based segmentation. Suboptimal cuts (a). Cuts created by our method (b). Medial surface colored by its importance metric (c).

4.3.1 *Skeletonization*

For a binary shape $\Omega \subset \mathbb{Z}^3$ with boundary $\partial\Omega$, its Euclidean distance transform $DT_{\partial\Omega} : \Omega \to \mathbb{R}_+$ is

$$DT_{\partial\Omega}(\mathbf{x} \in \Omega) = \min_{\mathbf{y}\in\partial\Omega} \|\mathbf{x}-\mathbf{y}\|. \tag{4.1}$$

The medial surface, or surface skeleton, of $\partial\Omega$ is next defined as

$$S_{\partial\Omega} = \{\mathbf{x} \in \Omega | \exists \{\mathbf{f}_1, \mathbf{f}_2\} \subset \partial\Omega, \mathbf{f}_1 \neq \mathbf{f}_2, \|\mathbf{x}-\mathbf{f}_1\| = \|\mathbf{x}-\mathbf{f}_2\| = DT_{\partial\Omega}(\mathbf{x})\} \tag{4.2}$$

where $\mathbf{f}_1$ and $\mathbf{f}_2$ are the contact (or feature) points with $\partial\Omega$ of the maximally inscribed ball in Ω centered at $\mathbf{x}$ [71, 148, 172]. These, in turn, define the so-called feature transform $FT_{\partial\Omega} : \Omega \to \mathscr{P}(\partial\Omega)$

$$FT_{\partial\Omega}(\mathbf{x} \in \Omega) = \operatorname*{argmin}_{\mathbf{y}\in\partial\Omega} \|\mathbf{x}-\mathbf{y}\|. \tag{4.3}$$

Medial surfaces are sensitive to small-scale noise on Ω, especially when using voxel-based models to sample the embedding space. To alleviate this, medial surfaces can be *regularized* by a computing a so-called importance metric $\rho : S_{\partial\Omega} \to$

$\mathbb{R}_+$, such as the medial geodesic function (MGF), which sets $\rho(\mathbf{x})$ to the length of the shortest path on $\partial\Omega$ between the two feature points of $\mathbf{x}$ [44, 145]. As the MGF monotonically increases from the medial-surface boundary to its center, upper thresholding it by a minimal importance ρ_{min} yields connected and noise-free simplified medial surfaces (though tunnel preservation requires additional work) [145]. Figure 4.1c shows a regularized medial surface obtained by upper-thresholding the MGF metric in [145].

4.3.2 *Cut model*

The first step of our PBS is to compute a rich set of cuts, or cut space $\mathscr{S}$, which all satisfy properties (1-5) listed in Sec. 4.3. To build a cut $c \in \mathscr{S}$, consider a point $\mathbf{x} \in S_{\partial\Omega}$. Here and in the following, we use for $S_{\partial\Omega}$ the simplified surface skeleton of Ω, obtained by upper-thresholding the MGF importance metric by a given value ρ_{min}. By definition, $\mathbf{x}$ has at least two feature points $\mathbf{f}_1$ and $\mathbf{f}_2$ on $\partial\Omega$ (Eqn. 4.2). Consider, for now, that there are precisely two such points. We first trace the shortest path $\gamma_1 \subset \partial\Omega$ between $\mathbf{f}_1$ and $\mathbf{f}_2$ (Fig. 4.2a), whose length is the MGF value for $\mathbf{x}$ (Sec. 4.3.1). Next, we find the midpoint $\mathbf{m}$ of γ_1, *i.e.* the voxel of γ_1 furthest in arc-length distance from both $\mathbf{f}_1$ and $\mathbf{f}_2$. We then trace a ray through $\mathbf{x}$ and oriented in the direction $\mathbf{x}-\mathbf{m}$, and find the point $\mathbf{o}$ where this ray 'exits' Ω (Fig. 4.2b). Intuitively, $\mathbf{o}$ is on the 'other side' of $S_{\partial\Omega}$ as opposed to $\mathbf{m}$. Finally, we trace the two shortest paths on $\partial\Omega$ connecting $(\mathbf{f}_1,\mathbf{o})$ and $(\mathbf{f}_2,\mathbf{o})$ respectively (Fig. 4.2 c,d). Our final cut c for point $\mathbf{x}$ is given by $\gamma_1 \cup \gamma_2 \cup \gamma_3$.

While c is piecewise geodesic (so locally smooth), it can be non-smooth at the three endpoints $\mathbf{f}_1, \mathbf{f}_2$ and $\mathbf{o}$ of γ_i. Also, our construction does not globally make c as tight as possible. To fix both issues, we perform 5 iterations of constrained Laplacian smoothing over c, with a kernel size of 10 voxels. We prevent c leaving the surface by reprojecting its voxels to their closest points on $\partial\Omega$ after each iteration. This smooths out possible 'kinks' at $\mathbf{f}_1, \mathbf{f}_2$ and $\mathbf{o}$, thus making c globally smooth and tight. If such kinks are very small or inexistent, smoothing has no effect, as c is globally geodesic. In that case, Laplacian smoothing shifts c's points along the surface normal, since c's acceleration c'' is normal to the surface, so reprojection moves the smoothed points back to their original location.

4.3.2.1 *Cut properties:*

Our cuts meet the desired properties we require for PBS, as follows:

1. *Tight:* Cut parts γ_i are piecewise-geodesic, thus shortest curves on $\partial\Omega$. Also, the constrained Laplacian smoothing shortens potential kinks present at the geodesic endpoints $\mathbf{f}_1, \mathbf{f}_2$, and $\mathbf{o}$, thus making the entire c wrap tightly around the shape;

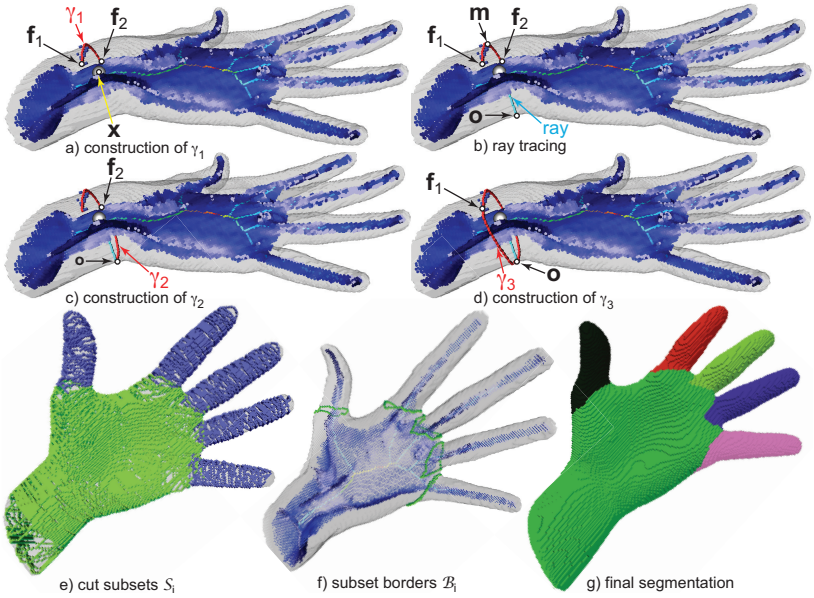


Figure 4.2: Cut construction (a-d) and cut space analysis (e-g) for part-based segmentation.

2. *Smooth:* Smoothness is guaranteed by the same properties as for tightness, *i.e.*, piecewise geodesicness and constrained Laplacian smoothing;

3. *Self-intersection free:* c is a geodesic triangle (three geodesics linking three *different* points on $\partial\Omega$) whose edges do not intersect except at endpoints, by definition;

4. *Locally orthogonal* to the symmetry axis: The cut $c(\mathbf{x})$ surrounds the medial surface $S_{\partial\Omega}$ around point $\mathbf{x}$, by construction. Hence, it also surrounds the so-called *curve skeleton* of $\partial\Omega$, which is a 1D structure locally centered within $S_{\partial\Omega}$ with respect to its boundary $\partial S_{\partial\Omega}$. While we do not have a formal proof of local orthogonality, we observed in practice that our construction always creates cuts that are visually orthogonal to the curve skeleton;

5. *Closed:* The cut c is a closed (Jordan) curve by construction.

4.3.2.2 *Implementation*

Our method requires the efficient and robust computation of regularized medial surfaces for 3D voxel shapes. For this, we tested the methods in [145] and [83]. As also described in [83], both methods produce very similar medial surfaces, and

also deliver a skeleton importance metric, required to simplify skeletons to *e.g.* eliminate noise or small details. As the method in [83] is on average 10 times faster, we use this technique to compute our surface skeletons.

To build γ_1, we need two feature points $\mathbf{f}_1$ and $\mathbf{f}_2$ for each medial surface point $\mathbf{x}$. Two issues exist here: (1) Computing the feature transform $FT(\mathbf{x})$ on digital shapes cannot be done via Eqn. 4.3, given the finite voxel grid resolution [139, 145]. To fix this, we compute the so-called extended feature transform $EFT(\mathbf{x})$ which finds all closest-points on $\partial\Omega$ to all 26 neighbors of $\mathbf{x}$, and which is a superset of $FT(\mathbf{x})$ [145]. From this superset, we select exactly two feature points that best represent the symmetric embedding of $S_{\partial\Omega}$ in Ω. For this, we select the two feature points $\{\mathbf{f}_1, \mathbf{f}_2\} \subset EFT(\mathbf{x})$ that maximize the angle $\widehat{\mathbf{f}_1\mathbf{x}\mathbf{f}_2}$. We trace the ray used to find $\mathbf{o}$ by Bresenham's 3D line-tracing algorithm on the voxel shape. We compute geodesics by Dijkstra's shortest-path algorithm on the connectivity graph of voxels of $\partial\Omega$, using A^* heuristics to speed the search, and using edge weights that approximate neighbor-voxel distances by Kiryati's scheme [93] for better path-length accuracy. Finally, we reproject Laplacian-smoothed points on the shape surface by using the fast ANN library for finding nearest-neighbors [122].

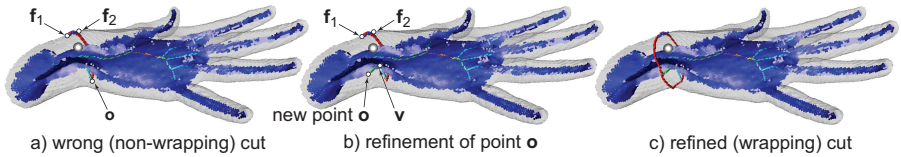


Figure 4.3: Refinement of cut construction.

In a few cases, point $\mathbf{o}$ found as above does not lie on the opposite side of $S_{\partial\Omega}$ with respect to $\mathbf{m}$, so the resulting cut will not wrap around the medial surface (Fig. 4.3 a). When this happens, we trace a ray in direction $\mathbf{f}_1 - \mathbf{f}_2$ from the midpoint $\mathbf{v}$ of the current ray, and set $\mathbf{o}$ to the voxel where this new ray exits Ω (Fig. 4.3 b). If the new $\mathbf{o}$ still does not yield a wrapping cut, we repeat the refinement (Fig. 4.3 c). This produces cuts wrapping around the medial surface for all our test shapes within 3 up to 4 refinement steps.

4.3.3 *Cut space partitioning*

For any voxel $\mathbf{x}$ of a shape's medial surface $S_{\partial\Omega}$, we can create a cut $c(\mathbf{x})$ which has good properties for PBS. Intuitively, $c(\mathbf{x})$ is a good way to cut the shape at point $\mathbf{x}$, *if* we want a cut there. We now must decide *where* we want to cut to get a PBS with desired global properties. Let $\mathscr{S} = \{c(\mathbf{x}) | \mathbf{x} \in S_{\partial\Omega}\}$ be the space of all cuts created from $S_{\partial\Omega}$. Given our cut properties, cuts on the same shape-part share similar properties *e.g.* position, orientation, and length. Cuts for different parts

have different properties. Consider our hand model: Finger cuts are short; wrist cuts have average length; and palm cuts are longest. For a shape having a rump and protruding parts, cuts for parts are shorter than cuts for the rump. We use these insights to partition $\mathscr{S}$ in subsets $\mathscr{S}_i$ so that $\cup_i \mathscr{S}_i = \mathscr{S}$ and $\mathscr{S}_i \cup \mathscr{S}_{j \neq i} = \varnothing$. We discuss next two ways to achieve this partitioning.

4.3.3.1 *Histogram-based partitioning*

A first way to do this is to use the histogram of cut lengths over $\mathscr{S}$, as described in [60]. This works as follows. Histogram peaks show large similar-length cuts, so partitioning it by thresholds in the valleys between peaks gives our desired subsets $\mathscr{S}_i$. Figure 4.4a shows the cut-length histogram for the hand model. Its three main peaks describe cuts on the fingers, wrist, and palm; the two valleys give the two thresholds needed to separate fingers from the palm and the palm from the wrist. Figure 4.4b shows the final segmentation computed by partitioning the histogram into the three aforementioned parts.

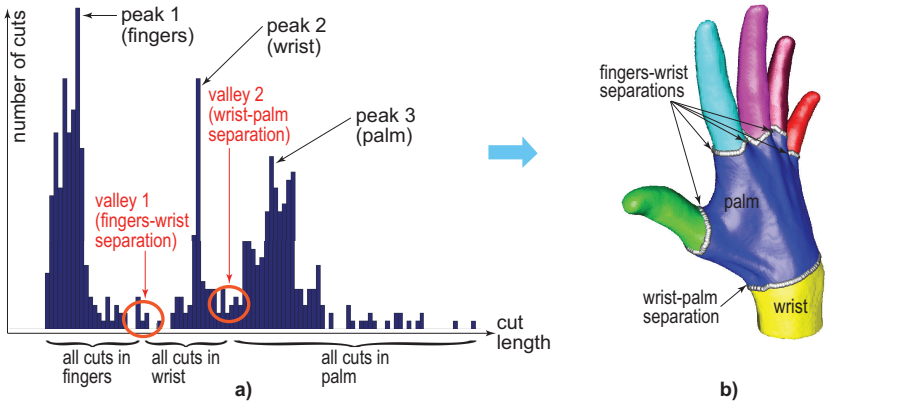


Figure 4.4: (a) Cut-length histogram for the *hand* model. (b) Segmented hand model based on left histogram (Sec. 4.3.3.1).

An important problem of the histogram-based partitioning is how to find its valleys *robustly* and *automatically*. As visible in Fig. 4.4a, the cut-length histogram is quite noisy, mainly due to the discrete nature of our cuts which are constructed in voxel space. Hence, robustly finding these valleys is a delicate process. To decrease the noise influence, we filter the histogram by mean shift [36]. This has the effect of 'sharpening' the cut-distribution and separate peaks from valleys more clearly. Following [60], we define a peak as a histogram value exceeding λ times the cut count $\|\mathscr{S}\|$, and a valley as a value less than a fraction μ of λ. Setting $\lambda \simeq 0.01$ and $\mu \simeq \lambda/3$ gives good results for a large range of shapes. However, problems appear

for shapes having small-scale surface details. Such small details, on the one hand, cause noise-level variations in the cut lengths; on the other hand, their cut counts $\|\mathscr{S}\|$ are low, and thus separated from each other by very shallow, thus hard to detect, valleys in the histogram. Figure 4.5 shows such an example. Here, ideally, we would like to segment the limbs, head, and details (fingers, hears, muzzle) of the armadillo model. The four instances in the figure show different results obtained for quite similar values of λ and μ. All these results show various degrees of over- or undersegmentation. A second issue of the histogram-based partitioning is that it does not offer an intuitive control of the parameters λ and μ: We cannot easily determine optimal values for them based on the number of segments we would finally like to obtain.

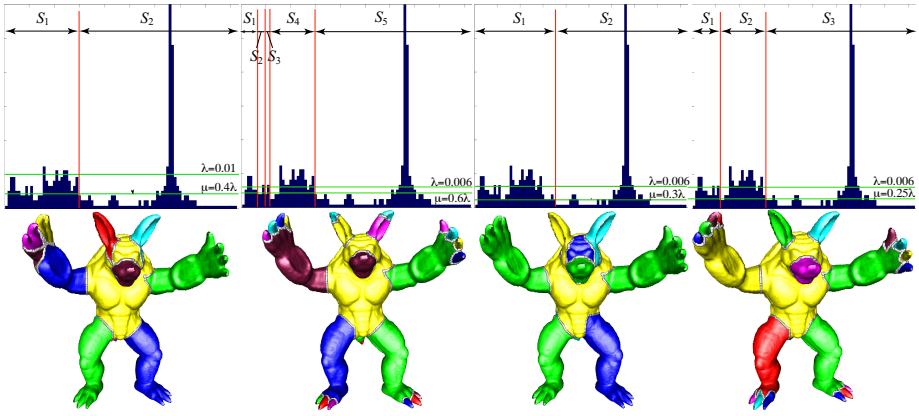


Figure 4.5: Subsets and corresponding segmentations obtained for different values of λ and μ (Sec. 4.3.3.1).

4.3.3.2 *Clustering-based partitioning*

To alleviate the aforementioned problems of histogram-based partitioning, we propose to partition the cut space $\mathscr{S}$ using a clustering approach. We first define a dissimilarity function $\delta : \mathscr{S} \times \mathscr{S} \to \mathbb{R}_+$ as

$$\delta(c_1 \in \mathscr{S}, c_2 \in \mathscr{S}) = \alpha \|l(c_1) - l(c_2)\| + \beta \|\mathbf{x}(c_1) - \mathbf{x}(c_2)\|, \tag{4.4}$$

where, for a cut c, $l(c)$ denotes the cut length and $\mathbf{x}(c)$ denotes the location of the skeleton-point from which c was generated, respectively (see Fig. 4.2); and $\alpha \in [0,1]$ and $\beta \in [0,1]$ are weight factors for the length, respectively distance, components of δ. To allow for a meaningful comparison between cut-lengths and cut-positions, we first normalize all cut lengths to the range $[0,1]$ and the voxel

shape to the range $[0,1]^3$, respectively. The function δ will thus take low values for cuts which are similar in length and close to each other, and high values for cuts of different lengths and/or located far away over $\partial\Omega$. Next, we use hierarchical bottom-up agglomerative clustering to iteratively group all cuts in $\mathscr{S}$, represented by the distance matrix given by δ. During this process, the most similar two cut-clusters, as determined by a so-called *linkage* function, are iteratively merged in a new cluster, until a single cluster containing all cuts is obtained. This creates a binary tree, or dendrogram, D. Full algorithm details, including a public implementation, are available at [78]. Cutting D at a desired level from its root next gives us a set of nodes, which are precisely our partitions $\mathscr{S}_i$.

Compared to the histogram-based partitioning, the clustering-based method is significantly more robust with respect to noisy cuts, as it has no thresholds or similar parameters. The only end-user parameter it requires is the number N of parts to create, which determines the level where to cut the dendrogram D. Compared to the parameters λ and μ of the histogram-based method, specifying the desired number of parts N is much simpler and more intuitive. As such, this is the method of choice for constructing the partitions $\mathscr{S}_i$ which we will use in the remainder of this paper.

4.3.3.3 *Segment border construction*

Subsets $\mathscr{S}_i$ do not (yet) coincide with our desired segments. Indeed, an $\mathscr{S}_i$ can contain logically disjoint cuts of similar lengths – *e.g.* all cuts on the fingers (blue in Fig. 4.2 e) are in the same subset. Also, $\mathscr{S}$ does not fully cover $\partial\Omega$, since we compute it from the *simplified* medial surface (Sec. 4.3.2).This is shown by the gaps between cuts in Fig. 4.2 d. To fix this, the method in [60] proposes to define a cut $c(\mathbf{x})$ as being a border $\mathscr{B}_i$ of subset $\mathscr{S}_i$ if $c(\mathbf{x})$ belongs to a different subset than any of the cuts $c(\mathbf{y})$, where $\mathbf{y}$ are the 26-neighbors of $\mathbf{x}$ on $S_{\partial\Omega}$. Using this definition, we can find the set of cuts $\{\mathscr{B}_i\}$ that represent the borders of our final segments (Fig. 4.2 f). Note that, if a cut is marked as border, at least one of its neighbor cuts will be in a different cut subset, by definition. Hence, that neighbor cut will also be a border, so more than one border will be produced from a $3 \times 3 \times 3$ voxel neighborhood. To remove such duplicates, we keep, for each such neighborhood, the shortest border.

While this method finds borders located close to areas where different partitions $\mathscr{S}_i$ meet, it has problems for parts which meet along so-called *ligature* skeleton branches [172]. To explain this, consider the situation sketched in Fig. 4.6a. The skeleton $S_{\partial\Omega}$ consists here of three branches that correspond to the three shape parts that meet at the central junction point. Consider now the vertical branch that describes the thinner (red) part. The first part of this branch corresponds to so-called regular skeleton points, which have a one-to-one mapping with the shape

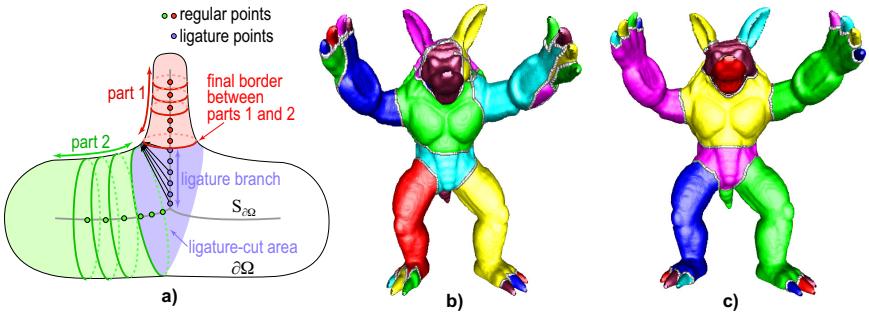


Figure 4.6: (a) Border construction problems for ligature regions. Part borders computed by (b) the original method in [60] and (c) our new ligature-sensitive method (Sec. 4.3.3.3).

surface $\partial\Omega$ via the feature transform $FT_{\partial\Omega}$. The second part of this branch contains ligature points (blue), which have a many-to-one mapping to $\partial\Omega$, as also indicated by the black feature vectors in the drawing. Ligature points do also generate cuts in our cut space, like regular points. However, as compared to regular cuts, such as the red and green ones drawn in the figure, ligature cuts are far less stable – they can fall anywhere in the blue surface area indicated in the figure. Separately, note that our desired red and green segments will meet precisely in this ligature area, so their separating border, when computed by the method in [60], can fall anywhere in this area.

We propose next a way to fix this problem. Consider two parts $\mathscr{S}_1$ and $\mathscr{S}_2$ which are adjacent, *i.e.*, have at least two neighbor cuts in the sense described earlier in this section, *e.g.* the green and red parts in Fig. 4.6a. Let l_1 and l_2 be the average cut-lengths over $\mathscr{S}_1$ and $\mathscr{S}_2$ respectively. We next decide that the border $\mathscr{B}$ separating $\mathscr{S}_1$ from $\mathscr{S}_2$ should come from the part $\mathscr{S}_i$ that has the smaller average-length l_i. This heuristic models the idea that we want to cut the smaller part $\mathscr{S}_1$ as precisely as possible from the larger adjoining part $\mathscr{S}_2$. To find the exact location of this border, we proceed as follows. Let $\mathscr{S}_1$ be the part having the smaller average-length, *i.e.*, $l_1 < l_2$. We next collect all cuts $P = \{c_i\} \subset \mathscr{S}_1$ whose lengths $l(c_i)$ are smaller than $l_1 + a \cdot \sigma_1$, where σ_1 is the cut-length standard deviation over $\mathscr{S}_1$, and a is a constant set to 1.2 for all tested shapes. The set P skips the potential ligature-cuts in $\mathscr{S}_1$, which are longer than regular cuts. From this candidate border-set P, we next select the border $\mathscr{B}$ as being the cut which is geometrically closest to $\mathscr{S}_2$, *i.e.* cuts $\mathscr{S}_2$ as closely as possible with respect to $\mathscr{S}_2$. This also favors creating *short* borders, in line with requirement 1 (Sec. 4.3.2.1). For the shape in Fig. 4.6a, this yields the border $\mathscr{B}$ indicated in the drawing, which is outside the ligature area *and* is short. Figure 4.6 compares our new variance-based borders with the ones produced by the original method in [60] for a shape having many ligature regions

(*e.g.* palm-hand, arm-torso, and ears-head junctions). As visible, the new borders separate the perceived shape segments better than the original ones, and are also shorter, while producing the same overall segmentation (number and location of parts).

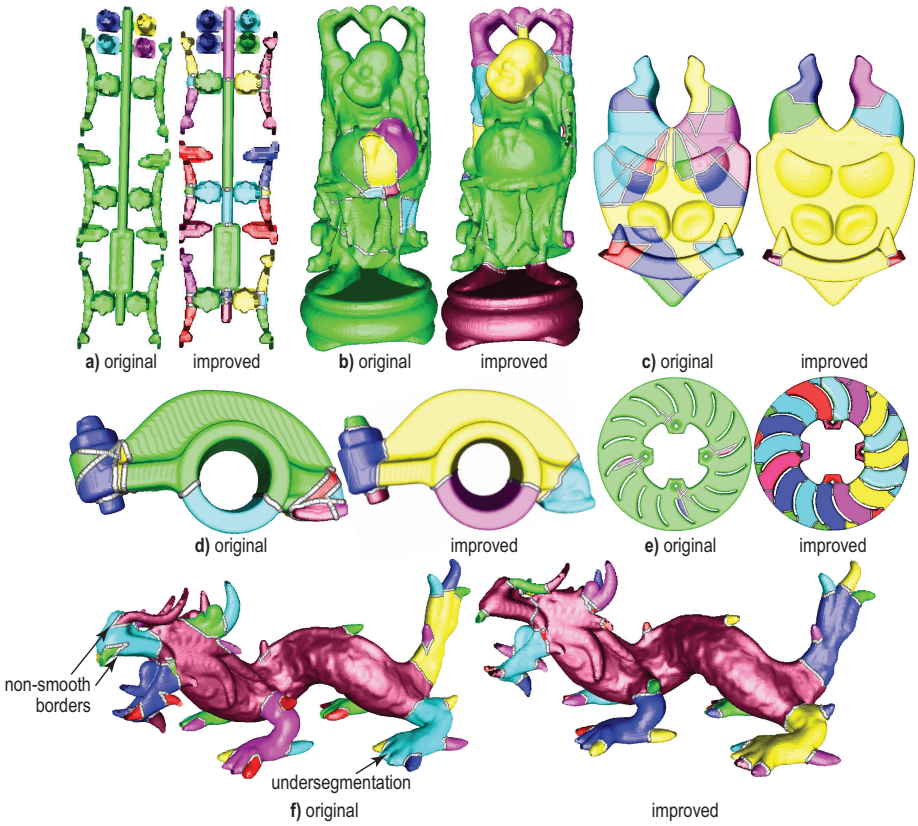


Figure 4.7: Comparison of original border construction [60] and improved ligature-sensitive method (Sec. 4.3.3.3).

Once the part borders $\mathscr{B}_i$ are determined, we compute the final segments by finding the connected components of $\partial\Omega$ separated by these borders, via a simple flood-fill algorithm on $\partial\Omega$, and visualize these segments, for illustration, by coloring them so that adjacent segments get different colors (see Fig. 4.4b and following figures in the paper).

4.3.3.4 *Final results*

Figure 4.7 shows several examples where we compare our improved segmentation pipeline (using clustering-based partitioning of the cut space and ligature-sensitive border creation) with the original segmentation results in [60] (which use histogram-based partitioning and the ligature-agnostic border creation). For shapes (a,b,e) the new method removes the severe undersegmentation effects of the old method which are due to the difficulty of finding appropriate histogram thresholds λ and μ. For shapes (c,d), the new method removes the border instability due to skeleton ligature points, and creates tighter and better oriented segment borders. Image (f) shows that the new method detects more small-scale details and also creates smoother segment borders (see markers in figure). Additionally, images (b,d,e) show that our method can handle shapes of genus larger than zero, *i.e.*, having tunnels.

4.4 INTERACTIVE SEGMENTATION FOR SHAPE EDITING

While useful, fully automated segmentation is not a solution in many contexts. Consider, for instance, the task of editing a 3D shape to *e.g.* enhance, deform, or remove certain features. The main time-consuming part here is accurately *selecting* the shape parts to process. This is typically done by interactive selection tools such as 2D or 3D bounding boxes or lasso tools [116, 213]. While such tools are quite efficient in a 2D setting, selecting details from complex 3D shapes still requires considerable user effort [34, 208, 213].

We present next a method to assist the process of efficiently selecting parts of a 3D shape using our cut space model. The key idea is simple: Given a 3D shape, the user can select any salient protruding part thereof by simply clicking on it in a 2D rendering of the shape. Next, once such a part is selected in 3D, the user can decide how to process the part, *e.g.*, deform, remove, or paint it.

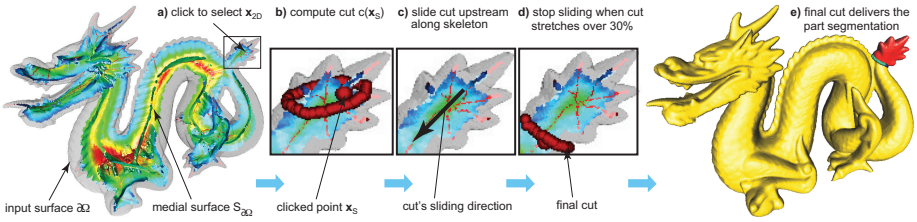


Figure 4.8: Interactive segmentation pipeline, starting from clicking a surface point (a) until obtaining the surrounding part (e).

Our proposal works as follows (see Fig. 4.8). Given a 3D rendering of the input shape Ω, the user clicks on a surface point $\mathbf{x}_{2D}$ thereof, in a classical 2D rendering of Ω (Fig. 4.8a). We next determine the corresponding 3D point $\mathbf{x}_{3D} \in \partial\Omega$. Thirdly, we find the closest surface-skeleton point $\mathbf{x}_S \in S_{\partial\Omega}$ to $\mathbf{x}_{3D}$, using the inverse of the feature transform FT of $\partial\Omega$, and construct the corresponding cut $c(\mathbf{x}_{3D})$, following the method outlined in Sec. 4.3.2 (Fig. 4.8b). This cut represents a way to 'slice' the input shape based on the clicked location $\mathbf{x}_{2D}$. Assuming the user clicked *anywhere* on a shape detail, this does not yet give us the *entire* shape detail containing the point $\mathbf{x}_{3D}$. To segment this detail from the rest of the shape, we proceed as follows. Let $\rho(\mathbf{x}_S)$ be the MGF importance of the skeleton point $\mathbf{x}_S \in S_{\partial\Omega}$ (Sec. 4.3.1). We then move $\mathbf{x}_S$ along the medial surface $S_{\partial\Omega}$ upstream, with a distance of one voxel, in strictly increasing order of the medial-surface importance $\rho(\mathbf{x}_S)$ (Fig. 4.8c). Since the importance ρ increases *monotonically* from the medial-surface boundary to its center [83, 145], the point $\mathbf{x}_S$ moves strictly 'upstream' along the medial surface $S_{\partial\Omega}$, towards the center of the skeleton $S_{\partial\Omega}$, which is the point of maximal importance [83, 145]. We stop this motion when the cut-length $\|c(\mathbf{x}_S)\|$ for the current skeleton point $\mathbf{x}_S$ increases over 30% as compared to the previous skeleton point in this upstream motion process. Practically, this stops the upstream motion of $\mathbf{x}_S$ once the sliding cut reaches the location where a part joins the main shape rump (Fig. 4.8d). Note that this location precisely corresponds to a large negative-curvature loop on the shape surface, which in turn is exactly the definition of the minima rules proposed by many researchers to segment protruding parts from a shape [23, 77, 88, 177]. When this event is detected, we use the current cut $c(\mathbf{x}_S)$ to separate the clicked shape part from the rump (Fig. 4.8e). Next, any shape-processing operations can be applied on this separated part, as desired by the user.

Figure 4.9 shows a simple example of such editing. Here, the user clicked three times, once inside each detail marked in blue in Fig. 4.9a, top-row. Using our part-selection procedure described earlier, we automatically select the three clicked parts, *i.e.*, the dragon's horn, tail, and hind leg spike. Next, we apply a simple erasing operation (for illustration purposes) to remove the selected details. Other shape-editing operations can be applied with the same ease, as desired. The final result is shown in Fig. 4.9b, top row. For comparison, Fig. 4.9 bottom-row shows the selection and editing operations performed by the related method of Clarenz *et al.* [34]. Our method achieves the same results, while being significantly simpler. Indeed, Clarenz *et al.* need to compute a differential surface classifier, encode it into a matrix, feed the matrix to an algebraic multigrid method that decomposes the matrix into a multiscale representation, select a suitable multiscale level, and threshold the basis functions representing the classifier on that level to find the clicked segment (for full details, we refer to [34]). In contrast, we only need to compute the shape's medial surface, select a point on it, and slide the cut generated

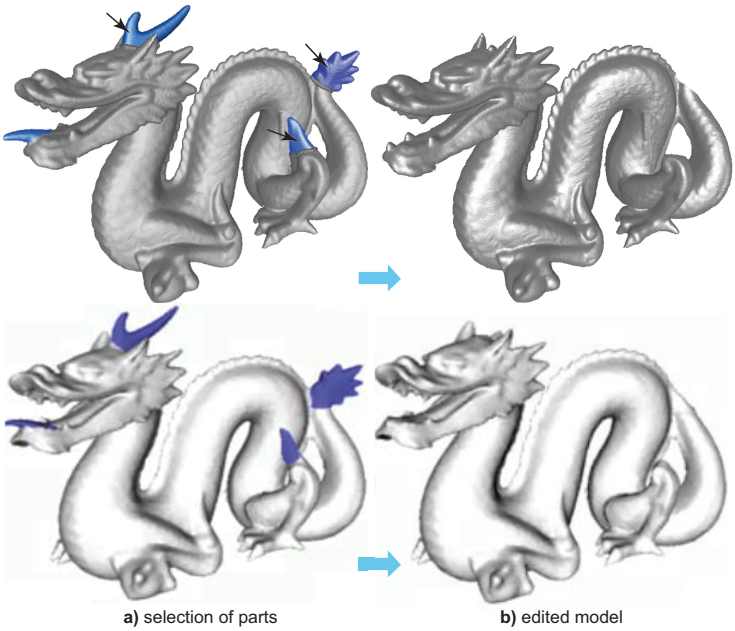


Figure 4.9: (a) Model with selected parts in blue. (b) Model with deleted parts. Bottom row: Method of Clarenz *et al.* [34]. Top row: our method.

by this point upstream the medial surface until its length increases by a desired threshold. Our interactive part selection method works in real-time as, upon a user click, we only need to compute a few tens of cuts from consecutive medial-surface points.

4.5 PARAMETER ANALYSIS

Our proposed segmentation pipeline involves several parameter values. For the method to be practically usable, end users need to understand (a) how these parameters affect the segmentation results, and (b) what are good preset values for them. In this section, we explore our method's parameter space and thereby address the above understanding goals. For this, we vary every parameter over its allowable range while keeping all other parameters at their preset values, and analyze the resulting segmentation results. The complete set of parameters of our method is listed in Tab. 2 and discussed next.

Description	**Introduced in**	**Allowed values**	Good preset
Skeleton simplification	Sec. 4.3.2	$\rho_{min} \in (0,1)$	$\rho_{min} = 0.01$
Cut dissimilarity δ	Eqn. 4.4	$\{length\text{–}only\ (\beta = 0),$ $length\text{–}and\text{–}position\ (\beta > 0)\}$	$length\text{–}only$
Linkage choice	Sec. 4.3.3.2	$\{single, centroid, full, average\}$	$average$
Input resolution	Sec. 4.1	$\|\Omega\| > 0$	$\|\Omega\| > 200^3$
Number of desired parts	Sec. 4.3.3.2	$N \in \mathbb{N}_{>0}$	task-dependent

Table 2: Complete set of method parameters with optimal preset values.

Simplification level: We use a simplified surface skeleton $S_{\partial\Omega}$ so as to avoid creating cuts from irrelevant spurious skeleton branches. Besides this, simplification allows removing skeleton details corresponding to small shape parts, to produce coarser segmentations. Thirdly, using simplified skeletons reduces computation time, as our method needs to create one cut per skeleton voxel. We empirically found that a skeleton simplification level of $\rho_{min} = 0.01\|\partial\Omega\|$ gives optimal results in terms of removing noise but keeping small shape details, and use this value as default for ρ_{min}. This result is in line with the independent observation that the same simplification level yields noise-free skeletons that capture all significant details of a 3D shape [145]. Additionally, simplified skeletons have voxels with large importance values, which in turn implies far-apart feature points $\mathbf{f}_1$ and $\mathbf{f}_2$ (see definition of the MGF importance metric in [44, 145]). This ensures that the ray casting used to compute cuts robustly finds cuts that wrap around the medial surface (Sec. 4.3.2). Figure 4.10 shows the effects of varying the simplification level ρ_{min} for the armadillo shape: Low ρ_{min} values capture finer-scale shape parts,

while higher values produce coarser segmentations.

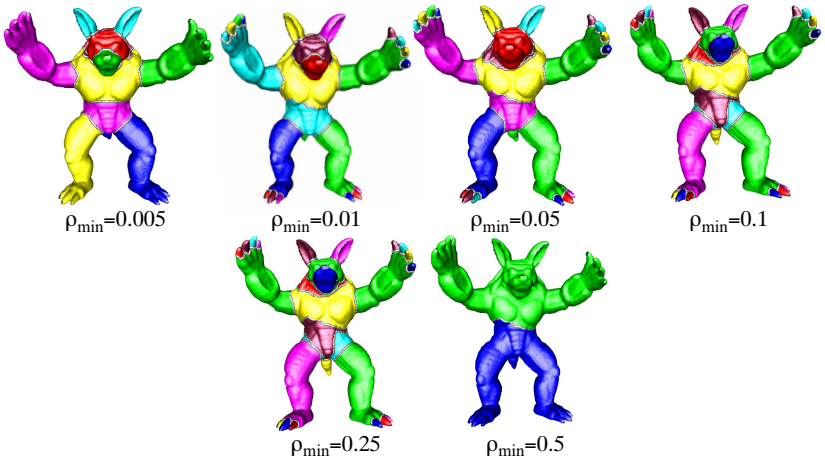


Figure 4.10: Segmentation results as function of skeleton simplification level ρ_{min}.

Linkage choice: Hierarchical bottom-up clustering works by iteratively merging the two most similar cut-clusters. To compute the similarity of two clusters $\mathscr{S}_1$ and $\mathscr{S}_2$, a so-called linkage function is used [78]. Well-known variants hereof are *single* linkage (the minimum of all pairwise distances between cuts in $\mathscr{S}_1$ and $\mathscr{S}_2$); *full* linkage (the maximum of all pairwise distances between cuts in $\mathscr{S}_1$ and $\mathscr{S}_2$); *average* linkage (the average of all pairwise distances between cuts in $\mathscr{S}_1$ and $\mathscr{S}_2$); and *centroid* linkage (distance between the averages of cuts in $\mathscr{S}_1$ and $\mathscr{S}_2$). We tested all four linkage strategies for the shapes presented in this paper. An example is shown in Fig. 4.11. Single linkage yields no segmentation, since border-cuts are shared by adjoining segments, so the single linkage of such segments is zero. Centroid linkage typically produces a visible degree of undersegmentation, as the cut averaging acts like a low-pass filter eliminating the effect of small shape details. Full linkage, in contrast, yields a small amount of oversegmentation, due to the maximum function involved in its computation. Finally, average linkage yields, in all tested cases, a balanced segmentation. As such, we set average linkage as the default value for our pipeline.

Dissimilarity function: As explained in Sec. 4.3.3.2, we construct partitions by clustering by comparing cuts based on their length only or length-and-position, as determined by the ratio of the parameters α and β in Eqn. 4.4. To test the effect of these parameters, we fix $\alpha = 1$ (since we always want to compare cut lengths), vary β between 0 and 1, and analyze the produced segmentations. Figure 4.12

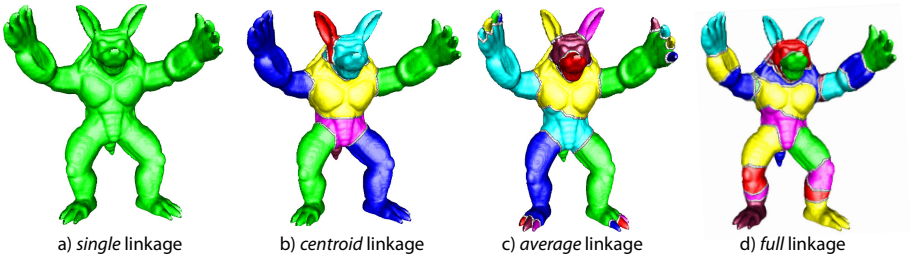


Figure 4.11: Segmentation results as function of the linkage method used in hierarchical clustering.

shows several results. For testing, we use here a shape exhibiting both thick and very thin parts and also having several elongated parts, so that both components of the dissimilarity function δ become important (Eqn. 4.4). We see that, when we use a non-zero importance β for the cut position (Fig. 4.12b), we obtain an oversegmentation of the length-only result: Long tubular-like parts, such as the trident shaft, torso, or limbs, are split into shorter segments. Also, we see a slight undersegmentation of details which only slightly differ in terms of local thickness, such as the bulge at the basis of the trident fork (Fig. 4.12a). Increasing β further yields an undersegmentation of the length-only result (Fig. 4.12c), as close cuts will be grouped in the same segment, regardless of their length – see *e.g.* the grouping of the trident spikes or fingers in the same segment. If oversegmentation of long tubular parts is not desired, then setting δ to length-only ($\beta = 0$) is a good default value. This is the value used for all examples in this paper except Fig. 4.12.

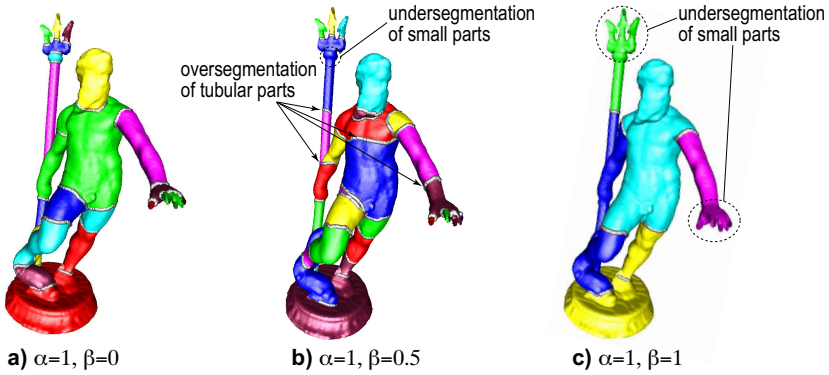


Figure 4.12: Segmentation results as function of the dissimilarity function δ. (a) Length-only. (b,c) Length-and-position.

Resolution: As our entire pipeline works in voxel space, the sampling resolution, or number of voxels used to represent our input shape, its skeleton, and the cut space, is an important parameter to examine. Figure 4.13 shows the segmentation results for four different resolutions. Overall, we see that the same segments are detected in all four cases, which tells that our method is robust with respect to sampling resolution. This is due to the fact that, once the used resolution is fine enough to capture skeletal details corresponding to small shape parts, then segments for those parts will be detected. Separately, we notice however an effect of the resolution in terms of *smoothness* of the produced cuts (see marked cut on the armadillo torso in Fig. 4.13). Low resolutions produce less smooth cuts, since the extended feature transform EFT of the input shape becomes inaccurate (see Sec. 4.3.2.2), and thus the feature-points used in our cut construction get noisy. As the resolution increases, so does the accuracy of our EFT in approximating the true FT, and thus the cuts become smoother *and* more orthogonal to the local symmetry axis of the shape. Combining the previous observations, we noticed, in practice, that a resolution of 400^3 voxels is sufficient to capture all salient shape segments and also produce smooth and well-oriented cuts.

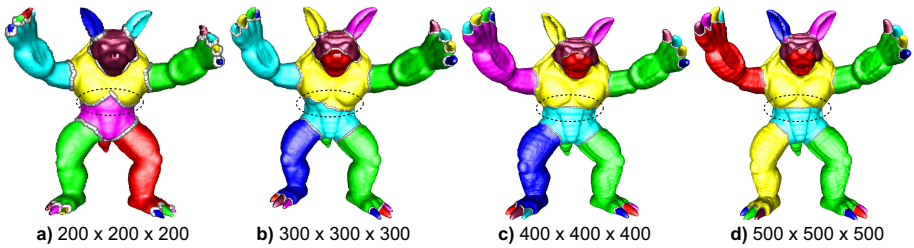


Figure 4.13: Segmentation results as function of the voxel resolution of the input shape.

Number of desired parts: The last parameter of our pipeline determines the number of desired parts to be produced by segmentation (value N, Sec. 4.3.3.2). This is the *single* free parameter of our method. Its setting depends largely on the specific application context, *e.g.*, what is the scale of details that we consider relevant and thus want to segment separately; and what is the amount of noise that is present on the input shape, which we do not want to yield separate segments. As such, we leave the setting of N to the end user. Segmentations for different N values can be created *interactively*, since the most expensive part of our pipeline, skeleton computation and cut creation, needs to be done only once for a given shape (see Sec. 4.6, Tab. 3 next). Figure 4.14 shows three settings for N for two different models which have a clear part-whole structure. As visible, increasing N produces more detailed segmentations, in a multiscale fashion.

