MEDIAL DESCRIPTORS FOR 3D SHAPE SEGMENTATION, RECONSTRUCTION AND ANALYSIS

JACEK ŁUKASZ KUSTRA

.

Cover: Medial Point Cloud Rendering of Author's face

Medial Descriptors for 3D Shape Segmentation, Reconstruction and Analysis

Jacek Łukasz Kustra
Thesis University of Groningen

ISBN 978-94-6259-649-8 (printed version)
ISBN 978-94-6259-654-2 (electronic version)

university of groningen

Medial Descriptors for 3D Shape Segmentation, Reconstruction and Analysis

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

Monday 18 May 2015 at 12.45 hours

by

JACEK ŁUKASZ KUSTRA

born on 31 May 1981
in Częstochowa, Poland

THESIS SUMMARY

Three dimensional shape analysis is an important component in several applications such as path planning, shape matching, or shape segmentation. Most methods in this area use, as input, a representation of the shape boundary or surface. However, such representations present several limitations when we are interested to reason about global shape properties. An alternative shape representation was proposed in 1967 by Harry Blum, and is known under the names of medial representation, medial axis, or skeleton. Medial representation describe shapes in terms of their symmetry set, and allow a natural way to reason about both local and global shape properties. Such representations have been used in many 2D shape analysis problems. However, their extension to large and complex 3D (polygonal) shapes is far from trivial and poses several conceptual and computational challenges.

This thesis addresses several of the above challenges for computing 3D shape skeletons and next using these skeletons to support a range of shape analysis applications. We present a new robust, fast, and accurate method to extract multiscale medial representations from large polygonal models, and also explore the relationship between 3D skeletons and 2D skeletons of projections of the same shape. Next, we show how we can enrich and adapt our medial representations with additional information, such as the relationships between medial points and surface points, to support a wide range of applications, including skeleton reconstruction, shape reconstruction, edge detection, and shape simplification and segmentation. We demonstrate our methods on a large collection of real-world 3D shapes.

SAMENVATTING

Driedimensionale shapeanalyse is een belangrijk element in applicaties zoals path planning, vorm matching en vormsegmentatie. De meeste methodes op dit gebied gebruiken, als invoer, een representatie van het vormoppervlak. Dergelijke representaties zijn echter beperkt wanneer men wil redeneren over globale vormeigenschappen. Een alternatieve vormrepresentatie is geïntroduceerd in 1967 door Harry Blum en is bekend onder de naam van mediale representatie, mediale as, of skelet. Mediale representaties beschrijven vormen via hun symmetriestructuur en bevorderen een natuurlijke manier om te redeneren over lokale en globale vormeigenschappen. Dergelijke representaties zijn gebruikt bij veel 2D vormanalyseproblemen. Hun uitbreiding naar grote en complexe 3D (polygonale) vormen is echter ver van triviaal en brengt meerdere conceptuele en computationele uitdagingen met zich mee.

Dit proefschrift beantwoord een aantal van de bovenstaande uitdagingen in het berekenen van 3D skeletten en het gebruiken ervan in een aantal vormverwerkingsapplicaties. Wij introduceren een nieuwe, snelle, en robuuste methode voor het extraheren van multischaal mediale representaties uit grote polygonale modellen en uitleggen de relatie tussen 3D skeletten en 2D skeletten van projecties van dezelfde vorm. Vervolgens laten wij zien hoe deze mediale representaties verrijkt met extra informatie, zoals de relatie tussen mediale punten en oppervlampunten, kunnen worden, en ook toegepast kunnen worden voor applicaties zoals skeletreconstructie, vormreconstructie, randdetectie, vormsimplificatie, en vormsegmentatie. We illustreren onze methodes op een grote verzameling van realistische 3D vormen.

To Mahsa

All our dreams can come true,
if we have the courage to pursue them

– Walt Disney –

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

I don't pretend we have all the answers. But questions are certainly worth thinking about.

— Arthur C. Clarke

1.1 SHAPES

We live in a world which, through our senses, we perceive as three dimensional. At this level of perception, a basic property of everything that surrounds us is its *shape*. Shape determines the way we identify objects and interact with our surroundings. It is an intrinsic dynamic property of virtually everything that we perceive, therefore its understanding is of extreme importance. Furthermore, as our visual and tactile perception of an object is often limited to its boundary, *i.e.* the outer interface between the object and the space surrounding it, our visual interpretation of an object is driven by the way we perceive its boundary.

Given the above, it is natural that many computer applications have been developed in the last decade to represent, manipulate, and analyze 3D shapes. Such applications range from classical computer graphics and virtual reality (which use shapes to depict realistically-looking 3D environments), augmented reality (which add computer-generated shapes to images and video capturing a real-world environment), computer-aided medicine (which use shapes extracted from 3D scans to depict and analyze various parts of the human body), to data visualization (which use 2D and 3D synthetic shapes to depict phenomena captured by large amounts of spatial or non-spatial data).

To be able to deal with 3D shapes, a computer needs ways to represent these shapes and to measure and determine its properties. Briefly put, any software application that is concerned with such shapes needs two main components: a shape *representation* part, which deals with the actual storage of the shape and its properties, and a shape *processing* part, which deals with the operations that the application wishes to execute on the shape representation. These two components, which parallel the fundamental data structures and algorithms in any computer program, are outlined next.

1.2 SHAPE REPRESENTATION

The first step towards reasoning about a shape is having a convenient way of representing it. The representation choice is dependent on several factors such as the type of sensor used to acquire the shape and the type of analysis one wants to perform on the representation. In practice, such representations can be classified as *continuous* or *discrete*. Continuous representations are able to faithfully capture all details of a shape, and allow in principle any type of analysis to be performed thereon without loss of precision. However, they are not practical or often not even feasible, as we cannot (easily) construct them for all but simple shapes. Discrete representations have the key advantage of allowing one to capture practically any type of shape in existence by a number of relatively simple data structures. While they typically cannot represent all details of any shape, they allow a controlled trade-off between representation size and complexity (on one hand) and representation costs and accuracy (on the other hand). As such, discrete shape representations are predominant in most computer-based shape processing applications.

Besides the above, shape representations can be classified into *surface* or *volumetric* ones. Surface, or boundary, representations capture only the form of the shape's surface or interface that separates the inner part of the shape from its surroundings. They are useful in cases when we are interested to reason only about the shape boundary or in cases when we only have information on this boundary, and not on the shape's interior. Discrete boundary representations can be readily acquired by *e.g.* laser scanners or constructed by modeling tools, and consist typically in a point-sampling of the object surface, with optional additional connectivity. These two representations are also known under the names of unstructured point clouds and polygonal meshes, respectively. Volumetric representations capture both the shape's surface and its interior. They can be acquired from 3D volumetric scanners, such as CT and MRI scanners, or constructed synthetically, *e.g.* by using field-based methods. Discrete volumetric representations usually consist of an uniform sampling of the shape's interior on a regular grid, *i.e.*, of a collection of voxels whose values indicate the distinction between shape interior and exterior and, optionally, the value(s) of one or more measured properties inside the shape. In contrast to boundary representations, volumetric representations require more memory and computing power to handle.

1.3 SHAPE PROCESSING

Both boundary and volumetric representations outlined above are essentially anchored in the $\mathbb{R}^3$ (three-dimensional) space in which the shape is embedded. However, this is not the only space in which shapes can be represented. For instance, shapes can be represented in the Fourier domain. This allows certain shape processing operations, such as *e.g.*

smoothing, matching, or filtering, to be implemented in simpler and/or more efficient ways than in the spatial domain. Conversely, the spatial domain supports better other types of shape processing, such as analyses pertaining to metrology or topology.

Shape processing operations encompass a huge spectrum of manipulations, such as matching, compression, simplification, denoising and smoothing, segmentation, and rendering. As outlined above, different shape *representations* support different types of *processing* operations better (or worse) in terms of simplicity, accuracy, robustness, and computational scalability. As such, the quest of the best match between specific processing operations and existing (or new) shape representations is an important evolving field of research.

1.4 MEDIAL REPRESENTATIONS

As outlined in Section 1.2, the boundary and volumetric representations are not the only spatial representations of 3D shape known. Although these are arguably the most convenient representations for shape acquisition and display, other representations can offer important advantages for specific shape processing operations.