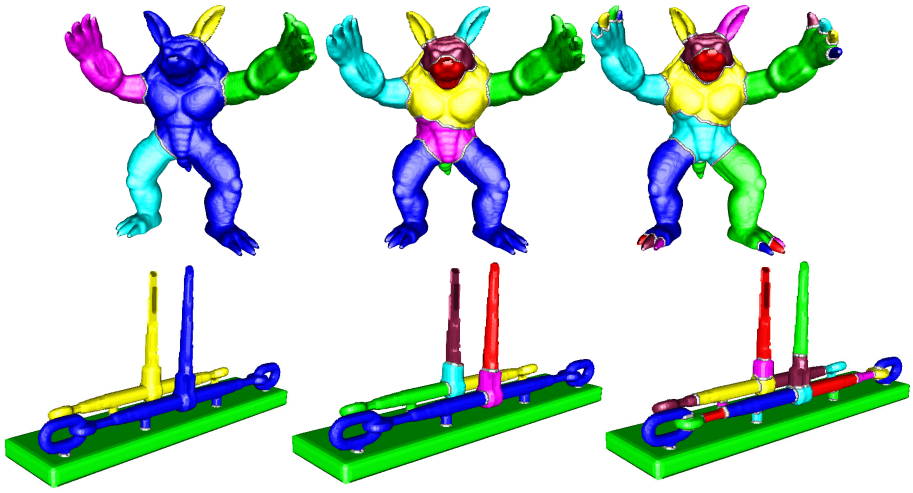


Figure 4.14: Segmentation results for different numbers of desired parts (increasing from left to right).

4.6 RESULTS AND COMPARISON

To start with, Fig. 4.15 presents several results of our method on a set of simple shapes. As visible, the produced segmentations are plausible, and, we argue, in line with what other part-based segmentation methods deliver (and also what a user would expect from these shapes).

To further evaluate our method, we consider several more complex shapes. We have tested our method on over 70 shapes provided as 3D polygon meshes, from the well-known shape repositories [2, 115], which we voxelized by *binvox* [123] at resolutions between 100^3 and 500^3 voxels. Results and comparison with related methods are discussed next.

Medial PBS methods: We first compare our results with [141], the best voxel-based PBS method that we are aware of which also uses medial descriptors for segmentation. We get very similar results, but find more fine-grained segments than [141] – see finger and ear details of the animal models, pig tail, dragon spikes, and microscope lens. Segment borders are smooth and locally orthogonal to the shape's symmetry axis, *i.e.*, similar to how a human would cut the shape at the respective places (Sec. 4.3). Our method finds segments of various sizes, ranging from details (dragon's tail, hound's ears), to large parts (limbs of various models).

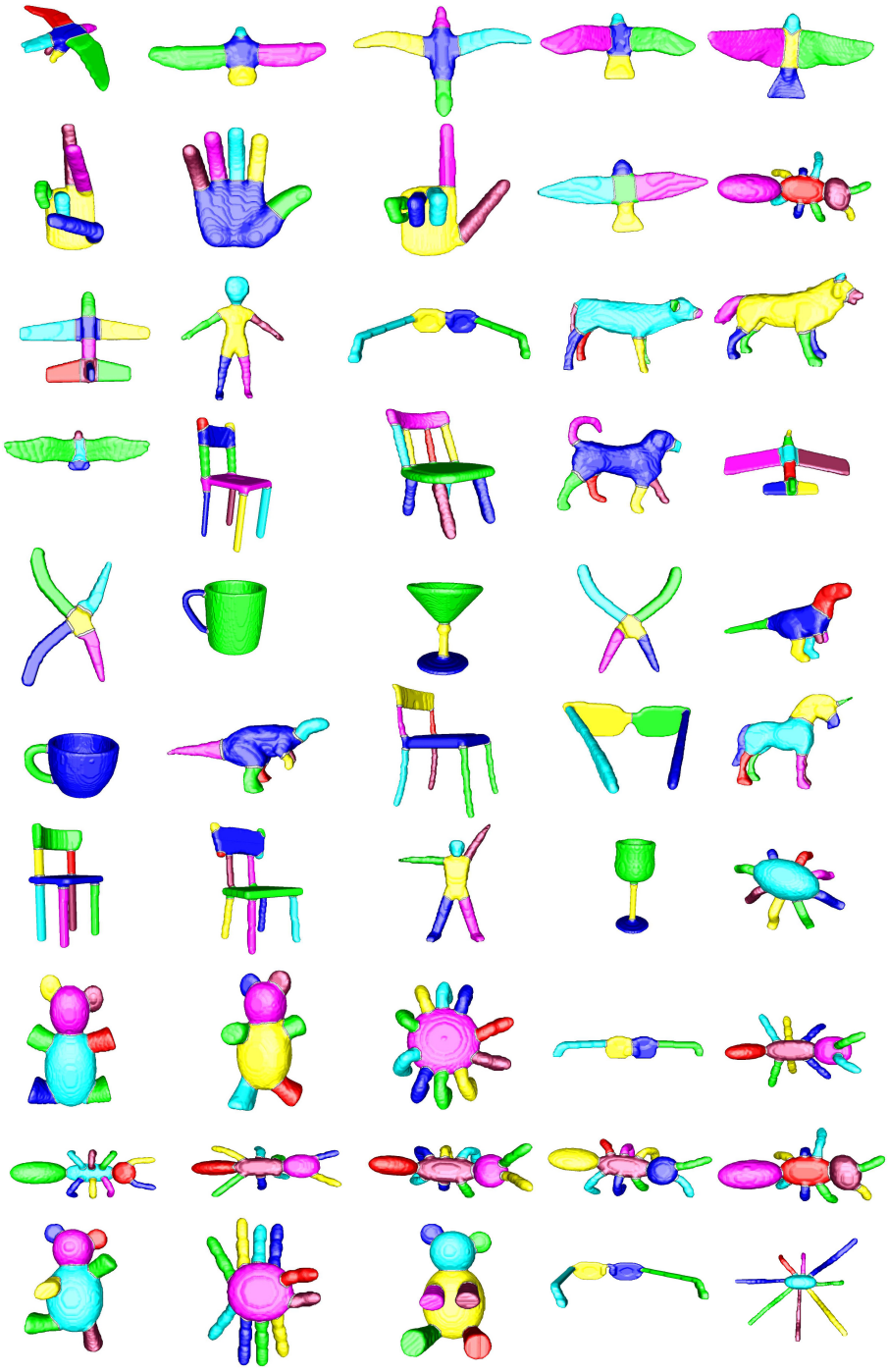


Figure 4.15: Results of our method on a set of simple shapes.

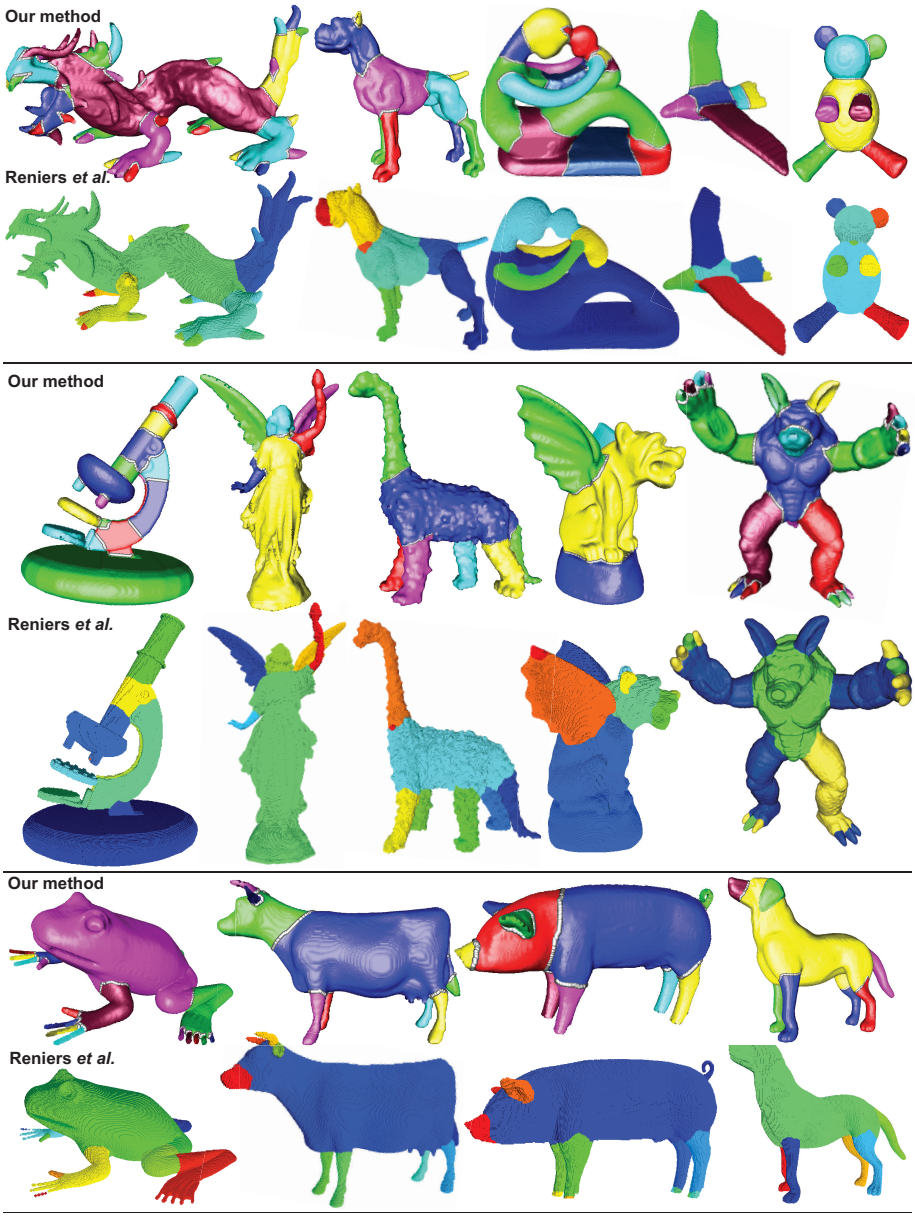


Figure 4.16: Part-based segmentations of our method *vs* Reniers *et al.* [141] (Sec. 4.6).

General PBS methods: We next compare our method with a larger class of general-purpose PBS methods (Fig. 4.17 a-k). The considered methods are [10, 100, 103, 105, 107, 140, 141, 201]. Here, Reniers *et al.* (1) denotes [140], and Reniers *et al.* (2) denotes [141]. These methods span from voxel-based to mesh-based, and use various segmentation heuristics (skeleton, curvature, salience, and topology-based). We argue that our method creates equally or, in some cases, more plausible PBSs. Since both our method and [141] use medial descriptors, computed by the same underlying method [145], a relevant question is how the two methods differ. We use (a) medial *surfaces*, while [141] uses *curve* skeletons; and (b) we find segment borders by analyzing *all* possible cuts, while [141] places such borders around the curve-skeleton branch junctions. Fig. 4.17 l-p shows five examples where the public implementation of [141] fails to segment at all. We find two causes for this: The shape parts in Fig. 4.17 l cannot be well described by curve-skeleton branches, as they are nearly rotationally symmetric. As few (if any) such junctions exist, [141] fails. The shape in Fig. 4.17 n is described by a mix of medial surfaces (base plate) and curve skeletons (tubular parts). As [141] only uses curve skeletons, data on the base plate is incomplete or missing. For the shapes in Fig. 4.17 m-p, the many heuristics in [141] to select cuts centered on the curve-skeleton fail, as they imply that such cuts should be nearly planar. This does not happen for the above shapes.

Multiscale: As described in Sec. 4.5, we can produce a multiscale segmentation by simply changing the number N of desired parts. This is a much simpler way to specify the desired level-of-detail than the earlier proposal in [60], where one had to simultaneously control two parameters λ and μ to yield the same result (see Sec. 4.3.3.1). Figure 4.17 r shows three such scales for the armadillo shape.

Invariance: Our method is pose invariant, as shown in Fig. 4.17 s. Indeed, our cut space essentially captures local shape thickness, which does not depend on pose. Additionally, as the cut clustering essentially depends on the *relative* difference in cut lengths and positions, and not on their absolute values, our method is also scale, translation, and rotation invariant.

Performance: Table 3 shows the time for creating cuts (t_{cuts}), medial surfaces (t_{skel}), cut space analysis (t_{space}), the total time of the original method in [60] (t_{total}), and total time for [141] ($t_{Reniers}$), for our method coded in C++ on an 8-core 3.5 GHz PC. As cuts are computed independently, we parallelized our method by *pthreads*, getting a speed boost factor of 7, close to the optimal value of 8 for our hardware. As visible, the original method [60] is slightly faster than [141]. We observe that most of the time is spent in the cut computation (t_{cuts} *vs* t_{total}). As such, we optimized the A^* method used to trace geodesics for cut construction

(Sec. 4.3.2.2), by using a fast priority queue implementation. The performance yielded by this optimization (Tab. 3, t_{optim}) is now significantly higher than the original method (t_{total}) and also much higher then [141]. Finally, we note that our method could successfully segment all tested shapes, while [141] failed on several shapes (Tab. 3, empty cells in column $t_{Reniers}$).

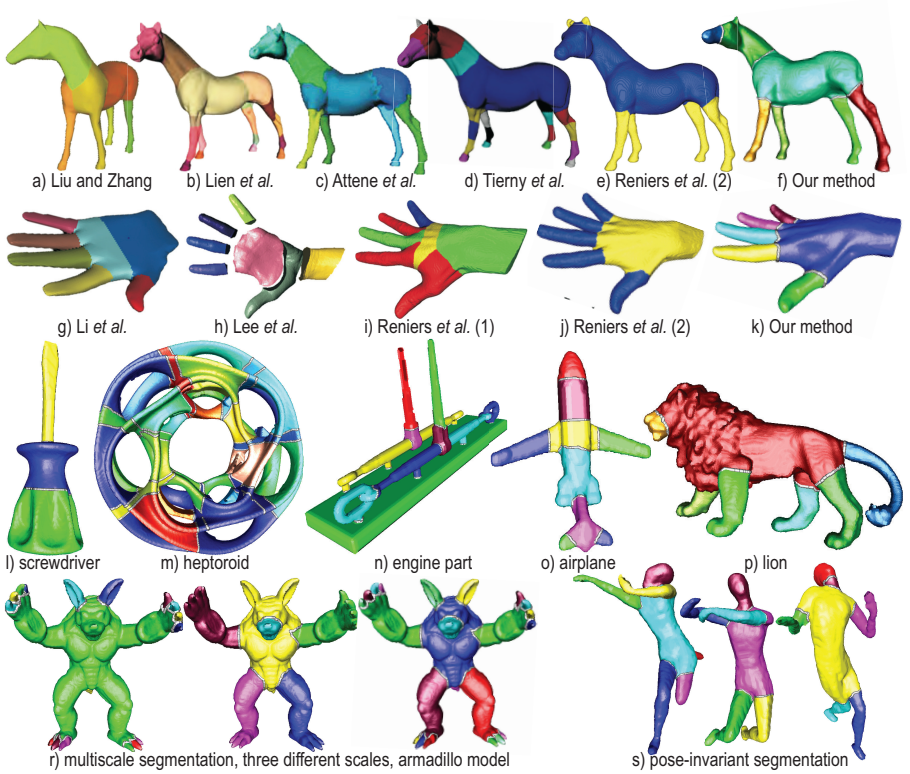


Figure 4.17: Comparison of our method with eight PBS methods (a-k). Our results for shapes where Reniers *et al.* fails (l-p). Multiscale (r) and pose-invariant (s) segmentations.

4.7 DISCUSSION

We next discuss several aspects of our proposed part-based segmentation method.

Global search: We create a PBS by finding all part-inducing cuts from the medial surface, and selecting a cut-subset by globally optimizing for part-similarity as

Shapes	**cuts** $\|\mathscr{S}\|$	**voxels** $\|\Omega\|$	**voxel volume**	t_{cuts}	t_{skel}	t_{part}	t_{total}	t_{optim}	$t_{Reniers}$
Dragon	2789	283238	400*400*400	50.8	1.90	0.03	52.73	9.19	40.26
Hound	1530	245759	300*300*300	23.24	1.51	0.01	24.76	6.39	25.1
Hyptoroid	4873	651478	400*400*400	400.5	3.36	0.04	403.90	47.5	-
Fertility	1354	199581	300*300*300	20.85	2.02	0.01	22.88	4.83	22.89
Gargoyle	488	129420	300*300*300	12.62	3.26	0.005	15.885	7.28	69.89
Microscope	1397	307863	300*300*300	44.14	1.58	0.01	45.73	8.12	198.02
Lucy	6201	1.04×10^6	300*300*300	68.01	0.63	0.09	68.73	12.7	52.65
Engine part	1501	135416	300*300*300	15.55	0.27	0.01	15.83	1.50	-
Screwdriver	1372	306480	300*300*300	13.14	0.60	0.01	13.75	4.48	-
Noisydino	1375	194117	300*300*300	14.79	1.19	0.015	16.00	3.72	20.2
Cow	1009	143938	256*256*256	8.15	0.96	0.01	9.12	2.41	14.34
Neptune	1908	211723	420*185*251	34.7	1.22	0.02	35.94	22.67	-
Airplane	741	76700	300*300*300	6.00	0.28	0.08	6.37	0.91	-
Bird	476	45638	300*300*300	2.28	0.18	0.003	2.47	0.40	7.98
Hand	584	58071	200*84*140	2.15	0.22	0.004	2.37	0.51	-
Lion	2181	381968	300*300*300	23.16	1.08	0.02	24.27	6.58	-
Horse	884	109555	142*300*251	9.58	1.24	0.008	10.83	2.56	-
Pig	959	145215	300*300*300	10.97	1.51	0.01	12.50	3.01	22.26
Dog	1241	184805	300*300*300	15.65	1.29	0.02	16.97	3.63	18.87
Hippo	838	166932	300*300*300	12.13	2.41	0.01	14.55	4.40	25.18
Rhino	1746	403399	300*300*300	25.20	2.15	0.03	27.39	7.57	-
Armadillo	2242	436933	300*229*252	47.55	2.67	0.03	50.26	12.21	-

Table 3: Shape sizes and segmentation times (in seconds) for [60], our optimized method, and Reniers *et al.* [141].

captured by cut lengths and/or positions. In contrast to purely topological PBS methods [140, 141], we search a much wider space of possible partitionings; yet, our search space is much smaller than that of other methods which look for cuts of *any* possible orientation [74], thereby achieving a good flexibility-performance balance. This is also visible if we compare our running times (Tab. 3, t_{optim}) with those reported in [74]: We process voxel shapes having tens up to hundreds of thousands of surface voxels ($\|\partial\Omega\|$) in under 10 seconds; on similar hardware, [74] processes meshes having *only* 4000 triangles in 4 minutes on average.

Simplicity: In our approach, we can use *any* medial surface skeletonization method, *e.g.* [7, 83, 145, 148, 175], as long as it outputs regularized skeletons. This makes our method directly applicable to mesh-based shapes, which allow fast medial-surface extraction [82], without the additional cost of voxelization.

Multiscale: Multiscale PBS occurs at two levels: (1) Simplified medial surfaces yield cuts only for important shape parts; (2) The user can specify the number of parts to be extracted from the shape.

Invariance: Our method is scale, translation, rotation, and pose invariant [141, 176], as shown by the model in Fig. 4.17 s (which is also used in [176] to show pose invariance). Note that pose-invariance is not guaranteed by default by other cut space segmentation methods, *e.g.* [74]. Figure 4.18 shows additional results computed by our method which illustrate the pose invariance we can obtain. As visible, in general, the method produces very similar segmentations for the same shape under different poses. However, certain detail differences exist. While undesired, such differences are not larger than other methods which claim pose invariance [141, 176].

Robustness: We robustly segment noisy or detail-rich surfaces, *e.g. dragon* and *dino* (Fig. 4.16) or *lion* (Fig. 4.17). Segment borders are smooth by construction (Sec. 4.3.2). Since our segmentation uses a subset of these cuts, and only considers *integral* cut properties (length, position) rather than differential ones (*e.g.* curvature), noise and/or small-scale details are robustly handled. Moreover, we avoid constructing segment borders from unstable cuts created from ligature skeletal points (Sec. 4.3.3.3).

Limitations: Our method's cost is $O(\|S_{\partial\Omega}\|\|\partial\Omega\|\log\|\partial\Omega\|)$. As our method parallelizes easily (Sec. 4.6), its practical cost is much lower than other skeleton-based PBS methods [140, 141] or cut-based methods [74]. For space constraints, we compare with only eight related PBS methods. More PBS methods exist, and quantitative metrics can be further used to measure segmentation quality [110]. Yet, even

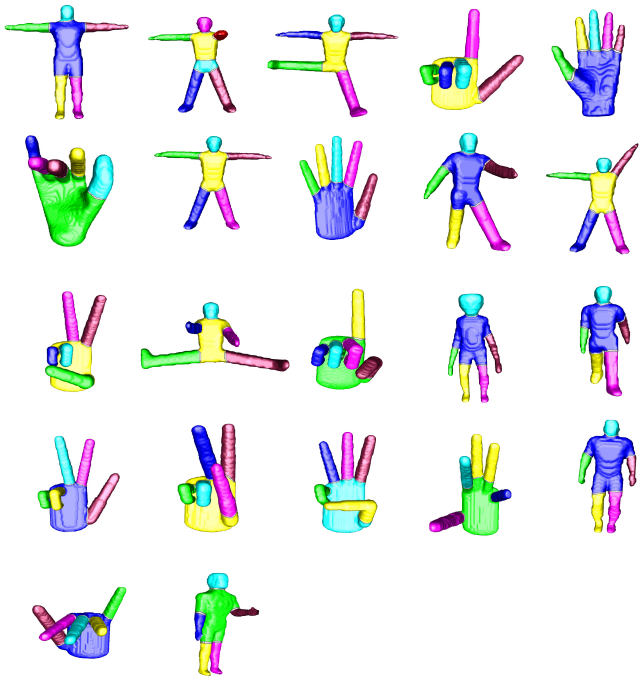


Figure 4.18: Results of our method illustrating the achieved pose invariance.

without such extra insights, we argue that our goal of showing that *surface* skeletons have added both theoretical and practical value for PBS, as opposed to the well-known use of *curve* skeletons for PBS, is well defended.

4.8 CONCLUSIONS

In this chapter, we have presented a refinemen of the method introduced in Chapter 3 for part-based segmentation of 3D voxel shapes. Both the original method and its refinement presented here work by analyzing the entire space of potential partitioning cuts constructed by using the shape's medial surface. To our knowledge, our approach is the first which uses medial *surfaces* for part-based segmentation, and thereby shows the added-value of medial surfaces for segmentation, as opposed to the well-known use of curve skeletons for the same task. We demonstrate our method on a wide variety of 3D shapes, and compare it with eight related segmentation methods. Our method can produce similar segmentations with less computational effort, and has a single intuitive end-user parameter to set – the number of desired segments. Compared to the original method in Chapter 3, we present various algorithmic improvements, most notably a new way to partition the cut space histogram using hierarchical clustering. Additionally, we discuss the parameters of our method in detail, and present both quantitative performance results and qualitative comparison results with other related methods.

Different ways to partition the cut space can be easily tried, *e.g.* cut similarities based on *e.g.* curvature, eccentricity, and orientation. This would lead to an entire family of PBS methods in a single simple implementation. Separately, cut-length-and-position histograms computed by our method could be an effective shape descriptor for retrieval and matching [160]. Separately, as already mentioned in the conclusion of Chapter 3, improvements of performance and/or scalability in model size could be obtained by adapting our cut space method to work on mesh representations. This aspect is discussed separately in Chapter 6.

5 A DESCRIPTOR FOR VOXEL SHAPE RETRIEVAL BASED ON THE SKELETON CUT SPACE

Chapters 3 and 4 have introduced the cut space descriptor and have shown its applications to the construction of part-based segmentations of 3D voxel shapes. In this chapter, we stay within the context of voxel shapes, but switch to a different application: shape matching and retrieval. Two-dimensional medial axes and three-dimensional curve skeletons have been long used for shape retrieval tasks. In contrast, and despite their ability to fully capture shape geometry and topology, three-dimensional surface skeletons have seen much less usage in this context. We present here a framework for shape matching and retrieval based on such surface skeletons. To this end, we employ the cut space generated from the surface skeleton, which has desirable invariance properties with respect to shape size, rotation, translation, pose, and noise, as described in Chapters 3 and 4. Next, we extract a histogram-based descriptor from this cut space, and discuss three different metrics to compare such histograms for shape retrieval. We illustrate our proposal by showing our descriptor's effectiveness in shape retrieval using a known shape-database benchmark.[1]

5.1 INTRODUCTION

The increase in computational resources and scanning technologies has made large collections of 3D shapes readily available. With this increase has come the demand for methods and techniques for searching for specific shapes in such databases [67, 158]. Within this field, *content-based shape retrieval* (CBSR) focuses on efficiently finding the most similar shapes to a given example shape from a given shape collection. Many classes of CBSR methods exist, based on various techniques, such as parametric templates [70], descriptor-based methods [193], and skeleton-based methods [186, 205].

Among the above, skeleton-based methods have the important advantage as being able to compare shapes at a high level, and based not only on their geometry, but also, specifically, their topology. This supports CBSR applications where the search is driven by *global* shape properties, such as structure and topology [37, 186], in addition to the more commonly used *local* geometry and texture

1 The text in this chapter is based on the paper: A descriptor for voxel shapes based on the skeleton cut space (C. Feng and A. Jalba and A. Telea), *Proc. Eurographics Workshop on 3D Object Retrieval (3DOR)*, eds. A. Ferreira and A. Giachetti and D. Giorgi, Eurographics, 2016.

properties. Classical medial axes (in 2D) and curve skeletons (in 3D) have proven here to be efficient and effective descriptors [161, 186]. In the 3D case, however, curve skeletons can capture well shape topology, and partially geometry, only for locally quasi-tubular shapes [39]. In contrast, surface skeletons capture well both topology and geometry for any 3D shape [85]. However, to our knowledge, such skeletons have not yet been used for CBSR.

We present in this chapter a shape descriptor that uses surface skeletons of 3D voxel shapes for CBSR. From these skeletons, we construct a so-called cut space that describes the local shape thickness properties in a multiscale way, *i.e.*, ignores small details which are less relevant for CBSR, but captures the actual shape thickness well. We next reduce this cut space to a histogram descriptor that efficiently and effectively captures thickness properties, and is also invariant to isometric shape transformations, pose, noise, and sampling resolution. We next study two existing metrics, and propose two new metrics, to compare such histograms for CBSR. We demonstrate our proposal by applying it to a well-known 3D shape benchmark database. Summarizing, the main contribution of this chapter is showing that surface skeletons have the potential to be efficient and effective instruments for constructing compact thickness-descriptors that, next, perform well for CBSR applications.

This chapter is structured as follows: Section 5.2 reviews related work, with a focus on skeleton-based CBSR. Section 5.3 presents our method. Section 5.4 illustrates our method on a variety of 3D shapes. Section 5.5 discusses our method. Section 5.6 concludes the chapter.

5.2 RELATED WORK

Many methods have been proposed for CBSR [193]. Most such methods use a two-step approach:

1. First, a so-called descriptor is computed from a shape, aiming to compactly capture the relevant shape properties for the search process, and also to be resistant to shape changes deemed as irrelevant for the same context, such as isometric transformations and noise [89].

2. Next, the best-match for the given descriptor is searched for in a database of shapes, which is organized (indexed) so as to speed up the search process;

Our focus in this work is on both above steps, *i.e.*, descriptor computation and descriptor comparison. To this end, we next extend the discussion on shape retrieval from Sec. 2.2.3 to review in more detail typical descriptors used in CBSR. Next, we focus on skeletal descriptors, which are central, and specific, to our method.

Shape descriptors for CBSR: Osada *et al.* [125] describe a shape by a distribution capturing its geometric properties such as angle, distance, and volume between random point-pairs on the shape surface. Kazhdan *et al.* [89] propose a method to create a descriptor which is rotation invariant, based on spherical harmonics. The descriptor is further refined in [90] to capture a shape's reflective and rotational symmetry. However, no exact accuracy result is provided, except for a stated 15% improvement in precision-recall plots as compared to PCA alignment of models prior to retrieval. The spherical extent function (SEF) measures the thickness of a shape along rays passing through its origin [157]. While simple, this descriptor does not work well for thin and entangled shapes.

Liu *et al* [108] propose the directional histogram model (DHM) that is very similar in spirit with, and also shares the limitations of, the SEF. Recently, Schmidt *et al.* [159] extended the DHM to a two-dimensional descriptor combining shape thickness and depth complexity (number of intersections of a ray with the shape), thereby capturing both shape geometry and topology. Both above methods, however, require thousands of 2D views of the shape to be rendered and analyzed, which becomes expensive even when using GPU acceleration.

Jain and Zhang [80] propose to represent each shape by the eigenvectors of a certain affinity matrix constructed such that normalization against rigid-body transformations, uniform scaling and bending is achieved. They also show that the retrieval performance of the light-field [29] and spherical-harmonic [89] descriptors is improved by 5 . . . 10% when applied on the proposed spectral embedding. Other state-of-the-art methods include covariance [188] and diffusion [25, 59, 185] -based descriptors.

Apart from the above descriptors that use only shape geometry, other descriptors use additional data such as texture and lighting. For instance, the DB-VLAT descriptor proposed in [194] combines dense SIFT descriptors taken from several 2D shape views. While showing very good precision and recall, this descriptor requires careful prior shape alignment, and appears to be very computationally intensive. More importantly, texture and/or lighting data is not always available, such as in the case of raw 3D voxel shapes.

Finally, machine learning techniques such as the bag-of-features and deep neural networks (DNNs) have also been used to learn shape descriptors, thus improving their retrieval performance, see, *e.g.*, [59, 104] and references therein. However, training DNNs can be challenging, as this requires numerous examples to learn from.

Medial descriptors: Skeletons, also known as medial axes, were introduced by Blum for the 2D case [20]. For the 2D case, medial axes jointly capture a shape's geometry and topology by the so-called medial axis transform, which is an exact dual representation of a shape as compared to a standard boundary representation

(see also Sec. 5.3.1). Siddiqi *et al.* [174] define the shock graph extracted from the shock grammar [171] and match such graphs to find similar 2D shapes. However, while this method can effectively capture topology, it is less good in capturing geometry, and was not extended to 3D shapes. Sebastian *et al.* [161] also use shock graphs to recognize shapes, based on a relatively expensive edit-type distance metric (minimum amount of changes needed to modify a shock graph into another one, *e.g.*, Levenshtein distance [101]). However, this method can only handle 2D shapes. Similarly, Xie *et al.* compare shapes by segmenting them using their skeletons' junctions and performing a part-by-part matching [205]. As [13], this method can only handle 2D shapes.

In 3D, the by far most used medial descriptors are *curve* skeletons, which essentially capture a shape's local circular symmetry, and are relatively easy to compute [39, 179]. Curve skeletons have been introduced in Sec. 2.1.2.2. Given their aforementioned properties, several methods have used curve skeletons for CBSR. Sundar *et al.* [186] use 3D curve skeletons to match shapes by extracting individual skeletal branches to construct a graph, and next using graph-matching algorithms to compare such shape graphs. However, this method is quite complex and computationally expensive. The method is next refined in [37] to use the Earth Mover's Distance (EMD) technique to compare 3D curve skeletons to match shapes [151]. For 2D shapes, Xiang *et al.* [13] propose an alternative to [186] by first pruning 2D medial axes to remove irrelevant branches (expensive for matching) using the DCE method [14], and next using paths between skeletal endpoints to construct a graph. While faster than [186], the method cannot effectively handle shapes having few salient skeletal endpoints, *i.e.*, shapes which are not well described by curve skeletons, and, as mentioned, can only treat 2D shapes.

Apart from curve skeletons, 3D shapes admit also *surface* skeletons, which fully capture their geometry and topology regardless of the shape type and are effectively dual representations of shape boundary descriptions [179, 189]. Surface skeletons have been introduced in Sec. 2.1.2.2.

However, efficiently computing surface skeletons for complex 3D shapes has been considerably harder than computing curve skeletons, due to the higher complexity of such descriptors, but also to their well-known sensitivity to small perturbations of the shape surface. Recent algorithms have changed this, allowing the near-real-time computation of surface skeletons both from mesh-based models [85] and voxel-based models [84]. Key to their usability, such methods also regularize the produced surface skeletons by efficiently computing so-called multiscale importance metrics, which ensure that only large surface details will generate skeletal branches [45, 145].

Summarizing the above, 3D curve skeletons have proved to be useful and efficient descriptors for CBSR, but are limited to mainly tubular shapes. Surface skeletons overcome the descriptive power of curve skeletons for any 3D shape type, but

have only recently shown to be efficiently and robustly computable for complex 3D shapes. As such, the main aim of this chapter is to show that recent advances in 3D surface skeleton computation make these descriptors more efficient, more effective, and simpler to implement tools for CBSR than the well-known curve skeletons used for the same task.

5.3 METHOD

We next explain our CBSR descriptor proposal. Section 5.3.1 outlines relevant skeleton definitions. Section 5.3.2 outlines the construction of the so-called 'cut space' which we use next to build our histogram-based shape descriptors (Sec. 5.3.3). Finally, Section 5.3.4 shows how such histogram descriptors can be efficiently and effectively compared. Figure 5.1 illustrates the proposed CBSR pipeline.

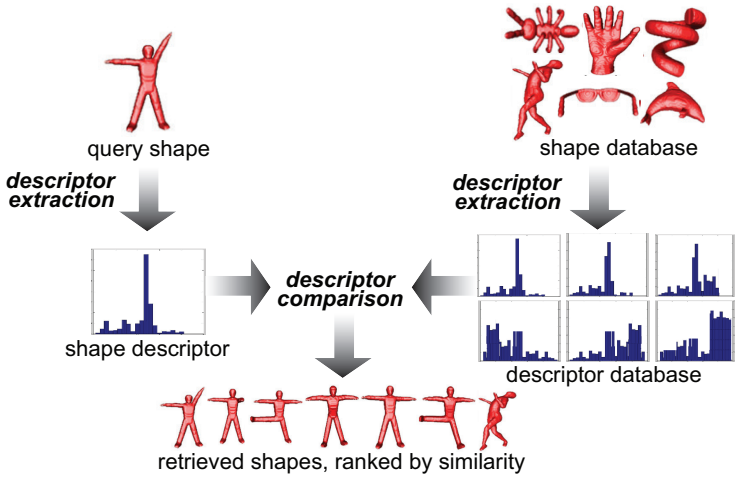


Figure 5.1: Proposed CBSR pipeline. Our focus is the descriptor extraction and comparison (marked in bold).

5.3.1 *3D Skeletonization*

Let the Euclidean distance transform $DT_{\partial\Omega} : \Omega \to \mathbb{R}_+$ of a binary voxel shape $\Omega \subset \mathbb{Z}^3$ with boundary $\partial\Omega$ be denoted by

$$DT_{\partial\Omega}(\mathbf{x} \in \Omega) = \min_{\mathbf{y}\in\partial\Omega} \|\mathbf{x} - \mathbf{y}\|. \tag{5.1}$$

The medial surface, or surface skeleton, of $\partial\Omega$ is next defined as

$$S_{\partial\Omega} = \{\mathbf{x} \in \Omega | \exists \{\mathbf{f}_1, \mathbf{f}_2\} \subset \partial\Omega, \mathbf{f}_1 \neq \mathbf{f}_2, \\ \|\mathbf{x} - \mathbf{f}_1\| = \|\mathbf{x} - \mathbf{f}_2\| = DT_{\partial\Omega}(\mathbf{x})\} \tag{5.2}$$

where $\mathbf{f}_1$ and $\mathbf{f}_2$ are the contact, or feature, points with $\partial\Omega$ of the maximally inscribed ball in Ω centered at $\mathbf{x}$ [71, 148]. These define the feature transform $FT_{\partial\Omega} : \Omega \to \mathscr{P}(\partial\Omega)$

$$FT_{\partial\Omega}(\mathbf{x} \in \Omega) = \underset{\mathbf{y} \in \partial\Omega}{\operatorname{argmin}} \|\mathbf{x} - \mathbf{y}\|. \tag{5.3}$$

The medial surface implied by Eqn. 5.2 consists of a complex set of 2D manifolds embedded in 3D. Their direct computation, following Eqn. 5.2, is sensitive to small-scale noise on $\partial\Omega$, especially when using voxel-based discretizations of Ω. To alleviate this, $S_{\partial\Omega}$ can be regularized by a computing a metric $\rho : S_{\partial\Omega} \to \mathbb{R}_+$ such as the medial geodesic function (MGF) which sets $\rho(\mathbf{x})$ to the length of the shortest path on $\partial\Omega$ between the two feature points of $\mathbf{x}$ [45]. As the MGF monotonically increases from the medial surface boundary to its center, upper thresholding it always yields connected and noise-free simplified medial surfaces [85, 145]. A similar regularization metric, which is faster to compute than the MGF, is proposed in [84] based on a mass advection process from $\partial\Omega$ onto $S_{\partial\Omega}$. In our work next, we will use the regularized surface skeletons produced by [84].

5.3.2 *Cut space construction*

The first step of our method is to construct a so-called *cut space* from the surface skeleton $S_{\partial\Omega}$, following the process proposed in [61] (see Fig. 5.2a-e, for a hand model): For each point $\mathbf{x} \in S_{\partial\Omega}$, we construct a closed curve $C(\mathbf{x}) \subset \partial\Omega$ as the union $\gamma_1 \cup \gamma_2 \cup \gamma_3$, where γ_1 is the shortest path on $\partial\Omega$ between the two feature points $\mathbf{f}_1$ and $\mathbf{f}_2$ of $\mathbf{x}$ (Fig. 5.2a); and γ_2 and γ_3 are the shortest-paths on $\partial\Omega$ between $\mathbf{o}$, the opposite point on $\partial\Omega$ of the midpoint $\mathbf{m}$ of γ_1 with respect to $\mathbf{x}$, and $\mathbf{f}_1$ and $\mathbf{f}_2$, respectively ((Fig. 5.2b-d). For full implementation details, we refer to [61] and also to Chapter 3 where the cut space is introduced in detail.

5.3.3 *Cut thickness histogram*

Key to our shape descriptor idea is that the length $\|C(\mathbf{x})\|$ of a cut $C(\mathbf{x})$ is a good local descriptor of the shape's properties around point $\mathbf{x}$. We argue this by following the analysis in [61]: The cut-space $CS = \bigcup_{\mathbf{x} \in S_{\partial\Omega}} C(\mathbf{x})$ captures, locally, the shape's symmetry and thickness in an effective and natural way. That is, CS contains all cuts which are locally smooth, tight-wrapping around $\partial\Omega$, and oriented orthogo-

nal to Ω's local symmetry axis. As these cuts have proven to be good for shape segmentation [61], we argue that they are also good for describing local shape thickness. Fig. 5.2e shows the cut space CS for our hand model, colored to emphasize the difference between short and long cuts. For all implementation details of the construction of CS, we refer further to [61].

As mentioned in [61], CS contains a small number of cuts which are not orthogonal to the local symmetry axis, or curve skeleton, of Ω. Upon closer examination, we noticed that these correspond to very short geodesics γ_1, *i.e.*, skeleton points $\mathbf{x}$ whose two feature points $\mathbf{f}_1$ and $\mathbf{f}_2$ are very close on $\partial\Omega$. The presence of such cuts unnecessarily perturbs the computation of $H(\Omega)$. This problem is well known in 3D skeletonization: Skeleton points having small feature-vector angles tend to be very unstable [65]. Hence, we remove from CS all cuts $C(\mathbf{x})$ for which $\|\gamma_1\|/\|C(\mathbf{x})\| < 0.4$, an empirically determined value which gave good results in all our tests. The resulting regularized cut-space CS^r is next used to robustly compute our shape descriptor. Note that using the angle of feature vectors $\mathbf{x}-\mathbf{f}_1$ and $\mathbf{x}-\mathbf{f}_2$ can also be used instead. We have found the geodesic length criterion to be better in removing skeleton points for which unreliable cuts can be computed. The difference makes, indees, sense: The angle criterion quantifies how reliable a *skeleton point is*; the geodesic length criterion quantifies how reliable the cut-construction algorithm, and thus its produced *cuts*, are. Since we are interested ultimately in stable cuts, we use the latter criterion.

We stress that the local shape-thickness estimation $\|C(\mathbf{x})\|$ adapts itself to both the shape *orientation* (C is orthogonal to Ω's local symmetry axis, or curve skeleton, of Ω) and the shape's local *geometry* (C is piecewise geodesic by construction, thus smooth). This is in high contrast to [108, 159], who estimate the local shape thickness in *arbitrary* directions, *i.e.*, regardless of the shape's curve-skeleton orientation. The above papers recognize that this can yield significant thickness-estimation noise. Indeed, formally, just one single direction (orthogonal to Ω's curve-skeleton) yields the 'true' shape thickness, as this thickness value is unique.

Given our cut space CS^r, we next build a histogram $H(\Omega)$ of all cut lengths $\{\|C(\mathbf{x})\| | \forall \mathbf{C} \in CS^r\}$. Since $S_{\partial\Omega}$ is rotation and translation invariant with respect to Ω, so is CS^r, and thus so is $H(\Omega)$. To achieve scaling invariance, we normalize the cut lengths by their median value over CS^r. This essentially makes cut lengths relative to the shape size, and thus makes next $H(\Omega)$ invariant to the scale (size) of Ω. To make H independent on the number of cuts (which depends in turn on the voxelization resolution of Ω and $S_{\partial\Omega}$), we normalize the bin-values $H(\Omega)^i$ by the total cut count $\|CS^r\|$. Using $B = 20$ bins $H(\Omega)^i$, $1 \leq i \leq B$, for constructing $H(\Omega)$ proved to be a very good balance between accuracy and level-of-detail. Note that the histogram normalization is not necessary for the shape segmentation applications of the cut space (Chapters 3, 4). Indeed, for segmentation we are only

interested to cluster similar cuts for a single given shape. For matching, we are interested in comparing two or more shapes, so normalization of histograms is needed.

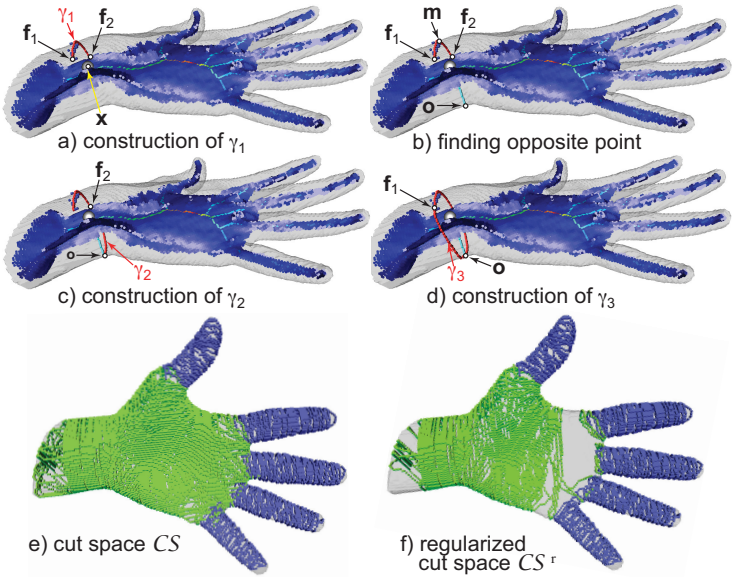


Figure 5.2: Construction of the regularized cut space CS^r.

5.3.4 *Shape matching distances*

Given two shapes Ω_1 and Ω_2 and their respective histograms $H(\Omega_1)$ and $H(\Omega_2)$, denoted next H_1 and H_2 for brevity, we compare these by using a distance function $d(H_1,H_2) \in [0,1]$. To this end, we studied four such distances, as follows.

Hellinger distance: We first consider the simple Hellinger distance $d_H(H_1,H_2) = \sum_{i=1}^{B}\left(\sqrt{H_1^i}-\sqrt{H_2^i}\right)^2$. Although d_H is not an optimal metric to compare shapes, as we will also show next, it has been used in previous CBSR applications [159, 193], so we include it here for completeness. Upon close examination, we noticed that d_H causes false negatives, *i.e.*, high distance values between two visually very similar shapes. Figure 5.3 shows such an example, where $d_H = 0.575$ for two quite similar shapes. We found similar problems when using other simple histogram-distances, such as the L_1 norm, Chi-square, and correlation metrics.

This problem can be understood if we consider the *coarse* nature of our cut-space: CS^r contains at most as many cuts as the size $\|S_{\partial\Omega}\|$ of the surface skeleton,

which is bounded by the voxelization-resolution of our input shapes Ω, and next reduced by regularization of CS to CS^r. In practice, for a shape voxelized at 300^3 resolution, CS^r contains a few thousand cuts. In contrast, the thickness estimation in [159] uses 500K thickness samples, which is *over three orders of mangitude* more. Similar high sampling resolutions are used in [108, 157]. While super-sampling alleviates the false-negative problem, it is also very expensive. Hence, one key challenge for our method is how to increase the *robustness* of comparing histograms without increasing the voxelization resolution, and thus *cost*, of our pipeline. Apart from reducing computational costs, this will also allow using our method on coarsely-sampled shapes.

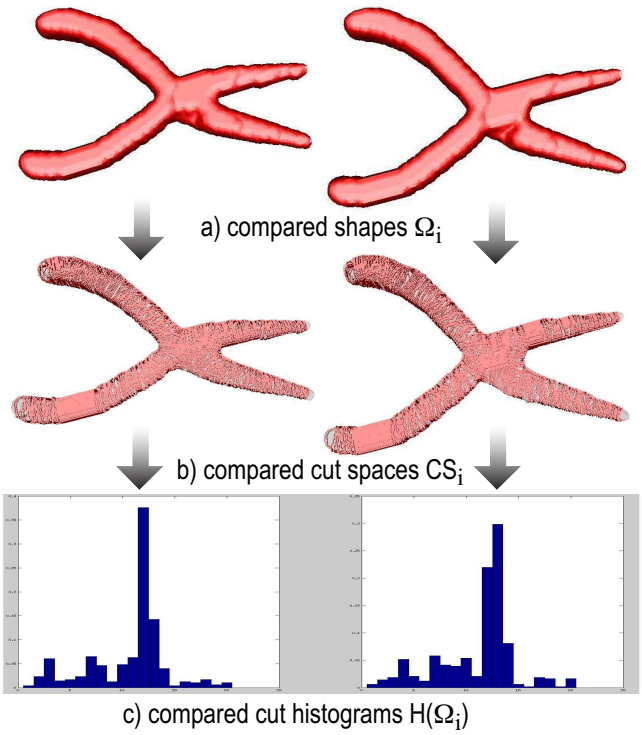


Figure 5.3: Two similar shapes and their cut-spaces and histograms.