This thesis focuses on one such alternative spatial representation for 3D shapes: the *medial* representation. This representation, first proposed by Blum in 1967 [15], encodes a (2D or 3D) shape into the collection of the medial loci of the maximally inscribed (2D or 3D) balls inside the shape's boundary. For 2D shapes, medial axes (also called 2D skeletons), consist of a collection of 1D curves. For 3D shapes, several medial representations are known, as follows: Medial surfaces, also called surface skeletons, consist of a (complex) collection of 3D manifolds with boundaries (Figure 1.1). They provide a full representation of the entire shape boundary, but are relatively complex to compute. Curve skeletons, in contrast, consist of a (relatively simple) collection of 3D curves. They can be computed relatively easily, but only capture the topological characteristics and main geometry characteristics of the shape.

Medial representations, also called medial descriptors, are very effective in supporting shape processing applications which focus on analyzing, or manipulating, the topological and symmetry-related properties of a shape. Intuitively put, medial descriptors capture the essential 'branching structure' or part-in-whole structure of a shape, as well as its symmetry characteristics. As such, they are often used in applications where such properties are important, such as shape matching, simplification, part-in-whole segmentation, and recognition.

In this thesis, we explore the usage of 3D medial descriptors for supporting a range of shape analysis and processing operations. Hence, we next overview the different types of medial representations (Section 1.4.1), applications that such representations can support (Section 1.4.2), and outline the key challenges that medial representations face when supporting such application (Section 1.4.3).

1.4.1 *Types of medial representations*

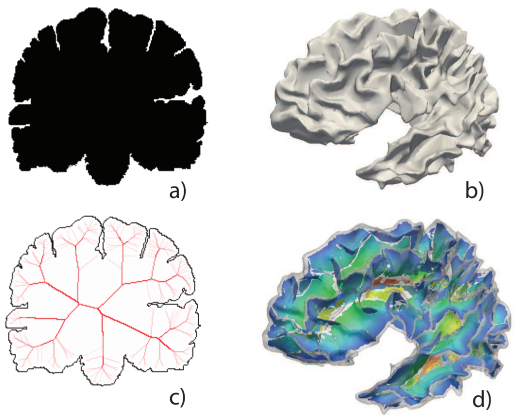


Figure 1.1: 2D and 3D shape (brain) boundary representation (a, b) and its corresponding medial representation (c, d)

As briefly outlined above, several types of medial representations have been proposed for several types of shapes and shape-processing applications. For 2D shapes, the widest-used medial representation directly follows the maximally inscribed ball definition outlined earlier. This representation has also been the most explored (and used) in practice, due to its simple and efficient computation. Currently, many algorithms exist for computing 2D skeletons, *e.g.* [45, 118, 122, 155, 183]. Such algorithms are able to robustly deliver accurate 2D skeletons of large and complex shapes, represented either as pixel images or polygonal contours, at interactive-frame rates. As such, 2D skeletons are frequently used in many applications on 2D shape matching, simplification, retrieval, recognition, and analysis (Figure 1.1).

For 3D shapes, curve skeletons are the widest-used medial representation, as these are relatively simple and efficient to compute and analyze. Similar to the 2D skeleton case, many curve-skeleton extraction algorithms have been proposed for both polygonal and volumetric shape representations, *e.g.* [7, 16, 64, 172, 174, 181, 192]. Such algorithms can efficiently extract accurate curve skeletons of large and complex 3D shapes, and have been used in applications such as 3D shape recognition and matching, path planning, and virtual navigation.

However, as already outlined, curve skeletons cannot fully capture the entire geometry and topology of a 3D shape. As such, they are most effective in describing shapes which have a locally tubular structure, such as blood vessel networks or plants. In contrast, medial surfaces do capture all topological and geometric information present on the boundary of a 3D shape. As such, they enable more complex shape processing applications, such as shape reconstruction and simplification, denoising, and smoothing. Several methods exist for extracting 3D medial surfaces from both volumetric and polygonal shape representations, *e.g.* [43, 50, 59, 101, 119, 126, 135]. However, most if not all such methods are considerably more complex, slower, and less scalable than 2D skeleton or 3D curve-skeleton extraction methods. As such, 3D surface skeletons have been used considerably less often in shape processing applications than their other two counterparts.

1.4.2 *Applications of medial representations*

Medial representations have been used for a wide range of analysis and processing operations involving both 2D and 3D shapes. Below we overview a number of important applications where such descriptors have been used, without the claim of being exhaustive.

Path planning and navigation: Given a (2D or 3D) shape, the skeleton thereof represents the locus of points which are locally centered with respect to the shape boundary. This implies that such points are, also, locally at maximal distance from the shape's boundary. As such, the skeleton can be used to construct a navigation path for a vehicle that moves inside the respective shape and avoid colliding with the shape's boundary. Following this idea, skeletons have been used for path planning applications in both 2D map-like contexts [24, 54, 55, 109, 166] and 3D contexts, such as navigating a virtual camera inside tubular organs such as arteries, bronchies, or the colon [79, 97, 192]. For 3D applications, curve skeletons have been mainly used, given that the desired path planning involved tube-like structures.

Segmentation: Given a (2D or 3D) shape, the branches of its skeleton correspond to protrusions (convex bumps) of the shape's boundary. As such, the natural idea arose to segment these protrusions from the main shape body or rump, by cutting the shape around points corresponding to the branch junction points or around points located on the skeletal ligature branches. This leads to so-called *part-based* segmentations, which work well for natural objects. Examples of part-based segmentation using skeletons include [147] for 2D shapes and [129] for 3D. In both examples, skeletons consisting of 1D structures (*i.e.*, 2D skeletons and 3D curve skeletons) are used.

Surface skeletons can be also used to segment shapes, leading to a different class of so-called *patch-based* segmentations, where segments represent quasi-flat shape surface areas separated by sharp creases. Examples of patch-based segmentations using surface skeletons are shown in [130].

Both part-based and patch-based segmentation approaches are further explored in Chapters 4 - 6.

Shape matching and retrieval: Skeletons represent the geometry and topology of a shape in a compact manner, *i.e.*, in a space which is typically of lower dimension than the given shape. As such, they can be used in shape retrieval and matching contexts. The overall idea is simple: The skeleton is reduced to an attributed graph, where nodes represent skeleton branches and edges represent skeleton junctions where several branches meet. Using this compact descriptor, shapes can be searched or matched following a top-down comparison process based on graph distances. Examples of such applications include [9, 60, 169, 196]. Similar to the segmentation case, in most current applications 2D skeletons and 3D curve skeletons are used, as surface skeletons are deemed too complex and/or computationally expensive for this context.

Shape analysis: Shape analysis concerns itself with finding specific features on the surface of 3D shapes, such as edges, corners, or concave or convex regions. Such regions can be often found easier by analyzing the shape skeleton, and using known connections between the skeleton and shape to locate them on the shape surface. For example, convex curvature maxima (ridges and corners) of 3D shapes can be found by locating the surface skeleton, respectively curve skeleton, boundaries [134]. Similar approaches can be used for 2D shapes, leading to enhanced descriptors for shape classification [8]. Compared to classical curvature-based shape analysis methods, skeletal methods have the advantage of enhanced robustness in the presence of small-scale surface noise.

Shape simplification: Shapes can be simplified by using their skeletons. Applications include removing small-scale surface noise and/or accentuating the important sharp edges thereof. The main idea here follows the fact that the boundary of a shape can be reconstructed from its skeleton. By applying suitable skeleton simplification techniques, the shape reconstructed from the simplified skeleton can thus emphasize certain important details and remove other irrelevant ones. Examples of such techniques include [41, 62, 175, 177]. Among other applications, this process leads to the production of a multiscale representation of shapes, based on progressively simplified skeletons thereof. However, as for the applications mentioned above, 3D curve skeletons and 2D skeletons are the main tools used here, the 3D surface skeletons being rarely used. Besides simplification of shapes represented by their boundaries, skeletons also have been used to simplify other types of spatial datasets, such as 2D grayscale and color images [199].

Mesh construction: Since the medial axis can be seen as the 'symmetry locus' of a boundary, it offers a useful instrument to construct a partitioning of the shape that links 'opposite' boundary parts. This property is instrumental in producing quad-dominant meshes from both 2D and 3D domains [5, 48, 137, 143]. Creating such quad meshes is of great importance for highly-accurate and efficient numerical simulations of complex phenomena, such as air flows around road vehicles and planes. Such meshing methods based on medial descriptors have also found their way into state-of-the-art commercial products [187].

Information visualization: As mentioned above, medial descriptors compactly capture the symmetry locus, or symmetry set, of a complex boundary. As such, they can encode the 'essence' of complex boundaries using more spatially compact shapes than the boundaries themselves. This property is useful, among others, in information visualization applications, where one design goal is to squeeze as much information as possible in a limited screen space – the so-called 'space filling' design approach [163]. In this context, medial axes have been used for the visualization of large graphs, by simplifying the drawing of such graphs using an edge-bundling approach [44, 179].

1.4.3 *Medial representation challenges*

For a medial descriptor to be useful and usable in practice, it should meet several conditions. Globally put, these conditions regard two activities: the *computation* of the medial descriptor from an input shape; the *interpretation* of the medial descriptor in terms of supporting relevant shape processing operations. We discuss these two aspects below.

Medial computation: The extraction of medial representations, specially for 3D surface skeletons, is a challenging proposal. For a medial extraction method (also called skeletonization method) to be usable, it has to meet several desirable properties, as follows:

- *accurate:* The computed medial descriptors have to match the skeleton definition (locus of maximally-inscribed balls) as well as possible, subject to the inherent limitations of the sampling resolution used to capture both the input shape and the output descriptor. Even small inaccuracies can lead to significant errors in the ensuing shape processing applications, such as wrong classifications or incorrect smoothing.

- *scalable:* Medial descriptors should be computable for large and complex 3D shapes, such as the ones produced by modern 3D scanners or modeling applications. These can easily have millions of polygons (for boundary representations) or billions of voxels (for volumetric representations). Additionally, the speed of computing such medial descriptors has to be high – seconds or even less for such complex shapes, in case we want to support interactive applications.

- *robust:* Skeletonization methods should be able to produce 'clean' medial descriptors from shapes having different sampling densities and/or variable amounts of sampling noise. By this, we mean that the resulting descriptors should not be strongly affected by the input sampling or noise characteristics.

- *generic:* Skeletonization methods should be able to handle shapes of a wide variety in terms of geometry and topology, *e.g.* closed or open boundaries, shapes of genus 0 or higher, and shapes representing smooth (natural) objects or shapes having a faceted structure (*e.g.* man-made objects). This way, such methods can be used for the largest possible class of applications.

While many 3D skeletonization methods exist, very few, if any, fully comply with *all* above requirements. For instance, thinning methods working on volumetric representations are simple and scalable, but are not very accurate [43, 119, 126]. Curve skeletonization methods are relatively scalable and robust, but are not generic [31, 32, 37, 64, 172]. Surface skeletonization methods are generic and accurate but relatively slow and less robust [108]. Without complying with all above requirements, the applicability of 3D skeletonization methods in real-world shape processing applications will be inherently limited.

Medial interpretation: Producing a 'raw' skeleton from a 3D shape is only the first step to actually using the skeleton for shape processing. Indeed, one further needs to extract information from the skeleton which is relevant to the specific processing operations we aim to support. This process is also known as computing higher-level medial *features*. Examples of such features are the feature points (contact points of each skeletal point with the input shape's boundary), local thickness (distance from each skeletal point to the closest input shape point), skeleton importance (a metric that encodes the relevance of each skeletal point to the description of the input shape), skeletal components (the distinct manifolds or branches that form the skeleton), junctions (the intersection points or curves where the skeletal manifolds meet), skeleton topology (the graph formed by the manifolds and their intersections), skeleton boundary (the skeletal points which appear on the border of the skeleton itself). Such features can be, in turn, refined to compute higher-level features, which support more advanced analysis and processing operations on the input shape. For example, the skeleton boundary and junctions can be further classified to detect convex and concave parts of the input shape; the skeleton importance can be analyzed to remove small-scale noise details of the input shape; skeletal boundaries can be used to detect (sharp) edges on the shape; and feature points can be analyzed to detect so-called ligature branches that correspond to protrusions, or convex parts, of the input shape.

As for skeletonization itself, several techniques have been proposed to compute higher-level features from 3D curve and surface skeletons. However, many such methods suffer from the same accuracy, scalability, robustness, and genericity problems outlined above for the skeleton extraction. As computing refined skeletal features is, thus, hard, using such features to support shape processing applications is inherently limited. On the other hand, the difficulty of computing refined skeletal features has made it hard to *explore* new ways by which such features could be used to support existing shape processing applications.

Research question: Based on the above points, we can state that computing 3D curve skeletons and their respective features is a relatively well developed field, including many applications. In contrast, we find a major challenge in the practical computation of 3D surface skeletons and their relevant higher-level features, and in the usage of such features to support shape processing applications. As such, the actual usability and usefulness of 3D surface skeletons in practice is still an open question that needs to be answered. Separately, we see that the costs of handling volumetric representations of 3D shapes (and their skeletons) appear to be intrinsically higher than the comparative costs involved by boundary representations.

Given this context, we can now formulate our **first research question**:

How can we compute 3D surface skeletons and related refined features accurately, scalably, robustly, and generically for a wide set of 3D shapes encoded by a discrete boundary representation?

To answer this question, we explore the development of novel skeletonization methods for surface skeletons, and validate them in practice by using them to compute such skeletons from a wide range of 3D shapes and several shape processing applications. This leads us to our **second research question**:

How can we use refined skeletal features extracted from 3D surface skeleton to efficiently support shape processing applications?

To answer this question, we will study how the refined skeletal features computed by the methods developed under the scope of our first question can be used to capture relevant aspects of 3D shapes, such as curvature, parts, and edges. Next, we will use the identified feature-to-shape-characteristic mappings to implement several shape processing operations, such as shape classification and shape segmentation. Finally, we will compare the results of our shape processing with results of established methods in the same class. Thereby, we will validate the usefulness and effectiveness of 3D surface skeletons (and their extracted features) for these applications.

Content-wise, with respect to the above two research questions, this thesis has two contributions:

- **Theory:** The usage of new or existing surface-skeletal *features* to support various shape processing applications highlights connections between shapes and their surface skeletons which have not been explored so far (partly due to the practical inability of computing such skeletons and features). We expect this will strengthen the interest of researchers in exploring additional skeletal features and their use in similar or different applications.
- **Practice:** The creation of efficient and effective 3D surface-skeletonization *algorithms* will support the practical applicability of surface skeletons in real-world applications. Slightly simplifying the discourse, one of the aims of this work is to show that surface skeletons can be made to be as easy and efficient to use in practice as the better-known curve skeletons.

1.5 STRUCTURE OF THIS THESIS

In line with the two main research questions stated above, this thesis has the following structure:

CHAPTER 2 provides an overview of 3D shape representations and introduces several skeleton-related concepts and definitions which will be used on all subsequent chapters. Additionally, it provides an overview of existing skeletonization methods and methods for computing related skeletal features, with a focus on 3D surface skeletons. Related work which is, by its nature, more specific to individual chapters is addressed next in the context of the respective chapters.

CHAPTER 3 presents a method for the fast, robust, and scalable extraction of 3D surface and curve skeletons from large and complex 3D shapes represented as point clouds and polygonal meshes. Additionally, we present how the proposed method can compute related skeleton features such as feature-points and skeleton importance values, used further for skeleton regularization. We also present how the resulting surface skeletons can be used to efficiently reconstruct the input shape, and how to reconstruct compact (meshed) representations of the surface skeletons from a skeleton point cloud. Given the scalability and accuracy of the presented method, this method will form the backbone for our subsequent work on extracting refined skeletal features and using such features in shape processing applications in the following chapters. Separately, we present here a different method that extracts curve skeletons from 2D views of 3D shapes. Similar to our surface-skeletonization proposal, this method is accurate, scalable, robust, and generic, and can handle complex and large shapes represented by point clouds or polygons. Globally, the two skeletonization methods presented in this chapter address the issues of scalability, robustness, and ease of computation of 3D surface and curve skeletons raised by our research questions.

CHAPTER 4 dives deeper into the challenges presented by the analysis of surface skeletons represented by point clouds, such as produced by our method proposed in Chapter 3. Two challenges are addressed here: (1) the extraction of smooth surfaces from noisy point clouds, which enables the use of our surface skeletonization methods directly on such point clouds; and (2) the extraction of the separate manifolds that compose a surface skeleton, which is an important type of refined skeletal feature. To underscore the added-value of our method, we also show its application for the denoising, segmentation, and extraction of meshed surfaces from general point clouds apart from skeletal ones. As such, this chapter partially answers the question of extracting refined features from 3D surface skeletons – in this context, these are skeletal manifolds.

CHAPTER 5 investigates the density properties of a 3D surface-skeleton point cloud. By exploring and exploiting these properties, unique to this type of skeleton representation (in contrast to *e.g.* voxel based representations), we show next how we can support shape segmentation in contexts where known segmentation methods fail to produce good results. We demonstrate our proposal by a practical application for the segmentation of orthodontic dental casts. This chapter thus targets our second research question by showing how the skeletal point-cloud density is instrumental in supporting segmentation applications.