EMD: Another popular way to compare histograms is the Earth Movers Distance (EMD) [151]. When comparing two histograms, this metric treats one histogram H_1 as mass and the second one H_2 as a container. The least amount of mass required to be moved from H_1 to H_2 to make the two histograms identical gives the distance $d_{EMD}(H_1, H_2)$. The EMD distance is known to be, in general, a robust way to compare two histograms [152]. In our case, d_{EMD} creates fewer false neg-

atives than d_H, but still too many of them. A similar effect has been reported for CBSR by [159].

Median value separation: Studying the above-mentioned false negatives, we noted that they are mainly caused by small differences in the location (cut-length) of the largest peaks (longest bars H^i) in the two compared histograms. These correspond to slightly different thickness-distributions of the two compared shapes Ω_1 and Ω_2. To alleviate this, we work as follows: For a shape Ω, we compute its median cut-length value m, and use it to divide its histogram H into two sub-histograms H^l and H^r which contain cuts shorter or equal than, respectively larger than, m. For H^l, we list bars in decreasing cut-thickness order; for H^r, bars are listed in increasing thickness order. Both histograms H^l and H^r use, together, the same fixed number of bins B as the original H. When comparing two shapes Ω_1 and Ω_2, we pad H_1^l and H_1^r with zero-size bins to the right so as to match the corresponding sizes of their counterparts H_2^l and H_2^r. Note that right-padding is made possible by the reordering of bars in H^l mentioned above. Finally, we use the EMD metric to compare the corresponding histogram-pairs, yielding the distance metric $d_{MV}(H_1,H_2) = \frac{1}{2}[d_{EMD}(H_1^l,H_2^l) + d_{EMD}(H_2^r,H_2^r)]$. Figure 5.7 middle shows this for the shapes, for which we get $d_{MV} = 0.281$, which reflects much better their visual similarity than d_{EMD} and d_H studied before.

Multilevel distance: As outlined above, small changes in the locations of tall bars in cut-space histograms can create large undesired distance differences. The median-value distance d_{MV} partially fixes this by effectively 'aligning' two histograms along their median values. We next refine this fix to also consider *gaps* in a histogram. These are near-empty bins separating blocks of non-empty bars (see Fig. 5.5), and correspond to cut sizes appearing rarely in a shape. Since a gap is, by definition, surrounded by two blocks of non-empty bars, it corresponds to shape zones showing rapid local-thickness transitions, such as joints between parts. Gaps were used in [61] to segment shapes along these joints. In contrast, we use gaps to compare shapes more robustly, as follows. First, we define gaps g^i in a histogram H as bars H^i containing less than $\tau\|CS\|$ of the total number of cuts, where $\tau = 0.01$ has given good results for all our experiments. Gaps effectively partition H into several bin-blocks B^i, each being defined by two consecutive gaps (g_l^i, g_r^i) to its left, respectively right. Given two such histograms H_1 and H_2, we compare their block-sets $B_1 = \{B_1^i\}$ and $B_2 = \{B_2^i\}$ using d_{MV} on block-pairs in B_1 and B_2 whose center distance $|g_l^i + g_r^i|/2$ is less than $\beta = 3B/20$ bars, which corresponds to three bars for our preset $B = 20$ number of bars in a histogram. Note that the complexity of this comparison is $O(\beta \cdot \max(|B_1|,|B_2|))$, *i.e.*, basically linear in the number of blocks, since a block in, say, B_1 only gets compared worst-case with 2β other blocks in B_2.

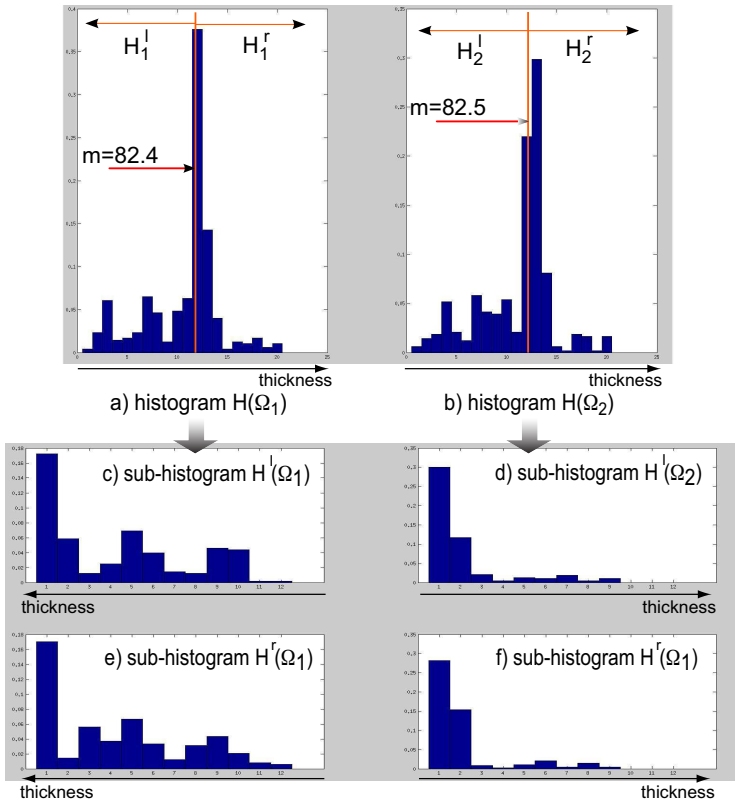


Figure 5.4: Median value separation histogram-distance d_{MV}.

Denoting the set of such matching blocks by $BP \subset B_1 \times B_2$, we get our multilevel distance metric $d_{ML}(H_1,H_2) = \sum_{(B_i,B_j)\in BP} d_{MV}(B_i,B_j)$. Figure 5.5 shows this idea for our two plier shapes (Fig. 5.3), whose histograms show one gap each, and thus have been cut into two blocks. Intuitively, we see that block-detection 'aligns' two histograms independently from the longest-bar alignment provided by d_{MV}. For our plier shapes, we obtain $d_{ML} = 0.0321$, which reflects the (very high) visual similarity of the two shapes better than all metrics considered so far.

5.4 IMPLEMENTATION AND RESULTS

We implemented our shape descriptor using the 3D skeletonization algorithm in [84], which is one of the fastest, most robust, exact, and easy to use methods to extract regularized 3D-multiscale surface-skeletons in existence. Cut spaces *CS* were built following the method in [61], described in Chapter 3. To test our proposed descriptor, we chose to use the McGill 3D Shape Benchmark [115],

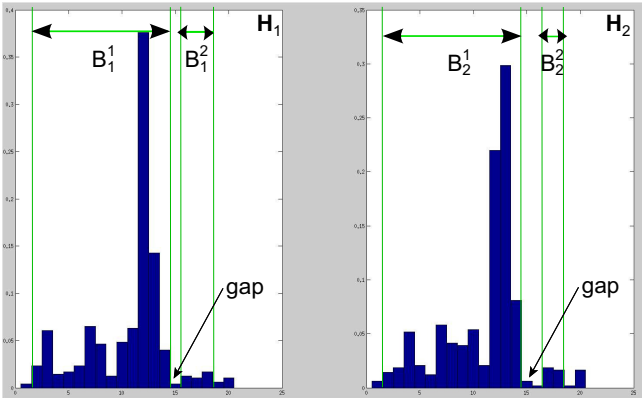


Figure 5.5: Multilevel histogram-distance d_{ML}.

which is well-known in the shape retrieval and computer vision community. The database contains 309 shapes, grouped into 13 semantic classes. Each class contains objects having various scales, poses, and articulations (see Fig. 5.6 for a sample subset). Since both [84] and [61] are voxel-based methods, we voxelized all our test shapes to resolutions ranging between 200^3 and 500^3 voxels, using *binvox* [123]. Our entire pipeline, written in C++, was executed on 3.5 GHz Linux PC.

Table 4 shows the performance of our cut-space-based descriptor using the three distance metrics in Sec. 5.3.4. To test retrieval accuracy for the used database, we used two methods:

Fine grained: Given a shape $\Omega \in D$ from a database D, we compute the query's *precision* for a query-size $Q = 10$, *i.e.*, find the most similar 10 other shapes $\Omega_i \in D$ according to $d(H(\Omega), H(\Omega_i))$, and count how many of these are in the same class as D.

Coarse grained: We apply the same procedure as above, but merge classes having structurally similar objects into three so-called superclasses: limbed objects (humans, hands, dinosaurs, four-limbs, teddies); highly articulated objects (spiders, ants, octopuses, spectacles); and low-articulated objects (fishes, dolphins).

We applied both above tests to all 309 shapes in the database and then averaged the results per shape-class. Table4 shows the results. Figure 5.7 gives more detailed insights, showing the six most-similar shapes retrieved for a query shape for the distance metrics d_{ML}, d_{EMV}, and d_{MV}. The Hellinger distance d_H was left out as it yielded significantly poorer results. From these data, we see that d_{ML} gives

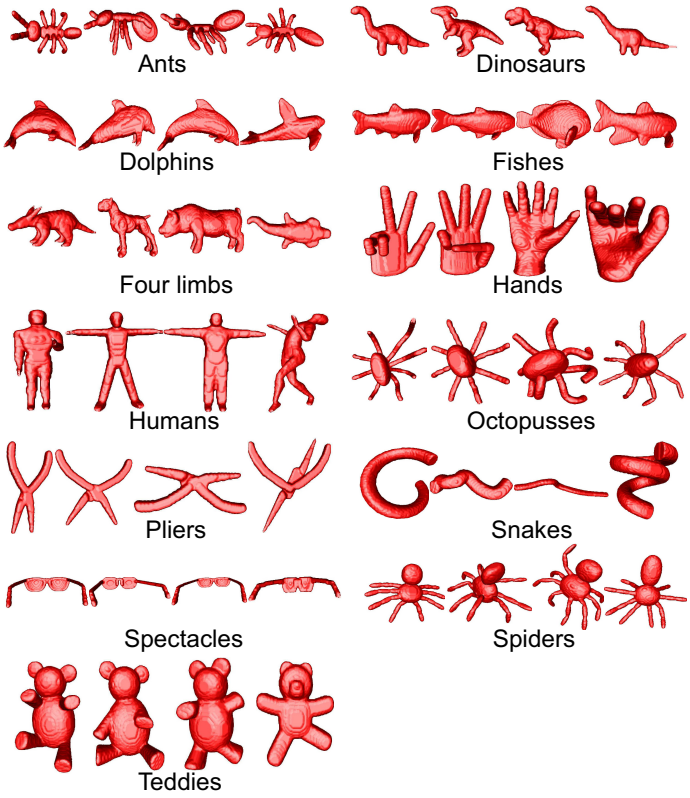


Figure 5.6: Example shapes from the used benchmark [115].

the visually best results, followed by d_{MV} and next by d_{EMD}.

Noise robustness: To test the robustness of our descriptor to noise, we created a database where noise was added to the shapes in [115], in the form of Gaussian bumps of two heights (small – 3% of the shape diameter $\Phi(\Omega)$, and large – 6% of $\Phi(\Omega)$); and two standard deviations (small – 4% of $\Phi(\Omega)$, and large – 8% of $\Phi(\Omega)$). Noise bumps were Poisson distributed over $\partial\Omega$. We next queried for a clean shape, and observed that we got both noised versions thereof and clean versions of shapes in the same class as the top-hits (Fig. 5.8 top row). We also queried the four types of noised shapes corresponding to the clean shape Ω, and noticed that we got, in three cases, three of the noised variants of Ω, and in the fourth case all four variants (Fig. 5.8, bottom 4 rows). In all cases, the clean shape appeared in the top-six most similar retrieved shapes. This shows that our descriptor is both robust to noise (it retrieves the correct same-class shapes and does not introduce

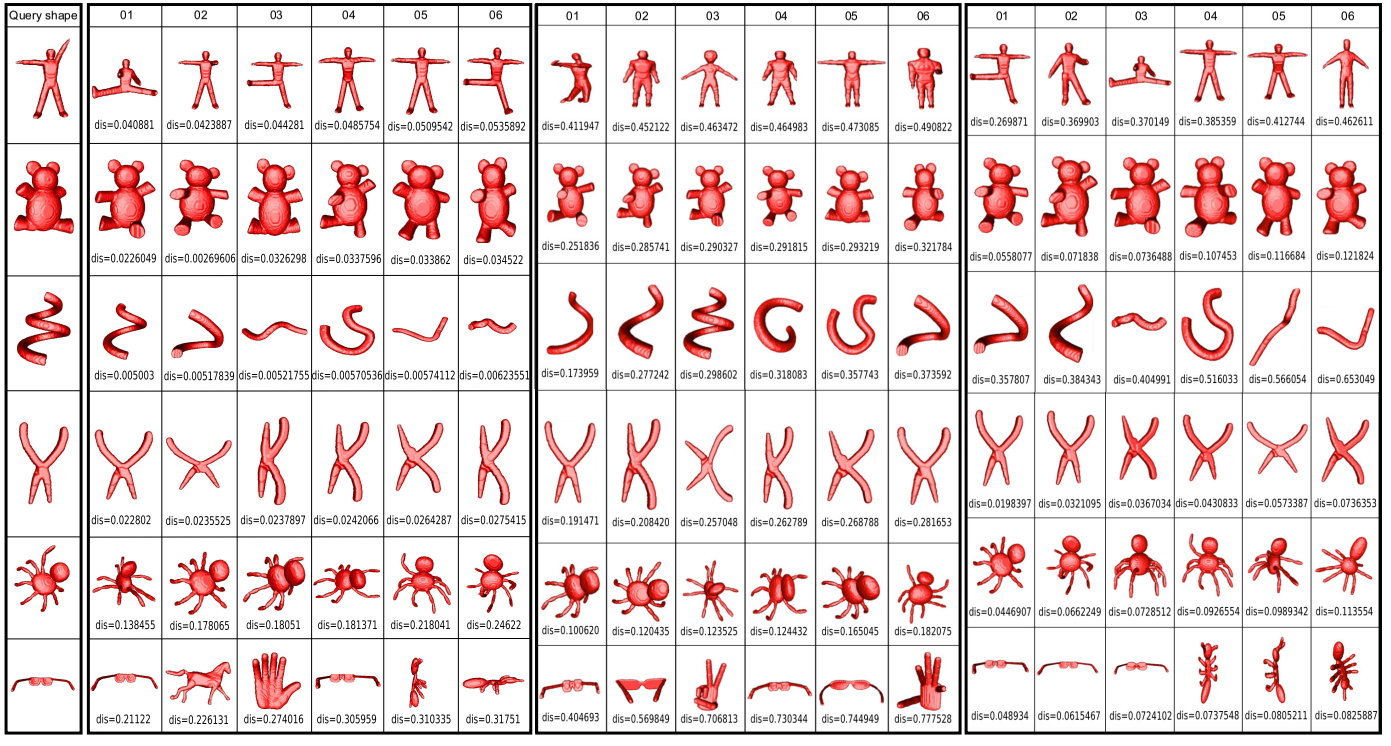


Figure 5.7: Examples of shape retrieval results for the distance metrics d_{EMD} (left), d_{MV} (middle), and d_{ML} (right). For each distance metric, shapes are listed from left to right in increasing distance order. Actual distance values are indicated under the retrieved shapes.

Classes	Class sizes	d_{EMD} (fine)	d_{EMD} (coarse)	d_{MV} (fine)	d_{MV} (coarse)	d_{ML} (fine)	d_{ML} (coarse)
humans	29	0.807	0.945	0.755	0.831	0.776	0.941
hands	20	0.345	0.7	0.36	0.6	0.435	0.81
four limbs	31	0.494	0.813	0.439	0.771	0.442	0.858
dinosaurs	18	0.439	0.844	0.389	0.828	0.339	0.878
spiders	31	0.787	0.900	0.777	0.974	0.797	0.939
ants	30	0.793	0.837	0.62	0.877	0.737	0.803
octopus	25	0.608	0.852	0.652	0.828	0.66	0.956
fishes	23	0.517	0.709	0.522	0.704	0.674	0.791
dolphins	12	0.4	0.683	0.467	0.642	0.442	0.642
pliers	20	0.92	0.92	1	1	0.87	0.87
teddy	20	0.72	0.72	0.775	0.775	0.885	0.885
spectacles	25	0.292	0.292	0.34	0.34	0.432	0.432
snakes	25	1	1	0.684	0.684	0.624	0.624

Table 4: Shape retrieval results with three for the distance metrics d_{MV}, d_{EMD}, and d_{ML}.

false-positives because of noise); and also sensitive to shape (it retrieves most of the noised versions of the input shape).

5.5 DISCUSSION

We nest discuss several relevant aspects of our proposed descriptor.

Robustness: The proposed descriptor is rotation, translation, scaling, pose, noise, and voxelization-resolution invariant. This is guaranteed by the corresponding properties of the underlying skeleton cut-space and histogram normalization (Sec. 5.3.3). Note that, while these properties hold by construction in the continuous space $\mathbb{R}^3$, they are preserved in the voxel space $\mathbb{Z}^3$ only as well as the underlying skeletonization method can manage to do so. The method we use here achieves very good results in this respect, as discussed in detail in [84]. Note also that, to ensure similar properties, other CBSR methods require more complex techniques, such as spherical harmonics [89, 90, 108, 157] or delicate, and time-consuming, manual model alignment [194].

Comparison: Our evaluation, while limited, is in line with several related papers. Osada [125] uses a database of 133 models grouped in 25 classes. For evaluating the performance of their method, they measure the query precision, and the percentage of all queries where the top match was from the queried shape's class, and obtain values of 30%, respectively 70%. In comparison, our corresponding values are of 70% (for the d_{ML} metric), respectively 100% (all metrics) (see Tab. 4). Fur-

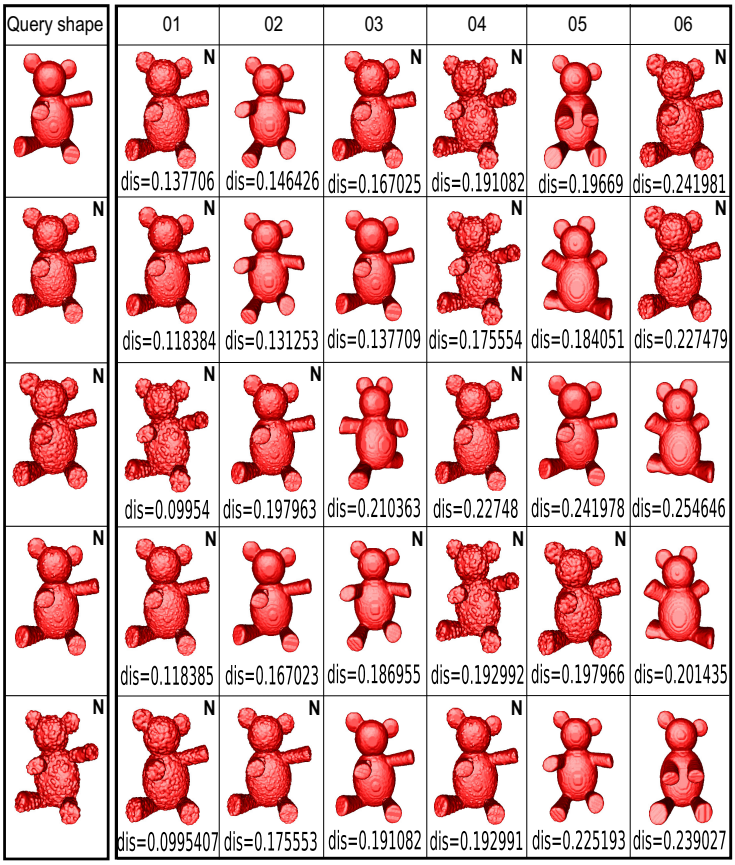


Figure 5.8: Retrieval robustness in presence of noise. Shapes marked '**N**' contain artificially-added noise.

thermore, the spectral descriptors of Jain and Zhang [80] yield 70...75% precision performance and outperform both the light-field [29] and spherical-harmonic [89], whose precision is 68% and 60%, respectively. On the same dataset, other query-precision results include 49% – shape-distributions [125], 45% – covariance [188] and 50% – diffusion [59], as reported in [59, 104]. When comparing the above figures with ours, we should stress that the obtained insights are, of course, limited by the fact that we use different databases. This is due to the lack of access to the above-mentioned implementations (so we couldn't use them on our database) and the difficulty and cost to convert the databases used in the above papers, when these were publicly accessible, to our voxel representation (so we couldn't easily use our implementation on these databases).

The skeleton-based descriptor used by [161] was tested on two databases (99 shapes, 9 classes; and 144 shapes, 8 classes). Its accuracy, measured by the so-called first tier (precision with the query size equal to the class size of the queried shape), yielding a value of 97% on average – in our case, the corresponding average value is of 80%. However, we note that [161] can only treat 2D shapes, which are known to be much easier to match than 3D shapes. Similarly, the skeleton-based path similarity matching method in [13] shows quite good retrieval results, with an average precision of 95% for a query size of 10 shapes, but is also limited to 2D shapes. Sundar [186] tested their 3D curve-skeleton-based descriptor on a database of 100 shapes. However, while a few retrieval results are shown, no statistics are provided. Cornea *et al.* compared their 3D curve-skeleton-based descriptor on a 1000 shape database [37]. They report a first-tier value of 17% and nearest-neighbor value of 71%. While both [37] and our method use the EMD to compare shape descriptors, [37] uses the entire curve-skeleton as descriptor, while we use the cut-space thickness-histogram. The latter is significantly more invariant to irrelevant shape changes than the former, which explains, at least partially, our better results.

Efficiency and ease of use: The proposed descriptor can be computed automatically, with no user parameters needed to be set or tuned. For a shape Ω, its computation complexity is $O(\|S_{\partial\Omega}\|D)$, where $D = \max_{\mathbf{x}\in\Omega} DT_{\partial\Omega}(\mathbf{x})$ indicates the maximal shape thickness and $\|S_{\partial\Omega}\|$ the number of voxels of the shape's surface skeleton. In practice, this allows computing our descriptor in subsecond times on shapes up to a few hundred voxels cubed on the platform indicated in Sec. 5.4. Significant speed-ups could be easily achieved, if desired, by trivially parallelizing the cut space computation on the CPU or GPU, if desired.

Limitations: As outlined earlier, our descriptor essentially captures the local shape thickness only. As such, its discriminative power is lower, in general, than more advanced descriptors, in general. However, for a fair comparison, one should

relate our proposal chiefly to other thickness-based descriptors in CBSR (or, more generally, descriptors with the same small size, in our case, $B = 20$ values). Indeed, comparing descriptors of highly different sizes can be perceived as unfair, since different descriptors have different complexities like time, memory. Separately, our proposal can be directly used to replace other less accurate thickness descriptors known in the literature, *e.g.* [108, 159]. A second limitation relates to the benchmark we used [115], which contains mainly locally-tubular shapes. For a better insight, more varied shapes should be added to such a benchmark, a task which we consider for future work. Finally, besides precision values, CBSR benchmarks also typically use nearest-neighbor and first-tier values [37, 161]. These metrics can be easily added in future evaluations of our proposed descriptor.

5.6 CONCLUSIONS

In this chapter, we have presented a new method for characterizing 3D shapes via a thickness-histogram descriptor. Our descriptor captures the *local* thickness of a 3D shape in an accurate and computationally-savvy way, by using the 3D surface-skeleton of a shape to construct a regularized space of smooth cuts which are tight around the shape, and also optimally oriented across the local shape curve-skeleton to capture the shape's local thickness. We next studied four histogram-distance metrics to compare such descriptors, and showed that two such novel metrics achieve shape matches which reflect the visual similarity of 3D shapes in better ways than classical histogram-comparison metrics used in the literature. Our method is simple to implement, automatic, and robust to shape variations such as pose, scale, rotation, noise, and voxelization resolution.

It is interesting to compare now the features of the cut space that support both shape segmentation (Chapters 3, 4) and shape matching (this Chapter). By doing this, we can highlight even better the added-value of this descriptor. We distil the following key aspects:

Invariance: The cut space is by construction invariant to scale, translation, rotation (up to inevitable small-scale variations caused by using a fixed voxel grid), and largely invariant to noise and pose (articulation). All these properties are actually inherited from the corresponding properties of surface skeletons. Note that such properties are *not* evident for curve skeletons. Indeed, as explained in Sec. 2.1.2.2, surface skeletons do have a unique formal description from which their properties can be inferred and quantified subject to sampling (Eqn. 2.2). Curve skeletons do not admit such a unique formal definition. As such, it is much harder to reason about their invariance properties.

Histograms: We use histograms of cut space metrics, such as length, to support both shape segmentation and shape matching. Yet, one important difference exists: For shape segmentation, we use both position and cut-length information; for shape matching, we are only interested in the cut lengths (the actual positions of cuts, which we needed to explicitly find segment borders in segmentation, are irrelevant for shape matching).

Segmentation-matching link: The fact that cut spaces can support both segmentation and matching is not fortuitous. After all, what determines where a shape's segments end, are variations in the cut space; and the way a shape can be split into segments tells something about how similar it is to other shapes. At a high level, this analogy is present in many skeleton-based segmentation methods (see Sec. 5.2). The difference between those methods and ours is that we do not *explicitly* segment a shape to match it.

Future work directions consider the integration of additional shape processing results in our cut histograms, such as the cut shape, eccentricity, and orientation, as well as more detailed testing our descriptor on larger shape benchmarks, and its (easy) integration with other existing shape descriptors in the literature.

6

UNIFIED PART-PATCH SEGMENTATION OF MESH SHAPES USING THE SKELETON CUT SPACE

In the previous three chapters, we have explored the use of the skeleton cut space to support segmentation and matching of voxel-based shapes. However, as noted in several instances, voxel-based methods have some significant limitations. First and foremost, they need large amounts of memory to accurately represent fine-detail shapes, since they use uniform sampling. Secondly, the fixed-grid positions of voxel centers limits the accuracy of various operations that one can perform on such representations (as opposed to using an ideal continuous shape representation). At the same instances, we mentioned the interest of exploring the extrapolation of the cut space concept and methods related to it to other types of shape representation.

This chapter covers the last above-mentioned point. We propose here a definition of the skeleton cut space for mesh-based shape representations (b-reps). Next, we explain how such a cut space can be computed, using suitable skeletonization methods for mesh shapes. Thirdly, we demonstrate that a mesh-based cut space can be used equally well as a voxel-based cut space to support shape segmentation. We support this by comparing segmentations of mesh-based and voxel-based shapes using the cut space, and also comparing these with third-party segmentation methods. Finally, we propose a way to extend the part-based segmentation idea introduced in Chapters 3 and 4 to include patch-based segmentation.[1]

6.1 INTRODUCTION

Shape segmentation is an important problem in many application domains such as computer-aided design, computer graphics, scientific visualization, and medical imaging. Informally put, shape segmentation aims at partitioning a given shape into several components, or segments, that capture application-specific part-whole relations as well as possible. Segmentation enables several shape processing and shape analysis tasks such as editing and content creation, identifying important features that occur in a large dataset, and shape matching, retrieval, and registration [11, 168]. Segmentation methods can be roughly classified into *part-based*

1 The text of this chapter is based on the paper: Unified part-Patch Segmentation of Mesh Shapes using Surface Skeletons (J. Koehoorn, C. Feng, J. Kustra, A. Jalba, A. Telea), *Skeletonization: Theory, Methods, and Applications*, Chapter 2, eds. P. K. Saha, G. Borgefors, and G. S. di Baja, Elsevier, 2016

methods, which aim to split articulated shapes into the components that would be perceived as naturally distinct by humans [141]; and *patch-based* methods, which aim to find quasi-flat components separated by edges on synthetic, faceted, models [145].

Shapes are typically encoded following a boundary representation or a volume representation. Boundary (explicit) representations capture the surface that partitions the shape interior from the surrounding exterior space, using various sampling and reconstruction schemes, *e.g.* polygonal meshes or point clouds [21]. Volume (implicit) representations, such as voxel models, store a densely-sampled labeling of the space in which shapes are embedded, marking points as interior *vs* exterior [19]. Both representations have their advantages and limitations: Voxel volumes are easy to create and manipulate, but can be very expensive when high resolution is needed; surface meshes can efficiently model high-resolution shapes, but mainly support operations focusing on the shape's surface, rather than its volumetric structure.

Skeletal representations are a third way to represent shapes. Informally put, skeletons jointly capture the geometry, symmetry, and topology of a shape in compact ways. 3D shapes admit two types of skeletons: *Surface* skeletons are the locus of centers of maximally-inscribed spheres in the shape, and as such capture geometry, symmetry, and topology. They generalize to 3D the well-known concept of a 2D symmetry axis [20]. *Curve* skeletons are one-dimensional structures locally centered in the shape which mainly capture a shape's part-whole structure. Skeletons combine the compactness of boundary representations with the ability to model and reason about volumetric properties. Additionally, they provide explicit and efficient access to a shape's symmetry structure and part-whole properties. As such, skeletal representations are a valuable tool to design shape segmentation methods [189].

Many skeleton-based segmentation methods have been proposed [189]. However, most of these methods use only the topological information encoded by *curve* skeletons. As such, they target mostly part-based segmentation of organic, articulated, shapes. Producing patch-based segmentations of faceted, synthetic, shapes, or more generally mixed part-patch segmentations of shapes that fall between the two categories (articulated, faceted) is rarely handled [28, 98, 145].

Recently, surface skeletons have been used to produce part-based segmentations of 3D shapes, as described by us in Chapters 3 and 4 (see also [60, 63]). Key to this idea is the construction of a so-called *cut space* containing a large number of well-designed cuts that partition the shape in ways similar to how a human would cut it. By analyzing this space, a small number of cuts is retained to yield the final shape segmentation. While the method was shown to deliver good results, it has several limitations. First, it only produces part-based segmentations, although it uses the surface skeleton which fully describes any type of shape (articulated or faceted).

Secondly, it only handles voxel representations, and as such is very expensive, or even prohibitive to use, for high-resolution models.

In this chapter, we extend the skeleton cut space segmentation proposal in [63] with the following main contributions:

- *Mixed segmentations:* We show how to compute part-based, patch-based, and mixed part-patch segmentations that cover a wide range of 3D shapes, using only surface skeletons; as such, we show that surface skeletons are *effective* tools for handling complex 3D shape segmentation problems;
- *Mesh models:* We propose a fully mesh-based implementation of our method that can efficiently handle very large, high-resolution, mesh models with low computational and memory costs; as such, we show that surface skeletons are *efficient* tools for handling complex 3D shape segmentation problems.

The goals of this chapter are twofold: On the practical side, we show how we can improve on existing part and patch based segmentation methods. On the theoretical side, we show that 3D surface skeletons, so far used only rarely in practice, can effectively and efficiently be used to support such applications.

The remainder of this chapter is structured as follows. Section 6.2 reviews related work in part-based and patch-based shape segmentation, with a focus on skeleton-based methods. It also introduces the cut space idea in [63]. Section 6.3 details our method, explaining the changes and enhancements proposed to the original cut space segmentation. Section 6.4 presents several results of our method and compares these with related skeleton-based segmentation methods. Section 6.5 discusses our proposal. Section 6.6 concludes the chapter.

6.2 RELATED WORK

Given that our focus is skeleton-based segmentation methods, we proceed by introducing necessary background on skeletonization (Sec. 6.2.1). Next, we overview several part- and patch-based skeleton-based segmentation methods, outlining their advantages and limitations (Sec. 6.2.2).

6.2.1 *Skeletonization*

To define skeletons, we first introduce the Euclidean *distance transform* $DT_{\partial\Omega} : \Omega \rightarrow \mathbb{R}^+$ that associates to every point in a shape $\Omega \subset \mathbb{R}^3$ the distance to the closest boundary point

$$DT_{\partial\Omega}(\mathbf{x} \in \Omega) = \min_{\mathbf{y} \in \partial\Omega} \|\mathbf{x} - \mathbf{y}\|, \tag{6.1}$$

where $\|\cdot\|$ denotes Euclidean distance. The so-called medial surface, or surface skeleton, $S_{\partial\Omega} \subset \Omega$ is next defined as

$$S_{\partial\Omega} = \{\mathbf{x} \in \Omega | \exists \{\mathbf{f}_1, \mathbf{f}_2\} \subset \partial\Omega, \mathbf{f}_1 \neq \mathbf{f}_2, \|\mathbf{x}-\mathbf{f}_1\| = \|\mathbf{x}-\mathbf{f}_2\| = DT_{\partial\Omega}(\mathbf{x})\} \quad (6.2)$$

In the following, we use for simplicity S to refer to $S_{\partial\Omega}$ when the shape to be skeletonized is implicitly clear from the context. In Eqn. 6.2, $\mathbf{f}_1$ and $\mathbf{f}_2$ are two of the contact points with $\partial\Omega$ of the maximally inscribed sphere in Ω centered at $\mathbf{x}$ and of radius $DT(\mathbf{x})$. Such points are also known as *feature points* [148, 189], while the vectors $\mathbf{f}_i - \mathbf{x}$ are known as *spoke vectors* [172]. These definitions are captured by the so-called *feature transform*

$$FT_{\partial\Omega}(\mathbf{x} \in \Omega) = \arg\min_{\mathbf{y} \in \partial\Omega} \|\mathbf{x}-\mathbf{y}\| \quad (6.3)$$

which associates to any point inside the shape all its feature points on the shape boundary.

Surface skeletons of 3D shapes implied by the definition in Eqn. 6.2 consist of a set manifolds with boundaries, or skeletal sheets, that meet along so-called Y-intersection curves [42]. The pair $(S_{\partial\Omega}, DT_{\partial\Omega}|S_{\partial\Omega})$ is called the medial axis transform (MAT) of Ω [172]. The MAT can be used to fully reconstruct a shape, *e.g.* by computing the union of balls centered at points on $S_{\partial\Omega}$ and having as radii the values of $DT_{\partial\Omega}$ at the respective points. As such, the MAT provides a third type of shape representation, along boundary and volumetric ones.

Skeleton points can be classified by the order of tangency of their maximally inscribed spheres with $\partial\Omega$ [71]. This classification enables several applications such as robust edge detection on 3D surfaces [98, 144], finding Y-intersection curves for patch-based segmentation [102, 142], and surface reconstruction from point clouds [28]. Until recently, surface skeletons have been hard to compute for large and complex shapes [145, 148]. Recent methods significantly alleviated such issues for both voxel [7, 83] and mesh [82, 112] representations.

Besides surface skeletons, 3D shapes admit also *curve* skeletons. These are generically defined as one-dimensional (curve) structures which are locally centered within the shape [39, 189]. Their lower dimensionality implies a simpler structure (branches meeting at junctions), which makes them easier to use for part-based segmentation [12, 44, 141, 169]. Also, curve skeletons are easier to compute for large and complex 3D shapes as compared to surface skeletons, both for mesh representations [12, 191] and voxel representations [7, 83]. However, in contrast to the MAT, curve skeletons do not fully represent shapes, except when these have locally circular symmetry. In the following, we will focus exclusively on surface

skeletons, as the use or curve skeletons in shape segmentation is well covered in the literature (see also Sec. 6.2.2.1).

Skeletons are well-known to be unstable to small-scale noise on the input shape surface $\partial\Omega$ [172, 189]. As such, several so-called regularization methods have been proposed. *Local* methods use information such as the angle between feature vectors [65, 148], distance transform values, or divergence of the distance transform's gradient [175] to prune skeletal points caused by noise. *Global* measures approximate the amount of boundary that 'collapses' to, or corresponds to, a skeletal point. This approximation can be done by computing the length of the geodesic path between the feature points of a skeleton point (the so-called medial geodesic function or MGF [44, 82, 145]) or by explicitly simulating the advection of mass from $\partial\Omega$ onto S along the gradient of the distance transform [83]. Important skeleton points are next defined to correspond to larger parts of the input boundary. Thresholding global importance measures can deliver a so-called multi-scale skeleton, which reflects the input shape at a user-chosen level of detail [189].

6.2.2 *Shape Segmentation*

Let $\Omega \in \mathbb{R}^3$ be a three-dimensional shape with boundary $\partial\Omega$. Segmenting Ω typically amounts to computing a so-called partition $\mathscr{C}$ of Ω into components C_i, so that $\cup_i C_i = \Omega$ and $C_i \cap C_j = \varnothing, \forall i, j, i \neq j$. In other words, the set $\mathscr{C} = \{C_i\}$ consists of disjoint components that fully cover Ω. We next denote the borders of these segments by ∂C_i. Such borders are closed, non-intersecting, tight, and smooth curves embedded in $\mathbb{R}^3$ (see Sec. 3.3.2.1).

As mentioned in Sec. 6.1, segmentation methods can be classified into part-based and patch-based. The difference between the two classes amounts to different constraints put on the segments C_i. Regardless of the method type, however, segmentation can be seen as a combination of two key decisions: (1) finding *where* to cut a shape, or where to place the segment borders ∂C_i; and (2) finding *how* to cut, or which properties the segment borders should respect. We next discuss part- and patch-based segmentation methods using skeletons from this perspective. In the following, we denote by C_i^{part} segments resulting from a part-based segmentation $\mathscr{C}^{part}$, and by C_i^{patch} segments resulting from a patch-based segmentation $\mathscr{C}^{patch}$. The notations $\mathscr{C}$ and C_i are used for the unified part-patch segmentation, and respectively its segments, that we aim to compute. For additional details on part-based and patch-based segmentation, we refer to Sec. 2.2.2.

6.2.2.1 *Part-Based Segmentation*

As already stated, most part-based segmentations focus on natural articulated shapes, such as humanoids, animals, plants, or other objects showing a clear part-

whole hierarchical structure. Parts are typically defined as elongated regions of a shape that significantly 'stick out' of the shape's rump. Such parts are separated from the rump by a negative curvature region, a principle also known as the 'minima rule' in cognitive theory [23, 77]. Such parts are easily detectable using curve skeletons, as they correspond roughly one-to-one to the curve skeleton terminal branches [38]. Separately from the above heuristic that tells where to place cuts, part-based segmentation methods exploit other perceptual principles to constrain the cut shapes, such as the 'short cut rule', that states that a cut should be as short (and wiggle free) as possible [177]. Lee *et al.* segment mesh models using the minima rule and optimizing for short cuts using snake models [99]. Additionally, a 'part salience rule', which captures how much a part sticks out of the shape's rump, can be used to limit oversegmentation [100].

Several part-based segmentation methods use curve skeletons in their design. Li *et al.* sweep the curve skeleton with a plane to cut the shape, and keep those cuts which have important geometric and topological changes that indicate a part joining the shape's rump [103]. Au *et al.* compute curve skeletons by iteratively contracting 3D meshes [12]. This enables them to backproject each skeletal point to one or several surface points. Hence, segmenting the curve skeleton into separate branches, and next backprojecting each branch, enables part-based segmentation. Along the same line, Reniers *et al.* first define curve skeletons as those points in Ω having at least two equal shortest-paths between their feature points [145], and then use closed loops (cuts) formed by such shortest paths placed at the curve-skeleton junctions to segment a shape [146]. Using shortest paths (geodesics) to construct cuts guarantees that these obey the desirable short cut rule. The method was refined to discard cuts that are far from planar, which reduces unneeded oversegmentation [141]. Serino *et al.* refine this idea by detecting three kinds of skeletal parts (simple curves, complex sets, and single points) which, by back-projection to the input shape, partition it in parts that protrude from a rump (called simple regions and bumps) and the rump itself (called a kernel) [162]. In comparison to [141], this method can yield better segment boundaries and suffers less from oversegmentation. The method in [162] was subsequently refined to use a computationally efficient and simple to implement curve-skeleton extraction based on the selection of a small number of centers of maximally inscribed balls (so-called anchor points) that guide an iterative voxel removal, or thinning, process. Apart from shape segmentation, this method can be also used in other contexts where a 3D curve skeleton is required. The problem of oversegmentation due to the potentially large number of curve-skeleton junctions, which lead to skeleton branches that do not map to salient shape parts, also discussed in [141], is elegantly addressed in [163] by so-called 'zones of influence', which compare the local shape thickness at junctions with inter-junction distances.

Separately, part-based segmentation and skeletonization have been shown to be related operations, which allows computing the latter from the former [105]. Conversely, Shapira *et al.* note the same relationship, but use it to segment a shape by computing a shape-diameter function (SDF) based on the boundary-to-curve-skeleton distance and finding cuts in places where the SDF has sharp variations [169]. Finally, Tierny *et al.* analyze the Reeb graphs computed for scalar functions defined on the shape surface to yield a hierarchical shape segmentation [201]. While Reeb graphs are not identical to Euclidean (curve) skeletons, they share their ability to capture topology, and thus such methods can be seen as skeleton-based segmentation techniques.

Overall, curve skeletons have established themselves as good descriptors for producing part-based segmentations for articulated shapes whose parts have a (near) tubular local geometry. Multiscale or hierarchical segmentations can also be easily obtained by considering the curve skeleton's hierarchical structure, or by pruning less important curve skeleton branches [146]. Since curve skeletons are locally centered in a shape, they yield largely pose-invariant segmentations. Finally, curve skeletons are easy and fast to compute for both voxel and mesh representations (Sec. 6.2.1).

Recently, *surface* skeletons of voxel models have also been used for part-based segmentation [60]. The key idea is to construct a *cut space* $\mathscr{CS} = \{c_i\}$ that contains a large set of cuts that have suitable properties to act as segment boundaries. This solves the problem of *how* to cut. Next, a subset of these cuts is selected to become segment borders, thereby solving the problem of *where* to cut. Cut spaces are constructed by building closed loops formed by shortest paths on $\partial\Omega$ between the two feature points of each surface skeleton point. Next, cuts that represent suitable segment boundaries are found by analyzing a histogram of the cut lengths [60] or, alternatively, clustering cuts in terms of length [63]. Related methods of analyzing cut spaces for segmenting shapes have also been proposed, though not using skeletons to construct the cuts, *e.g.*, [74, 87].

6.2.2.2 *Patch-Based Segmentation*

In contrast to part-based methods, patch-based segmentation methods focus mainly on synthetic, faceted, objects such as those produced by CAD applications. A segment, or patch, is here defined as a region of the shape surface that is relatively flat and is separated from it surroundings by a high-curvature area, such as edges or creases. Like for part-based segmentation, patch boundaries should usually be wiggle free; however, they need not be tight.

Most patch-based methods work directly on the shape surface by unsupervised clustering, or grouping, mesh facets found to be similar [34, 107, 114, 127]. While such methods can produce very good results, they generally are parameter-

sensitive. Closer to our interest, surface skeletons have been used for patch-based segmentation. The first such method we are aware of backprojects the (voxel-based) surface skeleton boundaries to the input surface, using the inverse feature transform (Eqn. 6.3) to yield segment boundaries [142]. A different approach is to segment the surface skeleton manifolds using Giblin's skeletal point classification [71] and backproject these to the input surface [28]. All such methods produce good patch segmentations even for shapes having soft edges. Such methods require high-throughput skeletonization tools, as well as the delicate analysis of the resulting surface skeletons to detect boundaries and isolate manifolds.

A simpler patch-based segmentation method using surface skeletons was recently presented in [98]. Briefly put, this method implements the same strategy as [142], *i.e.*, finding segment borders by backprojecting the boundary of the surface skeleton. However, in contrast to [142], this method handles point-cloud skeletons; and it has a much simpler implementation than [28]. We will use this method further in our unified segmentation pipeline (Sec. 6.3.8.1), and refer to it as the skeleton boundary backprojection (SBB) method.

Other methods: Outside the class of skeleton-based methods, many other methods exist for shape segmentation. Given our focus on using skeletons for this task, these methods are of a lesser interest. For a general survey on 3D shape segmentation, we refer the interested reader to [11, 168].

6.2.3 *Summary of challenges*

Summarizing the above discussion on skeleton-based segmentation methods, we identify the following requirements and challenges. A good method of this kind should

1. be able to efficiently and directly handle high-resolution mesh models, since computing skeletons of high-resolution voxel models is prohibitive in terms of memory and time;

2. guarantee smoothness of the resulting segment borders;

3. be able to produce patch-based, part-based, and mixed-type segmentations, thereby handling all types of shapes.

No existing skeleton-based segmentation method complies with all the above, to our knowlege. In the remainder of this chapter, we propose such a method based on a complete re-casting of the cut space idea in [63] (Chapters 3, 4) to use mesh models and to handle patch, part, and mixed segmentations.

6.3 METHOD

6.3.1 *Preliminaries*

To start with, we briefly outline the idea of the cut space part-based segmentation in [60, 63], which we next refer to as the 'voxel cut space segmentation' (VCS) (see also Fig. 6.1 top). Given a voxel shape $\Omega \subset \mathbb{Z}^3$, its surface skeleton S is first computed, using the method in [83]. Next, a simplified skeleton S_τ is extracted from S by removing voxels having an importance lower than a small predefined value τ. These are essentially voxels close to the boundary ∂S that correspond to small-scale details on the surface $\partial\Omega$ (Fig. 6.1b, top). This regularization makes sure that cuts (to be computed next) are only created from stable, important, skeleton parts. In this step, for each voxel $\mathbf{x} \in S_\tau$, a cut $c(\mathbf{x}) \in \partial\Omega$ is computed by tracing three shortest paths $\gamma_1, \gamma_2, \gamma_3$ on $\partial\Omega$ between the feature points $\mathbf{f}_1$ and $\mathbf{f}_2$ of $\mathbf{x}$ (γ_1), $\mathbf{f}_1$ and $\mathbf{m}$ (γ_2), and $\mathbf{f}_2$ and $\mathbf{m}$ (γ_3), where $\mathbf{m}$ is the reflection on $\partial\Omega$ of γ_1's midpoint with respect to $\mathbf{x}$ – see red curve in Fig. 6.1c, top. Cuts are piecewise-geodesic (thus, smooth and tightly wrapping around Ω), closed, and locally orthogonal to the object's symmetry axis (see again Fig. 6.1c, top). These properties ensure that cuts form good candidates for segment boundaries. The cut space $\mathscr{CS} = \{c(\mathbf{x}) | \mathbf{x} \in S_\tau\}$ is next partitioned into several cut-sets $\mathscr{K}_i$ containing similar-length cuts, using either an analysis of the cut-length histogram [60] or hierarchical clustering [63] (Fig. 6.1d, top). Next, the borders ∂C_i^{part} of the segments are computed by searching for cuts that separate the cut-sets $\mathscr{K}_i$ (Fig. 6.1e, top). Finally, the actual segments C_i^{part} are computed by searching for connected components on $\partial\Omega$ separated by the borders ∂C_i^{part} (Fig. 6.1f, top). For full implementation details, we refer to [60, 63], and also to Chapters 3 and 4, respectively.