CHAPTER 6 extends our quest for the computation of refined skeletal abstractions. We show how we can compute features such as edges, medial sheets, sheet-intersection curves, and skeleton point classifiers from the surface-skeleton point clouds delivered by our method proposed in Chapter 3. Next, we show how such features can be effectively and efficiently used to support applications in shape classification and segmentation, and compare our results with traditional techniques in these areas. As such, this chapter addresses both the first research question (extracting higher-level skeletal features) and the second research question (using the extracted features to support shape processing applications).

CHAPTER 7 concludes this thesis by discussing our answers to the two main research questions stated in Section 1.4.3 and outlines potential directions for future work in the area of using surface skeletons for additional shape processing applications.

2 MEDIAL REPRESENTATIONS OF OBJECTS

Symmetry is what we see at a glance

Blaise Pascal

2.1 INTRODUCTION TO SHAPE REPRESENTATIONS

SHAPE *noun* the external form, contours, or outline of someone or something.

According to the above definition, taken from the Oxford Dictionary, *shape* is an intrinsic property of just about everything which we perceive as an individual object separated from its surroundings. However, shapes can also be abstract mathematical representations, not necessarily representing some entity of what we call the *real-world* – consider, for instance, an m-dimensional hypersurface embedded in $n > m$ dimensions. The focus of the work presented in this thesis is to enable the use of medial representations to perform analysis of shapes of objects that we typically find in three dimensions.

To be able to analyze a shape computationally, we need some mathematical *representation* thereof. For instance, we can describe a shape by an explicit or implicit function; as being the solution of an equation; or by a (dense enough) set of sample points taken on its surface or its interior. In this sense, we can classify mathematical shape representations into *analytic* representations and *discrete* representations. Analytic representations follow the examples of the implicit/explicit function or equation solution listed above. They have the significant advantage of offering a way to exactly reason about the described shape – as long, of course, as the mathematical representation we use accurately captures the shape of interest. However, they also have the crucial disadvantage of being impractical: It is very hard to describe any possible (3D) shape in compact analytic form; moreover, even in cases where such an analytic representation may exist, it is not evident how to create it for a given shape.

Discrete representations offer an efficient and effective trade-off for the above problems. They essentially represent a shape as a dense-enough set of sample points taken either on the surface $\partial\Omega$ of the shape, or in the interior Ω of the shape. The key advantages of this representation are simplicity and generality: Given a dense-enough set of sample points, we can represent any desired shape up to a given accuracy level, by a single, relatively simple, model: a set of sample points.

Many types of discrete (or sampled) representations of shape exist. They differ in terms of *what* is sampled (*e.g.*, the boundary $\partial\Omega$ or the interior of the shape Ω); the so-called interpolation functions used to construct a (piecewise) continuous representation of the shape from its samples; the distribution of the sample points within the sampled object; and storage schemes for the sample points. In the following subsections, we provide a brief overview of a number of popular discrete representations of shape.

2.1.1 *Volumetric representations*

Discrete volumetric representations essentially sample the 'inside' of a shape Ω into a set of sample points $\mathbf{x}_i \in \mathbb{R}^n$, where n is the dimension of the space in which the shape is embedded (typically 2D or 3D). The essential information recorded by such a sample point $\mathbf{x}_i$, as such, is the shape property (or properties) recorded at the respective spatial location.

Arguably the best-known, and widest used, sampling scheme for such volumetric representations is the one using *uniform* samples. That is, sample points $\mathbf{x}_i$ are distributed on a regular lattice covering a compact axis-aligned region of the embedding n-D space. Such samples are known as pixels (for $n = 2$) or voxels (for $n = 3$). The main advantage of this sampling scheme is its simplicity: Essentially, all we need to store is an n-D matrix of sample values; the actual sample coordinates can be next easily inferred from the so-called structured coordinates, or indices, of a sample (pixel or voxel) inside the dataset. The main disadvantage of this uniform sampling scheme is its (very) high memory demands: Essentially, we have a uniform sampling density, thus we need to dedicate the same amount of sample points (per unit of n-D volume) to any zone. However, some zones may contain more interesting information than other zones.

Uniform volumetric sampling is used by many shape processing applications, ranging from the ubiquitously known image viewers that show images acquired *e.g.* by digital cameras, to medical imaging applications that show 3D CT or MRI images acquired by volumetric scanning procedures, that subsequently encode the type, density, or dynamics of tissues in a 3D spatial region.

2.1.1.1 *Binary Volumes*

In many cases, we are not interested in representing, or reasoning about, the internals of a shape Ω, but only about its *boundary* $\partial\Omega$. Typical examples include classical 3D computer graphics, where we want to show a number of surfaces (but usually are not concerned with what lies inside these surfaces); CAD-CAM applications, where we want first to design the boundary of a 3D shape, but are not concerned (at this stage) about the internals of the respective shape; or medical applications in which we want to separate, or segment, a shape having a given morphology (structure) from its surroundings, such as a tumor from enclosing tissue, a hard bone from enclosing soft tissue, or an arterial tree from surrounding less-vascularized tissue; or computer vision applications, where we want to isolate specific shapes of interest, such as cars or pedestrians, from surrounding landscape. In all these cases, we actually want to reason about any point $\mathbf{x}$ in the embedding space (*e.g.* $\mathbb{R}^2$ or $\mathbb{R}^3$) as being *inside*, *on the boundary*, or *outside* one or several shapes of interest.

A simple and efficient way to encode the above point classification is to use binary volumetric sampling, or binary volumes. In such a volume, each sample point essentially has a binary value, indicating whether the respective point is either inside or outside our given shape(s) of interest. Points on the boundary between 'inside' and 'outside', or on the surfaces $\partial\Omega$ of our shapes, can be easily found using this binary classification as *e.g.* all inner points having at least one outside neighbor point, or conversely (all outside points having at least one inside neighbor point).

Binary volumes provide a straightforward way of representing a segmentation in a volume. Each voxel is classified in a binary fashion an the marked voxels usually represent the voxels belonging to the segmented structure. The key disadvantage of this representation, is that the segmentation resolution is limited to the resolution of the volume. However, since voxel volumes in itself sample the space, the actual contours lie somewhere in between the voxel volumes. Implicit surfaces provide means to overcome this limitation. The analysis of digital volumetric shapes, or digital images, is the focus of a specific field of research, called digital geometry [80].

2.1.1.2 *Implicit surfaces*

Although binary volumetric representations offer a way to identify surfaces separating the inside of our shapes of interest from surrounding space, they present a key disadvantage: The represented surface is limited in resolution to the grid used to represent the binary function. In many cases, we have datasets, where representing a surface in this way, results in loss of accuracy. In addition, the representation accuracy is affected by transformations to the original data, such as rotation, scale or translation. Consider for example a 3D CT scan: This is essentially a 3D volume of points $\mathbf{x}_i$ whose scalar values $s(\mathbf{x}_i) \in \mathbb{R}$ represent some shape property, such as *e.g.* tissue density. Although the acquired data is sampled at the discrete positions $\mathbf{x}_i$, the boundaries of the various organs typically lie *in between* these positions. A binary voxel surface representation cannot capture such boundaries, thereby generating potential problems for the further interpretation and/or processing of such surfaces.

Implicit surfaces offer an alternative representation. Consider again our example of a 3D CT scan represented as a volume of points $\mathbf{x}_i$ whose scalar values $s(\mathbf{x}_i) \in \mathbb{R}$ represent some shape property, such as *e.g.* tissue density. We can now define an entire *family* of surfaces $S_j \subset \mathbb{R}^3$ having the tissue density equal to some user-given value $\tau_j \in \mathbb{R}$. These are also often called isosurfaces, contours, or level-set surfaces of the function s [146]. To define such isosurfaces, we must first be able to interpolate the sampled values $s(\mathbf{x}_i)$ to all positions $\mathbf{x}$ in our volume. This is typically done by using trilinear interpolation of the sampled values $s(\mathbf{x}_i)$ at the vertices of cubic cells defined by the sample points $\mathbf{x}_i$. Having the interpolation $s(\mathbf{x})$ at any point $\mathbf{x}$, we can now define $S_j = \{\mathbf{x} \in \mathbb{R}^3 | s(\mathbf{x}) = \tau_j\}$.

Implicit surfaces are a particular class of isosurfaces. Given a 3D shape, consider that we can define a so-called scalar indicator function $f : \mathbb{R}^3 \rightarrow \mathbf{R}$, so that f is negative inside the shape, and positive outside this shape. In this case, and assuming that f varies continuously, the actual surface of the shape is represented by the isosurface for scalar-value zero, or the zero level-set, of the indicator function. Indicator functions can be represented using a sampled voxel representation.

Implicit surfaces and isosurfaces can be extracted efficiently from a sampled volume s using *e.g.* marching cubes or dividing cubes algorithms [100]. The resulting implicit surfaces represent a piecewise-linear (polygonal mesh) approximation of the surface of our shape of interest. Several shape operations such as *union* or *intersection* can be easily computed using implicit representations, as they map to arithmetic and/or logical operations on the respective indicator functions. Other advantages of implicit representations are the ease of generation of a meshed surface representation (convenient for further processing and/or visualization), easy handling of surfaces having complex shapes and/or topologies, and guaranteed orientability and water-tightness of the resulting surfaces. Implicit representations are very popular among medical image segmentation algorithms [113].

2.1.2 *Boundary sampling representations*

Unlike the binary or implicit surface representations, which require an underlying voxel volume to represent the surface, boundary sampling representations encode the shapes surface *explicitly*, i.e. the boundary positions are directly encoded. In case we know that we want to represent, and next reason about, a *single* surface of our shape (rather than *e.g.* a family of surfaces, or the varying material properties of the interior of a volume), the only information we need to encode is the shape of the respective surface.

There exist several ways to represent such surfaces, as follows. Note that, for the sake of simplicity, we next consider only surfaces embedded in $\mathbb{R}^3$. The same reasonings apply to surfaces embedded in other dimensions.

2.1.2.1 *Point cloud representations*

A point cloud representation of a surface $S \subset \mathbb{R}^3$ is essentially a point sampling of this surface; in other words, it is a collection of points $\mathbf{x}_i \in S \subset \mathbb{R}^3$. Such point sets, also called point clouds, can be acquired by computing points on an implicit surface; by computing points on the interface between inner and outer voxels in a binary volume representation; or, more interestingly, by using 3D scanning techniques which acquire points on the visible surface of a set of given 3D objects using a laser scanner or a time-of-flight camera [49, 104]. The key advantage of such point cloud representations is their compactness: we can easily encode high detail present on complex 3D surfaces up to sub-millimeter precision in point clouds having a few million samples with significantly lower cost that encoding the same information in binary voxel volumes. In many cases, such as the use-case outlined above of recording the visible surfaces of 3D natural objects, it is much easier (and cheaper) to construct a surface point cloud (using scanning techniques) than acquire a true volumetric representation of the same object *e.g.* using CT or MRI scanning techniques. A third advantage of point clouds is that they naturally support non-uniform spatial sampling schemes: We can distribute the 3D cloud points any way we want; in contrast, in a typical volumetric sampling, the sample points are organized in a regular lattice or uniform grid. Thus, 3D point clouds are significantly more flexible in recording 3D shapes having a highly non-uniform spatial variation, by *e.g.* allocating more sample points to high-detail or high-frequency areas than to low-frequency, uninteresting, areas.

However, a major limitation of point cloud representations is that they essentially *only* encode a set of samples on our surface of interest, but *not* the shape of the surface as such. In other words, point clouds do not (explicitly) say, or model, what happens between the samples. To have such information, we need to *interpolate* the sample information in-between. This is the area covered by the representations and methods discussed next.

2.1.2.2 *Polygonal representations*

Polygonal shape representations extend the point cloud surface representation by adding connectivity information to the points. From a practical (and historical) perspective, polygonal surface representations are probably the oldest, best known, and most frequently used representations of 3D surfaces. From a theoretical perspective, polygonal representations extend point cloud sample-sets by adding an interpolation mechanism that estimates the surface between sample points, using linear basis functions or interpolants. In simpler terms, this amounts to adding a mesh representation atop of a point cloud that describes a set of polygons (typically triangles) whose vertices lies at the sample point positions, whose union represents our surface of interest, and which do not intersect –

thus, they form a *partition* of the desired surface. Such mesh representations are computationally inexpensive, compact in terms of memory usage, and simple to implement. As such, the vast majority of applications representing or processing 3D surfaces use polygonal representations. To give just an example, the marching cubes implicit-surface extraction algorithm mentioned earlier represents implicit surfaces by triangle meshes.

If a polygonal representation can be constructed for a given point cloud, numerous surface analysis and processing operations can be easily performed on it, *e.g.* normal estimation (by averaging polygon vertices at sample points); smooth shading (using either the linear Gouraud interpolation of shading computed at sample points or the slightly more complex Phong interpolation of surface normals at each surface pixel); surface curvature estimation [176]; surface segmentation [26], and many others. Fundamentally, many of these operations become possible due to the partition of the surface of interest offered by the polygonal representation, and the subsequently easy construction of interpolation mechanisms on this polygonal representation.

We should note that the need for an interpolation mechanism on a surface S that (in turn) interpolates or approximates our point cloud samples $\mathbf{x}_i$ does not directly and always require the construction of a polygonal interpolation (mesh) from $\mathbf{x}_i$. Mesh-less, or grid-less methods have been proposed. Such methods essentially provide most of the shape processing operations targeted by mesh representations, by creating different sets of basis functions than the classical bilinear ones that live on mesh triangles. Examples are constructing local point couplings by using nearest-neighbor projections to local tangent planes [27, 28, 133]. Such couplings essentially define a stiffness matrix, which next enables the direct application of many processing methods to take place on the 'implicit' surface defined by the couplings, *e.g.* anisotropic diffusion, denoising, texture generation, inpainting, or simplified rendering. While the coupling estimation using local tangent planes strongly resembles methods used for reconstructing meshed surfaces from point clouds [1, 195], such mesh-less methods do *not* explicitly deliver a manifold representation of a surface. As such, mesh-less shape processing methods are less attractive than methods that extract an explicit mesh surface from a point cloud and deliver it to any subsequent processing task.

However, given just a raw point cloud of positions $\mathbf{x}_i \in \mathbb{R}^3$, computing a polygonal representation of a surface that interpolates (or approximates) these points is far from trivial. To be useful in subsequent surface processing and/or analysis operations, such a polygonal surface should meet several criteria, *e.g.* be free of self-intersections; contain only cells (polygons) having a non-zero area and good aspect ratio; be orientable; contain a simple manifold structure; or even be watertight (describe a closed volume in $\mathbb{R}^3$ without boundaries). The class of methods aiming at creating such surfaces from 3D raw (or unstructured) point clouds, known also under the name of surface reconstruction methods, is briefly outlined next.

2.1.2.3 *Surface reconstruction*

Given the usefulness of polygonal surface representations outlined above, and the prevalence of 'raw' surface representations in terms of unstructured point clouds, many method have emerged that aim to create polygonal surfaces from such point clouds, satisfying various quality criteria for the generated polygonal mesh (such as polygon aspect ratio, manifoldness, orientability, and watertightness). Such methods are globally known under the name of *surface reconstruction* methods.

Globally speaking, surface reconstruction methods can be classified into (1) approximation and (2) interpolation methods. Given a point cloud $\{\mathbf{x}_i\} \in \mathbb{R}^3$, approximation methods generate a surface S that is globally as close as possible to the points $\mathbf{x}_i$, and also meets a number of global quality criteria, such as the ones stated at the beginning of this section. The degree of freedom of not having to pass precisely through the sample points gives some additional room for optimizing desirable surface properties, such as smoothness, orientability, or polygon size. A salient example of approximation methods is the Poisson surface reconstruction technique of Kazhdan *et al.*, which computes the approximating surface S by minimizing a global distance function between S and the sample points $\mathbf{x}_i$ subject to certain smoothing assumptions [77], or similar global functionals [138]. As it will be discussed later in this thesis, this technique works well for certain point-sampling distributions, but creates unwanted results for highly non-uniform point distributions. Interpolation methods generate a surface S that is guaranteed to pass through the sample points $\mathbf{x}_i$. This creates an overall better control of the result, but imposes implicit constraints on global surface properties. For instance, if $\{\mathbf{x}_i\}$ contains (many) noisy samples, interpolation will necessarily create a noisy surface with limited smoothness. Methods in this class involve 3D Delaunay triangulations and variants thereof, such as the power crust and variations [2, 3]; the ball pivoting method [12]; or local triangulations based on the point cloud covariance matrix [27, 96]. We examine the extraction of surfaces having desirable properties from point clouds, *e.g.* smooth self-intersecting manifolds with boundaries embedded into sampling noise, further in Chapter 4.

2.2 MEDIAL REPRESENTATIONS

In the previous sections, we discussed several representations that encode a shape Ω (or, alternatively, its boundary $\partial\Omega$) by various ways of sampling and reconstruction of the respective surface from data samples. In this section, an alternative shape representation is introduced – the *medial* representation. As we shall see, this representation offers several important advantages compared to more traditional volumetric or boundary representations; however, at the same time, additional challenges are brought on, which have to be addressed.