In the remainder of this section, we outline how we adapt the above voxel-based pipeline to work on mesh-based shapes admitting a point-cloud skeleton, and also handle mixed part-patch segmentations. Sections 6.3.2 – 6.3.7 describe our part-based pipeline (steps in Fig. 6.1b-g, bottom). Section 6.3.8 describes the patch-based pipeline and how this is unified with the part-based one (Fig. 6.1h-l, bottom).

6.3.2 *Regularized Surface Skeleton Computation*

Step 1 of our proposed method parallels step 1 of VCS: We compute a simplified surface skeleton of the input shape. However, we have to skeletonize mesh shapes rather than voxel volumes. For this, we use the technique in [82], which is, to our knowledge, the fastest method to compute surface skeletons of mesh shapes to date. This method outputs a *point cloud* representation of the skeleton S. Also, per skeleton point, the method computes two feature points, *i.e.*, $\mathbf{f}_1$ and $\mathbf{f}_2$ in Eqn. 6.2. In contrast, the voxel skeletonization method in [83] used by VCS outputs a voxel-

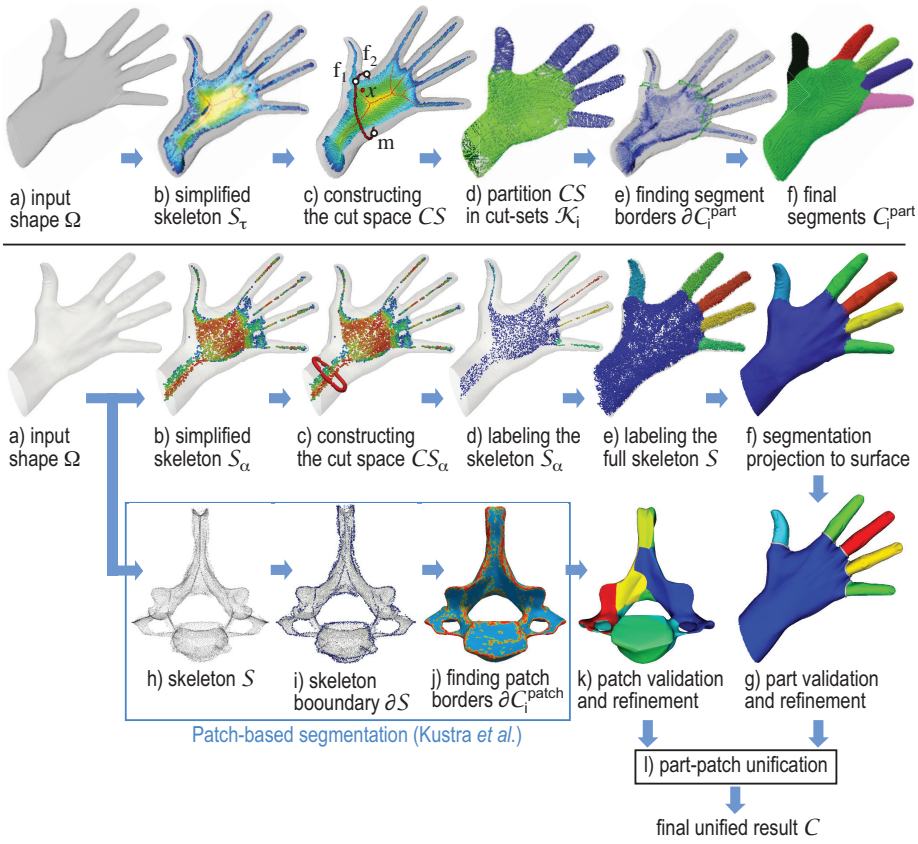


Figure 6.1: Top: Cut-space segmentation pipeline from [60, 63]. Bottom: Our proposed pipeline.

based skeleton. As we shall see, point-cloud skeleton representations introduce several challenges to be addressed.

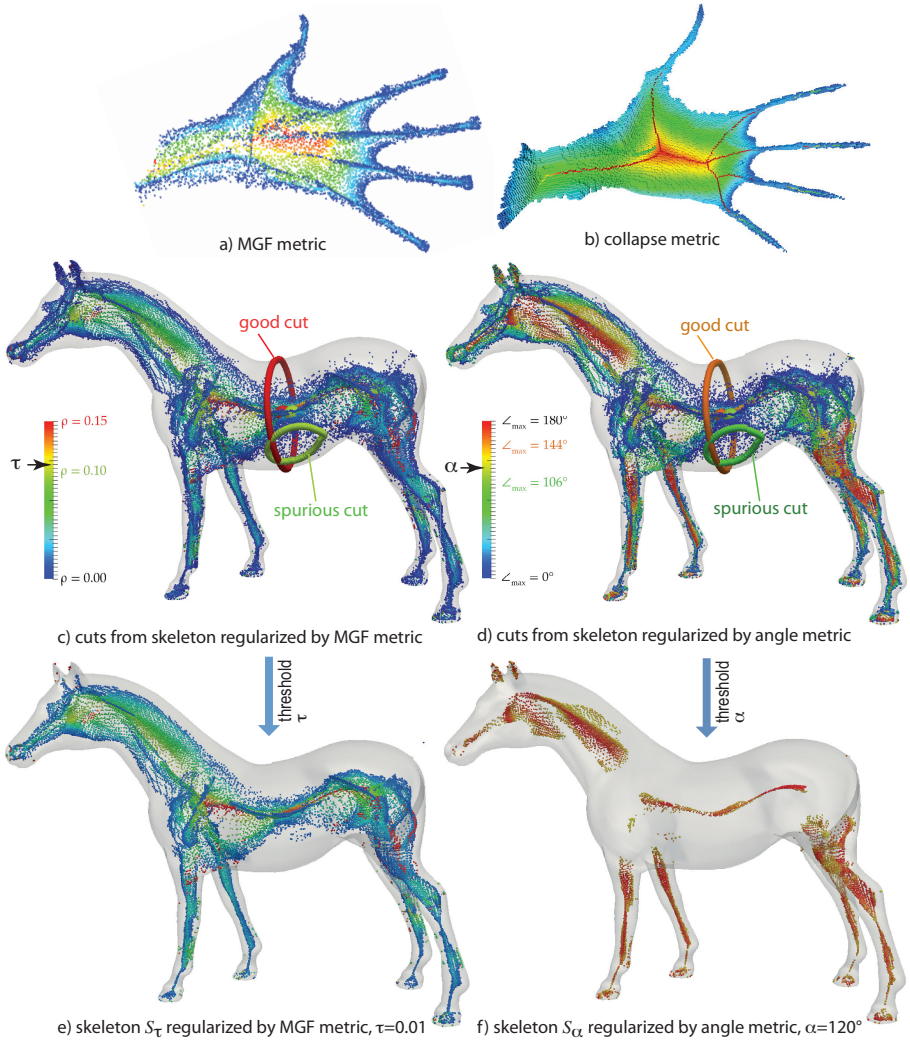


Figure 6.2: (a,b) Comparison of MGF and collapse importance metrics. (c) Thresholding the MGF metric keeps skeleton points that generate spurious cuts. (d) Thresholding the angle metric allows robustly separating good from spurious cuts. All models are color-coded by the respective importance metrics.

Similar to VCS, we want to regularize $S_{\partial\Omega}$ to avoid creating cuts from unimportant skeletal points that are caused by small-scale noise on $\partial\Omega$. At first sight, we

could use for this the MGF importance metric delivered by the underlying skeletonization method [82] (see also Sec. 6.2.1). The rationale for this would be that the MGF metric is analogous to the collapse metric provided by [83] and used by VCS.

However, upon careful examination, we note several issues. If we compare the MGF and collapse metrics for the same shape (Figs. 6.2a,b), we see that they are similar, except for points close to the shape's curve skeleton, where the collapse metric attains much higher values than over the surrounding surface skeleton (red curves in Fig. 6.2b). Hence, thresholding the collapse metric, as done in VCS, eliminates noisy skeleton-points *and* also preserves the important curve-skeleton points, which capture the shape part-whole structure (Sec. 6.2.2.1). In contrast, thresholding the MGF metric eliminates noise, but also all skeleton points in thin shape areas, which is undesired. Figure 6.2c illustrates this: At the current threshold level, the shown surface skeleton contains both important points (such as the one generating the well-oriented red cut), but also unimportant points (such as the one generating the green cut). If we thresholded the MGF metric with values larger than the one corresponding to the green cut, *i.e.* around the value τ shown in the color legend, we would loose about 70% of the entire skeleton, including the complete legs and ears of the horse model. Even a very conservative setting of the threshold $\tau = 0.01$ immediately eliminates detail parts such as the ears (see Fig. 6.2e). Hence, we cannot use the MGF metric to reliably keep skeletal points that correspond to small shape parts *and* in the same time eliminate spurious skeleton points that generate badly oriented cuts.

To solve this problem, we note that surface-skeleton points close to the curve-skeleton have large angles between their feature vectors [145]. Separately, skeleton points created by small-scale noise on the input surface have low MGF values, thus close feature vectors. Hence, we regularize the point-cloud skeleton using the angle between the feature vectors $\mathbf{f}1$ and $\mathbf{f}_2$ of a skeletal point, which is a well-known local importance metric [65, 148]. We define the simplified skeleton as

$$S_\alpha = \{\mathbf{x} \in S | \angle(\mathbf{f}_1, \mathbf{f}_2) > \alpha\} \tag{6.4}$$

where $\alpha = 120^\circ$ is a fixed preset that delivered good results for all our tested shapes. Figure 6.2d shows the result: The angle metric consistenly gets high values close to the curve-skeleton branches of *all* shape parts (rump, legs, muzzle, ears), and gets low values close to the noisy boundary of the surface skeleton. Hence, if we threshold the surface skeleton above the value α indicated in the figure, we robustly eliminate spurious cuts like the green one, and keep cuts that wrap around the shape's local symmetry axis (curve skeleton), like the red one. The resulting simplified skeleton S_α is shown in Fig. 6.2f. Note that this skeleton is not connected, first because it is just a point cloud, and next due to the removal of

low-importance points. However, this does not pose any problem further on, as we use it only to construct our segmentation cuts.

The proposed angle-based regularization of the surface skeleton is simple to implement (involves a dot product of the normalized feature vectors). Additionally, it eliminates the need to compute the MGF metric for surface skeletons, which is the most expensive part of the skeletonization method for mesh shapes in [82].

6.3.3 *Cut-Space Computation*

Step 2 of our method computes the cut space $\mathscr{CS}_\alpha = \{c(\mathbf{x}) | \mathbf{x} \in S_\alpha\}$ from the regularized skeleton S_α. To construct cuts, we cannot use the VCS approach, which employs Dijkstra's algorithm to connect the feature points $\mathbf{f}_1$ and $\mathbf{f}_2$ of a skeleton point $\mathbf{x}$ by three shortest paths in the adjacency graph implied by the voxel surface $\partial\Omega$ (see again Fig. 6.1c and related explanation). If we did so, *e.g.* by using the adjacency graph implied by the surface mesh vertices and triangle edges, we would obtain heavily zig-zagging cuts, which would be useless for our task of inferring segment borders. To create cuts, we propose here to use the geodesic-tracing technique introduced in [82] for the different purpose of evaluating the MGF skeleton-importance metric (see discussion in Sec. 6.3.2).

In detail, we proceed as follows. A straightest geodesic (SG) $\gamma_S : \mathbb{R}^+ \to \partial\Omega$, represented as a parameterized curve, is defined as the unique solution of the initial-value problem $\gamma_S(0) = \mathbf{p}, \gamma_S'(0) = \mathbf{v}$, with $\mathbf{p} \in \partial\Omega$ being a point on the shape boundary having tangent vector $\mathbf{v} \in T_\mathbf{p}$, where $T_\mathbf{p}$ is the plane tangent to $\partial\Omega$ at $\mathbf{p}$. Jalba *et al* [82] proposed an extension to define shortest-and-straightest geodesics (SSGs) γ_{se} between two points $\mathbf{s} \in \partial\Omega, \mathbf{e} \in \partial\Omega$ to be an accurate approximation of the SG from $\mathbf{s}$ to $\mathbf{e}$. SSGs are computed by tracing multiple straightest geodesics over tangent vectors $\mathbf{v}_i \in T_\mathbf{s}$, and then selecting the one with shortest length $\|\gamma_{S,i}\|$, *i.e.*

$$\begin{aligned} \gamma_{S,i}(0) &= \mathbf{s}, \quad \gamma_{S,i}'(0) = \mathbf{v}_i \\ \gamma_{S,i}(\|\gamma_{S,i}\|) &= \mathbf{e} \\ \gamma_{se} &= \arg\min_i \|\gamma_{S,i}\| \end{aligned} \tag{6.5}$$

For computing the MGF metric, Jalba *et al* computed the SSG between feature points $\mathbf{f}_1$ and $\mathbf{f}_2$ of each skeleton point $\mathbf{x} \in S$. For our purpose of computing shortest cuts, however, we require the geodesic to start and end in $\mathbf{f}_1$ and pass $\mathbf{f}_2$ somewhere in-between. As such, we redefine $\gamma_{S,i}$ as

$$\gamma_{S,i}(0) = \mathbf{f}_1, \qquad \exists t \in \mathbb{R}^+ : \ \gamma_{S,i}(t) = \mathbf{f}_2, \qquad \gamma_{S,i}(\|\gamma_{S,i}\|) = \mathbf{f}_1. \tag{6.6}$$

To compute γ_{se} using Eqns 6.5 and 6.6, we proceed similarly to [82]: For each skeleton point $\mathbf{x} \in S$, we trace $M = 30$ straightest geodesics $\gamma_{S,i}, 1 \leq i \leq M$, with

starting directions $\mathbf{v}_i$ uniformly distributed in $T_{\mathbf{s}}$, and retain finally the one with minimal length. For each direction $\mathbf{v}_i$, we compute $\gamma_{S,i}$ by intersecting the mesh $\partial\Omega$ with the plane with normal $\mathbf{n}_i = \mathbf{f}_1 \times \mathbf{v}_i$ that passes through $\mathbf{s}$. In contrast to [82], we continue tracing until having found an intersection with both $\mathbf{f}_2$ and finally arrive back at $\mathbf{f}_1$. Finally, we gather all such SSGs to construct the cut space

$$\mathscr{CS}_\alpha = \{\gamma_{\mathbf{f}_1\mathbf{f}_2} | (\mathbf{f}_1, \mathbf{f}_2) \in FT_{\partial\Omega}|_{S_\alpha}\}, \tag{6.7}$$

i.e., all cuts generated by points of the simplified skeleton S_α.

6.3.4 *Cut-Space Partitioning*

In step 3, we identify how to partition the cut space $\mathscr{CS}_\alpha$ into cut-sets that correspond to the shape's segments. We employ a similar approach to the VCS method, *i.e.* use a histogram of cut lengths, in which peaks indicate many cuts with similar lengths, which likely belong to similar shape parts. With this assumption, we can find valleys separating the histogram's peaks (Sec. 6.3.4.1), which in turn provide us with length thresholds to partition $\mathscr{CS}_\alpha$ (Sec. 6.3.4.2).

6.3.4.1 *Histogram Valley Detection*

To automatically and robustly detect histogram peaks and valleys, we need to analyze the histogram's bins and their interrelationships. For this, VCS first searches for a bin high enough to be considered as peak, then continues searching for the next bin smaller than a certain quantity which is considered a valley. While this method does give an indication of where peaks and valleys are located, it suffers from not finding the smallest valley because of the greedy search for valleys. Moreover, what one would consider a valley depends on the neighboring bins. Hence, we propose to make the search for valleys take into account their surroundings in the histogram.

To do this, we proceed as follows (see also Fig. 6.3). First, we use the mean shift algorithm [36] on the histogram bin heights, to sharpen the differences between peaks and valleys. Next, we start scanning the histogram bins, left to right, for a peak that contains at least $\lambda_p = H_{\text{peak}} \cdot \|\mathscr{CS}_\alpha\|$ cuts. When found, we remember its height h_p, and continue scanning and updating h_p as long as higher peaks are found. In the same time, we search also for valleys, *i.e.* bins having fewer than $\lambda_v = H_{\text{decrease}} \cdot h_p$ cuts. This way, a valley's detection depends on the height of the peak (h_p) it follows. Just as for peak detection, when a bin having fewer than λ_v cuts is found, we remember its height h_v, and continue scanning and updating h_v as long as lower peaks are found. If, during this valley-scan, we find a bin taller than λ_p, we have found a new peak. We have now two peaks and a valley of height h_v in-between. We store the valley's height $\theta_0 = h_v$ and continue scanning the histogram

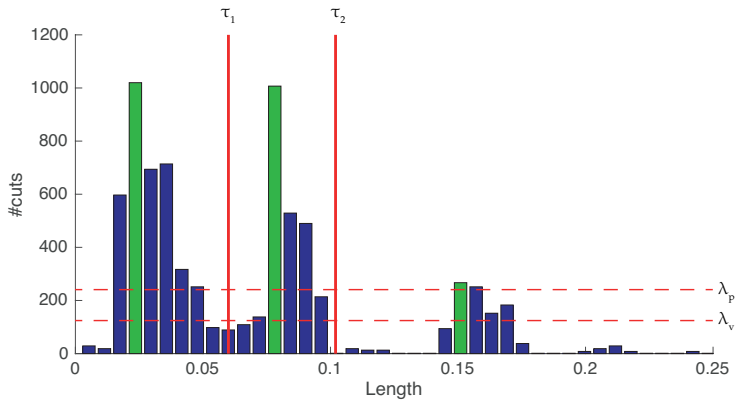


Figure 6.3: Histogram of cut-lengths of a horse shape from which two thresholds have been detected. The cuts in range $[0, 0.06)$ represent the horse's legs, range $[0.06, 0.1)$ corresponds with the neck and longer cuts correspond with the torso.

as above. After scanning the entire histogram this way, we obtain a set of valley heights $\Theta = \{\theta_i\}$. These will deliver our thresholds used to partition the cut space, as explained next in Sec. 6.3.4.2.

We established that good parameter choices are $H_{\text{peak}} = 0.01$ and $H_{\text{decrease}} = 0.25$, such that a peak should represent at least 1% of all cuts and a valley is smaller than a quarter of its accompanying peak.

6.3.4.2 *Histogram-based Cut Space Partitioning*

Using the set of valley thresholds Θ, we can now partition $\mathscr{CS}_\alpha$. This can be done in linear time in the cut space size $\|\mathscr{CS}_\alpha\|$, by iterating over histogram bins left-to-right and assigning all cuts between two consecutive thresholds θ_i, θ_{i+1} to the cut-set $\mathscr{K}_i$.

As for the VCS method, a cut-set $\mathscr{K}_i$ does not necessarily represent a compact shape segment, but could contain several such segments – for example, the five fingers of a hand have similar thickness, so they end in the same cut-set. To separate segments from cut-sets, we partition each cut-set into connected components, based on the connectivity of the skeleton points that generate the cuts. This was trivial to accomplish for VCS, given the explicit connectivity of skeleton voxels. In our case, the skeleton is an unorganized point-cloud lacking connectivity, as explained in Sec. 6.3.2. To address this, we define connectivity of skeletal points

in terms of the neighborhood relation $\mathcal{N}_S(\mathbf{x})$ of a skeleton point $\mathbf{x} \in S$ to other skeleton points within a distance r from $\mathbf{x}$ as

$$\mathcal{N}_S(\mathbf{x}) = \{\, \mathbf{y} \in S \mid \mathbf{y} \neq \mathbf{x} \wedge \|\mathbf{x} - \mathbf{y}\| < r \,\} \tag{6.8}$$

where setting r to 1% of the diameter of $\partial\Omega$ gives good results, following similar approaches in *e.g.* [82, 98].

The result of the cut space partitioning is a labeling $\mathscr{L}_\alpha : S_\alpha \to \mathbb{N}$, which associates to each point in the simplified skeleton (or, alternatively, each cut in $\mathscr{C}\mathscr{S}_\alpha$) an integer ID that tells the segment $C_i^{part} \subset \mathscr{C}^{part}$ these are part of. Given the nature of the neighborhood relation $\mathcal{N}_S$ and the imprecise nature of determining the thresholds in Θ, the labeling can be unstable for cuts whose skeleton points are very close to each other. To eliminate such variations, we apply a mode filter over all label values for points in the same neighborhood $\mathcal{N}_S(\mathbf{x})$, *i.e.*, we assign to $\mathscr{L}_\alpha(\mathbf{x})$ the most frequently occurring label over $\mathcal{N}_S(\mathbf{x})$.

6.3.5 *Partitioning the Full Surface Skeleton*

The labeling $\mathscr{L}$ computed as outcome of the cut space partitioning (Sec. 6.3.4.2) is not our final desired result, *i.e.*, the segmentation $\mathscr{C}^{part}$ of $\partial\Omega$. Indeed, $\mathscr{L}$ only assigns segment IDs to a *subset* S_α of the entire surface skeleton S. To obtain the final part-based segmentation $\mathscr{C}^{part}$, we proceed in two steps. First, we extend the labeling $\mathscr{L}_\alpha$ from the simplified skeleton S_α to a labeling $\mathscr{L}$ of the full surface skeleton S. This is described in this section. Next, we project the full labeling $\mathscr{L}$ from S to the shape boundary $\partial\Omega$. This is described in Sec. 6.3.6.

NAIVE SOLUTION: To interpolate $\mathscr{L}_\alpha$ over the entire point-cloud skeleton S, one could use several strategies. A simple (but naive) one is to use a nearest-neighbor scheme where $\mathscr{L}(\mathbf{x} \in S \setminus S_\alpha) = \mathscr{L}_\alpha(\arg\min_{\mathbf{y} \in S_\alpha} \|\mathbf{x} - \mathbf{y}\|)$. This approach has several problems, as follows. First, using the $\mathbb{R}^3$ Euclidean distance metric to determine nearest neighbors will not work for non-convex shapes which have non-convex surface skeletons. Figure 6.4b illustrates this for a dog model, shown in Fig. 6.4a. Here, colored points are labeled points in S_α, and gray points are points to be labeled in $S \setminus S_\alpha$. The marked (red) point $\mathbf{x}$, which is located on the neck skeleton, is closer to point $\mathbf{x}_1$ (on the dog's ear) than to $\mathbf{x}_2$ (on the neck skeleton), so it will wrongly get $\mathbf{x}_1$'s label, *i.e.*, be assigned as part of the ear. A second problem of this nearest-neighbor interpolation is that labels will always meet halfway between labeled points in S_α. This causes problems at junctions of parts having widely different thicknesses. Figure 6.4c illustrates this. We see here a detail of the dog model having two label values over S_α – dark blue for rump points, and cyan for points in the legs. The area where the two label values would

meet, when using the simple nearest-neighbor interpolation, is shown in red, and it is actually the border of a generalized Voronoi diagram having all blue, respective cyan, points as two sites. This partition is undesirable, as it would assign a large part of the rump skeleton to the segments corresponding to the legs. The desirable leg-rump partition is shown by the blue line in Fig. 6.4c.

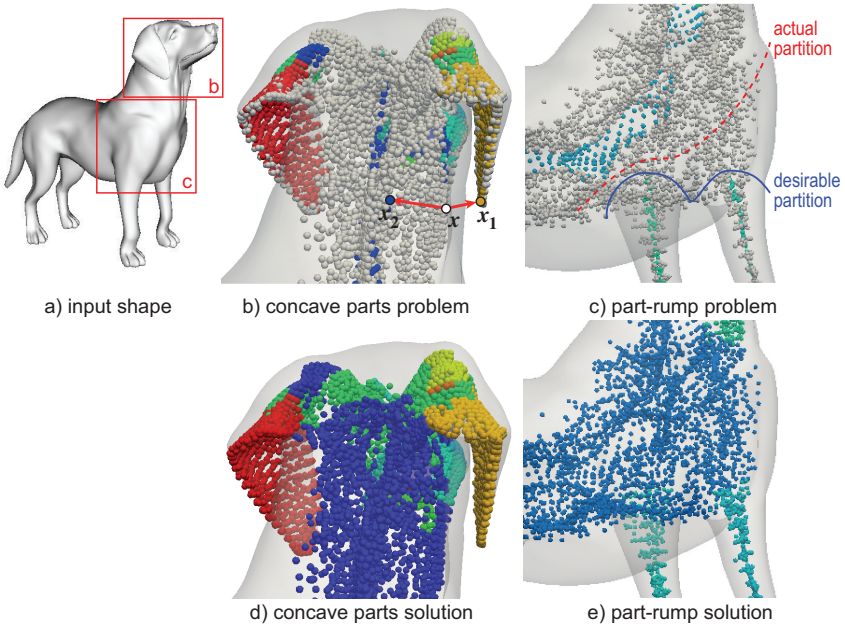


Figure 6.4: Label interpolation over the full skeleton. (a) Input shape. (b,c) Problems caused by naive nearest-neighbors interpolation. (d,e) Effective solution using the distance transform of the skeleton points.

EFFECTIVE SOLUTION: To correct the first problem, one way would be to consider geodesic distance along the S manifold rather than Euclidean distance in $\mathbb{R}^3$. Doing so is however not possible for skeletons represented as point clouds; and reconstructing smooth manifolds from such skeletons is a highly complex and expensive process [96]. Also, this would not correct the second problem. We propose next a much simpler and faster solution that addresses both problems. Key to this idea is the observation that the local shape thickness around a skeletal point should determine how far a label should be propagated. This local thickness is precisely represented by the distance transform $DT_{\partial\Omega}$ values on S. Hence, for a labeled point $\mathbf{x} \in S_\alpha$, we search all its neighbors $\mathbf{y}$ in a ball of radius $DT_{\partial\Omega}(\mathbf{x})$ and, if $DT_{\partial\Omega}(\mathbf{y}) < DT_{\partial\Omega}(\mathbf{x})$, assign $\mathscr{L}(\mathbf{y}) \leftarrow \mathscr{L}(\mathbf{x})$. The last distance condition is required to have labels of skeleton points in thick regions dominate those of

nearby skeleton points in thin regions, and also to ensure that the labeling does not depend on the order of visiting of the skeleton points. Finally, the (few) skeleton points that are still unlabeled after this operation are assigned the label of their nearest neighbor. Figures 6.4d,e show the results of our proposal. As visible, both concave-part and part-rump problems mentioned earlier are now solved as desired.

6.3.6 *Partition Projection to Surface*

Having now a part labeling defined on all skeleton points, we map, or project, this labeling from S to $\partial\Omega$. For this, the original VCS proposal proceeds as follows. All cuts are tested for being borders of the final part segments C_i^{part}. A cut $c(\mathbf{x})$ from a cut-set $\mathscr{K}_i$ is deemed to be a border candidate if at least one of the 26-neighbors of the skeleton voxel $\mathbf{x}$ has a cut that belongs to a different cut-set $\mathscr{K}_j$, $j \neq i$. Next, a single border is picked from a set of border candidates that separates two cut-sets $\mathscr{K}_i$ and $\mathscr{K}_j$, based on heuristics involving the cut length.

In our context, there are several issues with using this method to project the labeling from the skeleton to the surface, as follows. First, our point-cloud skeleton does not admit a direct equivalent of the 26-neighbors relationship used for voxel shapes. The nearest-neighbor relation $\mathscr{N}_S$ (Eqn. 6.8) is too coarse to capture such fine details. Much more critically, though, is the fact that our cuts are *planar* SSGs, while the ones used by VCS consist of three geodesic curves, not necessarily located in a plane (see Sec. 6.3.3). As such, VCS cuts have a much larger freedom to model flexible segment boundaries than our cuts. However, despite their flexibility, the VCS cuts are still constrained by their construction process to consist of three geodesic curves (see Sec. 6.3.1). While this appears to handle quite well part-based segmentation, it is arguably too rigid for producing good patch-based and mixed segmentations, which are our ultimate goal.

We propose a different way of projecting the segmentation information from the skeleton to the shape surface, that addresses all above issues, as follows. For each skeleton point $\mathbf{x} \in S$ having label $\mathscr{L}(\mathbf{x})$, we assign the label to all closest surface points to $\mathbf{x}$, *i.e.*, set $\mathscr{L}(\mathbf{y} \in FT_{\partial\Omega}(\mathbf{x}) \leftarrow \mathscr{L}(\mathbf{x})$. Note that this may not assign labels to boundary points in convex surface regions, since the skeletonization algorithm we use [82] only computes two feature points per skeleton point, rather than the full feature transform (Sec. 6.3.2). We compensate this by assigning labels to all yet unlabeled surface points by simple nearest-neighbor interpolation. Finally, we derive the part segments $C_i^{part} \subset \partial\Omega$ as the connected-components on $\partial\Omega$ that have the same label values.

6.3.7 *Part-based Partition Refinement*

The previous step delivered a partition $\cup_i C_i^{part} = \partial\Omega$ of the input surface into segments. The final step of our part-based segmentation pipeline takes this partition and refines it to yield the final part-based segmentation. Three operations are performed in this refinement, as follows.

Segment validation: First, segments C_i^{part} are checked as to their validity, in terms of what a human observer would qualify as being a 'part' or not. Recalling the assumptions of part-based segmentation [60, 63, 74, 141], we see that actual parts in a good part-based segmentation should be covered well by their associated cuts in the cut space. Intuitively put, if we cut through a part with tight cuts orthogonal to the shape's local symmetry axis, the cuts should stay in the part. Conversely, segments whose cuts cover far more than the segment itself, do not coincide with what is typically perceived as a part. To measure this coverage, let $K(C_i^{part}) = \{c \in \mathscr{CS} | c \cap C_i^{part} \neq \varnothing\}$ be the set of cuts in our cut space that intersects segments C_i^{part}. We then define the coverage of C_i^{part} by cuts as

$$\nu(C_i^{part}) = \frac{\|\{c \in K(C_i^{part}) \mid c \setminus C_i^{part} = \varnothing\}\|}{\|K(C_i^{part})\|}, \tag{6.9}$$

i.e. the fraction of cuts passing through a part that are fully confined to that part. High values of ν indicate segments which are well-covered by their cuts, and thus which are plausible parts. We keep these in the final segmentation. Lower values of ν – below an empirically determined threshold of 0.8 – indicate poor part properties. We remove such segments from the final result by merging them with one of their neighbor segments on $\partial\Omega$.

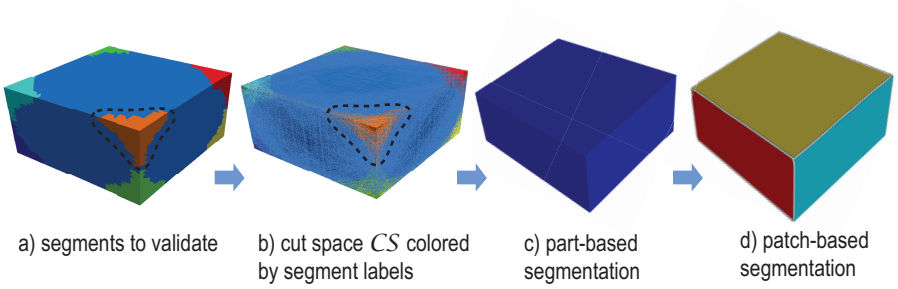


Figure 6.5: Part validation for a box shape. The produced segments (a) are validated by computing their cut-coverages (b). This results in a trivial part-based segmentation (c), which finally determines that a patch-based segmentation will handle this shape (d).

The coverage test effectively filters most, but not all, segments which are not part-like. A salient exception are corners of faceted objects, such as the box in Fig. 6.5a. These segments are covered by cuts which nearly all stay inside them, as shown by the cut space of the shape colored by the labels of the respective cuts (Fig. 6.5b). Hence, such segments will not be marked as invalid. A similar problem (without a solution) was highlighted by the curve-skeleton-based part segmentation technique in [141]. To discard such segments, we use a second heuristic: Parts should 'stick out' of the shape as much as possible [100], or in other words they should have as many points with opposite normals. To test this, we compute angles between all surface normals to points in a segment C_i^{part}, and require that the median of these angles be at least 90°. For the box shape, this invalidates the corner segments, yielding a single segment for the entire shape in terms of part-based segmentation (Fig. 6.5c). As such, the patch-based segmentation of the same shape, which we next discuss in Sec. 6.3.8 gets full freedom to process the shape, leading to the expected result (Fig. 6.5d).

For completeness, we should note that our part validation heuristic is not the only possible one. For instance, Serino *et al.* propose a so-called 'visibility criterion' [162], which aims to detect whether a peripheral part, far from the object's main rump, actively contributes to a meaningful segmentation. This heuristic uses only the curve skeleton in its computation in a voxel-based setting. As our pipeline only uses the *surface* skeleton, and computing the curve skeleton of mesh-based shapes is more complex when considering the technique in [82] which forms the backbone of our approach, the possibility of integrating this criterion in our framework is a topic of further investigation.

Segment border smoothing: Recall that segments C_i^{part} on $\partial\Omega$ are essentially back-projections of skeleton fragments sharing the same label values via the feature transform (Sec. 6.3.6). In the ideal continuous case, equivalent to an infinitely dense sampling of a smooth $\partial\Omega$, and a similar sampling of S, the segments would have smooth, continuous, borders. However, real-world meshes have a limited *and* often highly non-uniform sampling resolution. The used skeletonization method [82] essentially copies this sampling resolution to S. More critically, this method only uses the mesh vertices of $\partial\Omega$ as feature points for estimating the feature transform (Eqn. 6.3). As such, backprojecting labeled skeleton points to $\partial\Omega$ does not result in smooth segment borders. Figure 6.6a shows this for a densely, and uniformly, sampled hand model of 197K vertices. As visible, segment borders exhibit problematic small-scale fractal-like noise. We solve this issue by computing smooth segment borders as follows. First, we create borders ∂C_i^{part} of the segments C_i^{part} by connecting the midpoints of surface triangle-cell edges whose vertices have different labels (thus fall in different segments). Next, we apply classical Laplacian smoothing [196] to remove small-scale wiggles along these

boundaries. After each smoothing pass (10 in total), we reproject the smoothed boundary ∂C_i^{part} to $\partial\Omega$, since unconstrained 3D smoothing makes the boundary curve leave the shape's surface. The overall effect is identical to performing Laplacian smoothing *constrained* on the surface $\partial\Omega$. As the smoothed segment border moves on $\partial\Omega$, we update the labels of the vertices to ensure consistency. Figure 6.6b shows the result: The smoothed boundaries stay roughly in the same position as the initial ones, but are considerably tighter and smoother, thus similar in terms of desirable requirements to the original piecewise-geodesic boundaries of the VCS method. Finally, we draw these smooth boundaries as thick 3D tubes, for ease of perception (Fig. 6.6c).

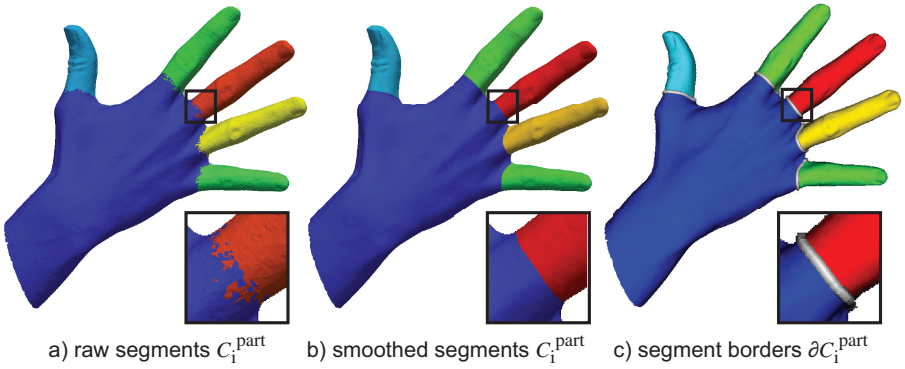


Figure 6.6: Computing part-segment borders. (a) Raw borders produced by skeleton-labeling backprojection. (b) Borders after Laplacian smoothing. (c) Visually emphasized borders.

Visual representation: Our segmentation $\mathscr{C}^{part}$ describes parts on $\partial\Omega$ in terms of *vertices* having the same label. For all practical purposes, such as visualization or further geometric processing, we need a *cell*-based description. For this, we split the triangle cells in $\partial\Omega$ in a Voronoi-like fashion to interpolate, in nearest-neighbor sense, the categorical vertex label values.

6.3.8 *Unified (Part and Patch) Segmentation*

So far, we described how to produce *part-based* segmentations $\mathscr{C}^{part}$ of mesh shapes. However, as outlined already in Sec. 6.1, our goal is to segment any shape, thus propose a way to combine part-based and patch-based segmentations in a flexible way. We present here a way to combine the two segmentation types in a new segmentation model, which we refer to as *unified* (part-and-patch) segmentation $\mathscr{C}$. We first introduce the patch-based segmentation method we use (Sec. 6.3.8.1).

Next, we discuss the desirable properties of a good unification method, and possible strategies to implement it (Sec. 6.3.8.2). Finally, the unification technique we propose is presented in Sec. 6.3.8.3.

6.3.8.1 *Patch-type Segmentation using Surface Skeletons*

For patch-based segmentation, we use the skeleton boundary backprojection (SBB) method of Kustra *et al.* [98], which has several desirable properties in our context. First, this method treats high-resolution mesh shapes, which is our application target. Secondly, the method is also based on surface skeletons, which matches our toplevel goal of showing how such skeletons can effectively support shape segmentation. Thirdly, the method uses the same skeletonization technique [167] as our part-based segmentation, which makes the technical combination of part and patch segmentation easy. We next briefly explain this method and also outline its limitations relevant to our unified segmentation goal.

As all other patch segmentation methods, Kustra *et al.* define patches C_i^{patch} implicitly by requiring that borders separating them should occur in high surface-curvature areas. To do this, they proceed as follows. First, the point-cloud surface skeleton S of the input shape Ω is computed using the technique in [167]. Next, points on the boundary ∂S of the surface skeleton, or so-called A_3 points [71], are detected as those skeletal points whose image on $\partial\Omega$, via the feature transform, contains a single compact cluster. However, as noted earlier in this paper, the underlying skeletonization method [167] does not compute the full feature transform, but only two feature points per skeleton point. Kustra *et al.* note that computing the exact (full) feature transform is sensitive, so they propose instead a so-called extended feature transform

$$FT^{\tau}_{\partial\Omega}(\mathbf{x} \in S) = \{\mathbf{f} \in \partial\Omega \,|\, \|\mathbf{x}-\mathbf{f}\| \leq DT_{\partial\Omega}(\mathbf{x}) + \tau\} \tag{6.10}$$

which gathers, for each skeletal point $\mathbf{x}$, all boundary points within a range $DT_{\partial\Omega}(\mathbf{x}) + \tau$, where τ is a small positive value. The extended feature transform provides a conservative approximation of the actual feature transform (Eqn. 6.3), and can be readily used to detect ∂S as outlined above. Similar conservative approximations of the feature transform are also proposed by the VCS method for voxel shapes [60, 63].

After the skeleton boundary is detected, the method projects ∂S to the shape surface via the extended feature transform, *i.e.*, compute the set

$$\Phi = \{\mathbf{f} \in FT^{\tau}_{\partial\Omega}(\mathbf{x}) \mid \mathbf{x} \in \partial S\} \tag{6.11}$$

which conservatively captures convex edges on $\partial\Omega$. Patches C_i^{patch} are now easily found as the connected components of $\partial\Omega \setminus \Phi$. Finally, points in Φ get assigned

to the closest established patch, thereby completing the patch-based segmentation $\mathscr{C}^{patch}$ of $\partial\Omega$.

Figure 6.7(top row) shows the results of the patch-based segmentation method described above on two shapes (*fandisk* and *horse*). For *fandisk*, which has clear and salient edges, a very good patch segmentation is produced (Fig. 6.7a). The *horse* shape has much softer and fuzzier edges, and as such yields a poor patch-type segmentation (Fig. 6.7b). This can be explained as follows: Finding a reliable set Φ that captures edges on the surface $\partial\Omega$ requires computing stable A_3 points to detect the skeleton boundary ∂S (Eqn. 6.11). However, the heuristic described earlier of finding A_3 points via the extended feature transform fails for cylinder-like parts of a shape, as discussed in [98]. While fine-tuning various parameters of the method of Kustra *et al.* sometimes improves results, patch-type segmentation does not work well for shape regions having too soft edges. In contrast, using our part-based segmentation method described in Secs. 6.3.2 – 6.3.7 yields very good results for tubular, soft-edged, shapes like *horse* (Fig. 6.7d). Conversely, part-based segmentation produces a poor result for the faceted shapes like *fandisk* (Fig. 6.7c). In this case, no part is identified, which is correct, given the part-validation criteria outlined in Sec. 6.3.7. Hence, to obtain the best segmentation results for all shape types, including those that have a mix of faceted and tubular parts, a unification of part- and patch-based segmentation is needed. Such a method is proposed next.

6.3.8.2 *Unification Desirable Properties*

Before designing a part-patch unification strategy, we must define its desirable properties. Based on our experience, we outline the following key properties:

1. *hybrid:* the strategy should handle shapes admitting a full part-based segmentation, shapes admitting a full patch-based segmentation, and also shapes admitting a mix of the two segmentation types in various areas;

2. *intuitive:* the unified segmentation should make sense according to human perception – that is, the produced parts, respectively patches, should visually make sense with respect to the part and patch properties established in Secs. 6.2.2.1 and 6.2.2.2;

3. *balanced:* unified segmentations should not be over-segmented due to incorporating both segment types;

The simplest unification approach would be to compute separate part-based and patch-based segmentations and then choose the result that maximizes the respective quality properties of the two segmentations. However, this solution cannot handle shapes that require a hybrid segmentation. An alternative is to compute the two segmentation types separately, but then *mix* their results in terms of selecting

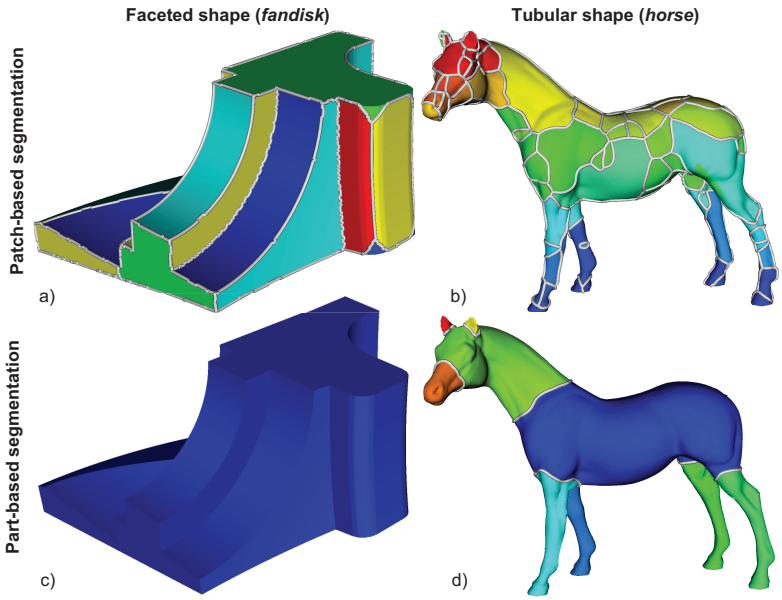


Figure 6.7: Two typical patch-type and part-type shapes, segmented by patch-based and part-based segmentation.

the optimal segments with respect to both local quality criteria (that decide which type of segmentation best fits a region of the shape) and global criteria (the user's preference for part- or patch-based segmentations). This approach can satisfy all above-mentioned requirements. As such, we choose this way next, as follows. For part-based segmentation $\mathscr{C}^{part}$, we use our pipeline presented in Secs. 6.3.2 – 6.3.7. For patch-based segmentation $\mathscr{C}^{patch}$, we use the method discussed in Sec. 6.3.8.1. Our unification method is discussed next.

6.3.8.3 *Unification Method*

As outlined in Sec. 6.3.8.2, our approach to a unified segmentation is to compute both part- and patch-based segmentations of the shape, and next decide which of the produced segments are *valid* in terms of specific part and patch requirements. Valid segments are kept and finally merged to yield the unified segmentation. The process is detailed next.

PART VALIDATION: For part-based segmentation, we use the segment validation procedure described in Sec. 6.3.7, which consists of the cut-coverage and angle-coverage criteria. As such, part-based segmentation will only produce valid

part-segments C_i^{part}, in areas where such segments can be computed. When and where valid segments cannot be computed, the shape will be left unsegmented.

PATCH VALIDATION: For patch-based segmentation, the method of Kustra *et al.* does not provide patch validation. Hence, this method may oversegment the shape or produce otherwise suboptimal patches, such as shown by the example in Fig. 6.7b. Merging such suboptimal patches with otherwise good parts will yield an overall poor unified segmentation. We address this by proposing a patch validation scheme, similar in spirit to the part validation discussed earlier.

As already outlined, a good (valid) patch C_i^{patch} should have its borders in high-curvature regions of the input surface $\partial\Omega$. Several methods exist for computing curvature on mesh surfaces, such as using the curvature tensor [120, 195]; using moment analysis [34]; or estimating the shape thickness over the surface-skeleton boundary [98]. The first two methods are quite sensitive in terms of setting the scale at which edges are detected, as also noted in [98]. The last method, which is also used to detect patch borders in the segmentation technique described in Sec. 6.3.8.1, does not have this problem, but captures also soft edges, leading to issues such as the oversegmentation in Fig. 6.7b.