The medial approach, first introduced by Harry Blum in 1967 [15], captures the boundary $\partial\Omega$ of a shape Ω in terms of its so-called 'symmetry-set' or 'symmetry locus'. That is, instead of explicitly encoding the points on the shape boundary, we encode points which are (in some sense) symmetric with respect to this boundary.

The basic concept underlying the medial representation is, in terms of intuition, quite simple: The shape boundary $\partial\Omega$ is represented as the locus of the centers of maximally inscribed discs ($\mathbb{R}^2$) or spheres ($\mathbb{R}^3$) located inside the shape Ω (that is, maximal disks or spheres fully located inside Ω). Together with the centers $\mathbf{p}_i$ of these disks, we can store their respective radius values $r_i \in \mathbb{R}^+$. The set $\{(\mathbf{p}_i, r_i)\}$ is known under many names – skeleton, medial axis, and medial axis transform. We will use next these terms interchangeably, pointing out to relevant differences in respective contexts. For a more formal definition of these terms, see *e.g.* [154].

Several alternative definitions (and, subsequently, computation methods) have been proposed for the medial axis. One of the most popular is the *grassfire* analogy [94]. Here, one assumes an isotropic (uniform) grass field covering the extent (interior) of our shape Ω. Next, we assume that the full boundary is set on fire at the exact same time t_0, and that the grass burns next with equal speed in all directions and along all points in Ω. The grass points where the fire front clashes at some given time time $t > t_0$ represent the medial axis. As we shall see next, this model can be used to derive several computation techniques for the medial axis, having their intrinsic strengths and limitations. Alongside this definition, alternative definitions exist for the medial axis, each having in turn their advantages and limitations. Such definitions and computation methods are discussed in Section 2.3.

However, the medial axis definition (and, subsequently, computation methods thereof) is not free from problems: Small perturbations on the boundary $\partial\Omega$ lead to large variations on the medial axis. Such issues are discussed in Section 2.3.4.

A separate aspect to mention at this moment is the *goal* behind medial axis computations. While, theoretically, computing medial axes is an interesting (mathematical) problem, the main practical aim we see is, ultimately, using such medial axes to enable the implementation of relevant shape processing operations with a clear practical relevance. As such, we do not see the medial axis as an *ultimate* goal, or product, of our work. Rather, the medial axis should be a simple, efficient and robust to compute, descriptor or tool that allows us to either perform relevant shape processing operations, or compute derived higher-level shape features that ultimately enable us to perform such operations. We shall cover the computation of such derived features and their use to perform useful shape analysis operations in Chapters 5 and 6. In this sense, a (loose) analogy could be made between medial *vs* boundary representations and time *vs* frequency representations of signals: Both representations in the above-mentioned pairs encode the exact same information in a different format and the usability of each representation is dependent on the application. Additionally, since we can convert between representations (boundary *vs* medial axis transform, or space *vs* frequency respectively), we can choose to use whichever representation suits us best for the desired processing operations in a given domain. Our shapes, or input signals, can be always transformed into the other domain as needed.

2.2.1 *Definitions*

Let us next present the formal definitions of medial structures we will be using in the context of this thesis.

2.2.1.1 *2D skeletons and surface skeletons*

Given a 2D or 3D binary shape $\Omega \subset \mathbb{R}^{n\in\{2,3\}}$ with boundary $\partial\Omega$, we first define its Euclidean *distance transform* $DT_{\partial\Omega} : \Omega \rightarrow \mathbf{R}^{+}$ as

$$DT_Y(\mathbf{x} \in \Omega) = \min_{\mathbf{y}\in Y} \|\mathbf{x} - \mathbf{y}\| \tag{2.1}$$

For all definitions in the context of this thesis, Y is set to $\partial\Omega$. For text conciseness purpuses, the $DT_{\partial\Omega}$ notation is used. $DT_{\partial\Omega}$ is called *Euclidean* distance transform as it is based on the Euclidean distance metric $\|\cdot\|$ between points. Other distance metrics induce different distance transforms, see *e.g.* [166]. However, in the context of this thesis, we will use only the Euclidean distance transform, as this is the most used metric for computing medial descriptors in practice.

Intuitively, the distance transform gives us, for each point inside the shape Ω, the minimal distance to any point on the shape's boundary $\partial\Omega$. From the definition, we already see that the distance transform is a positive scalar field which assumes increasingly large values as the point $\mathbf{x}$ is deeper situated in Ω with respect to its boundary $\partial\Omega$. Also, we see that $DT_{\partial\Omega}$ takes zero values on the entire boundary $\partial\Omega$. As we shall see next, $DT_{\partial\Omega}$ is an important ingredient in defining (and computing) the medial axis of Ω.

The skeleton of Ω is next defined as

$$\begin{aligned} S_{\partial\Omega} = \quad & \{\mathbf{x} \in \Omega | \exists \mathbf{f}_1, \mathbf{f_2} \in \partial\Omega, \mathbf{f}_1 \neq \mathbf{f_2}, \\ & \|\mathbf{x} - \mathbf{f}_1\| = \|\mathbf{x} - \mathbf{f_2}\| = DT_{\partial\Omega}(\mathbf{x})\}, \end{aligned} \tag{2.2}$$

where $\mathbf{f_1}$ and $\mathbf{f_2}$ are two of the contact points with $\partial\Omega$ of the maximally inscribed disc in Ω centered at $\mathbf{x}$. The mapping

$$FT_{\partial\Omega}(\mathbf{x} \in \Omega) = \arg\min_{\mathbf{y}\in\partial\Omega} \|\mathbf{y} - \mathbf{x}\| \tag{2.3}$$

also called the *feature transform* (FT) of the boundary $\partial\Omega$ [66, 166], contains all boundary points $\mathbf{y}$ that are at minimal distance from any interior point $\mathbf{x}$. These points $\mathbf{y}$ are also called feature points. The vectors $\mathbf{y} - \mathbf{x}$ are also called spoke vectors, in analogy to the spokes of a wheel (whose circumference is represented by the boundary $\partial\Omega$) [164, 165].

Note that $FT_{\partial\Omega}$ is multi-valued – or, in other words, its co-domain is the power set of $\mathbb{R}^3$. Indeed, an inscribed ball can have at least two, but potentially many more, contact points with $\partial\Omega$, depending on the actual position of the ball center $\mathbf{x} \in \Omega$. For instance, if Ω is a disk, the value for $FT_{\partial\Omega}$ for the circle center will be the entire circle boundary $\partial\Omega$. As we shall see next in Chapter 6, the cardinality of $FT_{\partial\Omega}$ allows us to reason about the (local) variations of the boundary $\partial\Omega$.

Skeletons defined by Equation 2.2 have an interesting intrinsic relation to the dimension n of the space embedding our shape Ω. That is, if $n = 2$, skeletons will be sets of 1D curves; these are called 2D skeletons (with reference to the embedding dimension $n = 2$), medial axes, or even simpler, skeletons. If $n = 3$, skeletons will be sets of 2D manifolds (surfaces). Such skeletons are typically called surface skeletons or medial surfaces. Note that in the trivial case $n = 1$ (our shapes are compact 1D line segments), skeletons will be sets of isolated points. As such, the skeleton dimension is one less than the dimension of the space embedding Ω. For $n = 3$, this already highlights an inherent problem concerning the accurate and efficient computation and analysis of such skeletal descriptions, which are sets of complex 2D manifolds. Obviously, the complexity of such skeletons will only increase with the size of the dimension n. Since the case $n = 3$ is challenging enough, this thesis will not focus on skeletons of shapes in higher dimensional spaces such as given by $n > 3$.

The skeleton defined by Equation 2.2 is *homotopic* to the input shape Ω – that is, it preserves the topological elements of the input shape [126, 154, 168]. For instance, if the input shape is of genus g, then the skeleton also has to be of genus g. The definition results in a connected skeleton, as long as the input shape Ω consists of a single connected surface. In the case that Ω consists of multiple disconnected components, the solutions presented for computing the the skeletons still hold, however the discussion of homotopy becomes implicitly more complex[185][186]. In the context of this thesis, unless explicitly expressed otherwise, the input shape Ω is considered to consist of one and only one connected component.

Having the skeleton of a shape, the medial axis transform (MAT) $M_{\partial\Omega}$ of a shape Ω can be next defined as a function that associates, to each skeletal point $\mathbf{x}$, the minimal distance of $\mathbf{x}$ to the boundary $\partial\Omega$. This distance is equal to the value of the distance transform $DT_{\partial\Omega}$ at point $\mathbf{x}$. Thus, we can define the MAT as

$$MAT_{\partial\Omega}(\mathbf{x} \in S_{\partial\Omega}) = (\mathbf{x}, DT_{\partial\Omega}(\mathbf{x})). \tag{2.4}$$

An important property of the MAT is that it provides a full encoding of the shape described in $\partial\Omega$. Indeed, given the MAT of a shape embedded in $\mathbb{R}^n$, we can reconstruct the respective shape Ω as being the union of n-dimensional balls centered at the MAT points and having as radii the distance values provided by the respective MAT points. In other words

$$\Omega = \bigcup_{\mathbf{x} \in S_{\partial\Omega}} B(\mathbf{x}, DT_{\partial\Omega}(\mathbf{x})). \tag{2.5}$$

Here, $B(\mathbf{x}, \rho > 0)$ denotes an nD ball centered at $\mathbf{x}$ and of radius ρ. Subsequently, given the reconstruction of the shape Ω, we can easily find its boundary $\partial\Omega$, if so desired.

This equivalence of a shape with its medial axis transform is an important element for the work presented in this thesis. In brief, it suggests that shape analysis and processing operations that are traditionally defined on a surface $\partial\Omega$ can be adapted to work on the medial axis transform representation $MAT_{\partial\Omega}$. As we shall see, this transposition is possible, and can lead to several interesting insights and practical advantages.

2.2.1.2 *Curve Skeletons*

As we have seen above, surface skeletons are relatively complex shapes consisting of a set of intersecting manifolds with boundaries. Both researchers and practitioners have since long recognized many of the computational and practical challenges of using such skeletons in real-world applications, where the input shapes are embedded in 3D. We discuss several such challenges next in Section 2.3.

As such, other simpler and thus more practical descriptors of 'centrality' for 3D shapes have been proposed. One such descriptor type is formed by *curve skeletons*. Topologically speaking, curve skeletons $C_{\partial\Omega}$ are sets of 1D curves. Given their lower dimensionality as compared to surface skeletons, curve skeletons are simpler to represent, analyze, and use. However, in contrast to the formal definition of surface skeletons (Eqn. 2.2), curve skeletons know several definitions in the literature. Apart from their dimensionality (sets of 1D curves), such definitions have in common the idea that curve skeletons should be 'locally' centered within the shape Ω, in analogy to the local centeredness of surface skeletons. However, there are many ways in which one can define curves that are locally centered within a shape. As such, many curve skeleton definitions and computation methods have emerged. Since there is no uniquely accepted formal definition of curve skeletons, researchers have tried to characterize these by a number of qualitative 'desirable properties' [32, 158]. Such properties include

- *centeredness:* Each point $\mathbf{x} \in C_{\partial\Omega}$ should be centered with respect to a neighborhood of points $\mathbf{y} \in \partial\Omega$ that is close to $\mathbf{x}$;
- *thinness:* Curve skeletons $C_{\partial\Omega}$ of 3D shapes are sets of 1D curves;
- *topology:* The topology of curve skeletons $C_{\partial\Omega}$ should capture the topology of the input shape Ω. More specifically, $C_{\partial\Omega}$ should be homotopic to Ω;
- *transformation invariance:* Given an isometric transformation $T : \mathbb{R}^3 \rightarrow \mathbb{R}^3$, such as translation, scaling, or rotation, the skeleton should be invariant to it. That, is $C_{T(\partial\Omega)} = T(C_{\partial\Omega})$, for any such T.

Other desirable properties of curve skeletons are mentioned such as computational stability in the presence of noise on the input shape, computational efficiency, and minimal thickness. While important, such additional properties are related to choices involving space discretization and computation algorithms, and therefore not key to our discussion here on the *definition* of curve skeletons. Separately, note that all above desirable properties apply as well to 2D medial axes and 3D surface skeletons.

From a definition viewpoint, two additional noteworthy approaches should be mentioned. One of the first formal and explicit definitions of curve skeletons is the locus of points $\mathbf{x} \in \Omega$ which admit at least two (equal-length) shortest paths, or geodesics on $\partial\Omega$, between their feature points [37, 125, 135]. This definition is equivalent to stating that the curve skeleton of a shape Ω is the ridge set, or local maxima in at least one dimension of the so-called medial geodesic function $MGF : \mathbb{R}^3 \rightarrow \mathbb{R}^+$, defined as the length of the shortest path on $\partial\Omega$ determined by any two feature points of the point $\mathbf{x}$. By definition, MGF can be only computed on the surface skeleton $S_{\partial\Omega}$, since points $\mathbf{x} \notin S_{\partial\Omega}$ have a single feature point, thus cannot allow constructing the above-mentioned shortest paths. This leads to the implicit additional property that curve skeletons are contained in the surface skeletons of the respective shapes, or $C_{\partial\Omega} \subset S_{\partial\Omega}$. This observation has been further exploited in a second, separate, definition of curve skeletons as the ridge set of the distance transform $DT_{\partial S_{\partial\Omega}}$, which measures the shortest (geodesic) distance from any point $\mathbf{x} \in S_{\partial\Omega}$ to the boundary $\partial S_{\partial\Omega}$ of the surface skeleton [181]. Several relations between curve and surface skeletons, as computed by a number of recent methods, are discussed in [159].

Due to their simple structure, curve skeletons are useful in a variety of applications such as shape matching, shape registration, path planning, 3D metrology, and virtual navigation [32]. In particular, curve skeletons are effective shape descriptors for *tubular* objects, *i.e.* objects which can be well described by the extrusion of a (near) circular contour along a set of 3D curves. Such objects occur in many applications, *e.g.* blood vessel trees obtained from CT or MRI scanning techniques, intestine or colon structures obtained by similar scanning procedures, or plant branches or roots, measured by laser scanning or vision techniques. However, compared to surface skeletons, curve skeletons do not admit the definition of a MAT that would allow the perfect reconstruction of the input shape Ω. As such, it is arguable that many surface analysis and processing operations can *not* be fully translated to a shape representation involving only curve skeletons, but require the richer surface skeletons. This is an additional justification for the subsequent focus of this thesis on surface skeletons.

2.2.2 *Classification of Medial Points*

As outlined earlier, 2D skeletons are formed by sets of 1D curve segments that intersect at a number of points (also called junctions); 3D medial surfaces are formed by sets of 2D surface manifolds with boundaries, that intersect along several curves (also called Y-intersection curves [23, 34, 93]; and curve skeletons are formed by sets of 3D curve segments that intersect at a number of junction points. As such, we can say that skeletons (of all types discussed so far) exhibit a high amount of *structure*. Computing and understanding this structure and its relationships with the input shape Ω is of great use in analyzing and processing shapes by using their medial descriptors. We give below a brief overview of several definitions and results in this direction. For additional information, we refer to [23, 34, 93, 154].

In two dimensions, the medial axis $S_{\partial\Omega}$ of a shape $\partial\Omega$ consists, in the generic case, of a collection of 1D curve segments. Each such segment has thus two end points. These can be next classified as 'terminal' points, also called tip points or tips; and intersection points, or junctions. *Tips* are end points which are shared by a single skeleton branch. They form, thus, the boundary $\partial S_{\partial\Omega}$ of the medial axis. It can be next shown that tips map, by means of the feature transform, to fragments of the boundary $\partial\Omega$ where this boundary exhibits local curvature maxima. These correspond to convex areas, or 'cusps', on the shape boundary. For instance, consider a 2D rectangle shape: Its skeleton has four tip points, which precisely map, via the feature transform, to the four corners of the rectangle. If we consider the same rectangle, but slightly 'round off' its four corners, *e.g.* by replacing each boundary fragment centered at a corner by a small quarter-circle, the resulting skeleton will still have four tips. However, these tips will now map, via the feature transform, to four quarter-circle fragments on the boundary $\partial\Omega$, capturing the boundary regions represented by the rounded corners. *Junctions* are points where three or more curve fragments of $S_{\partial\Omega}$ meet. They correspond, intuitively, to points where several distinct parts of the shape Ω get joined together. Overall, 2D skeletons $S_{\partial\Omega}$ can be thus represented as graphs whose nodes encode the separate skeleton curve segments (or skeleton branches), and edges represent the adjacency relations of branches meeting at a junction point (or conversely). Node and/or edge weights can be added to represent geometric properties measured on the shape elements corresponding to the respective skeleton fragments or junctions, such as (average) distance-to-boundary, curvature, or length. Such graphs thus represent a compact, though not lossless, encoding of the main properties of the analyzed shape. They are frequently used to implement shape retrieval and shape matching operations in terms of graph comparison operations [42, 102, 169].