For our patch-segmentation context, we need to compute segment borders that follow as precisely as possible salient edges on $\partial\Omega$. We propose a simple but effective edge detector for this, as follows. Let $V = \{(\mathbf{x}, \mathbf{n}(\mathbf{x}))\}$ be the vertices, and their normals, of the surface mesh $\partial\Omega$, and let $E = \{(\mathbf{x}, \mathbf{y}) \mid \mathbf{x} \in V, \mathbf{y} \in V\}$ be the mesh edges. For a mesh point $\mathbf{x} \in V$, denote by

$$\mathcal{N}_V(\mathbf{x}) = \{\, \mathbf{y} \in V \,|\, \|\mathbf{x} - \mathbf{y}\| < r \,\} \tag{6.12}$$

the nearest neighbors of $\mathbf{x}$ in the mesh within a radius r. Next, let

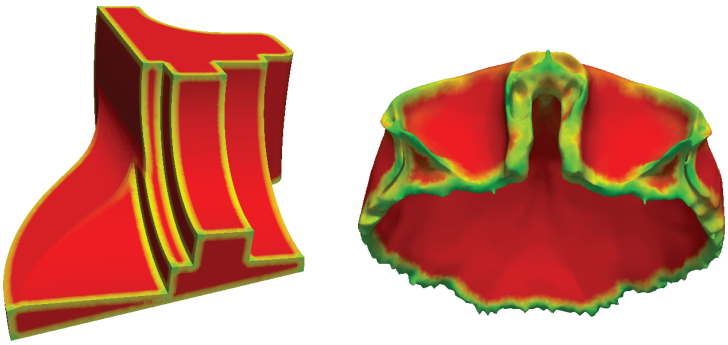
$$\partial\mathcal{N}_V(\mathbf{x}) = \{\, \mathbf{y} \in V \setminus \mathcal{N}_V(\mathbf{x}) \,|\, (\mathbf{x}, \mathbf{y}) \in E \,\} \tag{6.13}$$

be the boundary of the neighborhood $\mathcal{N}_V(\mathbf{x})$. We approximate the curvature $\kappa(\mathbf{x})$ of the surface at point $\mathbf{x}$ by the mean-angle of all point combinations in $\partial\mathcal{N}_V(\mathbf{x})$, *i.e.*

$$\kappa(\mathbf{x}) = \text{mean}\{\, \angle(\mathbf{n}(\mathbf{y}), \mathbf{n}(\mathbf{z})) \,|\, (\mathbf{y}, \mathbf{z} \neq \mathbf{y}) \subset \partial\mathcal{N}_V(\mathbf{x}) \times \partial\mathcal{N}_V(\mathbf{x}) \,\}. \tag{6.14}$$

where $\times$ denotes Cartesian product.

Figure 6.8 shows our curvature estimator κ for two shapes. As visible, κ captures all zones where salient edges exist, both for the *fandisk* model, which exhibits clear edges, and for the *frontal bone* model, which has much noisier edges. Here, the neighborhood size r (Eqn. 6.12) is set to roughly 2 to 5% of the diameter of $\partial\Omega$. This setting has given good results for all other tested shapes.



a) fandisk b) frontal bone

Figure 6.8: Proposed curvature detector κ for two shapes. Red indicates quasi-flat areas. Yellow and green indicate edges.

Using the curvature field κ we can now find and remove invalid patches. For this, we consider each border fragment $\mathscr{B}_{ij}$ that separates partches C_i^{patch} and $C^{patch}j$, which is defined as

$$\mathscr{B}_{ij} = \left\{ \mathbf{x} \in C_i^{patch} \,\middle|\, \exists \mathbf{y} \in C_j^{patch}, (\mathbf{x},\mathbf{y}) \in E \right\} \cup \left\{ \mathbf{x} \in C_j^{patch} \,\middle|\, \exists \mathbf{x} \in C_i^{patch}, (\mathbf{x},\mathbf{y}) \in E \right\}. \tag{6.15}$$

For each such border fragment, we compute the median curvature $\hat{\kappa}(\mathscr{B}_{ij})$ over all its vertices $\mathbf{x} \in \mathscr{B}_{ij}$. If $\hat{\kappa}(\mathscr{B}_{ij})$ is larger than a threshold $\beta = \pi/4$, determined empirically for our studied models, then $\mathscr{B}_{ij}$ is a valid (good) boundary of our patch-based segmentation, so we keep it. If not, then we erase $\mathscr{B}_{ij}$ from the segmentation, *i.e.*, merge patches C_i^{patch} and C_j^{patch}. To ensure a deterministic patch-validation result, we process patch border segments $\mathscr{B}_{ij}$ in increasing order of $\hat{\kappa}(\mathscr{B}_{ij})$.

MERGING SEGMENTATIONS: After removing invalid patches, we have both a validated part-based ($\mathscr{C}^{part}$) and patch-based ($\mathscr{C}^{patch}$) segmentation. We now construct the unified segmentation $\mathscr{C}$ by merging $\mathscr{C}^{part}$ and $\mathscr{C}^{patch}$ as follows. We first initialize $\mathscr{C}$ with $\mathscr{C}^{part}$, *i.e.*, keep all part segments which have been already validated (see Sec. 6.3.8.3). Next, we merge in $\mathscr{C}$ the patches C_i^{patch}. This essentially refines, or subdivides, those parts which have been found to consist of several patches in $\mathscr{C}^{patch}$.

Let us explain how the unified segmentation yields the desirable results for the shapes in Fig. 6.7. For the *fandisk* shape, $\mathscr{C}^{patch}$ contains patches having very high curvature along their borders (Fig. 6.7a). Hence, all borders shown in the figure will be validated and kept as shown. In contrast, its $\mathscr{C}^{part}$ contains a single single

part (Fig. 6.7c). The merging process outlined above will split this single part into precisely the patches of $\mathscr{C}^{patch}$. Hence, the unified result will be identical to $\mathscr{C}^{patch}$, as desired. For the *horse* shape, $\mathscr{C}^{patch}$ contains patches having (very) low curvature values along their borders (Fig. 6.7b). As such, all these patches will be invalidated, yielding $\mathscr{C}^{patch} = \varnothing$. Hence, the merging process will keep $\mathscr{C}^{part}$ unchanged. This is the desired result, since $\mathscr{C}^{part}$ for this shape is of good quality (Fig. 6.7d).

6.4 RESULTS

We next present several segmentation results obtained with our unified method.

We start by comparing part-based segmentations. Figure 6.9 compares our method with the original voxel-based cut space segmentation (VCS) in [63]. As visible, our results are very similar in terms of position, smoothness, and tightness of the delivered segments. We also see that our method succeeds in cases where VCS visibly undersegments the shape (*tool*, *mask*). Small-scale details may differ between the two segmentations, such as for the *rhino* model where VCS undersegments the toes, while our method undersegments the horn and ears. In this particular case (*rhino*), the VCS segmentation is likely better, as separating the larger horns and ear details should be more important than separating the smaller toes. These variations are explainable by the two different parameter settings of the two methods.

Figure 6.10 shows the added-value of the unified segmentation for a set of shapes having relatively soft (shallow) edges. We see how the unified segmentation (bottom row) refines the already-discussed part-based segmentation (top row) by splitting segments along lines of high curvature, *e.g.* the edge of the bird's wing or of the scapular bone. This allows getting a mix of parts, which capture the shape's topology, and faces, or patches, which capture the shape's geometry. The unified segmentation can thus be seen as a refinement of the part-based segmentation.

Figure 6.11 compares our unified segmentation with the SBB method of Kustra *et al.* [98], which is also using surface skeletons to produce patch-based segmentations of 3D shapes. The figure contains the anatomic shapes used as benchmark in [98]. These are quite challenging to segment, as they have complex geometries, a mix of sharp and soft tortuous edges, and consist of both parts and patches. As visible, our method is able to find all patch-like segments that the SBB method can. However, our method generates visibly smoother segment borders, which also better follow the shape's edges, even in the case of very complex geometries such as *frontal bone* and *gyrus*. We also see that SBB places two segment borders in areas where no apparent patch or part transition occurs (*scapula* and *spleen*, marked details). In contrast, our unified method does not allow such borders to exist, due to its part and patch validation steps (Secs. 6.3.7 and 6.3.8.3).

Figure 6.9: Comparison of segmentations between VCS (Feng *et al.*, [63]) and our part-based method.

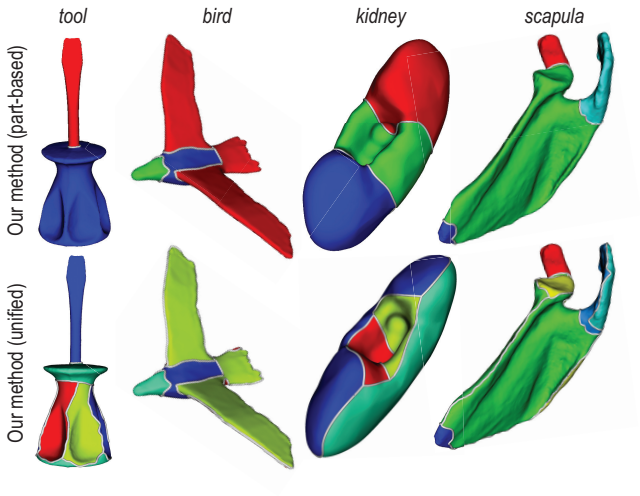


Figure 6.10: Soft-edged shapes segmented by part and unified techniques with our method.

6.5 DISCUSSION

We next discuss several aspects of our unified segmentation method.

COMPARISON: Section 6.4 has compared our method with the two main related methods which use surface skeletons to segment shapes (VCS, [63] and SBB, [98]). It is also interesting to compare our method with the larger class of shape segmentation techniques out there. While it is impossible to do so in the same detail as provided in Sec. 6.4, Figure 6.12 shows a qualitative comparison of our method with seven well-known part-based segmentation methods. Images (a-d) correspond to methods that do not use skeletons. Images (e-f) correspond to methods that use either curve or surface skeletons. For an overview of these methods, we refer to Sec. 6.2.2.1.

The most salient observation on Fig. 6.12 is that skeleton-based methods tend, in general, to provide more 'natural' part-based segmentations than the other studied methods, in terms of positioning and smoothness of the segments' borders. This is due to the inherent ability of skeletons to model a shape's local axis of symmetry, across which segment borders can be fit. In contrast, non-skeleton-based methods cannot ensure this proper border orientation. Secondly, we see that the cut space based methods (Fig. 6.12g,h) do not *oversegment*. This observation is also confirmed by all other earlier examples showing our method (Figs. 6.9, 6.10, 6.11). This is due to two design elements in our method: (1) the cut space partitioning ensures that similar-length cuts will never yield different parts (Sec. 6.3.4); and (2)

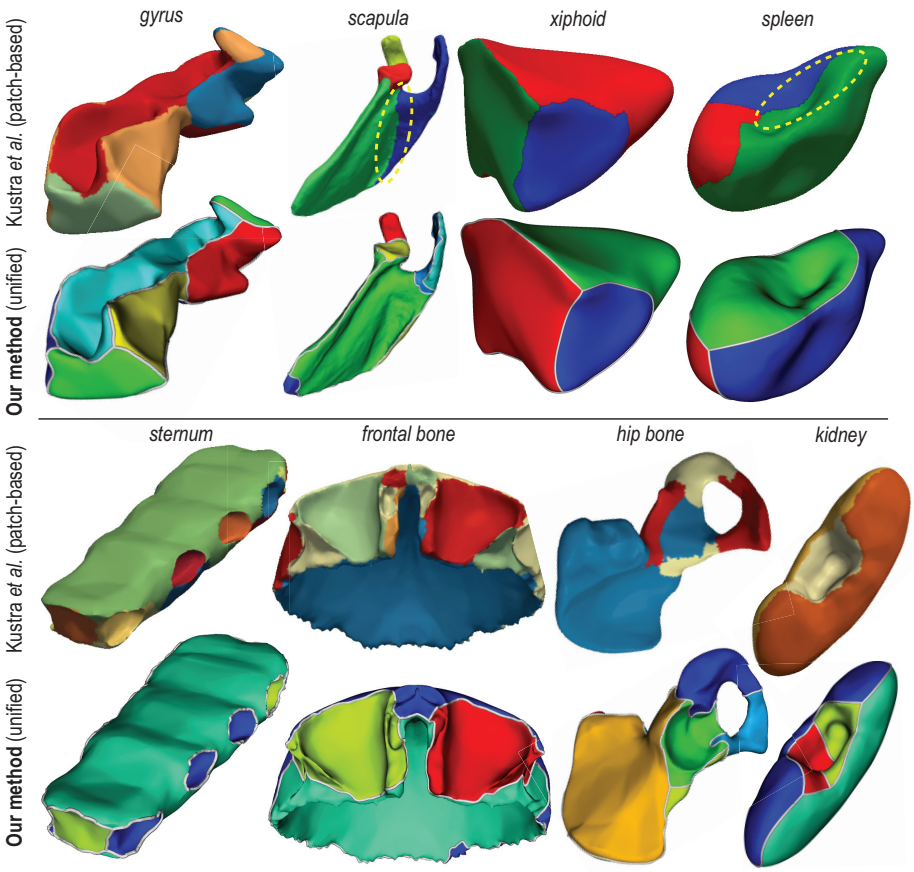


Figure 6.11: Anatomic shapes segmented by the SBB method of Kustra *et al.* [98] and our unified method.

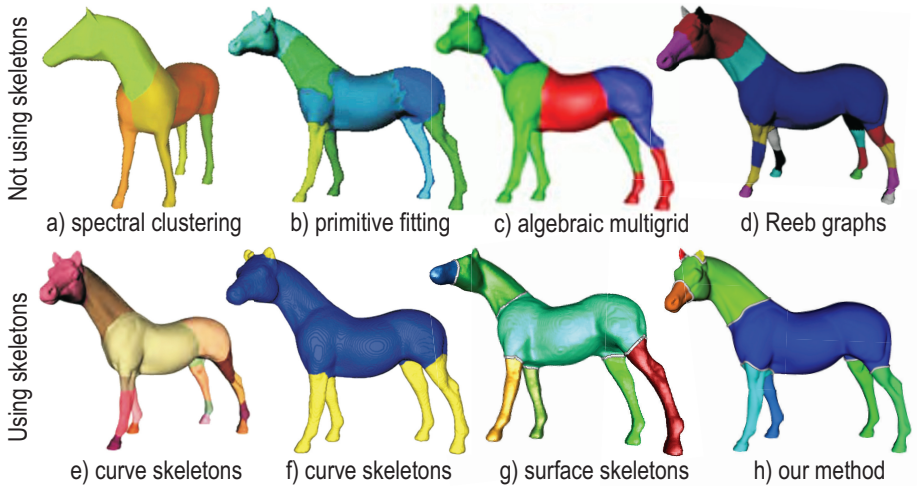


Figure 6.12: Comparison of eight part-based segmentation methods. (a) Liu and Zhang [107]; (b) Attene *et al.* [10]; (c) Clarenz *et al.* [34]; (d) Tierny *et al.* [201]; (e) Lien *et al.* [105]; (f) Reniers *et al.* [141]; (g) Feng *et al.* [63]; (h) our method.

the part and patch validations ensure that superfluous parts and patches are automatically removed (Secs. 6.3.7, 6.3.8.3). This is in contrast to several of the other methods depicted here (Fig. 6.12b-e). All in all, the above observations strongly plead for the added-value of skeletons for shape segmentation.

The fact that there is quite some variation in the segmentations produced by different methods and, implicitly, in their perceived quality, should however be interpreted with care. The comparison presented here, as is far from exhaustive, cannot thus be used to derive generalizing value judgments of one method *vs* another. For instance, the examples in Fig. 6.12e,f show that curve-skeleton-based methods can sometimes oversegment (see horse legs, Fig. 6.12e) and sometimes undersegment (see horse nech and rump, Fig. 6.12f). The fact that surface-skeleton-based methods do not show such artifacts (Fig. 6.12g,h) should not be interpreted as pointing to a general superiority of such methods as opposed to curve-skeleton-based methods. The main conclusion from this comparison is limited to showing that segmentation methods using *surface* skeletons are an interesting and viable alternative for *part* based shape segmentation.

UNIFICATION: If we study the state-of-the-art in shape segmentation methods [11, 168], we see that most such methods are, implicitly or explicitly, focused on handling either part-based or patch-based segmentations, but rarely both. This is easy to explain, as we have seen that the criteria defining (good) parts and patches

are very different, as the two have different natures. Indeed, parts are inherently volumetrically described; while patches are best described on the shape's surface [141, 142]. As such, one typically needs to know in practice what is the nature of shapes one wants to segment, in order to choose the best segmentation method for that task; or else, one needs to manually run several such methods and hope that one of them will be optimal for the shapes at hand. Our unified segmentation proposed here shows that one can use a single descriptor – surface skeletons – to automatically compute *and* combine both segmentation types. This allows users to simply 'drop' their shapes in the tool and let it choose the optimal mix of parts and patches to segment with.

PERFORMANCE: Table 5 shows the performance of our unified method, implemented in single-threaded C++, on an Intel Core i7 3.8 GHz PC with 32GB RAM. The tested shapes vary considerably in terms of mesh resolution ($\|\partial\Omega\|$) and mesh-sampling uniformity, and also in terms of the number of cuts we compute ($\|\mathscr{C}\mathscr{S}_\alpha\|$). We next see that the total cost is dominated by the cut space computation. At first sight, the overall cost appears to be quite high. To better assess this cost, we compute the throughput of our method

$$T_{unified} = \frac{1}{1000}\frac{|\partial\Omega\| \cdot \|\mathscr{C}\mathscr{S}_\alpha\|}{t_{cuts}} \tag{6.16}$$

i.e. the number of cuts, for a given resolution of the mesh (in thousands of vertices), that the method can deliver per second. Using the mesh resolution in Eqn. 6.16 accounts for the fact that the cut computation cost is proportional with the mesh resolution. Table 6 shows the throughput T_{VCS} for the original voxel-based VCS method. Here, $\|\partial\Omega\|$ denotes the number of voxels on the shape surface. The ratio of the average T_{VCS} to the average $T_{unified}$ is 2.32, *i.e.*, the VCS method is 2.32 times faster than ours. However, the original VCS method does use CPU parallelism (8 threads) to compute cuts, while our method is purely sequential. Parallelizing our cut computation is very easy, as cuts are traced completely independently. This makes our method potentially over 3 times faster than the VCS method. Further using GPU parallelism to compute the cuts, following the technique originally proposed by [82] for computing surface-skeleton importance, would make our method significantly faster than the VCS method, which lends itself far less to GPU parallelization. Interestingly, we see that the relative throughput of our method as compared to VCS increases significantly for shapes where many cuts are used, *e.g. scapula* and *heptoroid* – for the second shape, our method has an even higher throughput than VCS. Memory-wise, our method needs to store only the input mesh, point-cloud skeleton, and two feature points per skeleton point, its memory cost being $O(\|\partial\Omega\|)$. In contrast, VCS needs to store four full densely-sampled volumes, yielding a memory cost of $O(\|\partial\Omega\|^{3/2})$ (for details,

see the underlying skeletonization method [83]). All in all, we see that our method scales far better than VCS with respect to shape size.

Shape	$\|\partial\Omega\|$	$\|\mathscr{C}\mathscr{S}_\alpha\|$	t_{skel}	t_{cuts}	t_{patch}	t_{unif}	$\mathbf{T_{unified}}$
Horse	49,749	7,453	5.6	63.2	5.2	6.4	5866
Hound	16,158	364	0.7	5.1	2.0	2.7	1153
Cow	137,862	16,447	23.0	242.3	10.9	63.0	9357
Bird	47,184	26,485	6.3	262.0	5.3	14.1	4769
Pig	4,800	756	0.2	2.0	0.6	0.6	1814
Vertebra	22,789	7,779	0.2	57.4	2.4	2.2	3088
Scapula	117,432	83,340	17.3	2325.7	11.0	26.6	4208
Heptoroid	79,056	63,876	5.0	539.1	6.3	5.8	9367
Kidney	30,389	9,986	6.5	177.7	4.0	27.0	1707
Hand	49,546	2,815	7.3	47.9	4.2	11.1	2911

Table 5: Performance of the proposed unified segmentation method.

Shape	$\|\partial\Omega\|$	$\|\mathscr{C}\mathscr{S}\|$	t_{skel}	t_{cuts}	$\mathbf{T_{VCS}}$
Horse	109555	884	1.24	9.58	10109
Hound	245759	1530	1.51	23.24	16179
Cow	143938	1009	0.96	8.15	17820
Bird	45638	476	0.18	2.28	9527
Pig	145215	959	1.51	10.97	12694
Vertebra	68632	683	0.37	8.56	5472
Scapula	285854	4329	30.0	301.3	4106
Heptoroid	651478	4873	3.36	400.5	7926
Kidney	31874	403	0.16	3.91	3278
Hand	58071	584	0.22	2.15	15773

Table 6: Performance of the VCS part-based segmentation method of Feng *et al.* [63].

IMPLEMENTATION: While at first sight complex, our unified method requires only a few ingredients to be implemented – the point-cloud skeletonization method for mesh models in [82], which delivers surface skeletons, distance transforms, and feature transforms; and a way to find the nearest-neighbors in a point cloud

(Eqns. 6.8 and 6.12). The former is provided by the respective algorithm, and the latter is readily available via the ANN package [122].

LIMITATIONS: While delivering good-quality segmentations of complex shapes, our unified method still has several limitations, as follows. First and foremost, its quality essentially depends on the underlying qualities of the parts and patches delivered by its two branches (Sec. 6.3). While the part and patch validation steps we introduce considerably help delivering good quality parts and patches, the overall result still depends on the possibility of *independently* finding good parts and patches on a shape. There are, obviously, cases when this assumption does not hold. In such cases, designing a new 'joint segment' model able to capture the full continuum between parts and patches is desirable, and is a challenge for future work.

Secondly, our unified method is technically more complex than other part-based and patch-based segmentation methods taken separately. Indeed, it comprises a full part-based segmentation pipeline (7 steps, Secs. 6.3.2–6.3.7), a full patch-based segmentation pipeline (Sec. 6.3.8.1), and a unification pipeline (two steps, Sec. 6.3.8.3). However, this is justified by the fact that our method is, to our knowledge, the only existing one that can handle mixed part-patch based segmentations with good results.

The comparisons of our method with related segmentation methods [10, 34, 60, 63, 98, 105, 107, 141, 201] is, of course, not covering the entire spectrum of segmentation methods out there. This would be highly challenging, if not impossible, to do given the available space and the availability of implementations of such methods. More effort (from the entire shape segmentation community) is required here. Yet, we argue that our main point – showing that our method can leverage surface skeletons to generate part-patch based segmentations than other tools in the same class – has been well defended by the presented results.

6.6 CONCLUSION

In this chapter, we have presented a novel method to create segmentations of 3D mesh shapes. In contrast to most existing segmentation methods out there, we show that it is possible to design a single method that effectively handles tubular (articulated) shapes, faceted shapes, and shapes containing a mix of the two. On a more fundamental level, we show that surface skeletons are an effective tool to support complex segmentation tasks. Thereby, we strengthen the existing evidence that this type of medial descriptors, so far sparsely used in actual applications, are effective tools with practical added value. We support our claims by comparing our method with the most relevant part-based and patch-based segmentation methods

using 3D skeletons, on a collection of shapes ranging from purely faceted to purely tubular.

In particular, this chapter shows that the usage of the cut space to segment voxel shapes (Chapters 3 and 4) can be extended to handle mesh-based shape representations. As such, the cut space can be deemed to be representation independent (at least with respect to voxel and mesh representations). On the one hand, this is obviously good news that supports the added-vale of the cut space concept. On the other hand, the implementation of the cut space computation and processing of its results to perform segmentation for meshes, as shown in this chapter, is significantly more complex from a technical and implementation perspective than its voxel-based counterpart. As such, voxel-based surface skeletons, and their usage, should not be directly discarded in favor of using mesh-based representations. We discuss the particular aspect of this issue regarding the accurate and efficient computation of surface skeletons for voxel models (the principal disadvantage of these models as compared to mesh models) in Chapter 7.

Future work can handle several directions. First and foremost, the current results show that it is possible to create hybrid part-patch segmentations using a single descriptor type (surface skeletons). As such, it is interesting to explore refinements of the presented part and patch detection and merging heuristics, to design methods where users can control the resulting segmentation more easily and intuitively. Secondly, low-hanging fruits include the GPU acceleration of all steps of the proposed pipeline, to yield a single method able to compete, speed-wise, with current state-of-the-art segmentation methods. Thirdly, we expect that the mesh-based cut space presented here can be translated to handle shape matching and retrieval along the techniques presented for voxel models in Chapter 5.

7

MULTISCALE 2D MEDIAL AXES AND 3D SURFACE SKELETONS BY THE IMAGE FORESTING TRANSFORM

As illustrated by all Chapters so far, surface skeletons have multiple applications in supporting shape analysis and processing. However, as the same Chapters also illustrate, *efficiently* computing such *accurate* multiscale skeletons is hard. For voxel models, there are only a handful of multiscale surface skeletonization methods available [82, 145]. Both these methods have limitations. As outlined, [145] is reasonably accurate but very slow. In contrast, [83] is reasonably fast but not too accurate. In more detail, our experience with [83] in computing simplified surface skeletons for shape matching (Chapters 3, 4) has shown that this method can sometimes yield disconnected and/or noisy skeletons, which in turn require applications using them to devise various ad-hoc regularization heuristics. For mesh models, we have fast and accurate methods, *e.g.*, [82]. However, this method is highly complex; more critically, processing the point-cloud skeletons it delivers is significantly more complex than processing voxel-based skeletons (compare the same segmentation method using the two skeleton types presented in Chapters 3 and 6).

As such, we argue that voxel-based surface skeletons have not yet reached their full potential. If we were able to compute a highly accurate voxel-based multiscale surface skeleton, in a computationally scalable way, and using a simple implementation, such skeletons could strongly compete with mesh-based ones. This chapter addresses precisely this task. We present here a new method based on the Image Foresting Transform [53, 58] framework, which achieves such results for medial axes of 2D shapes and for surface skeletons of 3D shapes. Our approach relies on simple and efficient algorithms, faster than several methods based on the same importance metric, simpler than such other methods, far less sensitive to numerical noise than state-of-the-art in this area [83]. In the same time, our method attains similar quality of centeredness, smoothness, thinness, and ease to simplify the skeleton. These conclusions are substantiated with a comparative analysis of our method's results on a wide set of 2D and 3D real-word shapes against its multiscale counterparts.[1]

1 This chapter is based on the following publication: Multiscale 2D medial axes and 3D surface skeletons by the image foresting transform (A. Falc ao, C. Feng, A. Telea), *Skeletonization: Theory, Methods, and Applications*, Chapter 4, eds. P. K. Saha, G. Borgefors and G. Sanniti di Baja, Elsevier, 2016.

7.1 OUTLINE

Medial descriptors, or skeletons, are used in many applications such as path planning, shape retrieval and matching, computer animation, medical visualization, and shape processing [172, 189]. In two dimensions, such descriptors are typically called medial axes. Three-dimensional shapes admit two types of skeletons: *Surface* skeletons are sets of manifolds with boundaries that meet along a set of so-called Y-intersection curves [28, 42, 102]; *curve* skeletons are 1D structures locally centered in the shape [37].

A fundamental, and well-known, problem of skeleton computation is that skeletons are inherently unstable to small perturbations of the input shape [189]. This leads to the appearance of so-called spurious branches, which have little or no application value, and considerably complicate the analysis and usage of skeletons. To alleviate this, various simplification methods have been proposed to eliminate such branches. In this context, *multiscale* methods are particularly interesting. They compute a so-called importance metric for skeletal points, which encodes the scale of the shape details, and next offer a continuous way for simplifying skeletons by simply thresholding that metric.

Several robust, simple to implement, and efficient methods exist for computing 2D multiscale skeletons, *e.g.*, [58, 124, 198]. For surface skeletons, the situation is very different: Only a few such methods exist, and these are either computationally expensive [45, 138, 145], complex [82], or sensitive to numerical discretization [84].

In this chapter, we address the joint problem of computing multiscale 2D medial axes and 3D surface skeletons by a new method. For this, we cast the problem of computing the importance metrics proposed in [58, 124, 198] (for 2D skeletons) and in [82, 145] (for 3D skeletons) as the search for an optimal path forest using the Image Foresting Transform framework [54]. In 2D, the skeletons are one-pixel-wide and connected in all scales for genus-0 shapes. In 3D, the surface skeletons are one-voxel-wide and can be connected in all scales for genus-0 shapes if the curve skeleton points are detected [82, 145], which we do not address here. Next, we provide simple and efficient algorithms to compute these metrics for both 2D and 3D binary images. Compared to 3D techniques that use the same multiscale importance metric, our method is faster [45, 138, 145] or alternatively considerably simpler to implement [82]. Compared to other 3D multiscale techniques, our method is far less sensitive to numerical noise [84]. Compared to all above techniques, our method yields the same quality level in terms of centeredness, smoothness, thinness, and ease to simplify the skeleton.

This chapter is structured as follows. In Section 7.2, we overview multiscale skeletonization solutions and challenges. Section 7.3 introduces the Image Foresting Transform and its adaptations required for multiscale skeletonization. Sec-

tion 7.3.3 details our multiscale skeletonization algorithms for 2D and 3D shapes. Section 7.4 compares our method with its multiscale competitors on a wide set of 2D and 3D real-world shapes. Section 7.6 concludes the chapter, summarizing our main contributions and outlining directions for future work.

7.2 RELATED WORK

In this section, we provide the basic definitions related to skeletonization and discuss skeleton regularization based on local and global measures.

7.2.1 *Definitions*

Given a shape $\Omega \subset \mathbb{R}^d$, $d \in \{2,3\}$ with boundary $\partial\Omega$, we first define its Euclidean distance transform $DT : \mathbb{R}^d \to \mathbb{R}^+$ as

$$DT(\boldsymbol{x} \in \Omega) = \min_{\boldsymbol{y} \in \partial\Omega} \|\boldsymbol{x} - \boldsymbol{y}\|. \tag{7.1}$$

The Euclidean skeleton of Ω is next defined as

$$S = \{\boldsymbol{x} \in \Omega | \exists \boldsymbol{f}_1, \boldsymbol{f}_2 \in \partial\Omega, \boldsymbol{f}_1 \neq \boldsymbol{f_2}, \|\boldsymbol{x} - \boldsymbol{f}_1\| = \|\boldsymbol{x} - \boldsymbol{f}_2\| = DT(\boldsymbol{x})\} \tag{7.2}$$

where $\boldsymbol{f}_1$ and $\boldsymbol{f}_2$ are the contact points with $\partial\Omega$ of the maximally inscribed ball in Ω centered at $\boldsymbol{x}$ [71, 145], also called *feature points* of $\boldsymbol{x}$ [148], where the feature transform $FT : \mathbb{R}^d \to \mathscr{P}(\partial\Omega)$ is defined as

$$FT(\boldsymbol{x} \in \Omega) = \underset{\boldsymbol{y} \in \partial\Omega}{\arg\min} \|\boldsymbol{x} - \boldsymbol{y}\|. \tag{7.3}$$

The vectors $\mathbf{f} - \mathbf{x}$ are called *spoke vectors* [181]. By definition (Eqn. 7.2), for any $\mathbf{x} \notin S$, $FT(\mathbf{x})$ yields a single point, *i.e.*, $|FT(\mathbf{x})| = 1$; while for any $\mathbf{x} \in S$, $|FT(\mathbf{x})| \geq 2$. In practice, computing FT can be quite expensive and/or complicated due to its multi-valued nature. As such, many applications, see *e.g.* [145, 148], use the so-called single-value feature transform $F : \mathbb{R}^d \to \partial\Omega$, defined as

$$F(\boldsymbol{x} \in \Omega) = \mathbf{y} \in \partial\Omega \quad \text{so that} \quad \|\mathbf{x} - \mathbf{y}\| = DT(\mathbf{x}). \tag{7.4}$$

For $d = 2$, S is a set of curves which meet at the so-called skeleton junction points [58]. For $d = 3$, S is a set of manifolds with boundaries which meet along a set of so-called Y-intersection curves [28, 42, 102]. The pair $MAT = (S, DT)$ defines the medial axis transform (MAT) of Ω, which is a dual representation of Ω, *i.e.* allows reconstructing the shape $\Omega_{S,DT} = \bigcup_{\mathbf{x} \in S} B_{DT(\mathbf{x})}(\mathbf{x})$ as the union of balls $B_{DT(\mathbf{x})}(\mathbf{x})$ centered at $\mathbf{x} \in S$ and with radii $DT(\mathbf{x})$.

Except when it is explicitly mentioned, we consider only the case where $\Omega \setminus \partial\Omega$ and $\partial\Omega$ have the same number of components.

7.2.2 *Skeleton Regularization*

In practice, skeletons are extracted from discretized (sampled) versions of Ω, using either an implicit (boundary mesh) representation [15, 82, 124, 190] or an explicit (volumetric) representation [45, 58, 145, 198]. In this chapter, we focus on the latter case. Here, the d-dimensional space is discretized in a uniform grid of so-called *spels* (space elements) having integer coordinates, *i.e.*, pixels for $d = 2$ and voxels for $d = 3$. Due to this discretization, solving Eqn. 7.2 on $\mathbb{Z}^d$ rather than $\mathbb{R}^d$ yields skeletons that are not perfectly centered, not necessarily one-spel-thin, and not necessarily homotopic to the input shape. Discretization also causes skeletons to have a large amount of spurious manifolds (branches). Formally, this means that skeletonization is not a Cauchy or Lipschitz continuous operation with respect to the Hausdorff distance between two shapes, but a semi-continuous operation [189]. Informally put: Small variations of a shape can cause arbitrarily large variations of its skeleton.

To achieve Cauchy or Lipschitz continuity, desirable for most practical applications (which should not be sensitive to discretization issues), a *regularization* process is typically used. For this, one defines a so-called *importance metric* $\rho : \Omega \to \mathbb{R}^+$, whose upper thresholding by some desired value $\tau > 0$ removes, or prunes, branches caused by small-scale details or noise on $\partial\Omega$ [42, 166]. The regularized skeleton is defined as $S_\tau = \{\mathbf{x} \in S | \rho(\mathbf{x}) \geq \tau\}$. We distinguish between local and global importance measures, in line with [15, 82, 84, 145], as follows.

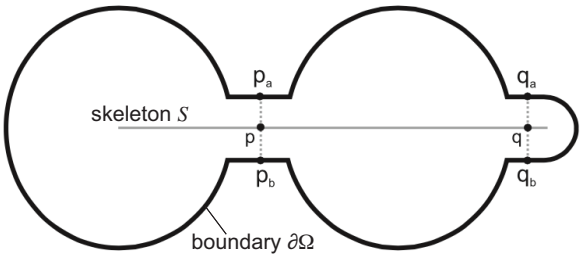


Figure 7.1: Problems of local regularization. For the shown shape, all local regularization metrics will yield the same values for points **p** and **q**. However, **p** is globally more important for the shape description than **q**. Image from [145].

Local measures essentially consider, for a skeletal point $\mathbf{x} \in S$, only a small neighborhood of $\mathbf{x}$ to compute $\rho(\mathbf{x})$. The main advantage of these measures is that they are simple to implement and fast to compute. Local measures include the angle between the feature points and distance-to-boundary [3, 65, 148, 184], divergence metrics [22, 175], first-order moments [153], and detecting the multi-valued points of ∇D [181, 182]. Local measures are, also, historically speaking, the first skeleton regularization techniques having been proposed. Good surveys of local methods are given in [172, 189]. However, local measures have a fundamental issue: They cannot discriminate between locally-identical, yet globally-different, shape contexts. Figure 7.1 illustrates this for a synthetic case: For the 2D shape with boundary $\partial\Omega$, the central skeletal point $\mathbf{p}$ is, clearly, more important to the shape description than the peripheral point $\mathbf{q}$ that corresponds to the right local protrusion. However, any local importance metric will rank $\mathbf{p}$ as important as $\mathbf{q}$, since their surroundings, including the placement of their feature points (shown in the figure) are identical. Similar situations can be easily found for 3D shapes.

Given the above, thresholding local measures can (and typically will) disconnect skeletons. Reconnection needs extra work [113, 134, 175, 184], and makes skeleton pruning less intuitive and harder to implement [166]. Without this kind of work, no local measure can yield connected skeletons for all shapes. Note that this is a fundamental issue related to the local nature of these metrics – see also the discussion in [84]. Secondly, local metrics do not support the notion of a *multiscale* skeleton: Such skeletons S_τ should ensure a continuous simplification (in the Cauchy or Lipschitz sense mentioned above) of the input shape Ω in terms of its reconstruction $\Omega_{S_\tau,DT}$ as a function of the simplification parameter τ [124, 189]. As such, while local measures can be 'fixed' by reconnection work to yield connected skeletons, they still cannot provide an intuitive and easy-to-use way to simplify skeletons according to a user-prescribed threshold τ.

Global measures monotonically increase as one walks along S from the skeleton boundary ∂S inwards, towards points increasingly further away from ∂S. For genus-0 shapes, such measures can be informally thought of as giving the removal order of skeletal points in a homotopy-preserving erosion process that starts at ∂S and ends when the entire skeleton has been eroded away. Given this property, thresholding them always yields connected skeletons which also capture shape details at a user-given scale. For shapes with genus greater than 0, like having holes (in 2D) or cavities (in 3D), simple suitable postprocessing of the importance metric guarantees the joint connectivity and multiscale properties. For 2D shapes, a well-known global measure is the so-called boundary-collapse metric used to extract multiscale 2D skeletons, and proposed by various authors in different contexts [58, 124, 197, 198]. For 3D shapes, the union-of-balls (UoB) approximation uses morphological dilation and erosion to define the scale of shape details cap-

tured by the regularized skeleton [15, 73]. Dey and Sun propose as regularization metric the medial geodesic function (MGF), equal to the length of the shortest-path between feature points [45, 138], and use this metric to compute regularized 3D curve skeletons. Reniers *et al.* [145] extend the MGF for both surface and curve skeletons using geodesic lengths and surface areas between geodesics, respectively. A fast GPU implementation of this extended MGF for meshed shapes is given in [82].

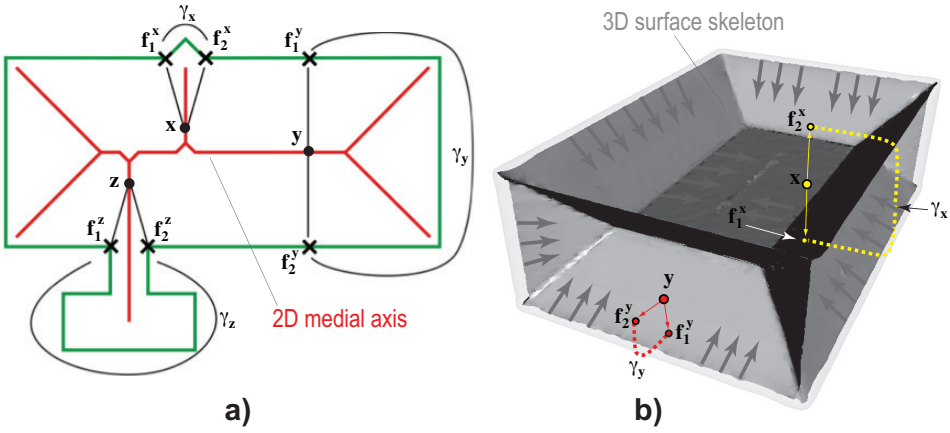


Figure 7.2: Multiscale collapse metric for 2D shapes (a) and 3D shapes (b).

The 3D MGF and its 2D boundary-collapse metric counterpart have an intuitive geometric meaning: They assign to a skeleton point $\mathbf{x} \in S$ the amount of shape boundary that corresponds, or 'collapses' to, $\mathbf{x}$ by some kind of advective boundary-to-skeleton transport. As such, skeleton points $\mathbf{x}$ with low importance values correspond to small-scale shape details or noise; points $\mathbf{x}$ with large importance values correspond to large-scale shape parts. Figure 7.2 illustrates the boundary-collapse principle for both 2D medial axes (a) and 3D medial surfaces (b). In both cases, for a skeletal point $\mathbf{x}$, the importance is equal to the length of the shortest-path $\gamma_{\mathbf{x}}$ that goes on $\partial\Omega$ between the feature points $\mathbf{f}_1^{\mathbf{x}}$ and $\mathbf{f}_2^{\mathbf{x}}$. This allows an intuitive and controllable skeleton simplification: Thresholding the MGF by a value τ eliminates all skeleton points which encode less than τ boundary length or area units. Since all above-mentioned collapse metrics monotonically increase from the skeleton boundary ∂S to its center, thresholding them delivers a set of nested skeleton approximations, also called a multiscale skeleton. Importantly, these progressively simplified skeletons correspond, via the *MAT* (Sec. 7.2.1), to progressively simplified versions of Ω. More precisely, for all above collapse metrics, the reconstruction of a shape Ω from its simplified skeleton S_τ yields a shape

where all details of Ω of size smaller than τ have been replaced by circle arcs (in 2D) or spherical caps (in 3D) [82, 145, 198].

The idea of a mass collapse process from $\partial\Omega$ to S was also used by other works to define multiscale skeletons. Couprie [41] proposed a discrete framework for computing 2D skeletons and 3D curve skeletons by a guided thinning process which, for the 2D case, yields very similar results to [58, 198]. However, 3D surface skeletons are not covered by this approach. Torsello *et al.* propose a conservative mass advection $\partial\Omega$ onto S to define ρ in 2D [8], which was next extended to 3D [149]. However, the numerical computation of this process is affected by serious stability issues. Recently, Jalba *et al.* extended this advection model to compute multiscale 2D skeletons and 3D surface and curve skeletons in a unified formulation [84]. While this method delivers convincing results, it still suffers from numerical stability problems, and it is also relatively complex to implement.

Summarizing the above, we argue that multiscale regularization metrics are net superior, both in theory and practice, to local regularization metrics. However, as outlined, and discussed next in more detail in Sec. 7.3.1, multiscale regularization is far from being simple and cheap. Our proposal, presented next, aims at solving these problems.

7.3 PROPOSED METHOD

To compute multiscale 2D and 3D skeletons of binary shapes efficiently and robustly, we propose next to use the *Image Foresting Transform* (IFT) methodology [54]. We start, in Sec. 7.3.1, by introducing our general idea, which details the strengths and weaknesses of the MGF and advection-based regularization techniques introduced above in Sec. 7.2.2. Next, we detail the use of the IFT to compute skeletons that combine the identified strengths of these two approaches (Sec. 7.3.2). Our final proposed skeletonization algorithms are detailed in Sec. 7.3.3.

7.3.1 *Multiscale regularization – strengths and weaknesses*

If we analyze all multiscale skeletonization methods surveyed in Sec. 7.2 [8, 58, 82, 84, 124, 145, 149, 197, 198], we notice that their various importance-metric definitions can be all explained, at a high level, by introducing a vector field $\mathbf{v} : \Omega \rightarrow \mathbb{R}^d$, as follows: If we imagine that the input shape's surface $\partial\Omega$ is covered by uniformly-distributed mass with density $\rho(\mathbf{x} \in \partial\Omega) = 1$, then all above methods explain the importance $\rho(\mathbf{x} \in \Omega)$ of a spel $\mathbf{x}$ as the amount of mass transported, or advected, by $\mathbf{v}$ from $\partial\Omega$ to $\mathbf{x}$. This observation was also made in [83, 145]. Studying the properties of $\mathbf{v}$ brings several insights into multiscale regularization, as follows.

First, we note that the importance values of non-skeletal spels $\mathbf{x} \notin S$ should be low and nearly locally constant, so that upper-thresholding ρ by this value allows one to reliably separate S. All above-mentioned methods define $\rho(\mathbf{x} \notin S)$ to be equal to the importance of the single feature point of $\mathbf{x}$, *i.e.*, $\rho(\mathbf{x} \notin S) = \rho(FT(\mathbf{x}))$. This property is realized if we define $\mathbf{v} = \nabla DT$ for all $\mathbf{x} \notin S$, as it is well known that gradient lines of the distance transform only intersect at skeletal points [22, 175].

To fully define the multiscale importance ρ over Ω in terms of an advection process, what remains to be done is thus to define $\mathbf{v}$ on S. Studying the above-mentioned multiscale methods, and considering for now the case of connected genus-0 shapes, we see here that all such methods aim to define a field ρ which monotonically increases from ∂S to its center, or root $\mathbf{r} \in S$. As explained earlier, this allows easily computing multiscale skeletons S_τ by simply upper-thresholding ρ with desired values τ. In advection terms, this is equivalent to defining a field $\mathbf{v}$ which transports mass along S from its boundary ∂S to $\mathbf{r}$, along paths that finally meet at $\mathbf{r}$. For $d = 2$, mass flows from $\partial\Omega$ to the one-dimensional medial axis S, then along the branches of S to its center $\mathbf{r}$. For $d = 3$, mass flows from $\partial\Omega$ to the two-dimensional surface-skeleton S, then along S towards its local center (which is the curve skeleton of Ω), and then along the curve-skeleton branches towards the center $\mathbf{r}$ thereof.

The different multiscale importance listed above can be explained in terms of different definitions of $\mathbf{v}$ over S, as follows. In 2D, and for genus-0 shapes, all surveyed methods essentially reduce to defining $\mathbf{v}$ as being locally tangent to S and pointing towards the root of the skeleton (which is, in this case, a tree) [58, 124, 197, 198]. In 3D, explaining multiscale importance in terms of a field $\mathbf{v}$ defined on S is more complicated. We distinguish here two classes of methods, as follows.

MGF methods: For genus-0 shapes, MGF-based methods do not actually give a formal definition of $\mathbf{v}$, but compute $\rho(\mathbf{x} \in S)$ as the length of the longest shortest-path on $\partial\Omega$ between any two feature points $\mathbf{f}$ and $\mathbf{g}$ of $\mathbf{x}$, *i.e.*

$$\rho(\mathbf{x} \in S) = \max_{\mathbf{f} \in FT(\mathbf{x}), \mathbf{g} \in FT(\mathbf{x})} GL(\mathbf{f}, \mathbf{g}) \tag{7.5}$$

where $GL(\mathbf{x}, \mathbf{y})$ is the length of the shortest path on $\partial\Omega$ between two points $\mathbf{x} \in \partial\Omega$ and $\mathbf{y} \in \partial\Omega$. This is based on the empirical observation that Eqn. 7.5 defines a field ρ which smoothly and monotonically increases from ∂S to its center [45, 145]. This allows one to compute regularized surface skeletons even for noisy and complex 3D shapes. Another advantage of the MGF is that it makes simplification intuitive to understand – thresholding ρ at a value τ eliminates all spels from S where the local thickness of the shape is larger than τ. However, a formal justification of the MGF in terms of an advection process on S has not yet been given. A

second limitation of the MGF model is that is becomes very expensive to compute for large 3D shapes, where one needs to trace (at least one) shortest-path on $\partial\Omega$ for each point $\mathbf{x} \in S$, so its cost is $O(|S| \cdot |\partial\Omega| \log |\partial\Omega|)$. Using GPU techniques can accelerate this process [82], but also massively complicates the implementation of the method. This is illustrated by our own usage of the method in [82] in Chapter 6.

Advection methods: Aiming to solve the above issues with the MGF, Jalba *et al.* define $\mathbf{v}$ on S as the result of a mass-conserving advection model, with topological constraints used to ensure skeleton homotopy with the input shape [84]. In contrast to the MGF, computation of all types of skeletons (2D medial axes, 3D surface skeletons, and 3D curve skeletons) is now fully captured by a single unified advection model. Also, the method in [84] is considerably faster than MGF techniques, its cost being $O(|\Omega|)$, and is also simpler to implement. However, setting the simplification threshold τ is now less intuitive than for MGF techniques, as this value does not have an immediate geometric interpretation. Also, all 3D advection methods we are aware of [8, 84, 149], suffer various amounts of from numerical diffusion, which means that the computed importance ρ will deliver regularized skeletons S_τ having jagged boundaries. This is reflected by the various heuristics we needed to introduce in Chapters 3, 4, and 5 to reliably use skeletons computed with this method for practical applications in shape segmentation and matching.

The method that we propose in the next section aims to combine the strengths, and limit the disadvantages, of the MGF and advection methods outlined above. In detail: We use an incremental, advection-like, computation of the intuitive MGF metric, which makes our method considerably faster than existing MGF methods. We provide an implementation which is simple and does not suffer from numerical diffusion issues. Our proposal delivers smooth-boundary regularized, centered, one-spel-thin, multiscale skeletons, of the same overall quality as skeletons delivered by existing state-of-the-art methods.

7.3.2 *Image Foresting Transform*

The Image Foresting Transform (IFT) interprets d-dimensional images as graphs and reduces image operators to the computation of an *optimum-path forest* followed by a local processing of its attributes. In essence, the IFT is Dijkstra's shortest-path algorithm modified to use multiple sources and more general connectivity (path-value) functions. In our context, the IFT will propagate, from $\partial\Omega$ to $\mathbf{x} \in \Omega \setminus \partial\Omega$, both the feature points $FT(\mathbf{x})$ and the advection field $\mathbf{v}(\mathbf{x})$. Figure 7.3 shows in yellow the spels s with undefined $\nabla DT(\mathbf{x})$. These are the leaves of the optimum-path forest whose paths follow the direction of $\mathbf{v}$.

In order to design an image operator based on the IFT, one needs to specify which image elements (points, edges, regions) are the nodes of the graph, an *ad-*

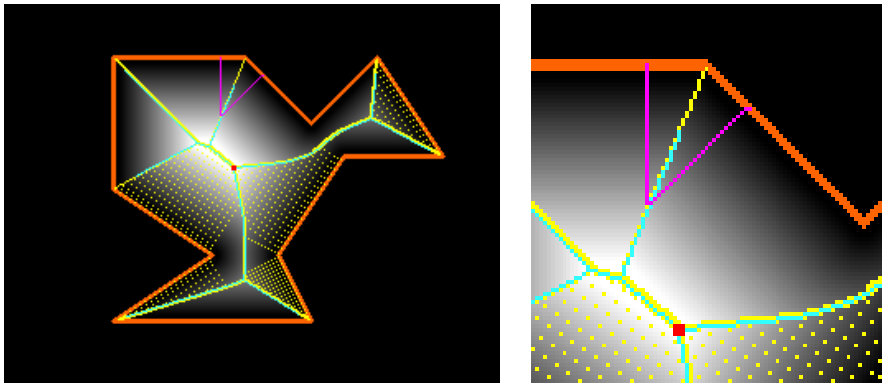


Figure 7.3: A polygon Ω with orange boundary $\partial\Omega$, its distance transform D (gray values), a plausible skeleton S (cyan), the root point r (red), the points with undefined $\nabla D(s)$ (yellow), and magenta lines connecting a given spel s with its feature points in $\partial\Omega$.

jacency relation between them, and a *connectivity function* that assigns a value (*e.g.*, strength, cost, distance) to any path in the graph. This methodology has been successfully used for boundary-based [52, 56, 118], region-based [54, 119], and hybrid image segmentation [31, 180]; connected filtering [57]; shape representation and description [5, 58, 154, 155]; and unsupervised [147], semi-supervised [4], and supervised data classification[129]. In this section, we show how the IFT can be adapted to propagate the single-point feature transform $F(\mathbf{x}) \in \partial\Omega, \forall \mathbf{x} \in \Omega \setminus \partial\Omega$ (Eqn. 7.4) and also to compute the length $GL(\mathbf{x},\mathbf{y})$ of the shortest-path on $\partial\Omega$ between two feature points $\mathbf{x},\mathbf{y} \in \partial\Omega$.

Graph definition: In the discrete space, the shape Ω is provided as a binary image $\mathbf{I} = (I, I)$, where $I \subset \mathbb{Z}^d$ is the image domain and each spel $\mathbf{x} \in I$ has a value $I(\mathbf{x}) \in \{0,1\}$. For instance, $\Omega \subset I$ may be the set of spels with value 1 and its complement $\overline{\Omega} = I \setminus \Omega$ be defined by the spels with value 0. The image $\mathbf{I}$ can be interpreted as a graph whose nodes are the spels in I and arcs are defined by the adjacency relation $A_{I,\delta}$,

$$A_{I,\delta} \quad = \quad \{(\mathbf{x},\mathbf{y}) \in I \times I \;\mid\; \|\mathbf{x}-\mathbf{y}\| \leq \delta\} \tag{7.6}$$

for a given value $\delta \in \mathbb{R}^+$. Let also $A_{I,\delta}(\mathbf{x})$ be the set of spels adjacent to a spel $\mathbf{x}$. The shape boundary $\partial\Omega$ is then defined by

$$\partial\Omega \quad = \quad \{\mathbf{x} \in \Omega | \overline{\Omega} \cap A_{I,1}(\mathbf{x}) \neq \emptyset\} \tag{7.7}$$

In order to compute the single-point feature transform F and the length $GL(\mathbf{f},\mathbf{g})$ of shortest paths between point-pairs $(\mathbf{f},\mathbf{g}) \in \partial\Omega \times \partial\Omega$, we constrain the adjacency relation $A_{I,\delta}$ to its subsets $A_{\Omega,\delta}$ and $A_{\partial\Omega,\delta}$, defined as

$$A_{\Omega,\delta} = \{(\mathbf{x},\mathbf{y}) \in \Omega \times \Omega \mid \|\mathbf{x}-\mathbf{y}\| \leq \delta\}, \tag{7.8}$$

$$A_{\partial\Omega,\delta} = \{(\mathbf{x},\mathbf{y}) \in \partial\Omega \times \partial\Omega \mid \|\mathbf{x}-\mathbf{y}\| \leq \delta\}, \tag{7.9}$$

respectively. Given the above, we are next interested in the graphs $(\Omega, A_{\Omega,\delta})$ and $(\partial\Omega, A_{\partial\Omega,\delta})$ which describe the shape's interior and boundary, respectively.

For the graph $(\Omega, A_{\Omega,\delta})$, let a *path* $\pi_{\mathbf{t}}$ be a spel-sequence $\langle \mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n = \mathbf{t} \rangle$ with terminal spel $\mathbf{t}$, such that $(\mathbf{x}_i, \mathbf{x}_{i+1}) \in A_{\Omega,\delta}$, $1 \leq i < n$. Let $\Pi\left(\Omega, A_{\Omega,\delta}\right)$ be the set of all paths in $(\Omega, A_{\Omega,\delta})$. A pair of spels is called *connected* in $(\Omega, A_{\Omega,\delta})$ when there exists a path in $\Pi\left(\Omega, A_{\Omega,\delta}\right)$ between them. A *connected component* in $(\Omega, A_{\Omega,\delta})$ is a subgraph, maximal for the inclusion, therein all pairs of spels are connected. The same definitions apply to the graph $(\partial\Omega, A_{\partial\Omega,\delta})$.

We next assume, with no generality loss, that the interior $\Omega \setminus \partial\Omega$ of Ω defines a single connected component in $(\Omega, A_{\Omega,1})$. When this is not the case, we simply treat each such connected component separately to yield a separate skeleton. To eliminate cases where Ω and $\Omega \setminus \partial\Omega$ have different numbers of components, *i.e.*, the removal of $\partial\Omega$ would disconnect the shape, one can preprocess Ω by *e.g.* morphological dilation with the distance of the size of one spel.

Objects with holes (in 2D) can be easily treated by merging the internal skeletons derived from each component in $(\partial\Omega, A_{\partial\Omega,\sqrt{d}})$ into a single one through the *skeleton by influence zones* (SKIZ) [58]. We will illustrate that for the 2D case, but since our examples of 3D objects do not present cavities, we will assume for simplicity (in 3D) that $\partial\Omega$ has a single connected component in $(\partial\Omega, A_{\partial\Omega,\sqrt{d}})$. Note also that the algorithm we next propose to compute the so-called interior skeleton of Ω can be also used to compute the so-called external skeleton of the complement $\overline{\Omega}$. For presentation simplicity, we focus here on the interior skeleton.

Distance and feature transforms: Using the graph $(\Omega, A_{\Omega,\sqrt{d}})$, we can propagate from $\partial\Omega$ the distance value $DT(\mathbf{x})$ to every interior spel $\mathbf{x} \in \Omega \setminus \partial\Omega$ by using the connectivity function

$$\begin{aligned} \psi_{edt}(\langle \mathbf{t} \rangle) &= \begin{cases} 0 & \text{if } \mathbf{t} \in \partial\Omega, \text{ and} \\ +\infty & \text{otherwise.} \end{cases} \\ \psi_{edt}(\pi_{\mathbf{s}} \cdot \langle \mathbf{s},\mathbf{t} \rangle) &= \|\mathbf{s} - F(\mathbf{s})\|, \end{aligned} \tag{7.10}$$

where $F(\mathbf{s}) \in \partial\Omega$ is the single-point feature transform of $\mathbf{s}$ (Eqn. 7.4) and $\pi_{\mathbf{s}} \cdot \langle \mathbf{s}, \mathbf{t} \rangle$ is the extension of the path $\pi_{\mathbf{s}}$ by the arc $(\mathbf{s}, \mathbf{t}) \in A_{\Omega,\sqrt{d}}$. The Euclidean distance transform DT thus becomes

$$DT(\mathbf{t} \in \Omega) \quad = \quad \min_{\forall \pi_{\mathbf{t}} \in \Pi\left(\Omega, A_{\Omega,\sqrt{d}}\right)} \{\psi_{edt}(\pi_{\mathbf{t}})\}. \tag{7.11}$$

Besides propagating the distance transform DT, the IFT method also propagates the closest point $F(\mathbf{x}) \in \partial\Omega$ to any $\mathbf{x} \in \Omega$. This yields the single-point feature transform F for all spels in Ω, which we will heavily use, as outlined next.

MGF importance: To compute ρ (Eqn. 7.5), we need the complete feature transform FT. As explained in Sec. 7.2.1, computing FT is difficult, especially for discrete-grid representations. In practice, this is often replaced by computing a so-called extended single-point feature transform FT_{ext} [145] defined as

$$FT_{ext}(\mathbf{s}) \quad = \quad \{F(\mathbf{t}) \in \partial\Omega \;\mid\; \mathbf{t} \in A_{\Omega,1}(\mathbf{s})\}, \tag{7.12}$$

which gathers the single-point feature transforms $F(\mathbf{t})$ of all adjacent spels $\mathbf{t} \in A_{\Omega,1}(\mathbf{s})$. We have also defined a particular type of EFT in Section 3.3.2.2.

Having FT_{ext}, we can now immediately write a simpler version of Eqn. 7.5 as

$$\rho(\mathbf{s} \in \Omega \setminus \partial\Omega) \quad = \quad \max_{\mathbf{f}=F(\mathbf{s}), \mathbf{g} \in FT_{ext}(\mathbf{s})} \{GL(\mathbf{f}, \mathbf{g})\}. \tag{7.13}$$

This simplification applies to multiscale planar and surface skeletons, leading to less shortest-path length computations, but Eqn. 7.5 is still important if one desires to merge 3D multiscale curve and surface skeletons, because the union of geodesic paths between all pairs of feature points of a spel $\mathbf{s}$ on the curve skeleton draws in $\partial\Omega$ a closed contour, splitting $\partial\Omega$ into two parts such that the geodesic surface area between them can be used as importance $\rho(\mathbf{s})$ [145].

To evaluate Eqn. 7.13, we compute the shortest-path length GL from $\mathbf{f} = F(\mathbf{s})$ to $\mathbf{g} \in FT_{ext}(\mathbf{s})$ on the boundary-graph $(\partial\Omega, A_{\partial\Omega,\sqrt{d}})$ by using the connectivity function ψ_{geo} defined as

$$\begin{aligned} \psi_{geo}(\langle \mathbf{w} \rangle) \quad &= \quad \begin{cases} 0 & \text{if } \mathbf{w} = \mathbf{f}, \text{ and} \\ +\infty & \text{otherwise,} \end{cases} \\ \psi_{geo}(\pi_{\mathbf{w}} \cdot \langle \mathbf{w}, \mathbf{h} \rangle) \quad &= \quad \psi_{geo}(\pi_{\mathbf{w}}) + \|\mathbf{h} - \mathbf{w}\| + \|\mathbf{g} - \mathbf{h}\|, \end{aligned} \tag{7.14}$$

where the term $\|\mathbf{g}-\mathbf{h}\|$ is the A^* heuristic optimization [76] used to reach $\mathbf{g}$ faster. Summarizing, we compute the shortest-path length $GL(\mathbf{f},\mathbf{g})$ as

$$GL(\mathbf{f},\mathbf{g}) \quad = \quad \min_{\pi_{\mathbf{g}} \in \Pi\left(\Omega, A_{\Omega,\sqrt{d}}\right)} \{\psi_{geo}(\pi_{\mathbf{g}})\}. \tag{7.15}$$

It is important to note that the IFT propagation for ψ_{edt} can also output an *optimum-path forest* P, i.e., a map that assigns a so-called predecessor $\mathbf{s} = P(\mathbf{t}) \in \Omega$ to every spel $\mathbf{t} \in \Omega \setminus \partial\Omega$, and a marker $P(\mathbf{t}) = nil \notin \Omega$ to spels $\mathbf{t} \in \partial\Omega$, respectively [54]. This defines a vector field $\mathbf{v}(\mathbf{t}) = \mathbf{t} - P(\mathbf{t})$ for every interior spel $\mathbf{t} \in \Omega \setminus \partial\Omega$. The forest P provides the direction of $\mathbf{v}$ for non-skeletal spels. The skeleton S is contained in the set of P's leaves (yellow lines in Figure 7.3). The vector field $\mathbf{v}$ describes how all information computed by the IFT – that is, DT, F, and ρ – is iteratively propagated, or *advected*, from $\partial\Omega$ to all spels in the shape's interior $\Omega \setminus \partial\Omega$. This fundamentally links the MGF importance model and the advection importance model. As explained in Secs. 7.2 and 7.3.1, these two importance models are typically used independently in the literature. The only work which we are aware of where an MGF model is linked with an advection model is [41]. However, our work here stands apart from [41] in terms of algorithmic model, and also by the fact that for the 3D case we compute multiscale *surface* skeletons (and not curve skeletons, while [41] approaches precisely the opposite).

Summarizing the above: To compute ρ (Eqn. 7.13), we need to compute the single-point distance transform F and the shortest-path length between feature points in the extended feature transform FT_{ext}. The algorithms for both above operations are described next in Secs. 7.3.2.1 and 7.3.3, respectively.

7.3.2.1 *Single-point feature transform*

The single-point feature transform F is computed by the same algorithm used for computing the Euclidean distance transform DT, but returns F rather than DT. The full algorithm we use for computing F is listed below. It also returns $\partial\Omega$, which we next need to compute shortest-paths between feature points, and a component label map $L_c\colon \mathbf{s} \in \Omega \rightarrow \lambda(\mathbf{s}) \in \{1,2,\ldots,c\}$, that assigns a subsequent integer number to each component of $\partial\Omega$ in $(\partial\Omega, A_{\partial\Omega,\sqrt{d}})$ and its closest spels in Ω. The map L_c is used for SKIZ computation in 2D. Indeed, the component label propagation to every spel $\mathbf{s} \in \Omega \setminus \partial\Omega$ is not needed, but it helps to illustrate the location of the SKIZ (Sec. 7.3.3).

Algorithm 1. – Single-Point Feature Transform

Input: An object Ω in dimension d represented on a uniform $\mathbb{Z}^d$ grid.
Output: The single-point feature transform F, object boundary $\partial\Omega$, and component label map L_c.
Auxiliary: Priority queue Q, distance transform DT, and variable $tmp \in \mathbb{R}$.

1. *Compute $\partial\Omega$ of Ω by Eqn. 7.7.*
2. **For each** $\mathbf{s} \in \Omega \setminus \partial\Omega$, $DT(\mathbf{s}) \leftarrow +\infty$.
3. **For each** $\mathbf{s} \in \partial\Omega$, **do**
4. $\quad DT(\mathbf{s}) \leftarrow 0$; $F(\mathbf{s}) \leftarrow \mathbf{s}$; $L_c(\mathbf{s}) \leftarrow \lambda(\mathbf{s}) \in \{1,2,\ldots,c\}$, *according to*
5. $\quad$ *its component in* $(\partial\Omega, A_{\partial\Omega,\sqrt{d}})$; *and insert* $\mathbf{s}$ *in* Q.
6. **While** $Q \neq \emptyset$, **do**
7. $\quad$ *Remove* $\mathbf{s}$ *from* Q, *where* $DT(\mathbf{s})$ *is minimal over* Q.
8. $\quad$ **For each** $\mathbf{t} \in A_{\Omega,\sqrt{d}}(\mathbf{s})$, *such that* $DT(\mathbf{t}) > DT(\mathbf{s})$, **do**
9. $\quad\quad tmp \leftarrow \|\mathbf{t} - F(\mathbf{s})\|^2$.
10. $\quad\quad$ **If** $tmp < DT(\mathbf{t})$, **then**
11. $\quad\quad\quad DT(\mathbf{t}) \leftarrow tmp$; $F(\mathbf{t}) \leftarrow F(\mathbf{s})$; $L_c(\mathbf{t}) \leftarrow L_c(\mathbf{s})$.
12. $\quad\quad\quad$ **If** $DT(\mathbf{t}) \neq +\infty$ **then**
13. $\quad\quad\quad\quad$ *Update position of* $\mathbf{t}$ *in* Q.
14. $\quad\quad\quad$ **Else**
15. $\quad\quad\quad\quad$ *Insert* $\mathbf{t}$ *in* Q.
16. **Return** $(F, \partial\Omega, L_c)$

Lines 1-5 essentially extract the object boundary $\partial\Omega$, initialize the trivial-path values of Eqn. 7.10 in $DT(\mathbf{s})$ for all $\mathbf{s} \in \Omega$, set $F(\mathbf{s})$ for $\mathbf{s} \in \partial\Omega$, assign a distinct integer to each component of $\partial\Omega$ in $(\partial\Omega, A_{\partial\Omega,\sqrt{d}})$, and insert $\mathbf{s} \in \partial\Omega$ in the priority queue Q. The main loop in Lines 6-15 propagates to every spel $\mathbf{t} \in \Omega \setminus \partial\Omega$ its single-point feature $F(\mathbf{t}) \in \partial\Omega$ in a non-decreasing order of distance values $DT(\mathbf{t})$ between $\mathbf{t}$ and $\partial\Omega$. In Line 7, when a spel $\mathbf{s}$ is removed from Q, $DT(\mathbf{s})$ stores the closest squared distance between $\mathbf{s}$ and $\partial\Omega$, $F(\mathbf{s})$ stores its single-point feature, and $L_c(\mathbf{s})$ indicates its closest component in $(\partial\Omega, A_{\partial\Omega,\sqrt{d}})$. The loop in Lines 8-15 evaluates if $\mathbf{s}$ can offer a lower squared distance value $\|\mathbf{t} - F(\mathbf{s})\|^2$ (value of an extended path $\pi_{\mathbf{s}} \cdot \langle \mathbf{s}, \mathbf{t} \rangle$ in Eqn. 7.10) to the current value assigned to an adjacent spel $\mathbf{t}$ in $DT(\mathbf{t})$ (Lines 9-10). If this is the case, then Line 11 updates distance, single-point feature of $\mathbf{t}$ with respect to $\partial\Omega$, its closest component in $(\partial\Omega, A_{\partial\Omega,\sqrt{d}})$, and Lines 12-15 update the status of $\mathbf{t}$ in Q.

Note that the use of the squared Euclidean distance $\|\mathbf{t} - F(\mathbf{s})\|^2$ in Line 9 allows to implement Q by bucket sorting [52], since all distances are integers on a pixel/voxel grid representation. As such, Algorithm 1 has average complexity $O(|\Omega|)$.

7.3.2.2 *Shortest-path length computation*

As mentioned earlier in Sec. 7.3.1, computing the multiscale regularization metric ρ for $d = 3$ heavily depends, cost-wise, on the rapid computation of shortest-

path lengths on $\partial\Omega$ between single-point features. Accelerating these shortest-path computations is key to accelerating multiscale 3D skeletonization. To achieve this, we maintain, for each $\mathbf{f} \in \partial\Omega$, a set $\mathscr{C}(\mathbf{f}) = \{\mathbf{s} \in \Omega \setminus \partial\Omega | F(\mathbf{s}) = \mathbf{f}\}$. We use $\mathscr{C}$ to *incrementally* compute all shortest-path lengths between $\mathbf{f}$ and other single-point features $\mathbf{g} \neq \mathbf{f}, \mathbf{g} \in \partial\Omega$, as follows: The IFT algorithm returns $GL(\mathbf{f},\mathbf{g})$ whenever $\mathbf{g}$ is reached; for any $\mathbf{h} \in \partial\Omega$ for which $GL(\mathbf{f},\mathbf{h}) \leq GL(\mathbf{f},\mathbf{g})$, we return immediately the already computed path-length $GL(\mathbf{f},\mathbf{h})$; and thus only continue computation for points $\mathbf{h} \in \partial\Omega$ where $GL(\mathbf{f},\mathbf{h}) \geq GL(\mathbf{f},\mathbf{g})$. To do the above, we store the computed shortest-path lengths $GL(\mathbf{f},\mathbf{g})$ (Eqn. 7.15) between a given feature point $\mathbf{f} \in \partial\Omega$ and all other spels $\mathbf{g} \in \partial\Omega$ into a map $L_{\mathbf{f}} : \partial\Omega \to \mathbb{R}^+, L_{\mathbf{f}}(\mathbf{g}) = GL(\mathbf{f},\mathbf{g})$. The computation of the shortest-path length between a given spel $\mathbf{f} \in \partial\Omega$ and all other spels $\mathbf{g} \in \partial\Omega$ is presented in Algorithm 4, Sec. 7.3.3.

For $d = 2$, the problem is trivial, since $\partial\Omega$ may consist of closed *one-dimensional* contours: For an arbitrary spel $\mathbf{f}_0$ in each contour $C \subset \partial\Omega$, we first compute in $L_{\mathbf{f}}(\mathbf{g})$ the path length from $\mathbf{f}_0$ to each spel $\mathbf{g} \in C$ while circumscribing C from $\mathbf{f}_0$ in a single orientation (clockwise or anticlockwise). Now, for any two spels $\mathbf{f}, \mathbf{g} \in C$, let $\Delta(\mathbf{f},\mathbf{g}) = |L_{\mathbf{f}}(\mathbf{g}) - L_{\mathbf{f}}(\mathbf{f})|$. The geodesic length $GL(\mathbf{f},\mathbf{g})$ between $\mathbf{f}$ and $\mathbf{g}$ is then given by

$$GL(\mathbf{f},\mathbf{g}) \quad = \quad \min\{|C| - \Delta(\mathbf{f},\mathbf{g}), \Delta(\mathbf{f},\mathbf{g})\}, \tag{7.16}$$

where $|C|$ is the perimeter-length of the contour C. However, for the purpose of finding one-spel-wide skeletons by Eqn 7.13, we can drop the absolute difference and redefine $\Delta(\mathbf{f},\mathbf{g}) = L_{\mathbf{f}}(\mathbf{g}) - L_{\mathbf{f}}(\mathbf{f})$ for $\mathbf{f} = F(\mathbf{s})$ and $\mathbf{g} \in FT_{ext}(\mathbf{s})$ on the boundary-graph $(\partial\Omega, A_{\partial\Omega,\sqrt{d}})$.

The next section presents the IFT-based multiscale skeletonization algorithms for $d = 2$ and $d = 3$, respectively.

7.3.3 *Multiscale skeletonization – putting it all together*

The complete 2D multiscale skeletonization algorithm is presented below.

Algorithm 2. – Multiscale skeleton computation in 2D

Input:	An object Ω in dimension $d = 2$.
Output:	Multiscale skeleton importance ρ.
Auxiliary:	Boundary $\partial\Omega$ with perimeter-length $\|\partial\Omega\|$; path length map $L_{\mathbf{f}}$; single-point feature transform F; component label map L_c; variable $tmp \in \mathbb{R}$.

1. $(F, \partial\Omega, L_c) \leftarrow$ Algorithm1(Ω).
2. ***For each*** *component* $C \in (\partial\Omega, A_{\partial\Omega,\sqrt{d}})$ ***do***
3. | *Select an arbitrary point* $\mathbf{f}_0 \in C$.
4. | ***For each*** $\mathbf{g} \in C$ *found by circumscribing* C *from* $\mathbf{f}_0$ ***do***
5. | | $L_{\mathbf{f}}(\mathbf{g}) \leftarrow$ *path length from* $\mathbf{f}_0$ *to* $\mathbf{g}$ *on* C.

6. ***For each*** $\mathbf{s} \in \Omega \setminus \partial\Omega$ ***do***
7. $\quad \rho(\mathbf{s}) \leftarrow 0.$
8. $\quad$ *Compute* $FT_{ext}(\mathbf{s})$ *by Eqn. 7.12.*
9. $\quad$ ***For each*** $\mathbf{g} \in FT_{ext}(\mathbf{s})$ ***do***
10. $\quad\quad$ ***If*** $L_c(\mathbf{g}) > L_c(F(\mathbf{s}))$ ***then*** *set* $\rho(\mathbf{s}) \leftarrow +\infty$ *and return to 6.*
11. $\quad\quad$ $tmp \leftarrow L_{\mathbf{f}}(\mathbf{g}) - L_{\mathbf{f}}(F(\mathbf{s}))$.
12. $\quad\quad$ ***If*** $tmp > |\partial\Omega| - tmp$ ***then*** $tmp \leftarrow |\partial\Omega| - tmp$.
13. $\quad\quad$ ***If*** $tmp > \rho(\mathbf{s})$ ***then*** $\rho(\mathbf{s}) \leftarrow tmp$.
14. ***Return*** ρ.

Line 1 finds the object boundary $\partial\Omega$, the single-point feature transform F, and the component label map L_c by Algorithm 1. The remaining lines follow the procedure described in Sec. 7.3.2.2 for $d = 2$. Lines 2-5 compute in $L_{\mathbf{f}}(\mathbf{g})$ the path length from an arbitrary spel $\mathbf{f}_0 \in C$, selected for each component $C \in (\partial\Omega, A_{\partial\Omega,\sqrt{d}})$, to every spel $\mathbf{g} \in C$, while circumscribing the contour C. The main loop in Lines 6-13 computes for each spel $\mathbf{s} \in \Omega \setminus \partial\Omega$ the shortest-path length by Eqn. 7.16 between point feature $F(\mathbf{s})$ and each $\mathbf{g} \in FT_{ext}(\mathbf{s})$ (Lines 11-12), and use them to update the MGF $\rho(\mathbf{s})$ in Line 13, as proposed in Eqn. 7.13. The SKIZ is detected whenever a point feature $\mathbf{g} \in FT_{ext}(\mathbf{s})$ comes from a distinct component than $F(\mathbf{s})$. For one-pixel-wide connected SKIZ, $\mathbf{s}$ is selected as belonging to the SKIZ whenever $L_c(\mathbf{g}) > L_c(F(\mathbf{s}))$. In this case, $\rho(\mathbf{s})$ is set to the maximum possible value and the algorithm returns to Line 6 (see example in Fig. 7.4). It should be clear that Algorithm 2 has complexity $O(|\Omega|)$.

A comment regarding the multiscale skeleton homotopy with the input shape is needed here. As visible from Fig. 7.4, the importance ρ has now a different variation across S than for genus-0 shapes (see *e.g.* Fig. 7.5). Clearly, for sufficiently high thresholds, the skeleton in Fig. 7.4 will get disconnected, *i.e.*, the three loops surrounding the holes in Ω will get separated from the central skeletal branch. Note that this *also* happens when using all other definitions of the same importance metric proposed by [58, 124, 198]. The root of the problem is that the collapsed-boundary importance metric used in all above works (and ours too) makes sense, in a multiscale way, only for genus-0 shapes whose skeleton is a tree. In other words: We know how to gradually simplify a tree (by removing its leafs), but we do not know how to do the same for a graph having loops. Issues here are how to assign an importance value to a loop (based on which geometric and/or topological criterion); and should the simplification of a loop remove it all at once, or should it allow its gradual disconnection. All these are (valid) questions which, however, go beyond our scope here.

For completeness, we note that disconnection of 2D non-genus-0 figures during simplification can be easily achieved, if this is a key issue. To do this, one can simply postprocess the computed skeleton S: Trace all shortest paths in S linking

each pair of loop-components in S, and assign spels along a value $+\infty$. This will only allow next the multiscale simplification of the tree parts of S.

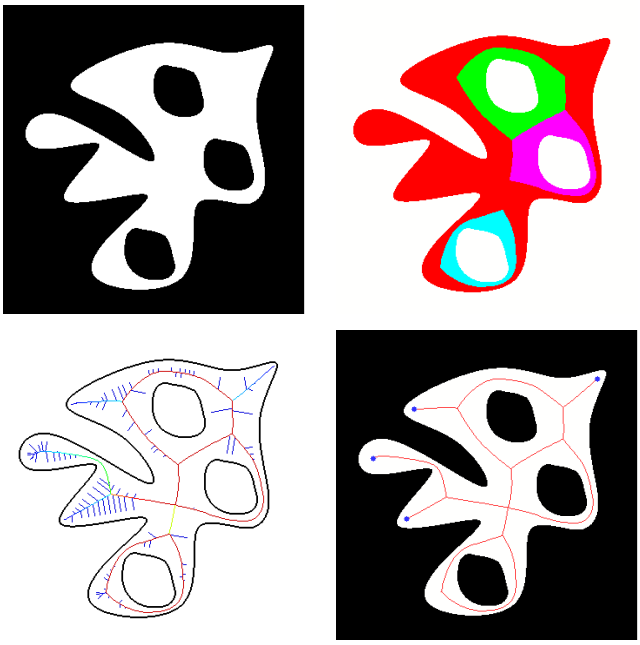


Figure 7.4: (a) A 2D object Ω with three holes. (b) The component label map L_c as computed by Algorithm 1. (c) The color-coded multiscale skeleton ρ of Ω, using the rainbow color map. The SKIZ is shown in red since its spels are assigned to the maximum importance in ρ. (d) A connected one-pixel-wide skeleton for a given scale of ρ, with its terminal points shown in blue.

Essentially, Algorithm 2 is identical to the methods presented in [58, 124, 198]. As such, its main added-value is of theoretical nature – showing that 2D multiscale skeletonization can be easily cast in the IFT framework.

The situation in 3D ($d = 3$) is however very different: Here, our proposed multiscale skeletonization is *both* conceptually similar to the 2D case, *and* very computationally efficient. This is in stark contrast with existing methods which are either similar in 2D and 3D, but quite complex and do not provide an *explicit* definition of the regularization metric [84]; or existing methods which provide strongly related metrics in 2D [58, 124, 198] and 3D [45, 145], but show a massive performance drop in the 3D case. The algorithm listed next shows our 3D multiscale skeletonization method. In contrast to the 2D proposal (Algorithm 2), we now use the efficient incremental shortest-path computation proposed in Algorithm 4.

Algorithm 3. – Multiscale skeleton computation in 3D

Input:	An object Ω in dimension $d = 3$.
Output:	Multiscale skeleton importance ρ.
Auxiliary:	Priority queue Q; list $\mathscr{V}$ of boundary points that have been inserted in Q; A^* path-cost map G; boundary $\partial\Omega$; sets $\mathscr{C}(\mathbf{s}), \forall \mathbf{s} \in \partial\Omega$; shortest-path length map $L_{\mathbf{f}}$; single-point feature transform F.

```
1.  (F, ∂Ω) ← Algorithm1(Ω).
2.  For each s ∈ Ω \ ∂Ω do
3.      Insert s in C(F(s)).
4.  For each f ∈ ∂Ω do
5.      L_f(f) ← +∞; G(f) ← +∞.
6.  Q ← ∅; V ← ∅.
7.  For each f ∈ ∂Ω do
8.      L_f(f) ← 0; G(f) ← 0; insert f in Q; insert f in V.
9.      While there exists s ∈ C(f) do
10.         Remove s from C(f).
11.         ρ(s) ← 0; compute FT_ext(s) by Eqn. 7.12.
12.         For each g ∈ FT_ext(s) do
13.             L_f(g) ← Algorithm4(∂Ω, g, Q, V, G, L_f).
14.             If L_f(g) > ρ(s) then ρ(s) ← L_f(g).
15.     For each g ∈ V
16.         L_f(g) ← +∞; G(g) ← +∞.
17.     Q ← ∅; V ← ∅.
18. Return ρ.
```

Line 1 finds the object boundary $\partial\Omega$ and the single-point feature transform F by Algorithm 1. We are not interested in L_c, since Algorithm 3 assumes that $\partial\Omega$ is a single surface. Lines 2-3 compute the sets $\mathscr{C}(\mathbf{f}) = \{\mathbf{s} \in \Omega \setminus \partial\Omega | F(\mathbf{s}) = \mathbf{f}\}$ that speed up shortest-path length computations, as described in Sec. 7.3.2.2 for $d = 3$. Note that G stores the A^* path costs, while $L_{\mathbf{f}}$ stores the desired path lengths, in Algorithm 4. The shortest-path lengths from each boundary point $f \in \partial\Omega$ to other boundary points $g \in \partial\Omega$ are *incrementally* computed in Algorithm 4 (Eqn. 7.15). Therefore, the trivial-path value initialization of ψ_{geo} (Eqn. 7.14) must be performed outside Algorithm 4 (Lines 4-5 before the main loop of Lines 7-17, and Lines 15-16 and 8 to restart computation for every initial boundary point $f \in \partial\Omega$). Lines 4-5 execute for the entire boundary $\partial\Omega$, so the purpose of set $\mathscr{V}$ is to revisit only the boundary points used in Algorithm 4, when reinitializing $L_{\mathbf{f}}$ and G. Line 8 initializes the priority queue Q and set $\mathscr{V}$ with one initial boundary point f for Algorithm 4. The loop of Lines 9-14 computes the 3D MGF $\rho(\mathbf{s})$ by Eqn. 7.13 for each spel $\mathbf{s}$ whose the single-point feature is the current point $\mathbf{f} \in \partial\Omega$. Line 10 removes a spel $\mathbf{s}$ from $\mathscr{C}(\mathbf{f})$, Line 11 initializes $\rho(\mathbf{s})$ and finds $FT_{ext}(\mathbf{s})$ by Eqn. 7.12. For each point feature $\mathbf{g} \in FT_{ext}(\mathbf{s})$, Line 13 finds $GL(\mathbf{f}, \mathbf{g})$ (Eqn. 7.15) and stores

it in $L_{\mathbf{f}}(\mathbf{g})$, and Line 14 updates $\rho(\mathbf{s})$ according to Eqn. 7.13. Algorithm 4 is presented next.

Algorithm 4. – Incremental shortest-path length computation

Input:	Boundary $\partial\Omega$; terminal node $\mathbf{g} \in \partial\Omega$; priority queue Q; boundary points $\mathscr{V}$ that have been inserted in Q; A^* cost map G; shortest-path-length map $L_{\mathbf{f}}$.
Output:	Shortest-path length $L_{\mathbf{f}}(\mathbf{g})$ at the terminal node with respect to the current starting node $\mathbf{f}$ chosen in Algorithm 3.
Auxiliary:	Variable $tmp \in \mathbb{R}$.

1. ***If*** $L_{\mathbf{f}}(\mathbf{g}) \neq +\infty$ ***then*** *return* $L_{\mathbf{f}}(\mathbf{g})$.
2. ***While*** $Q \neq \emptyset$ ***do***
3. ⎸ *Remove* $\mathbf{w}$ *from* Q*, where* $G(\mathbf{w})$ *is minimal over* Q.
4. ⎸ ***If*** $\mathbf{w} = \mathbf{g}$ ***then*** *return* $L_{\mathbf{f}}(\mathbf{g})$.
5. ⎸ ***For each*** $\mathbf{h} \in A_{\partial\Omega,\sqrt{d}}(\mathbf{w})$ *so that* $G(\mathbf{h}) > G(\mathbf{w})$ ***do***
6. ⎸ ⎸ $tmp \leftarrow L_{\mathbf{f}}(\mathbf{w}) + \|\mathbf{h} - \mathbf{w}\| + \|\mathbf{g} - \mathbf{h}\|$.
7. ⎸ ⎸ ***If*** $tmp < G(\mathbf{h})$ ***then***
8. ⎸ ⎸ ⎸ $G(\mathbf{h}) \leftarrow tmp$; $L_{\mathbf{f}}(\mathbf{h}) \leftarrow L_{\mathbf{f}}(\mathbf{w}) + \|\mathbf{h} - \mathbf{w}\|$.
9. ⎸ ⎸ ⎸ ***If*** $G(\mathbf{h}) \neq +\infty$ ***then***
10. ⎸ ⎸ ⎸ ⎸ *Update position of* $\mathbf{h}$ *in* Q.
11. ⎸ ⎸ ⎸ ***Else***
12. ⎸ ⎸ ⎸ ⎸ *Insert* $\mathbf{h}$ *in* Q *and in* $\mathscr{V}$.

Line 1 halts computation whenever the shortest-path length from $\mathbf{f}$ to $\mathbf{g}$ on $\partial\Omega$ has already been computed in a previous execution of Algorithm 4. The main loop of Lines 2-12 computes the shortest-path length to every boundary point $\mathbf{w} \in \partial\Omega$ in a non-decreasing order of the cost values in G, until it finds the terminal point $\mathbf{g}$ in Line 4. In Line 3, when a point $\mathbf{w} \in \partial\Omega$ is removed from Q, $G(\mathbf{w})$ stores the minimum A^* path cost and $L_{\mathbf{f}}(\mathbf{w})$ stores the shortest-path length from $\mathbf{f}$ to $\mathbf{w}$ on $\partial\Omega$, which may be used for early termination in Line 1 in a next execution of Algorithm 4. The loop in Lines 5-12 evaluates if $\mathbf{w}$ can offer a lower path cost $L_{\mathbf{f}}(\mathbf{w}) + \|\mathbf{h} - \mathbf{w}\| + \|\mathbf{g} - \mathbf{h}\|$ (value of an extended path $\pi_{\mathbf{w}} \cdot \langle \mathbf{w}, \mathbf{h} \rangle$ in Eqn. 7.14) to the current value assigned to an adjacent point $\mathbf{h} \in \partial\Omega$ (Lines 6-7). If this is the case, then Lines 8-12 update $G(\mathbf{h})$, $L_f(\mathbf{h})$, and the status of $\mathbf{h}$ in Q, accordingly.

The complexity of Algorithm 3 would be $O(|\Omega \setminus \partial\Omega||\partial\Omega| \log |\partial\Omega|)$ with a naive implementation of shortest-path length computation. In practice, however, Algorithm 4 finishes in Lines 1 or 4 much earlier than visiting all boundary points. This makes a considerable reduction in the processing time of Algorithm 3, as we will see next.

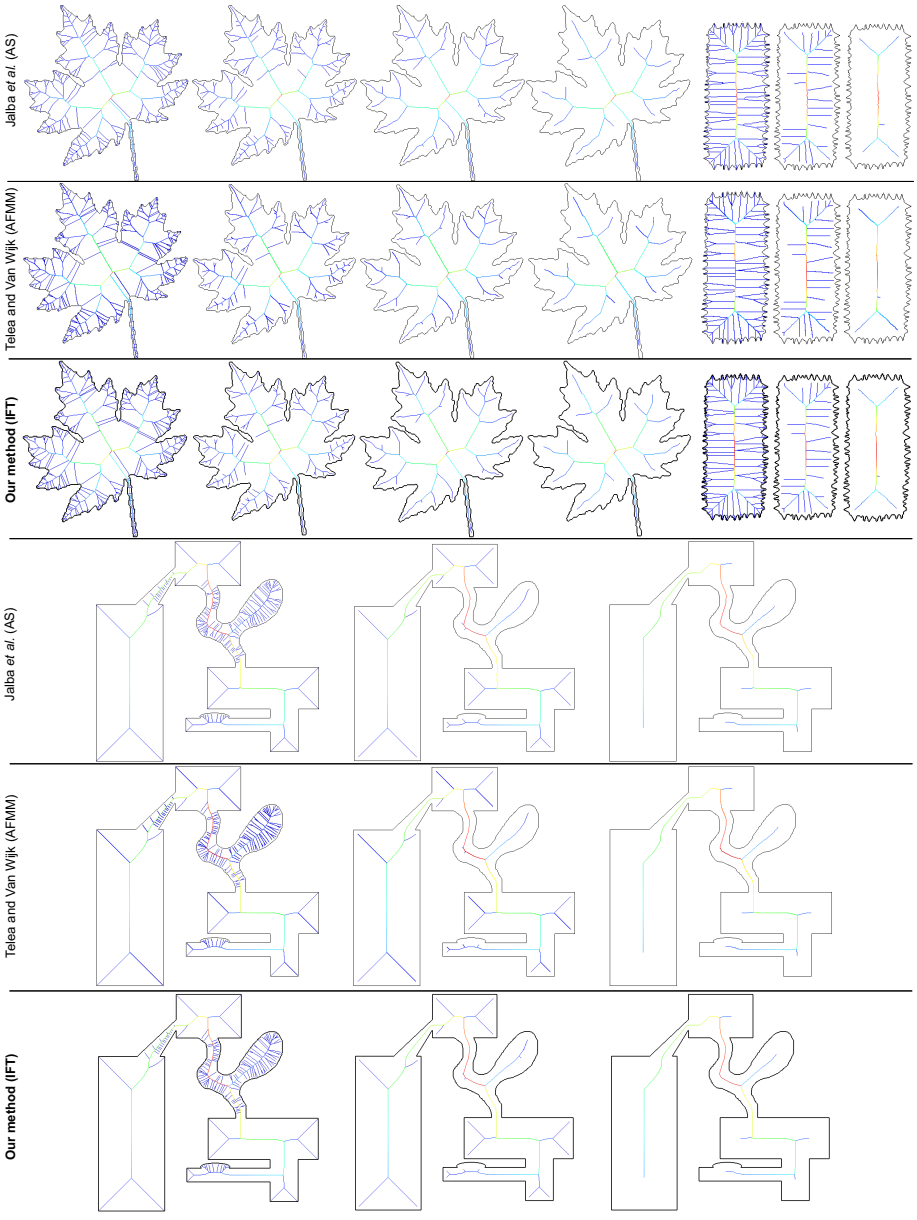


Figure 7.5: Our multiscale 2D skeletons (IFT) compared with AFMM and AS.

7.4 COMPARATIVE ANALYSIS

We next present and discuss our results as compared to other state-of-the-art multiscale skeletonization methods, as follows.

7.4.1 *2D medial axes*

We first consider medial axes of 2D objects. Here, we compare our IFT method with its two main competitors – the augmented fast-marching method (AFMM) [198] (basically identical to [58, 124]) and the more recent advection-based method (AS) in [84]. We compared the above three methods on a set of over 30 2D shapes, taken from relevant papers in the field [22, 124, 175, 198]. Figure 7.5 shows three such shapes, with their progressively simplified skeletons. It is clearly visible that all three methods yield nearly identical skeletons, both in terms of location *and* importance values. In other words, our IFT-based method can compute multiscale 2D skeletons which are nearly identical to existing methods. As visible, our method handles complex, noisy, variable-scale shapes with the same ease as the other two analyzed methods.

7.4.2 *3D medial surfaces*

7.4.2.1 *Global comparison*

For 3D shapes, we compared our IFT-based methods with two classes of competing techniques. First, and most interesting, we considered all techniques that we are aware of that produce multiscale skeletons, in the sense described in Sec. 7.2.2. These are the multiscale MGF-based method in [145] (MS); the advection-based method in [84] (AS); and the multiscale ball-shrinking method that implements the MGF metric in [145] for mesh models [82] (MBS). Secondly, to illustrate the advantage of multiscale regularization, we compare our method with three local regularization, non-multiscale, methods: Hamilton-Jacobi skeletons (HJ [175]), the Integer Medial Axis (IMA [148]), and Iterative Thinning Process (ITP [86]). We have chosen these methods as they are well-known in the 3D skeletonization arena, are relatively efficient, produce good-quality 3D surface skeletons, and have public implementations.

Figure 7.6 shows the results of the above-mentioned comparisons for 7 shapes, processed by 7 skeletonization methods. The multiscale skeletons computed by MS and AS are color-coded to reflect the importance metric, using a rainbow colormap, just as in Fig. 7.5. Multiscale skeletons computed by MBS are not importance color-coded in Fig. 7.6 – the MBS importance is discussed separately in more detail in Sec. 7.4.2.2.

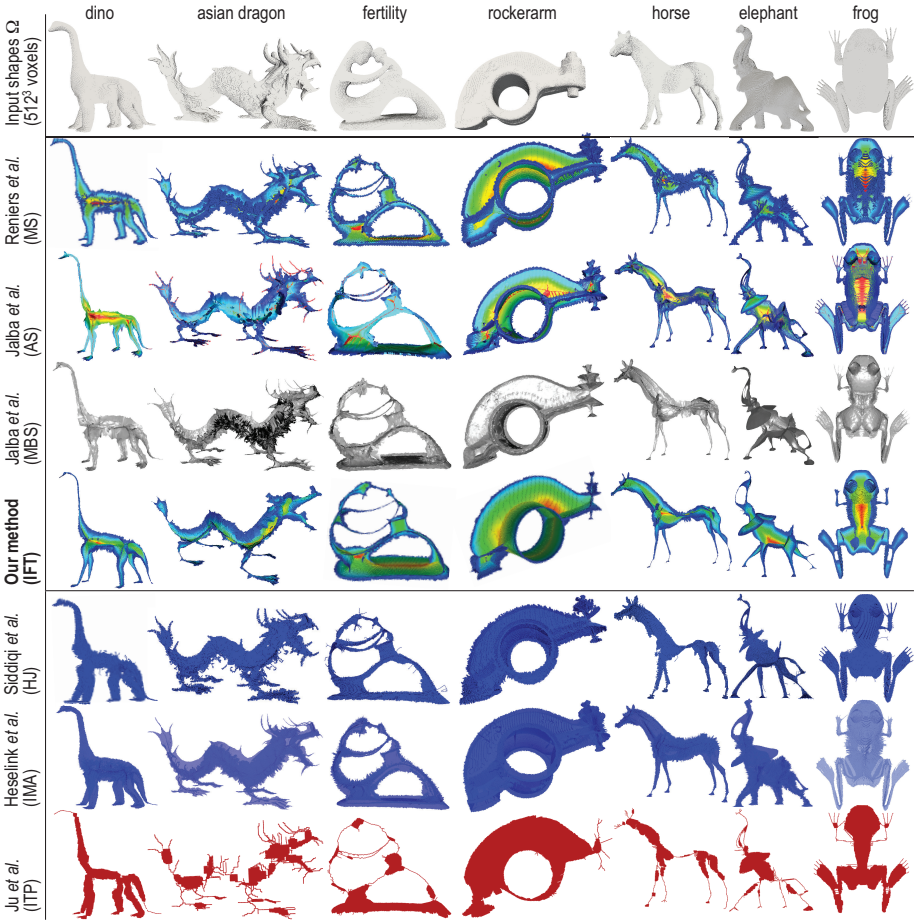


Figure 7.6: Global comparison of our 3D skeletonization method (IFT) with three multiscale skeletonization methods (MS, AS, MBS) and with three additional non-multiscale methods (HJ, IMA, ITP). See Sec. 7.4.2.1.

Quality-wise, our 3D surface skeletons are voxel-thin, centered within the shape (within the margin allowed by the voxel resolution), and have the same number of connected components and loops as the input shape, by construction. These are key properties, required by any skeletonization method [37]. For example, IMA yields centered and voxel-thin skeletons, but these can get disconnected when simplified too much, as this method essentially uses the local angle-and-distance based simplification criterion of the θ-SMA method of Foskey *et al.* [65]. Note that the disconnection implied above is not due to the existence of loops in the skeleton – IMA can easily disconnect also skeletons of genus-0 shapes. This does not happen with our method. Conversely, HJ yields connected skeletons, but for this the method uses a thinning process ordered by the divergence of the distance transform's gradient which must explicitly check to preserve homotopy [175]. The ITP method computes skeletons which are voxel-thin and homotopic to the input shape, but not well centered in the shape, as seen by the various zig-zag branches of the *dragon* model (Fig. 7.6, bottom row).

The sensitivity of the skeletons shown in Fig. 7.6 to noise or small-scale details on the input shape surface varies quite a lot. As known, local regularization methods such as HJ, IMA, and ITP are more noise-sensitive than global regularization methods such as MS, AS, and MBS [189]. Our method (IFT) falls in the latter class of global methods, so it is less sensitive to noise and produces smoother surface skeletons, as visible in Fig. 7.6, fourth row from bottom.

7.4.2.2 *Detailed comparison*

To gain more insight, we next compare our IFT method with several methods we are aware of that compute *multiscale* 3D surface skeletons (AS, MS, and MBS). The first two methods (AS, MS) are voxel-based, while the last one (MBS) is mesh-based. Figures 7.7 and 7.8 show results for a selected set of shapes. Since all above-mentioned methods produce multiscale skeletons, we regularized these by removing very low importance (spurious) skeleton points to yield comparably simplified skeletons. Several observations can be made when studying the compared methods, as follows.

Regularization: Figures 7.7 and 7.8 show that IFT delivers 3D surface skeletons which are, geometrically speaking, very similar to the ones produced by AS, MS, and MBS. This, in itself, is a good indication of quality of IFT. Indeed, surface-skeletonization methods should deliver similar results, given that they all aim to approximate the *same* surface skeleton definition (Eqn. 7.2). Secondly, we see that the IFT delivers the same degree of small-scale noise removal to create smooth and clean skeletal manifolds as AS, MS, and MBS, so it can be used for robust skeleton regularization. The IFT regularization is as easy to use as the one proposed by the

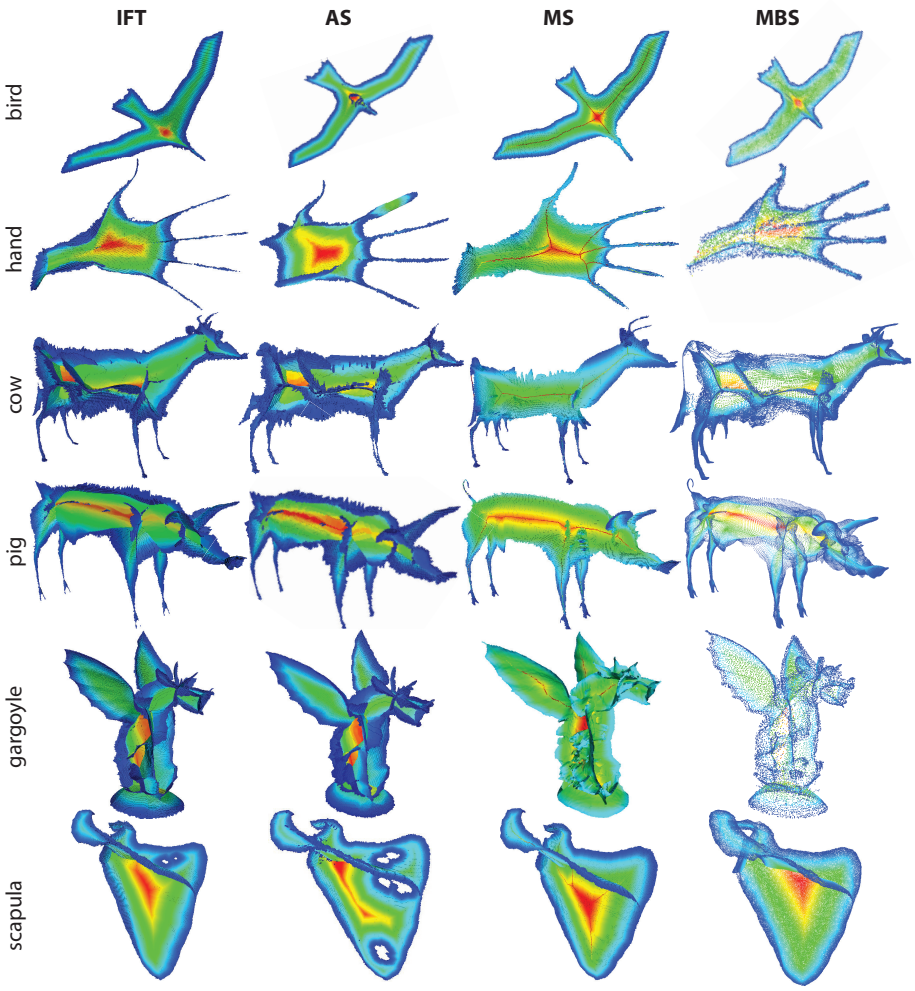


Figure 7.7: Detailed comparison of 3D surface skeletons computed by our method (IFT) and other multiscale methods (MS, AS, and MBS). See Sec. 7.4.2.2.

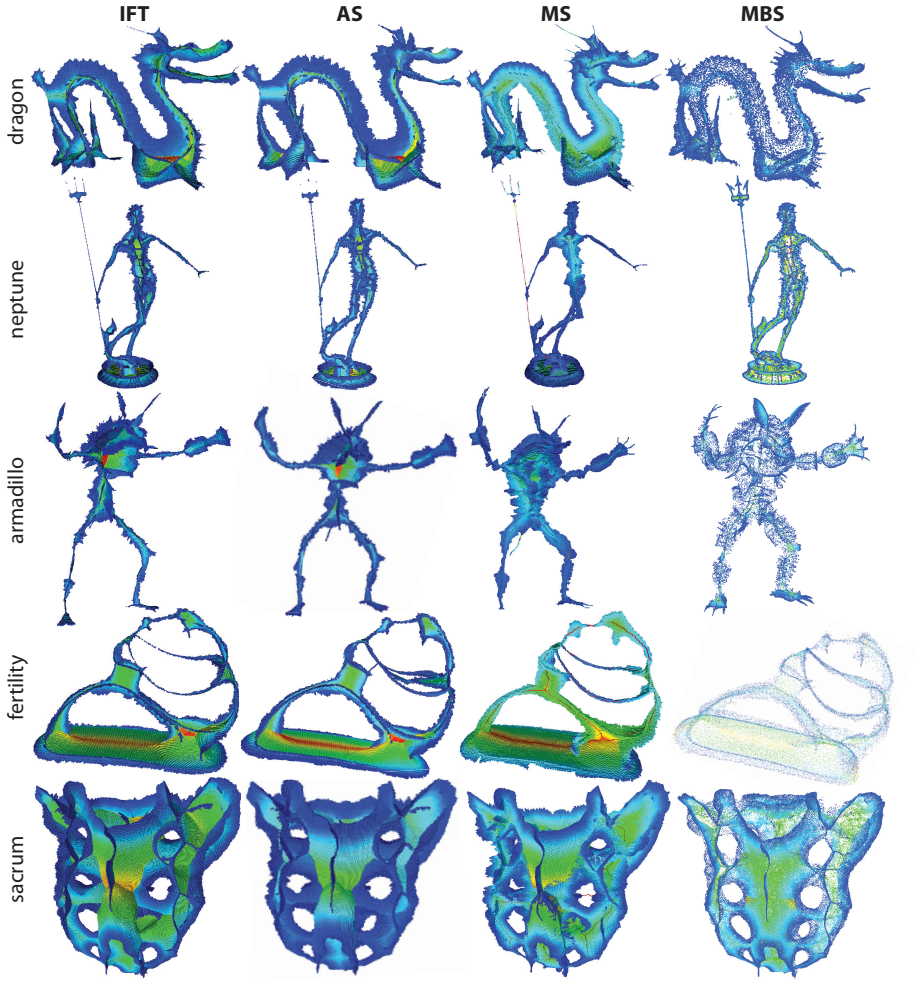


Figure 7.8: Additional examples of 3D multiscale skeletons computed by our method (IFT) and other multiscale methods (MS, AS, and MBS). See Sec. 7.4.2.2.

other methods – the setting of a single importance thresholding parameter τ. Note that this is far simpler than the regularization proposed by local methods, *e.g.* HJ, IMA, or ITP, which require the careful setting of one or several parameters to obtain comparable results.

A more subtle insight regards the gradient of the importance metric ρ from the skeleton boundary ∂S to its center, visible in Figs. 7.7 and 7.8 in terms of the blue-to-red color change. All tested methods (IFT, AS, MS, MBF) yield a ρ that increases monotonically from ∂S to the center of S. Separately, we see that ρ for IFT, MBS, and MS is not just increasing from ∂S to the center of S, but has a very similar gradient. This implies that our method (IFT) delivers an importance metric ρ which is very similar to the ones delivered by MS and MBS. Since MS and MBS compute the medial geodesic function (MGF) metric, it follows that IFT also computes a very similar metric. This is indeed the expected outcome given the IFT algorithm (see Sec. 7.3.3). In contrast, the gradient of ρ delivered by AS is quite different. This is explained by the fact that AS is the only multiscale skeletonization method in the studied set that does not explicitly use the MGF metric.

Dataset	$\lvert\Omega\rvert$	**MS** [145]	**AS** [84]	**IFT**	$\lvert\partial\Omega\rvert^{mesh}$	**MBS** [82]
bird	445690	64.21	13.86	8.64	46866	18.69
hand	776869	62.94	2.07	4.36	49374	15.5
cow	6343478	177.80	39.86	17.73	181823	96.54
pig	5496145	181.95	34.84	23.89	225282	142.02
gargoyle	6614154	566.52	25.66	79.43	25002	7.54
scapula	2394694	1717.37	29.99	609.33	116930	102.57
dragon	7017452	322.81	39.3	32.86	100250	49.01
neptune	2870546	322.75	47.25	68.72	28052	5.85
armadillo	1854858	45.43	7.2	4.25	172952	104.65
fertility	1264132	99.62	6.15	8.46	24994	6.15
sacrum	12637931	2015.59	39.83	417.54	204710	213.49

Table 7: Timings (seconds) for the compared surface skeletonization methods.

Connectivity: IFT, AS, and MS deliver a compact surface skeleton, while MBS delivers only a disconnected point cloud. This makes IFT (and AS and MS) more interesting than MBS for practical applications where one requires a compact surface skeleton. Indeed, point-cloud skeletons require complex post-processing methods for reconstructing a compact representation [82, 96]. Voxel skeletons do

not have this problem.

Scalability: We implemented all tested methods in C++ and run them on an Intel 3.5 GHz 8-core 32MB RAM PC. The methods MBS, AS, and MS use CPU multithreading parallelization, as described in the respective papers. No GPU parallelization was used for MBS. Our method (IFT) is purely serial. Table 7 shows the timings for the compared methods for the shapes depicted in Figs. 7.7 and 7.8. Column $|\Omega|$ gives the number of foreground voxels of the tested models with MS, AS, and IFT. For MBS, the comparable metric – number of sample points of the input mesh – is given in column $|\partial\Omega|^{mesh}$.

When testing scalability on large voxel volumes, we found that the MS implementation from [145] encountered problems: For the *scapula* shape (Fig. 7.7, bottom row), MS could not handle the 512^3 voxel resolution of our model, so we reduced the resolution to 370^3. At this resolution, the shape shows visible holes, due to the very thin wall thickness (a few voxels). In contrast, IFT and AS (which are both voxel-based methods) could handle 512^3 voxel volumes without problems.

Performance-wise, Table 7 shows that IFT is roughly 3 to 10 times faster than MS, which is the only voxel-based method which implements the same MGF importance metric. This is an important result, as it tells us that the IFT algorithm produces significant speed-ups for the geodesic length evaluation, which was one of its main goals. Compared to AS, IFT is faster on some models, but considerably slower on the *sacrum* and *scapula* models. This is explained by the fact that the complexity of AS is roughly $O(K|\partial\Omega|\log|\partial\Omega|)$, where K is $\max_{\mathbf{x}\in\Omega} DT(\mathbf{x})$, *i.e.* the shape thickness. In contrast, the complexity of MS is roughly $O(L|\partial\Omega|)$, where K is the average geodesic-path length between two feature points on $\partial\Omega$. For large and locally-tubular shapes, such as *cow* or *pig*, IFT is thus faster. For relatively thin and large-surface shapes, such as *scapula* and *sacrum*, the geodesic computation cost becomes very high, so IFT is slower than AS. However, as outlined earlier, this extra price of IFT delivers a higher-quality regularization in terms of smoothness of the importance metric. We note a similar effect when comparing IFT with MBS – for locally tubular shapes, IFT is faster than MBS, especially when the latter considers high-resolution mesh models. For locally thin and large-surface shapes, IFT becomes slower than MBS. Again, this extra price of IFT is counterbalanced by the higher-quality regularization metric it delivers, and also by the fact that IFT delivers connected skeletons, whereas MBS delivers only a skeletal point-cloud. All in all, we argue that the performance of IFT compares favorably with methods using the same importance metric (MS) but also with other multiscale skeletonization methods (AS, MBS). This is especially salient when considering that we implemented IFT as a purely *serial* algorithm, while MS, AS, and MBS all use CPU-side 8-core multithreading parallelization.

7.5 EXTENSIONS

As discussed so far, the IFT-based method we proposed can compute skeletons of planar 2D shapes and surface skeletons of volumetric 3D shapes by using the same formulation. Given this ability of our method to generalize between 2D and 3D, it is interesting to see whether the generalization can be pushed further.

One possible direction for generalization is to consider the computation of *curve skeletons*. As introduced in Sec. 2.1.2.2, these are one-dimensional curvilinear structures which are locally centered within a 3D shape. Related to the shortest-path concept used in the IFT skeletons, several authors define the curve skeleton of a shape Ω as the locus of points $\mathbf{x}$ in Ω which admit two equal-length, but different, shortest paths γ_1 and γ_2 between two feature points $\mathbf{f}_1$ and $\mathbf{f}_2$ of $\mathbf{x}$ [45, 82, 145]. Multiscale skeletons can be next defined for genus 0 shapes atop this curve-skeleton definition by assigning to each curve skeleton point $\mathbf{x}$ an importance equal to the area of the smallest surface component on $\partial\Omega$ delimited by the closed shortest-path ring formed by γ_1 and γ_2 [82, 145]. Given that our IFT method efficiently computes such shortest paths, it is thus attractive to consider its extension to compute such multiscale curve skeletons. Although we believe this should be efficiently possible, we have not explored this avenue further. One current blocker is finding a way to efficiently compute the area of the above-mentioned surface components. In [145], this is done by performing a flood fill on $\partial\Omega$ after each ring $\gamma_1 \cup \gamma_2$ has been computed for each curve skeleton point $\mathbf{x}$. This, however, as the authors also note, is quite slow, roughly $O(|\partial\Omega|^2)$. We believe that our IFT algorithm should be modifiable in a way that such areas can be computed incrementally, much as we compute shortest paths incrementally. However, a complication here is that shortest paths γ_i between feature points corresponding to different curve skeleton points $\mathbf{x}_i$ may partially overlap on $\partial\Omega$, more specifically in saddle areas of $\partial\Omega$. As such, a scheme is needed to acount for such overlaps, so that one does not count the same voxels on $\partial\Omega$ multiple times when evaluating areas. Given the above difficulties, we have not pursued this extension direction further.

Another equally (or even more) interesting direction is to consider the computation of skeletons of *curved surfaces*. In detail: consider a curved manifold with boundaries Ω embedded in $\mathbb{R}^3$, such as *e.g.* a twisted sheet of paper. Since this shape has a boundary $\partial\Omega$, we could define a skeleton for it by adapting definition used for 2D planar-shape skeletons and 3D surface skeletons, following Eqn. **??**, which we repeat below for reading convenience:

$$S_{\partial\Omega} = \{\mathbf{x} \in \Omega | \exists \{\mathbf{f}_1, \mathbf{f}_2\} \subset \partial\Omega, \mathbf{f}_1 \neq \mathbf{f}_2, \|\mathbf{x}-\mathbf{f}_1\| = \|\mathbf{x}-\mathbf{f}_2\| = DT_{\partial\Omega}(\mathbf{x})\}.$$

The key difference with the 2D planar case and the 3D volumetric case is that we cannot use the Euclidean 2D, respectively 3D, distance to define $\|\cdot\|$ in Eqn. **??**.

However, we can use the *geodesic* distance $\delta : \Omega \times \Omega \rightarrow \mathbb{R}^+$ which, given two points $\mathbf{x}_1$ and $\mathbf{x}_2$ located on the curved surface Ω, sets $\delta(\mathbf{x}_1, \mathbf{x}_2)$ to the length of the shortest path $\gamma \subset \Omega$ that connects $\mathbf{x}_1$ and $\mathbf{x}_2$. Accordingly, we define a feature point $\mathbf{f} \in \partial\Omega$ of a point $\mathbf{x} \in \Omega$ as a point for which the distance $\delta(\mathbf{f}, \mathbf{x})$ is minimal. This yields a definition of the skeleton of the curved surface Ω as

$$S_{\partial\Omega} = \{\mathbf{x} \in \Omega | \exists \{\mathbf{f}_1, \mathbf{f}_2\} \subset \partial\Omega, \mathbf{f}_1 \neq \mathbf{f}_2, \delta(\mathbf{x}, \mathbf{f}_1) = \delta(\mathbf{x}, \mathbf{f}_2) = DT_{\partial\Omega}(\mathbf{x})\}. \quad (7.17)$$

Here, $DT_{\partial\Omega} : \Omega \rightarrow \mathbb{R}^+$ is, by analogy with Eqn. 2.2, the distance transform of the curved boundary $\partial\Omega$, *i.e.*

$$DT_{\partial\Omega}(\mathbf{x} \in \Omega) = \min_{\mathbf{y} \in \partial\Omega} \delta(\mathbf{x}, \mathbf{y}). \quad (7.18)$$

Equations 7.17 and 7.18 give us a complete framework for curved surface skeletonization. We call the resulting skeletons *curved skeletons*, to distinguish them from all other skeleton types we discussed so far (skeletons of 2D planar shapes, surface skeletons of 3D shapes, and curve skeletons of 3D shapes).

If we study the properties implied by Equations 7.17 and 7.18, we see that they are very similar to those inferred by their classical counterparts (Equations 2.2 and 2.1, respectively): The distance transform is zero on $\partial\Omega$ and monotonically increases as one advances further in the shape from its boundary. Skeletons are located at the points where $DT_{\partial\Omega}$ is locally maximal. The gradient $\nabla DT_{\partial\Omega}$ points away from the boundary $\partial\Omega$ into the shape. Moreover, the classical Blum definition of skeletons as loci of maximally inscribed disks holds also for our curved skeleton case – the only difference now is that such a disk is a curved one, living on a curved surface.

Given the above, it is thus interesting to study how such curved skeletons can be computed so that they satisfy the desirable properties known for their classical counterparts (see Sec. 2.1.2.3). To this end, we propose next a computation method for curve skeletons using a voxel shape representation, and discuss its results and properties.

Surface definition: To proceed, we need a way to define curved surfaces Ω. Since we work with voxel models, we designed, for experimentation purposes, a simple way to achieve this. Given a volumetric shape Ω^{vol}, we compute its intersection with a cylinder whose spatial orientation and radius can be interactively controlled by the user. This allows quickly defining a rich family of 3D curved surfaces as the part of the boundary $\partial\Omega^{vol}$ which is located inside the cylinder. Figure 7.9 shows an example. Here, we slice a hand model with a cylinder whose axis is oriented roughly orthogonally to the palm's surface. As expected, we get a curved surface

consisting of two disk-like components (only one is visible in the figure, the other one is located on the invisible part of the hand).

Boundary definition: To apply Equations 7.17 and 7.18, we next need a way to define the boundary $\partial\Omega$ of our curved surface. Several so-called template-based detectors could be used to this end to find the voxels of Ω which meet specific properties that make them part of the boundary [164]. We have tried several such detectors, but none has shown to deliver a connected, voxel-thin, boundary. As such, we propose a new boundary detector, as follows: A voxel $\mathbf{x} \in \Omega$ of a curved voxel surface Ω is part of the boundary if one cannot construct a closed voxel-loop that does not pass through $\mathbf{x}$ and resides in the 26-neighbors of $\mathbf{x}$ located on Ω. This detector finds, in general, connected boundaries $\partial\Omega$ for the connected components of Ω. However, sometimes small gaps of a size of a few voxels appear on the boundary, as certain voxels are incorrectly classified as interior. To patch these, we connect boundary voxels at both sides of the gap by tracing shortest paths over Ω between them. This method delivers good, pixel-thin, and connected boundaries for the large majority of shapes we have tested (although, some gaps can still exist in corner cases). As an example, Figure 7.9c shows the boundary of the disk-like surface in Fig. 7.9b.

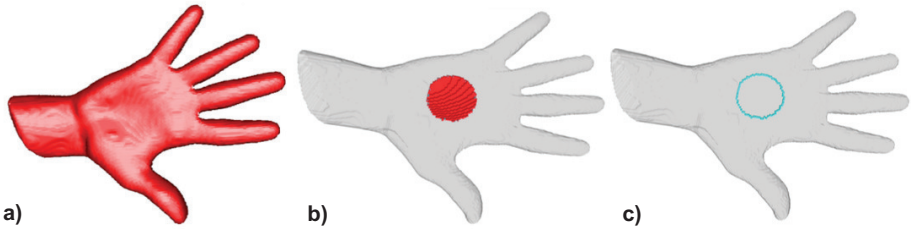


Figure 7.9: Construction of curved surfaces. a) Typical volumetric shape. b) Intersection of the shape surface with a cylinder yields a curved surface. c) Boundary of the curved surface.

Skeletonization: Having a boundary $\partial\Omega$ of our curved shape, we can now compute its curved skeleton. For this, we proceed as follows. For each point $\mathbf{x} \in \Omega$, we trace the shortest path to the closest boundary point $\mathbf{f} \in \partial\Omega$, which is a feature point of $\mathbf{x}$, by definition. This can be easily done using Dijkstra's algorithm. Next, we repeat the procedure to find all feature points $\mathbf{f}_i$ of the 26-neighbors $\mathbf{x}_i$ of $\mathbf{x}$ which are located in Ω. We then set the importance $\rho(\mathbf{x})$ to the length of the shortest path along $\partial\Omega$ formed by all pairs $(\mathbf{f}_i, \mathbf{f}_j)$. Note that this is essentially a generalization of the Augmented Fast Marching Method (AFMM) used earlier to compute skeletons of 2D planar shapes [198]. However, our approach has an important added value for the context of curved surfaces: If we were to compute

ρ as in the AFMM, we would need to construct a labeling of all voxels along ρ in arc-length-distance from some starting voxel on $\partial\Omega$. This is doable only for boundaries which have the same topology as a non-intersecting loop. However, in general, curved surfaces can have a more complex boundary topology – think, for instance, of the curved surface that is the surface skeleton of a 3D shape. For a box, for instance, this boundary is formed by six loops, one for each face of the box. Our importance computation can handle such cases, while the simple AFMM labeling scheme cannot.

The importance field ρ over all voxels in Ω defines the multiscale curved skeleton of Ω. That is, we an extract a curved skeleton of Ω at a given scale τ by thresholding, *i.e.*,

$$S^{\tau}_{\partial\Omega} = \{\mathbf{x} \in \Omega | \rho(\mathbf{x}) \geq \tau\}. \tag{7.19}$$

Note that this is in perfect analogy with the way multiscale 2D skeletons are computed by thresholding other importance metrics (Sec. 2.1.2.1).

Figure 7.10 shows curved skeletons for several curved surfaces obtaining by the above-mentioned slicing procedure from two 3D volumes. As visible, the curved skeletons share the same properties as classical 2D planar skeletons: They are connected, voxel-thin, locally centered in the shape, and capture the topology of the shape. Concerning the last point, indeed, if we look at Fig. 7.10b, we see that the boundary $\partial\Omega$ consists of two disconnected components – a large loop surrounding an isolated point. As such, the corresponding curved skeleton has a loop thar surrounds the isolated boundary point. However, for curved surfaces, the situation is more subtle than for 2D planar shapes: The fact that the boundary $\partial\Omega$ consists of several disconnected components does *not* mean the skeleton will always have loops. The example in Fig. 7.10f shows this. Here, the curved surface is a cylinder-like shape (without caps), having two rings as boundary. The curved skeleton of this shape consists of a single ring.

Figure 7.11 shows several additional aspects. Image (b) in that figure shows the actual importance field ρ computed for the shape in image (a), color coded using a rainbow colormap. Low values (blue) show points of zero importance, which have a single feature point on $\partial\Omega$. By definition, these are non-skeletal points. Warmer colors show skeletal points. As visible, the importance increases gradually from the tips of the skeleton branches to its center. However, we also see four salient red branches that extend all the way to the shape's boundary. When thresholding this skeleton, these branches remain, as shown in image (c). Upon examination, we found that these high-importance spurious branches are caused by very small disconnections along the voxel boundary $\partial\Omega$. These are, in turn, due to the imperfection of our boundary detector discussed above.

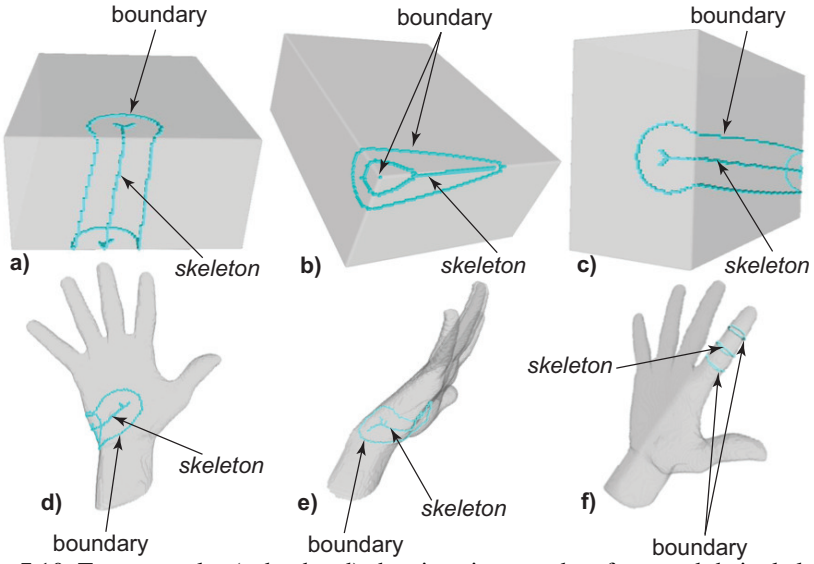


Figure 7.10: Two examples (cube, hand) showing six curved surfaces and their skeletons. The boundary of each curved surface, respectively its curved skeleton, are shown.

Concluding, we have shown an extension of the concept of skeletons (and related concepts such as feature points and distance transforms) for curved surfaces embedded in $\mathbb{R}^3$. Such curved skeletons can be defined analogously to their classical 2D and 3D counterparts, and can be computed in a multiscale way using shortest-path tracing. Since the IFT delivers a general framework for efficiently computing such shortest paths, the curved skeletonization problem can be cast in this framework, yielding thus a unification of the computation of 2D planar skeletons, skeletons of curved surfaces, and 3D surface skeletons of volumetric shapes. However, to compute curved skeletons, we need a robust way to detect the boundary of a curved voxel surface. Although we have proposed such a detector, its robustness is not sufficient to treat all types of curved surfaces.

This experiment opens some interesting avenues. First, if the aforementioned detector of boundaries of curved surfaces were available, we could immediately use our curved skeleton computation to treat more complex surfaces, such as surface skeletons. Computing 'skeletons of surface skeletons' is a highly interesting matter. As shown in [81], such structures are very similar to classical curve skeletons of 3D shapes. As such, and since our curved skeleton definition is a formal one and also analogous to the definition of surface skeletons, such a 'skeleton of surface skeleton' approach could lead to a formal definition of curve skeletons. Moreover, since a single definition (and computational framework) is used, this could yield a 'grand unification' of all skeleton types known in the literature. How-

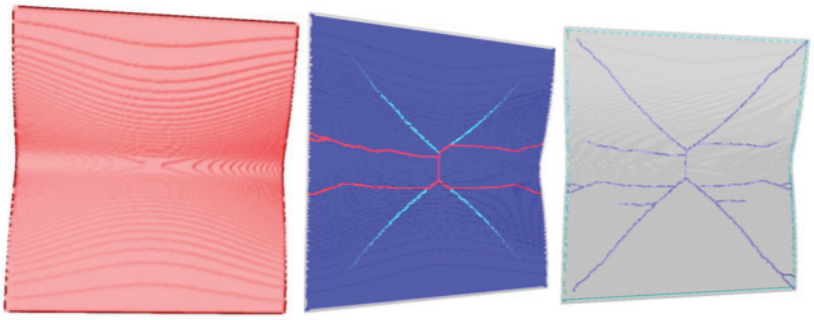


Figure 7.11: Multiscale curved skeleton importance. a) Input curved surface. b) Multiscale skeleton colored by importance metric. c) Thresholded skeleton.

ever, examining this topic is a subject for future work, as we still do not have a reliable boundary detector.

The question of foundary detection can be generalized beyond the finding of boundaries of curved surfaces. Following the above recursive idea for skeleton definitions, we can think of a recursive way to define the boundary of a shape. Starting with a 3D volumetric shape, we can find first its interior voxels. For this, we can *e.g.* use a template-based detector (the choice of which we do not comment further on, as this is not the point we are making here). By removing these interior voxels, we are left with a definition of the surface $\partial\Omega$, or a two-dimensional shape. Next, we can repeat the process, by finding voxels interior to $\partial\Omega$. By removing these, we are left now with the boundary of the shape-boundary, or $\partial\partial\Omega$, which contains one-dimensional structures, such as curves. By repeating the process, we find $\partial\partial\partial\Omega$, which contains the tips of these curves, or a zero-dimensional set of isolated voxels. In this process, lower-dimensional structures are the boundary of higher-dimensional structures. Figure 7.12 illustrates this recursive boundary detection for several 3D shapes.

The above idea of recursive boundary definition and detection is just an early experiment, and has to be further refined and tested. However, we believe it has several useful properties. First and foremost, the fact that we can generalize skeletonization as a recursive operation (as discussed above) strongly suggest that we could, and should, do the same for boundary detection. This would yield a uniform, and arguably easier to understand and use, framework for dealing with shapes of different dimensionalities. Separately, detecting boundaries of different dimensions is a required tool for practical application of skeletonization. Indeed, if we voxelize a general 3D shape, we will typically obtain a mix of volumes, surfaces, and curves (see *e.g.* the hull, sails, and respectively masts of the ship model in Fig. 7.12). As such, we cannot use a *single* type of skeleton to fully encode such a shape: For instance, for the volumetric part, surface skeletons have to be used;

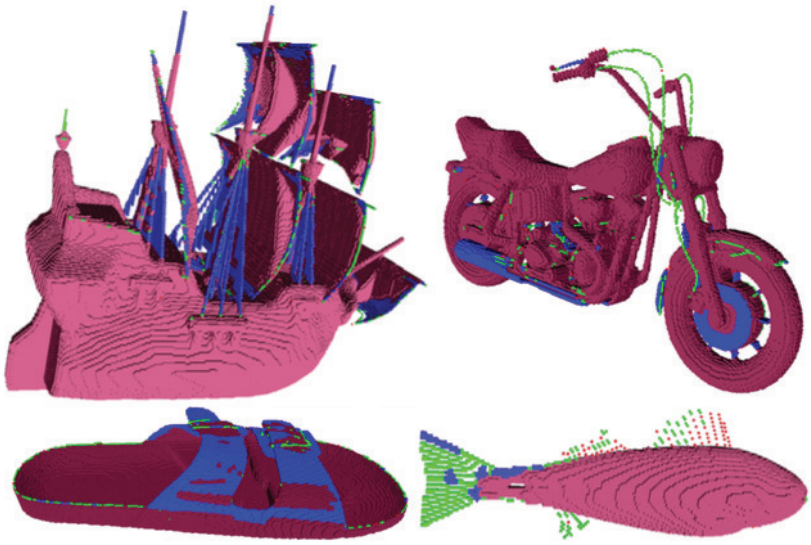


Figure 7.12: Recursive boundary detection. Three-dimensional voxels Ω are red. Two-dimensional voxels $\partial\Omega$ are blue. One dimensional ($\partial\Omega$) and zero-dimensional ($\partial\Omega$) voxels are green.

while for the curved-surface parts, the curved surface skeletons proposed in this section can be used. Interestingly enough, this issue has not been highlighted in the largest part of all skeletonization papers that we are aware of (except Tagliasacchi *et al.* [192] who introduce the concept of meso-skeletons which combine classical 3D curve and surface skeletons). Yet, being able to separate parts of a shape having different dimensionalities is critical for practical skeletonization, as most skeletonization methods in existence *assume* a certain dimensionality of the input to be able to process it correctly.

7.6 CONCLUSION

In this chapter, we have presented a novel way of computing multiscale 2D medial axes and 3D surface skeletons of image, respectively voxel datasets. For this, we cast the problem of computing the medial geodesic function (MGF) regularization metric, known for its ability to deliver high-quality multiscale skeletons, in the computation of optimal path forests with the Image Foresting Transform (IFT) framework. We show that the delivered 2D and 3D skeletons compare very favorably from the perspective of similarity and regularization with several other known multiscale skeletonization methods. Our IFT-based implementation is very simple

and delivers good performance. To our knowledge, our method is the second one that can compute both 2D and 3D multiscale medial skeletons with a unified formulation, aside [84].

This result is important in several contexts, as follows. First, for the work in this thesis, the desirable properties of IFT surface skeletons as compared to state-of-the-art methods (smoothness of importance metric, implementation simplicity, speed) make them strong competitors for those produced by the the method we used in Chapters 3 - 5, *i.e.*, [83]. However, we agree that the ultimate proof – showing that IFT skeletons can solve many of the regularization problems inherent to [83] for shape segmentation and shape matching – is not yet present (given our limited available time to work on this thesis). Yet, there is no single aspect of IFT that suggests that it would not reduce such problems, if and when used in practice. Secondly, in a broader context, the IFT is a serious competitor for all other multiscale surface-skeletonization methods out there. As such, we believe that its use in practice can improve any application relying on surface skeletons, beyond segmentation and matching. Finally, we introduced the concept of curved skeletons which generalize 2D planar skeletons to curved surfaces embedded in 3D. Curved skeletons can be computed using the same path-tracing mechanisms that the IFT proposes for computing 3D surface skeletons.

Several extensions of this work are possible. Performance-wise, extending IFT to use multithreaded parallelization has the potential to make this method the fastest (and most accurate) multiscale skeletonization technique for 2D skeletons and 3D surface skeletons on the CPU *in existence*. Application-wise, the IFT framework allows one to easily change the cost function, thereby enabling one to design a whole family of multiscale regularization metrics, beyond the MGF metric. Such metrics could, in turn, support various types of applications, such as feature-sensitive regularization. Finally, an interesting extension regards the computation of multiscale 3D curve skeletons, following *e.g.* the detection criteria in [45, 145]; and the refinement of the curved skeleton concept introduced in this chapter.

8 CONCLUSION

We conclude our work by revisiting the main research question (and derived sub-questions) stated in Chapter 1, Sec. 1.3:

How can we use surface skeletons of 3D shapes to efficiently and effectively support a range of shape processing applications?

In the following, we overview and discuss our main research results. Section 8.1 lists these, with a discussion focusing on the main technical contributions of each result. Section 8.2 discusses all our work from the general perspective of application requirements, such as genericity, scalability, and ease of use. Section 8.3 concludes with an outline of potential future work directions.

8.1 TECHNICAL CONTRIBUTIONS

Our main contributions can be organized along two directions: the skeleton cut space, and multiscale skeletonization, as follows.

8.1.1 *Skeleton cut space*

The key contribution of this thesis is the introduction of the *skeleton cut space* (Chapter 3). The skeleton cut space is a new 'refined feature' of the surface skeleton, in the sense introduced by Kustra *et al.* [98]. That is, it occupies the same relation to the surface skeleton $S_{\partial\Omega}$ as the feature points given by the feature transform $FT_{\partial\Omega}$ (Eqn 2.3), the skeleton boundary $\partial\S_{\partial\Omega}$ (Eqn. 2.5), the skeleton importance metric ρ (Eqn. 2.4), and the geodesic paths between feature points (γ, Sec. 2.1.2.3).

Briefly put, the cut space enriches the relation between a surface skeleton $S_{\partial\Omega}$ and the shape Ω it comes from by adding, for each skeletal point $\mathbf{x}$, a cut $\gamma \subset \partial\Omega$ that is tight (piecewise geodesic), smooth, closed, intersection-free, and locally orthogonal to the shape's symmetry axis or curve skeleton. Such cuts effectively generalize the closed cuts proposed by Dey and Sun [45] and Reniers *et al.* [145] in the sense of (a) being computed from the surface, rather than the curve, skeleton; (b) being locally smooth; and (c) being used for novel purposes (segmentation and shape matching). Intuitively, the cut space answers the question "given a surface skeleton point $\mathbf{x}$, how can I best cut the shape around this point?".

The cut space (set of all cuts having the above-mentioned properties) is an important separate concept. It allows one to analyze *all* cuts that the surface skeleton proposes for a given shape. In this sense, it is analogous to the randomized cuts idea of Golovinskyi *et al.* [74], *i.e.*, proposes a conservative set of all possible ways we can cut a shape, given certain cut properties. We argue that our cut space is more suited for applications such as segmentation and matching, as the constraints we put on our cuts are a superset of those in [74]. For instance, the method in [74] does not require cuts to be locally orthogonal on the shape's symmetry axis. This yields a large number of spurious, uninteresting, cuts which our cut space suffers far less from.

The above-mentioned properties of the cut space make it next an effective tool for shape processing applications. We demonstrate this by three kinds of applications, as follows.

Segmentation: First, we show how the cut space can be used to efficiently and effectively produce part-based (Chapters 3 and 4) and mixed part-and-patch based segmentations (Chapter 6) of voxel and mesh shapes. In particular, the unified part-and-patch segmentation is novel, as most skeleton-based segmentation methods that we are aware of handle well only one segmentation type (part or patch). For this, we propose several types of analyses of the cut space, based on its length histogram as well as on cut clustering. Effectively, this moves the segmentation problem from analyzing either the input surface (as in classical mesh segmentations) or the skeleton (as in 'pure' skeleton-based segmentation methods [12, 141, 142]) to the analysis of the intermediate cut space concept. The main advantage hereof is decoupling the computation of the skeleton from the actual segmentation procedure. Hence, we can easily use our cut space methods with any kind of skeleton structure that can deliver such a cut space. In turn, such skeletons can be computed by independent methods, *e.g.*, the voxel-based method in [83], the GPU mesh-based method in [82], or our own multiscale voxel-based method in Chapter 7. We believe this to be an important contribution, as it separates technical concerns on the design (and validation) of skeletonization and segmentation, which are often mixed in current papers.

Matching and retrieval: Our second application regards 3D shape matching and segmentation (Chapter 5). Our main contribution in this area is showing how *surface* skeletons can be used to this end, in contrast to the vast majority of related literature where only curve skeletons are used. For this, we propose a new shape dscriptor based on the cut space introduced in Chapter 3 which captures local shape characteristics in a more global way than existing methods. In detail, we estimate the local characteristics by the length of a cut, which captures more information than *e.g.* using the distance-to-boundary field. Our descriptor also has

good characteristics, such as scale, rotation, and translation invariance, and (up to large extents) pose invariance. Based on this descriptor, we propose a simple shape retrieval method, for which we test four different distance functions, two of each (the Median Value Separation and the Multilevel Distance) are novel. We compare this method qualitatively and in terms of several global precision metrics with several well-known retrieval techniques. Separately, we demonstrate that our method is robust to shape noise. The results show that our method can compete very well with other state-of-the-art retrieval methods. From a technical perspective, this brings the second evidence to our hypothesis that surface skeletons are useful tools for supporting shape processing applications.

Editing: Our third and last application shows that the cut space can be used to effectively support shape editing operations. Key to the effectiveness of such operations are tools that allow a *simple* selection of the shape details to edit. In line with earlier work [33], we show that the cut space allows one to select features such as protruding parts from a shape by a single click. The aforementioned properties of our cuts also make them ideal as selection boundaries, *i.e.*, plausible ways to separate a selected part from the rest of the shape (much like in shape segmentation). We should note that this use-case is much less elaborated than those for shape segmentation and/or shape matching. However, we believe that the idea behind the usage of the cut space to support interactive shape editing is straightforward, and as such it supports this type of use-case withour the need for further proof.

8.1.2 *Multiscale surface skeletons*

Our second main contribution is the proposal of a novel method for computing multiscale surface skeletons of voxel shapes (Chapter 7). As explained there, the motivation for this work resides in the various limitations we have found when using existing comparable state-of-the-art methods in our earlier work on computing cut spaces. These stem, first and foremost, from the fact that there are only very few methods able to compute *multiscale* surface skeletons. Moreover, several such methods listed in the literature are not available and/or easily replicable. Yet, other remaining methods are prohibitively slow [145].

This analysis left us with only two methods [82, 83] that meet the required characteristics for their practical usage in computing cut spaces – they are easy to use, they can handle large datasets (shapes) in a matter of seconds, they compute an importance metric allowing the continuous progressive regularization and simplification of surface skeletons, and they produce skeletons which are centered, connected, complete, thin, and invariant to isometric transformations (as described in [179]). However, practical usage of the methods in [82, 83] in our shape segmentation work (Chapters 3,4, and 6) showed that these methods also have limitations.

Specifically, the voxel-based method in [82] often produces simplified skeletons which are noisy; and the mesh-based method in [82] is highly complex and requires a very dense sampling of the input $\partial\Omega$.

We proposed to improve on the above limitations by focusing on voxel-based surface skeletons. Chapter 7 presents a new method for computing such skeletons using the Image Foresting Transform (IFT) [58]. Several properties of the IFT allow us to derive our skeletonization method in a principled way, and also to control the quality of the resulting skeletons. The examples shown in Sec. 7.4 show that our method produces skeletons showing a much smoother variation of the multiscale importance metric from the boundary $\partial\Omega$ inwards as all other multiscale skeletonization methods that we could compare it with, which include all earlier mentioned methods. As such, using our method gives better (smoother, less noisy) skeletons that can be used in any application, beyond shape segmentation and matching. From a technical viewpoint, our work extends the so far small family of multiscale surface skeletonization methods [82, 83, 145] with a fourth method. Additionally, our method handles 2D shapes in an identical way to 3D shapes. To our knowledge, a single other method exists that does this [82]. Finally, experimental evaluation shows that our method has competitive running times as compared to the other tested methods, and also has a simple implementation (see pseudocode in Sec. 7.3).

Separated from the above, we introduced the concept of curved skeletons which generalize 2D planar skeletons to curved surfaces embedded in 3D. We showed how curved skeletons can be compued by the same path-tracing mechanisms that the IFT uses to extract 3D surface skeletons. Although the curved skeleton concept is only in an easy phase, it shows potential in being extended to robustly and efficiently describe such curved shapes.

8.2 APPLICATION-DRIVEN REQUIREMENTS

We next briefly outline the aspects of our work which address practical applications.

Representation: We have demonstrated the application of cut space techniques to both voxel-based and mesh-based methods. Specifically, we demonstrated shape segmentation for both representation types (Chapters 3, 4, 6); and shape editing, retrieval, and skeletonization for voxel models (Chapters 4, 5, 7). The relatively limited presence of mesh-based applications is purely due to the limited time available for this research. From the material in Chapter 6, it is clear that the cut space concept, including processing thereof in terms of length histograms, is perfectly doable for mesh shapes. As such, extending the retrieval work in Chapter 5 to

handle mesh shapes is an easy task.

Practicality: We have validated all our proposals by actual implementations of the methods (segmentation, editing, retrieval, skeletonization) and evaluation thereof on a wide range of real-world shapes, including natural and man-made shapes, shapes having different poses and articulations, shapes of different genii, sampling resolutions, and both smooth and noisy shapes. Evaluation was done both qualitatively, in terms of comparing our results visually with those produced by competitive related methods; and qualitatively, in terms of comparing *e.g.* computational performance and accuracy with these methods. For all methods, parameters are presented and discussed in detail to support replicability. As such, we believe our results to be on a maturity level comparable to other good results in the related literature. Additionally, we believe our methods to be usable and useful in practice.

Generality: Our methods have not addressed a specific subtype (family) of shapes; nor have our applications. That is, we do not have prior constraints on the shapes we handle. The cut space concept is generic, in the sense that its construction and properties are identical for any input shape. Of course, the results shown for various applications (*e.g.* shape matching and shape segmentation) differ in terms of quality among the tested shapes. However, this is not due to explicit design decisions in the underlying methods. This is a consequence of the fact that surface skeletons capture the information in *all* types of 3D shapes equally well – in contrast to curve skeletons, which, as explained effectively capture locally tubular shapes but are not highly meaningful for other shapes.

8.2.1 *Usability*

We proposed a part-based segmentation method, and we used this method in practical dataset. In our first approach, the whole computation time of one segmentation result is dozens second. After optimization, our computation time is speed-up. In table3, I did not count feature transform and distance transform computation time. The computation of feature transform is not optimized, and I use a simplified surface skeleton for segmentation. Any way, the computation time of feature transform is usually less than ten seconds. And the computation time of Euclidean transform is in few seconds. So the whole segmentation process of one segmentation is in a reasonably quick speed. Besides, in theory, the whole segmentation process is built step by step. And in a computational way, more engineering work should be done for segmentation. So parameters in theory and practical are important for use. In histogram based segmentation process, we can make a visual histogram for users to get desired segmentation results. In clustering based method,

I fixed almost all parameters , and leave one parameter for hierarchical clustering. Of course, if there are more complicated shapes, parameter setting should be discussed. But in this thesis, I only use one parameter for hierarchical clustering.

8.3 FUTURE WORK

Summarizing the above, we believe to have defended well our initial hypothesis that surface skeletons are useful and usable instruments that can effectively and efficiently support various shape processing applications. This, and the specific results shown in this thesis, opens several potential directions for future work, as follows.

Cut space: The cut space proposed in Chapter 3 can be extended in several directions. So far, although we compute detailed cuts, we use their actual geometric information only for *drawing* borders of segments. For analyzing the cut space, *e.g.* to actually *compute* the segments or for shape matching, we use only the cut length and cut center information. A wealth of other cut features can be easily extracted and used to improve the description power of the cut space, *e.g.*, cut eccentricity, local orientation, or even information on the shape boundary along the cut, *e.g.* texture or colors. This would lead to a whole family of richer high-dimensional descriptors which have a higher chance of capturing relevant shape aspects. In turn, this can arguably lead to better segmentation and shape retrieval, but can also support new application areas, such as shape compression, simplification, abstraction, and modeling.

Applications: At a more detailed level, the applications discussed in this thesis can be further refined. For instance, for shape matching, more refined distance functions to compare two histogram-based descriptors can be thought of. One interesting idea in this respect is to weigh the importance of different shape parts differently. For instance, for a matching application where one knows that certain shape features, like protrusions or specific user-indicated shape parts, have a high importance, this information can be incorporated in the cut space and resulting histograms. Next, refined distance metrics can be designed to favor similarity over the high-importance parts. This would lead to a new family of *interest-driven* retrieval methods, where users can steer the search process by adding various types of information they are looking for.

Skeletonization: The multiscale method presented in Chapter 7 satisfies very well all functional and non-functional requirements that 3D surface skeletonization methods have, but has (still) limited scalability. This can be increased along two directions. First, computational speed can be massively increased by using CPU or

GPU parallelization, as several of the underlying IFT operations are independent on each other. Separately, but of equal or even higher importance, the maximal size of voxel models that the method could treat can be increased, by using sparse storage schemes. This last point is of great interest, since most (if not all) voxel-based skeletonization methods in existence are limited by the need to store one or several full volumes in memory, which makes them practically applicable only up to sizes of a few thousand voxel cubed. Using such storage schemes would drastically reduce the memory need, making our method competitive, in terms of resolution of shapes it can treat, with advanced mesh-based methods such as [82]. Combined with the aforementioned parallelization, this could lead to *the* 3D surface skeletonization method that all applications could next directly use.

BIBLIOGRAPHY

[1] A. Agates, I. Pratikakis, S. Perantonis, N. Sapidis, and P. Azariadis. 3D mesh segmentation methodologies for CAD applications. *Computer-Aided Design & Applications*, 4(6):827–841, 2007.

[2] Aim@Shape Consortium. Aim@Shape repository, 2015. `http://visionair.ge.imati.cnr.it/ontologies/shapes`.

[3] N. Amenta, S. Choi, and R. K. Kolluri. The power crust. In *Proc. ACM SMA*, pages 249–266, 2001.

[4] W. P. Amorim, A. X. Falcão, and M. H. D. Carvalho. Semi-supervised pattern classification using optimum-path forest. In *Proc. SIBGRAPI*, pages 111–118, 2014. DOI 10.1109/SIBGRAPI.2014.45.

[5] F. Andaló, P. Miranda, R. D. S. Torres, and A. Falcão. Shape feature extraction and description based on tensor scale. *Pattern Recognition*, 43(1):26 – 36, 2010.

[6] A. Angelidis and M. P. Cani. Adaptive implicit modeling using subdivision curves and surfaces as skeletons. In *Proc. ACM SMA*, pages 45–52, 2002.

[7] C. Arcelli, G. S. di Baja, and L. Serino. Distance-driven skeletonization in voxel images. *IEEE TPAMI*, 33(4):709–720, 2011.

[8] A.Torsello and E. Hancock. Correcting curvature-density effects in the Hamilton-Jacobi skeleton. *IEEE TIP*, 15(4):877Ð891, 2006.

[9] D. Attali and A. Montanvert. Computing and simplifying 2D and 3D continuous skeletons. *CVIU*, 67(3):261–273, 1997.

[10] M. Attene, B. Falcidieno, and M. Spagnuolo. Hierarchical mesh segmentation based on fitting primitives. *Visual Comput*, 22(3):181–193, 2006.

[11] M. Attene, S. Katz, M. Mortara, G. Patane, M. Spagnuolo, and A. Tal. Mesh segmentation – a comparative study. In *Proc. SMI*, pages 134–141, 2006.

[12] O. Au, C. Tai, H. Chu, D. Cohen-Or, and T. Lee. Skeleton extraction by mesh contraction. *ACM TOG (Proc. ACM SIGGRAPH)*, 27(3):441–449, 2008.

[13] X. Bai and J. Latecki. Path similarity skeleton graph matching. *IEEE TPAMI*, 30(7), 2008.

[14] X. Bai, J. Latecki, and W. Y. Liu. Skeleton pruning by contour partitioning with discrete curve evolution. *IEEE TPAMI*, 29(3):449–462, 2007.

[15] M. Balint, J. Giesen, and M. Pauly. Discrete scale axis representations for 3D geometry. *ACM TOG*, 29(4):1–10, 2010.

[16] I. Baran and J. Popović. Automatic rigging and animation of 3D characters. *ACM TOG*, 26(3):72(1–8), 2007.

[17] G. Bertrand. A parallel thinning algorithm for medial surfaces. *Pattern Recogn Lett*, 16(9):979–986, 1995.

[18] G. Bertrand and M. Couprie. Two-dimensional parallel thinning algorithms based on critical kernels. *J Math Imaging Vis*, 31(35):35–56, 2008.

[19] J. Bloomenthal, C. Bajaj, J. Blinn, M. P. Cani, A. Rockwood, B. Wyvill, and G. Wyvill. *Introduction to Implicit Surfaces*. Morgan Kaufmann, 1997.

[20] H. Blum. *A transformation for extracting new descriptors of shape*. Models for the perception of speech and visual form. MIT Press, 1967.

[21] M. Botsch, L. Kobbelt, M. Pauly, P. Alliez, and B. Lévy. *Polygon Mesh Processing*. A K Peters, 2010.

[22] S. Bouix, K. Siddiqi, and A. Tannenbaum. Flux driven automatic centerline extraction. *Medical Image Analysis*, 9(3):209–221, 2005.

[23] M. Braunstein, D. Hoffman, and A. Saidpour. Parts of visual objects: and experimental test of the minima rule. *Perception*, 18:817–826, 1989.

[24] M. Breuss, A. Bruckstein, and P. Maragos. *Innovations for Shape Analysis: Models and Algorithms*. Springer, 2013.

[25] A. M. Bronstein, M. M. Bronstein, R. Kimmel, M. Mahmoudi, and G. Sapiro. A Gromov-Hausdorff framework with diffusion geometry for topologically-robust non-rigid shape matching. *IJCV*, 89(2-3):266–286, 2010.

[26] F. Cazals, H. Kanhere, and S. Loriot. Computing the volume of a union of balls: A certified algorithm. *ACM Trans. Math. Softw.*, 38(1):1–20, 2011.

[27] CGAL. 3D Voronoi diagram computation. In *CGAL User and Reference Manual*. CGAL Editorial Board, 4.6.2 edition, 2015. URL `http://doc.cgal.org`.

[28] M. Chang, F. Leymarie, and B. Kimia. Surface reconstruction from point clouds by transforming the medial scaffold. *CVIU*, 113(11):1130–1146, 2009.

[29] D. Y. Chen, X. P. Tian, Y. T. Shen, and M. Ouhyoung. On visual similarity based 3D model retrieval. *CGF*, 22(3):223–232, 2003.

[30] L. Chen, H. Wei, and J. Ferryman. A survey of human motion analysis using depth imagery. *Pattern Recognition Letters*, 34(15):1995–2006, 2013.

[31] K. Ciesielski, P. Miranda, A. Falcão, and J. Udupa. Joint graph cut and relative fuzzy connectedness image segmentation algorithm. *Medical Image Analysis*, 17(8):1046–1057, 2013.

[32] U. Clarenz, U. Diewald, and M. Rumpf. Anisotropic geometric diffusion in surface processing. In *Proc. IEEE Visualization*, pages 232–230, 2000.

[33] U. Clarenz, M. Griebel, M. Rumpf, M. A. Schweitzer, and A. Telea. Feature sensitive multiscale editing on surfaces. *Visual Computer*, 20(5):329–343, 2004.

[34] U. Clarenz, M. Rumpf, and A. Telea. Robust feature detection and local classification for surfaces based on moment analysis. *IEEE TVCG*, 10(5): 516–524, 2004.

[35] D. Cohen-Or and A. Kaufman. Fundamentals of surface voxelization. *Graphical Models and Image Processing*, 57(6):453–461, 1995.

[36] D. Comaniciu and P. Meer. Mean shift: A robust approach toward feature space analysis. *IEEE TPAMI*, 24(5):603–619, 2002.

[37] N. Cornea, M. Demirci, D. Silver, A. Shoukofandeh, S. Dickinson, and P. Kantor. 3D object retrieval using many-to-many matching of curve skeletons. In *Proc. IEEE SMI*, pages 147–152, 2005.

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

[39] N. D. Cornea, D. Silver, and P. Min. Curve-skeleton properties, applications, and algorithms. *IEEE TVCG*, 13(3):87–95, 2007.

[40] L. F. D. Costa and R. M. Cesar. *Shape Analysis and Classification: Theory and Practice*. CRC Press, 2000.

[41] M. Couprie. Topological maps and robust hierarchical Euclidean skeletons in cubical complexes. *CVIU*, 117(4):355–369, 2013.

[42] J. Damon. Global medial structure of regions in $\mathbb{R}^3$. *Geometry and Topology*, 10:2385–2429, 2006.

[43] T. Delame, J. Kustra, and A. Telea. Structuring 3D medial skeletons: A comparative study. In *Proc. Vision, Modeling, and Visualization (VMV)*. Eurographics, 2016.

[44] T. Dey and J. Sun. Defining and computing curve-skeletons with the medial geodesic function. In *Proc. SGP*, pages 143–152, 2006.

[45] T. Dey and J. Sun. Defining and computing curve-skeletons with the medial geodesic function. In *Proc. SGP*, pages 143–152, 2006.

[46] T. Dey and W. Zhao. Approximate medial axis as a Voronoi subcomplex. *Comp. Aided Design*, 36(2):195–202, 2004.

[47] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, pages 269–271, 1959.

[48] M. P. Do Carmo. *Differential Geometry of Curves and Surfaces*. Prentice Hall, 1976.

[49] M. van Dortmont, H. van de Wetering, and A. Telea. Skeletonization and distance transforms of 3D volumes using graphics hardware. In *Proc. DGCI*, pages 617–629. Springer, 2006.

[50] R. Dougherty and K. Kunzelmann. Computing local thickness of 3D structures with ImageJ. In *Proc. Microscopy & Microanalysis Meeting, Ft. Lauderdale, Florida, Aug. 5-9, 2009*, 2007. `www.optinav.com/LocalThicknessEd.pdf`.

[51] O. Ersoy, C. Hurter, F. Paulovich, G. Cantareira, and A. Telea. Skeleton-based edge bundles for graph visualization. *IEEE TVCG*, 17(2):2364 – 2373, 2011.

[52] A. X. Falcão, J. K. Udupa, and F. K. Miyazawa. An ultra-fast user-steered image segmentation paradigm: live wire on the fly. *IEEE Transactions on Medical Imaging*, 19(1):55–62, 2000.

[53] A. X. Falcão, J. Stolfi, and R. A. Lotufo. The image foresting transform: Theory, algorithms, and applications. *IEEE TPAMI*, 26(1):19–29, 2004.

[54] A. X. Falcão, J. Stolfi, and R. A. Lotufo. The image foresting transform: theory, algorithms, and applications. *IEEE TPAMI*, 26(1):19–29, 2004.

[55] A. X. Falcão, C. Feng, J. Kustra, and A. Telea. Multiscale 2D medial axes and 3D surface skeletons by the image foresting transform. In P. K. Saha, G. Borgefors, and G. S. di Baja, editors, *Skeletonization: Theory, Methods, and Applications*, chapter 4. Elsevier Limited Press, 2016.

[56] A. Falcão and J. Udupa. A 3D generalization of user-steered live-wire segmentation. *Medical Image Analysis*, 4(4):389–402, 2000.

[57] A. Falcão, B. da Cunha, and R. Lotufo. Design of connected operators using the image foresting transform. In *Proc. SPIE*, volume 4322, pages 468–479, 2001. DOI 10.1117/12.431120.

[58] A. Falcão, L. da F. Costa, and B. da Cunha. Multiscale skeletons by image foresting transform and its applications to neuromorphometry. *Pattern Recognition*, 35(7):1571–1582, 2002.

[59] Y. Fang, J. Xie, G. Dai, M. Wang, F. Zhu, T. Xu, and E. Wong. 3D deep shape descriptor. In *Proc. IEEE CVPR*, pages 2319–2328, 2015.

[60] C. Feng, A. Jalba, and A. Telea. Part-based segmentation by skeleton cut space analysis. In *Mathematical Morphology and Its Applications to Signal and Image Processing (Proc. ISMM)*, pages 607–618. Springer LNCS 9082, 2015.

[61] C. Feng, A. Jalba, and A. Telea. Part-based segmentation by skeleton cut space analysis. In *Proc. ISMM*, pages 324–336, 2015.

[62] C. Feng, A. Jalba, and A. Telea. A descriptor for voxel shapes based on the skeleton cut space. In *Proc. Eurographics Workshop on 3D Object Retrieval (3DOR)*. Eurographics, 2016.

[63] C. Feng, A. Jalba, and A. Telea. Improved part-based segmentation of voxel shapes by skeleton cut spaces. *Mathematical Morphology – Theory and Applications*, 1:60–78, 2016.

[64] L. de Floriani and M. Spagnuolo. *Shape Analysis and Structuring*. Springer, 2008.

[65] M. Foskey, M. Lin, and D. Manocha. Efficient computation of a simplified medial axis. *Proc. ACM Symp. Solid Modeling*, pages 96–107, 2003.

[66] W. R. Franklin, V. Akman, and C. Verrilli. Voronoi diagrams with barriers and on polyhedra for minimal path planning. *Visual Computer*, 1:133–150, 1985.

[67] T. Funkhouser, O. Min, M. Kazhdan, and J. Chen. A search engine for 3D models. *ACM TOG*, 2(1):83–105, 2003.

[68] M. Garland, A. Willmott, and P. S. Heckbert. Hierarchical face clustering on polygonal surfaces. In *Proc. I3D*, pages 49–58, 2001.

[69] Y. Ge and J. Fitzpatrick. On the generation of skeletons from discrete Euclidean distance maps. *IEEE TPAMI*, 18:1055–1066, 1996.

[70] R. Getto and D. Fellner. 3D object retrieval with parametric templates. In *Proc. 3DOR*, 2015.

[71] P. Giblin and B. Kimia. A formal classification of 3D medial axis points and their local geometry. *IEEE TPAMI*, 26(2):238–251, 2004.

[72] P. J. Giblin and S. A. Brassett. Local symmetry of plane curves. *American Mathematical Monthly*, 92:689–707, December 1985.

[73] J. Giesen, B. Miklos, M. Pauly, and C. Wormser. The scale axis transform. In *Proc. SGP*, pages 106–115, 2009.

[74] A. Golovinskiy and T. Funkhouser. Randomized cuts for 3D mesh analysis. *ACM TOG*, 27(5):454–463, 2008.

[75] J. R. Haaga, D. Boll, V. S. Dogra, M. Forsting, R. C. Gilkeson, K. H. Ha, and M. Sundaram. *CT and MRI of the Whole Body*. Mosby, 2008.

[76] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[77] D. Hoffman and W. Richards. Parts of recognition. *Cognition*, 18:65–96, 1984.

[78] M. J. J. de Hoon, S. Imoto, J. Nolan, and S. Miyano. Open source clustering software. *Bioinformatics*, 19:1453–1454, 2004. Software available at `http://bonsai.hgc.jp/~mdehoon/software/cluster`.

[79] I. Hotz and H. Hagen. Visualizing geodesics. In *Proc. IEEE Visualization*, pages 311–318, 2000.

[80] V. Jain and H. Zhang. A spectral approach to shape-based retrieval of articulated 3D models. *Comput. Aided Des.*, 39(5):398–407, 2007.

[81] A. Jalba and A. Telea. Computing curve skeletons from medial surfaces of 3D shapes. In *Proc. Theory & Practice of Computer Graphics (TPCG)*, pages 99–106. Eurographics, 2012.

[82] A. Jalba, J. Kustra, and A. Telea. Surface and curve skeletonization of large 3D models on the GPU. *IEEE TPAMI*, 35(6):1495–1508, 2013.

[83] A. Jalba, A. Sobiecki, and A. Telea. An unified multiscale framework for planar, surface, and curve skeletonization. *IEEE TPAMI*, 2015. DOI:10.1109/TPAMI.2015.2414420.

[84] A. Jalba, A. Sobiecki, and A. Telea. An unified multiscale framework for planar, surface, and curve skeletonization. *IEEE TPAMI*, 38(1):30–45, 2016.

[85] A. C. Jalba, J. Kustra, and A. Telea. Surface and curve skeletonization of large 3D models on the GPU. *IEEE TPAMI*, 35(6):1495–1508, 2013.

[86] T. Ju, M. Baker, and W. Chiu. Computing a family of skeletons of volumetric models for shape description. *Comput. Aided Design*, 39(5):352–360, 2007.

[87] S. Katz and A. Tal. Hierarchical mesh decomposition using fuzzy clustering and cuts. *ACM TOG*, 22(3):954–961, 2003.

[88] S. Katz, G. Leifman, and A. Tal. Mesh segmentation using feature point and core extraction. *Visual Comput*, 21(8):649–658, 2005.

[89] M. Kazhdan, T. Funkhouser, and S. Rusinkiewicz. Rotation invariant spherical harmonic representation of 3D shape descriptors. In *Proc. SGP*, pages 156–164. Eurographics, 2003.

[90] M. Kazhdan, T. Funkhouser, and S. Rusinkiewicz. Symmetry descriptors and 3D shape matching. In *Proc. SGP*, pages 46–54, 2004.

[91] B. Kimia, A. Tannenbaum, and S. Zucker. Shapes, shocks, and deformations I: the components of two-dimensional shape and the reaction-diffusion space. *IJCV*, 15(3):189–224, 1995.

[92] R. Kimmel, D. Shaked, N. Kiryati, and A. Bruckstein. Skeletonization via Distance Maps and Level Sets. *CVIU*, 62(3):382–391, 1995.

[93] N. Kiryati and G. Szekely. Estimating shortest paths and minimal distances on digitized three-dimensional surfaces. *Pattern Recognition*, 26: 1623–1637, 1993.

[94] J. Koehoorn, C. Feng, J. Kustra, A. Jalba, and A. Telea. Unified part-patch segmentation of mesh shapes using surface skeletons. In P. K. Saha, G. Borgefors, and G. S. di Baja, editors, *Skeletonization: Theory, Methods, and Applications*, chapter 2. Elsevier Limited Press, 2016.

[95] W. Kropatsch, N. Artner, Y. Haximusha, and X. Jiang. *Graph-based representations in pattern recognition*. Springer, 2013.

[96] J. Kustra, A. Jalba, and A. Telea. Robust segmentation of multiple intersecting manifolds from unoriented noisy point clouds. *CGF*, 33(1):73–87, 2014.

[97] J. Kustra, A. Jalba, and A. Telea. Shape segmentation using medial point clouds with applications to dental cast analysis. In *Proc. VISAPP*, pages 169–172, 2014.

[98] J. Kustra, A. Jalba, and A. Telea. Computing refined skeletal features from medial point clouds. *Patt Recog Lett*, 2015. DOI:10.1016/j.patrec.2015.05.007.

[99] Y. Lee, S. Lee, A. Shamir, and D. Cohen-Or. Intelligent mesh scissoring using 3D snakes. In *Proc. IEEE Pacific Graphics*, pages 279–287, 2004.

[100] Y. Lee, S. Lee, A. Shamir, D. Cohen-Or, and H. P. Seidel. Mesh scissoring with minima rule and part salience. *CAGD*, 22:444–465, 2005.

[101] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10(8):707–710, 1966.

[102] F. Leymarie and B. Kimia. The medial scaffold of 3D unorganized point clouds. *IEEE TVCG*, 29(2):313–330, 2007.

[103] X. Li, T. Woon, T. Tan, and Z. Huang. Decomposing polygon meshes for interactive applications. In *Proc. I3D*, pages 35–42, 2001.

[104] Z. Lian, A. Godil, and X. Sun. Visual similarity based 3D shape retrieval using Bag-of-Features. In *Proc. SMI*, pages 25–36, 2010.

[105] J. Lien, J. Keyser, and N. Amato. Simultaneous shape decomposition and skeletonization. In *Proc. ACM SPM*, pages 219–228, 2005.

[106] L. Liu, E. Chambers, D. Letscher, and T. Ju. A simple and robust thinning algorithm on cell complexes. *CGF*, 29(7):2253–2260, 2010.

[107] R. Liu and H. Zhang. Segmentation of 3D meshes through spectral clustering. In *Proc. Pacific Graphics*, pages 298–305, 2004.

[108] X. Liu, S. B. Kang, and H. Y. Shum. Directional histogram model for three-dimensional shape similarity. In *Proc. IEEE CVPR*, pages 813–820, 2003.

[109] Y. Liu, J. Pu, H. Zha, W. Liu, and Y. Uehara. Thickness histogram and statistical harmonic representation for 3D model retrieval. In *Proc. 3D Data Processing, Visualization and Transmission (3DPVT)*, pages 896–903, 2004.

[110] Z. Liu, S. Tang, S. Bu, and H. Zhang. New evaluation metrics for mesh segmentation. *Computers & Graphics*, 37(6):553–564, 2013.

[111] W. E. Losensen and H. E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. In *Proc. ACM SIGGRAPH*, pages 163–169, 1987.

[112] J. Ma, S. Bae, and S. Choi. 3D medial axis point approximation using nearest neighbors and the normal field. *Visual Computer*, 28(1):7–19, 2012.

[113] G. Malandain and S. Fernandez-Vidal. Euclidean skeletons. *Image and Vision Computing*, 16(5):317–327, 1998.

[114] A. Mangan and R. Whitaker. Partitioning 3D surface meshes using watershed segmentation. *IEEE TVCG*, 5(4):308–321, 1999.

[115] McGill University. McGill 3D Shape Benchmark, 2015. `http://www.cim.mcgill.ca/~shape/benchMark`.

[116] MeshLab Consortium. MeshLab geometry processing tool, 2015. `meshlab.sourceforge.net`.

[117] M. Meyer, M. Desbrun, P. Schröder, and A. Barr. Discrete differential-geometry operators for triangulated 2D manifolds. In *Proc. VisMath*. Springer, 2002.

[118] P. A. V. Miranda, A. X. Falcão, and T. V. Spina. Riverbed: A novel user-steered image segmentation method based on optimum boundary tracking. *IEEE Transactions on Image Processing*, 21(6):3042–3052, 2012.

[119] P. de Miranda, A. Falcão, and J. Udupa. Synergistic arc-weight estimation for interactive image segmentation using graphs. *Computer Vision and Image Understanding*, 114(1):85 – 99, 2010. ISSN 1077-3142.

[120] H. Moreton and C. Séquin. Functional optimization for fair surface design. In *Proc. ACM SIGGRAPH*, pages 167–176, 1992.

[121] M. Mortara, G. Patané, M. Spagnuolo, B. Falcidieno, and J. Rossignac. Plumber: a method for a multi-scale decomposition of 3D shapes into tubular primitives and bodies. In *Proc. ACM SMA*, pages 339–344, 2004.

[122] D. Mount and S. Arya. Approximate nearest-neighbor search, 2015. www.cs.umd.edu/~mount/ANN.

[123] F. Nooruddin and G. Turk. Simplification and repair of polygonal models using volumetric techniques. *IEEE TVCG*, 9(2), 2003.

[124] R. L. Ogniewicz and O. Kubler. Hierarchic Voronoi skeletons. *Patt Recog*, (28):343–359, 1995.

[125] R. Osada, T. Funkhouser, B. Chazelle, and D. Dobkin. Shape distributions. *ACM TOG*, 21(4):807–832, 2002.

[126] N. Otsu. A threshold selection method from gray-level histograms. *IEEE Trans. Sys., Man. Cyber.*, 9(1):62–66, 1979.

[127] D. Page, A. Koschan, and M. Abidi. Perception-based 3D triangle mesh segmentation using fast marching watersheds. In *Proc. IEEE CVPR*, pages 27–32, 2003.

[128] K. Palagyi and A. Kuba. Directional 3D thinning using 8 subiterations. In *Proc. DGCI*, volume 1568, pages 325–336, 1999.

[129] J. Papa, A. Falcão, V. de Albuquerque, and J. Tavares. Efficient supervised optimum-path forest classification for large datasets. *Pattern Recognition*, 45(1):512 – 520, 2012.

[130] N. Pears, P. Bunting, and Y. Liu. *3D Imaging, Analysis and Applications*. Springer, 2012.

[131] D. Perchet, C. Fetita, and F. Preteux. Advanced navigation tools for virtual bronchoscopy. In *Proc. SPIE Image Processing*, pages 147–158, 2004.

[132] G. Peyre and L. Cohen. Geodesic computations for fast and accurate surface remeshing and parameterization. In *Progress in Nonlinear Differential Equations and Their Applications*, volume 63, pages 151–171. Springer LNCS, 2005. www.ceremade.dauphine.fr/~peyre.

[133] M. Pharr, W. Jakob, and G. Humphreys. *Physically based rendering – From theory to implementation*. Morgan Kaufmann, 3 edition, 2016.

[134] S. Pizer, K. Siddiqi, G. Szekely, J. Damon, and S. Zucker. Multiscale medial loci and their properties. *IJCV*, 55(2-3):155–179, 2003.

[135] S. M. Pizer, C. A. Burbeck, J. M. Coggins, D. S. Fritsch, and B. S. Morse. Object shape before boundary shape: Scale-space medial axis, 1992. Technical Report TR92-025, MEdical Image Display Group, Department of Computer Science, University of North Carolina, Chapel Hill.

[136] K. Polthier and M. Schmies. Straightest geodesics on polyhedral surfaces. In *Proc. ACM SIGGRAPH Courses*, pages 30–38, 2006.

[137] B. Preim and D. Bartz. *Visualization in Medicine: Theory, Algorithms, and Applications*. Morgan Kaufmann, 2007.

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

[139] D. Reniers and A. Telea. Tolerance-based feature transforms. In *Advances in Computer Graphics and Computer Vision*, pages 187–200. Springer, 2007.

[140] D. Reniers and A. Telea. Hierarchical part-type segmentation using voxel-based curve skeletons. *Visual Comput*, 24(6):383–395, 2008.

[141] D. Reniers and A. Telea. Part-type segmentation of articulated voxel-shapes using the junction rule. *CGF*, 27(7):1845–1852, 2008.

[142] D. Reniers and A. Telea. Patch-type segmentation of voxel shapes using simplified surface skeletons. *CGF*, 27(7):1837–1844, 2008.

[143] D. Reniers and A. Telea. Segmenting simplified surface skeletons. In *Proc. DGCI*, pages 132–140, 2008.

[144] D. Reniers, A. Jalba, and A. Telea. Robust classification and analysis of anatomical surfaces using 3D skeletons. In *Proc. VCBM*, pages 61–68. Eurographics, 2008.

[145] D. Reniers, J. J. van Wijk, and A. Telea. Computing multiscale skeletons of genus 0 objects using a global importance measure. *IEEE TVCG*, 14(2): 355–368, 2008.

[146] D. Reniers and A. C. Telea. Skeleton-based hierarchical shape segmentation. In *Proc. IEEE SMA*, pages 179–188, 2007.

[147] L. Rocha, F. Cappabianco, and A. Falcão. Data clustering as an optimum-path forest problem with applications in image analysis. *International Journal of Imaging Systems and Technology*, 19(2):50–68, 2009.

[148] J. Roerdink and W. Hesselink. Euclidean skeletons of digital image and volume data in linear time by the integer medial axis transform. *IEEE TPAMI*, 30(12):2204–2217, 2008.

[149] L. Rossi and A. Torsello. An adaptive hierarchical approach to the extraction of high resolution medial surfaces. In *Proc. 3DIMPVT*, pages 371–378, 2012.

[150] Y. Rubner, C. Tomasi, and L. J. Guibas. A metric for distributions with applications to image databases. In *Proc. IEEE ICCV*, pages 59–66, 1998.

[151] Y. Rubner, C. Tomasi, and L. J. Guibas. A metric for distributions with applications to image databases. In *Proc. IEEE ICCV*, pages 59–66, 1998.

[152] Y. Rubner, C. Tomasi, and L. J. Guibas. The earth mover's distance as a metric for image retrieval. *IJCV*, 40(2):99–121, 2000.

[153] M. Rumpf and A. Telea. A continuous skeletonization method based on level sets. In *Proc. VisSym*, pages 151–158, 2002.

[154] R. da S. Torres and A. Falcão. Contour salience descriptors for effective image retrieval and analysis. *Image and Vision Computing*, 25(1):3 – 13, 2007.

[155] R. da S. Torres, A. Falcão, and L. da F. Costa. A graph-based approach for multiscale shape analysis. *Pattern Recognition*, 37(6):1163–1174, 2004.

[156] P. K. Saha, G. Borgefors, and G. S. di Baja. A survey on skeletonization algorithms and their applications. *Patt Recog Lett*, 76(1):3–12, 2015.

[157] D. Saupe and D. Vranic. 3D model retrieval with spherical harmonics and moments. In *Proc. DAGM*, pages 392–397. Springer, 2001.

[158] S.Berchtold and H.-P.Kriegel. S3: Similarity search in CAD database systems. In *In ACM SIGMOD*, pages 564–567, 1997.

[159] W. Schmidt, J. Sotomayor, A. Telea, C. Silva, and J. Comba. A 3D shape descriptor based on depth complexity and thickness histograms. In *Proc. SIBGRAPI*, pages 433–440, 2015.

[160] W. Schmitt, J. Sotomayor, A. Telea, C. Silva, and J. Comba. A 3D shape descriptor based on depth complexity and thickness histograms. In *Proc. IEEE SIBGRAPI*, pages 261–267, 2015.

[161] T. Sebastian, P. Klein, and B. Kimia. Recognition of shapes by editing shock graphs. *IEEE TPAMI*, 26(5):550–571, 2004.

[162] L. Serino, G. S. di Baja, and C. Arcelli. Using the skeleton for 3D object decomposition. In *Proc. SCIA*, pages 447–456. Springer LNCS, 2011.

[163] L. Serino, C. Arcelli, and G. S. di Baja. From skeleton branches to object parts. *CVIU*, (129):42–51, 2014.

[164] J. Serra. *Image Analysis and Mathematical Morphology*. Academic Press, 1983.

[165] J. A. Sethian. A fast marching level set method for monotonically advancing fronts. In *Proceedings of the National Academy of Sciences of the United Stated of America*, volume 93 number 4, pages 1591–1595, Febrary 1996.

[166] D. Shaked and A. Bruckstein. Pruning medial axes. *CVIU*, 69(2):156–169, 1998.

[167] A. Shamir. A formulation of boundary mesh segmentation. In *Proc. 3DPVT*, 2004.

[168] A. Shamir. A survey on mesh segmentation techniques. *CGF*, 27(8):1539–1556, 2008.

[169] L. Shapira, A. Shamir, and D. Cohen-Or. Consistent mesh partitioning and skeletonisation using the shape diameter function. *Visual Comput*, 24(4): 249–259, 2008.

[170] K. G. Shin and R. D. Throne. Robot path planning using geodesic and straight line segments with Voronoi diagrams, 1986. Technical Report RSD-TR-27-86, University of Michigan, Ann Arbor.

[171] K. Siddiqi and B. B. Kimia. A shock grammar for recognition. In *Proc. IEEE CVPR*, pages 507–513, 1996.

[172] K. Siddiqi and S. Pizer. *Medial representations: mathematics, algorithms and applications*. Springer, 2008.

[173] K. Siddiqi, A. Shoukofandeh, S. Dickinson, and S. Zucker. Shock graphs and shape matching. *IJCV*, 35(1):13–32, 1999.

[174] K. Siddiqi, A. Shoukofandeh, S. Dickinson, and S. W. Zucker. Shock graphs and shape matching. *IJCV*, 35(1):13–32, 1999.

[175] K. Siddiqi, S. Bouix, A. Tannenbaum, and S. Zucker. Hamilton-Jacobi skeletons. *IJCV*, 48(3):215–231, 2002.

[176] K. Siddiqi, J. Zhang, D. Macrini, A. Shoukofandeh, and S. Dickinson. Retrieving articulated 3D models using medial surfaces. *Mach. Vis. Appl.*, 19: 261–275, 2008.

[177] M. Singh, G. Seyranian, and D. Hoffman. Parsing silhouettes: The short-cut rule. *Perception & Psychophysics*, 4(61):636–660, 1999.

[178] A. Sobiecki, H. Yasan, A. Jalba, and A. Telea. Qualitative comparison of contraction-based curve skeletonization methods. In *Proc. ISMM*, pages 425–439. Springer, 2013.

[179] A. Sobiecki, A. Jalba, and A. Telea. Comparison of curve and surface skeletonization methods for voxel shapes. *Pattern Recogn Lett*, 47:147–156, 2014.

[180] T. Spina, P. de Miranda, and A. Falcão. Hybrid approaches for interactive image segmentation using the live markers paradigm. *IEEE Transactions on Image Processing*, 23(12):5756–5769, 2014.

[181] S. Stolpner, S. Whitesides, and K. Siddiqi. Sampled medial loci and boundary differential geometry. In *Proc. IEEE 3DIM*, pages 87–95, 2009.

[182] S. Stolpner, S. Whitesides, and K. Siddiqi. Sampled medial loci for 3D shape representation. *CVIU*, 115(5):695–706, 2011.

[183] D. Storti, G. Turkiyyah, M. Ganter, C. Lim, and D. Stal. Skeleton-based modeling operations on solids. In *Proc. ACM SMA*, pages 141–154, 1997.

[184] A. Sud, M. Foskey, and D. Manocha. Homotopy-preserving medial axis simplification. In *Proc. SPM*, pages 103–110, 2005.

[185] J. Sun, M. Ovsjanikov, and L. Guibas. A concise and provably informative multi-scale signature based on heat diffusion. *CGF*, 29(5):1383–1392, 2009.

[186] H. Sundar, D. Silver, N. Gagvani, and S. Dickinson. Skeleton based shape matching and retrieval. In *Proc. ACM SMI*, pages 130–137, 2003.

[187] V. Surazhsky, T. Surazshky, D. Kirsanov, S. Gortler, and H. Hoppe. Fast exact and approximate geodesics on meshes. In *Proc. ACM SIGGRAPH*, pages 130–138, 2005.

[188] H. Tabia and H. Laga. Covariance-based descriptors for efficient 3D shape matching, retrieval, and classification. *IEEE Transactions on Multimedia*, 17(9):1591–1603, 2015.

[189] A. Tagliasacchi, T. Delame, M. Spagnuolo, N. Amenta, and A. Telea. 3D skeletons: A state-of-the-art report. *CGF*, 2016. DOI:10.1111/cgf.12865.

[190] A. Tagliasacchi, H. Zhang, and D. Cohen-Or. Curve skeleton extraction from incomplete point cloud. *ACM Transactions on Graphics*, 28(71):1–10, 2009.

[191] A. Tagliasacchi, I. Alhashim, M. Olson, and H. Zhang. Skeletonization by mean curvature flow. In *Proc. Symp. Geom. Proc.*, pages 342–350, 2012.

[192] A. Tagliasacchi, I. Alhashim, M. Olson, and H. Zhang. Mean curvature skeletons. *CGF*, 31(5):1735–1744, 2012.

[193] J. W. Tangelder and R. Veltkamp. A survey of content based 3D shape retrieval methods. *Multimed Tools Appl*, 39(3):441–471, 2008.

[194] A. Tatsuma, H. Koyanagi, and M. Aono. A large-scale shape benchmark for 3D object retrieval: Toyohasi shape benchmark. In *Proc. APISPA ASC*, pages 1–10, 2012.

[195] G. Taubin. Estimating the tensor of curvature of a surface from a polyhedral approximation. In *Proc. ICCV*, pages 902–907, 1995.

[196] G. Taubin. Geometric signal processing on polygonal meshes. In *Proc. Eurographics – STARs*. Eurographics Association, 2000.

[197] A. Telea. Feature preserving smoothing of shapes using saliency skeletons. In *Visualization and Mathematics*. Springer, 2011.

[198] A. Telea and J. van Wijk. An augmented fast marching method for computing skeletons and centerlines. In *Proc. EG/IEEE VisSym*, 2002.

[199] A. C. Telea. *Data Visualization: Principles and practice*. CRC Press, 2014. 2^{nd} edition.

[200] A. Telea and A. Jalba. Voxel-based assessment of printability of 3D shapes. In *Proc. ISMM*. Springer, 2011.

[201] J. Tierny, J. Vandeborre, and M. Daoudi. Topology driven 3D mesh hierarchical segmentation. In *Proc. SMI*, pages 215–220, 2007.

[202] A. Vilanova, R. Wegenkittl, A. Konig, and E. Gröller. Nonlinear virtual colon unfolding. In *Proc. IEEE Visualization*, 2001.

[203] M. Wan, F. Dachille, and A. Kaufman. Distance-field based skeletons for virtual navigation. In *Proc. IEEE Visualization*, pages 246–253, 2001.

[204] K. V. Wong and A. Hernandez. A review of additive manufacturing. *SRN Mechanical Engineering*, 22, 2012. DOI doi:10.5402/2012/208760.

[205] J. Xie, P. Heng, and M. Shah. Shape matching and modeling using skeletal context. *Patt Recog*, 41:1756–1767, 2008.

[206] S. Yoshizawa, A. Belyaev, and H. Seidel. Free-form skeleton-driven mesh deformations. In *Proc. ACM SMA*, pages 247–253, 2003.

[207] S. Yoshizawa, A. Belyaev, and H. Seidel. Skeleton-based variational mesh deformations. *CGF*, 26(3):255–264, 2007.

[208] L. Yu, K. Efstathiou, P. Isenberg, and T. Isenberg. Efficient structure-aware selection techniques for 3D point cloud visualizations with 2DOF input. *IEEE TVCG*, 18(12):2245–2254, 2012.

[209] A. Zaharescu, E. Boyer, K. Varanasi, and R. Horaud. Surface feature detection and description with applications to mesh matching. In *Proc. IEEE CVPR*, pages 373–380, 2009.

[210] Q. Zhou, T. Ju, and S. Hu. Topology repair of solid models using skeletons. *IEEE TVCG*, 27(3):675–685, 2007.

[211] M. van der Zwan, Y. Meiburg, and A. Telea. A dense medial descriptor for image analysis. In *Proc. VISAPP*, pages 133–140, Los Alamitos, CA:, 2013. IEEE Press.

[212] M. van der Zwan, V. Codreanu, and A. Telea. CUBu: Universal real-time bundling for large graphs. *IEEE TVCG*, 22(12):2550–2563, 2016.

[213] M. Zwicker, M. Pauly, and O. K. M. Gross. Pointshop 3D: an interactive system for point-based surface editing. In *Proc. ACM SIGGRAPH*, pages 322–329, 2002.

ACKNOWLEDGMENTS

It is now time to close my PhD journey. And this should be done by showing due appreciation to those who helped me.

First of all, I should thank Alex Telea, my main supervisor. I still remember that I sent a lot of emails to Alex for pursuing this PhD project before I started this journey. Alex helped me to have this chance. Besides, Alex has broad knoledge in the medial representation and related fields, which he shared widely with me. Additionally, as an expert in the field, Alex showed me many possible research directions and had many deep discussions with me.

I should also thank my co-supervisor, Andrei Jalba. Although we did not discuss a lot, I do strongly value, above all, his sharing of valuable code bases that helped me with my work. Also, I thank Andrei for helping me with my research and my papers directly and indirectly.

I think I should thank Harry Blum, the 'creator' of medial descriptors. Although I don't know Blum personally, for obvious reasons, it is based on his work that I did mine.

During the various conferences I attended (ISMM, 3DOR, Eurographics, EuroVis), I had deep discussions with related researchers. These discussions help me a lot in my research. As such, I want to acknowledge this help.

I enjoyed my lunch time with the SVCG members, during which we had a lot of interesting talks. In particular, prof. Jos Roerdink, the group leader of SVCG, gave me some inspiring ideas which were helpful for my research. Matthew van der Zwan, thank your for your opinions and your perspective. I should also thank Jasper van der Gronde. You helped me a lot with the starting period of my research.

To all secretaries in our institute, especially Esmee Elshof, Janieta de Jong-Schulkebir, Desiree Hansen, Ineke Schelhaas, and Lineke Koops, I value your help in all administrative work. Henk Broer, professor in our institute, you helped me to relax from high pressure with your laughing. To all members of our institute: I think you helped me directly or indirectly, and I thank all of you.

In the city of Groningen, I played football almost every week for a long time. With all members in our team, we enjoyed quite a lot joyful time. Here I thank you all your company.

I want to thank many people who appear in my daily life, but to whom I did not manage to usually talk.

I want to thank Wu Yingqiu for her support in many aspects.

I need to thank my family, my baba Feng Guoan, my mama Fang Aiguo, and my sister Feng Fang. Without your support, I would not have been able to write one single word of this thesis. You provided me with the full financial support for pursuing my PhD project, even by borrowing money. I love you deeply.

COLOPHON

This document was typeset using the typographical look-and-feel `classicthesis` developed by André Miede. The style was inspired by Robert Bringhurst's seminal book on typography "*The Elements of Typographic Style*". `classicthesis` is available for both LaTeX and LyX:

`https://bitbucket.org/amiede/classicthesis/`

Happy users of `classicthesis` usually send a real postcard to the author, a collection of postcards received so far is featured here:

`http://postcards.miede.de/`

Final Version as of June 27, 2017 (`classicthesis`).