In three dimensions, understanding the structure of medial surfaces becomes a considerably more complex problem. One way to tackle this higher complexity is to classify the medial points $\mathbf{x} \in S_{\partial\Omega}$ based on the so-called order of contact of maximally inscribed spheres centered at these points with the boundary $\partial\Omega$. This is, loosely speaking, equivalent to studying the values that the feature transform $FT_{\partial\Omega}(\mathbf{x})$ takes at such points. Giblin and Kimia proposed such a classification, where each medial point is denoted as being of type A_k^n is introduced, where n corresponds to the number of different k-fold tangencies and k to the contact order [58]. Note that this classification is equivalent to analyzing the number n of disjoint groups that $FT_{\partial\Omega}(\mathbf{x})$ consists of, and the dimensionality k of each such group respectively.

Following this classification, a medial surface $S_{\partial\Omega}$ can be decomposed into the following point types:

SHEETS

Medial sheets characterize the two dimensional elements of a medial surface – thus, each medial sheet is a manifold with boundaries. For typical shapes (excluding spheres and tubular shapes), the vast majority of medial points are located on medial sheets. Medial sheets contain A_1^2 medial points, *i.e.* are the locus of maximally inscribed spheres which touch the surface $\partial\Omega$ at exactly two different points. In other words, these medial points have each a feature transform consisting of exactly two different points on $\partial\Omega$.

CURVES

Curve points are located on the boundaries of the medial sheets. Two types of curve points can be next identified, depending on the boundary types. First, we have curves characterized by A_3 points. These curves are the locus of maximally inscribed spheres which have a single contact zone with $\partial\Omega$, which represents a surface fragment of $\partial\Omega$. Such curves represent the ‘open’ boundary of sheets. Their union creates the boundary $\partial S_{\partial\Omega}$ of the surface skeleton. The points on $\partial\Omega$ to which such boundary curves map correspond to the maxima of the surface curvature of $\partial\Omega$ – or, more informally, to convex edges of $\partial\Omega$. A_3 points are thus the 3D

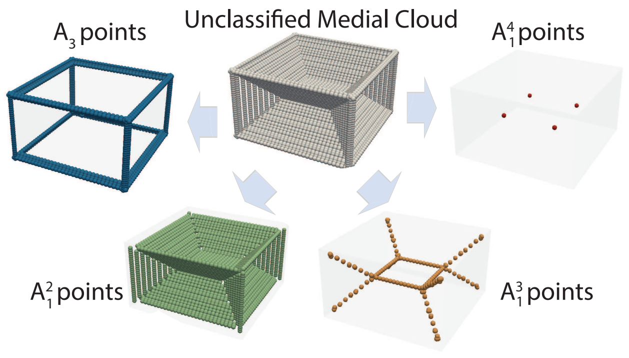


Figure 2.1: Medial cloud classification into different point types.

analogon of tip points for 2D skeletons. Secondly, we have curves characterized by A_1^3 points. These are thus the locus of centers of maximally inscribed balls which have three distinct tangent points with $\partial\Omega$. These curves occur at the intersection of three medial sheets, and are thus also called sometimes Y-intersection curves. They are the 3D equivalent of junction points of 2D skeletons. In other words, these curves contain points whose feature transform yields three distinct points on $\partial\Omega$.

POINTS

The medial curves defined above can be further characterized by studying their end points. Two types of curve end points can be defined, as follows. First, A_1^4 points are defined as the intersection of four A_1^3 curves meet. These are thus centers of maximally inscribed spheres which have four different contact points with $\partial\Omega$. Loosely put, we can think of them as the 'internal corners' of the medial sheets. Secondly, A_1A_3 points are defined as the intersection of an A_3 and an A_1^3 curve – or alternatively, the end points of A_3 curves. They are the centers of maximally inscribed spheres having one regular tangency point and one higher-order tangency contact zone with $\partial\Omega$. They can be loosely seen as the 'external corners' of the medial sheets. These points map, by means of the feature transform, to corners of $\partial\Omega$, where several surface edges meet.

The above point classification of medial surfaces has many important uses. First, it allows us to decompose (and reason about) the complex structure of a medial surface in terms of simpler separated elements and relationships thereof. For instance, the set of A_1^3 curves forms the so-called Y-network of a surface skeleton, which defines how different shape parts (characterized by A_1^2 sheets) are joined together. More generally, the entire set of sheets, curves, and curve endpoints, and the relations between them, can be described by a graph structure, also called the medial scaffold [93]. Analyzing this graph structure supports a number of important operations such as shape segmentation [23]. In Chapter 6, we shall show how such point classifications can be efficiently computed for large surface skeletons represented as point clouds, and how they can be subsequently used to support a variety of shape analysis and processing operations.

2.3 AN OVERVIEW OF SKELETONIZATION TECHNIQUES

In the previous sections, we have provided an overview of the main definitions and concepts associated with medial descriptors. Separately, Section 1.4.2 presented several classes of applications where such descriptors are useful.

However, to make medial descriptors – that is, both the 'raw' skeletons and their derived properties such as feature transforms, MATs, and skeletal point classifications – useful and usable in practice, we need efficient and effective ways to compute such descriptors from a given shape representation. Methods that compute medial descriptors given an input shape are called medial representation extraction methods, or more briefly, skeletonization methods. In this section, we provide a compact overview of the main classes of skeletonization methods known in the literature, together with several example algorithms in each class.

In line with the shape representation models discussed earlier in Sections 2.1.1 and 2.1.2, we can classify skeletonization methods into volumetric ones and boundary-based ones. These two classes of methods are discussed next in Sections 2.3.1 and 2.3.2 respectively. In these sections, we focus on the extraction of skeletons which follow the canonical definition given by Equation 2.2, *i.e.*, 2D medial axes and 3D surface skeletons. 3D curve skeletons represent a particular case, due to the strong dependence between the extraction method and the underlying curve-skeleton definition that the respective method aims to support (see Section 2.2.1.2). As such, we discuss such methods separately in Section 2.3.3. A separate important computational concern is the problem of regularization, or elimination of unwanted details and/or noise from the extracted skeletons. Since regularization can be seen as a crosscutting concern for virtually all skeletonization methods, we discuss it separately in Section 2.3.4.

2.3.1 *Volumetric Methods*

As suggested by their name, volumetric (or voxel-based) methods start with a voxel representation of the input shape Ω. Typically, this comes as a binary volume, with foreground voxels describing locations inside Ω, and background voxels describing locations outside Ω respectively. Note that, in this discussion, we use the term 'voxel' mainly because our interest is in computing 3D surface skeletons. However, our characterization of volumetric methods also coves the 2D case, where the equivalent term is 'pixel'. To represent the computed skeleton $S_{\partial\Omega}$, we also have two options: use a voxel representation (analogously to the one capturing the input shape), or use other representations, such as boundary sampling. Most, though not all, methods that use a voxel-based representation of Ω will subsequently also use a voxel-based representation for $S_{\partial\Omega}$. This has the advantage of simplicity – a single shape representation is used throughout the computational pipeline. However, as we shall see, this introduces several practical and conceptual challenges. In contrast, methods that use a voxel-based representation for Ω but a boundary-sample representation for $S_{\partial\Omega}$ avoid such challenges. However, they are more complex to implement, as the computational pipeline has now to deal with both volumetric and boundary-sampling representations.

Volumetric skeletonization methods can be in turn broadly divided into two categories: Morphological thinning and distance field based methods. These two categories are discussed next. For a recent overview of volumetric skeletonization methods, we refer to [159].

2.3.1.1 *Morphological thinning*

Morphological thinning approaches are among the first known skeletonization methods. Their idea is simple: Given a shape Ω represented by the set of foreground voxels, the method iteratively removes so-called boundary voxels (foreground ones which have at least one 4-connected (in 2D) or 6-connected (in 3D) background voxel). Thereby, the shape Ω is effectively 'thinned', until we obtain its skeleton $S_{\partial\Omega}$. During this process, two elements are essential: First, voxels should be removed in an *order* which ensures that the resulting skeleton is centered within the shape Ω, thereby satisfying the definition given by Equation 2.2. Secondly, voxel removal should *stop* at some point – otherwise, we obtain $S_{\partial\Omega} = \varnothing$. The stopping criterion has to accommodate further constraints:

- the skeleton is a 'thin' $n-1$ dimensional subspace of the space $\mathbb{R}^n$ in which the input shape Ω is embedded (see Section 2.2.1). If we use a volumetric representation for $S_{\partial\Omega}$, the skeleton should be as thin as possibly allowed by this representation, *e.g.*, a one-pixel-thick curve for $n = 2$, respectively a one-voxel-thick surface for $n = 3$. Thus, thinning should not stop too early, otherwise we obtain a 'thick' skeleton.

- the skeleton is homotopic to the input shape Ω (see Section 2.2.1). Thus, thinning should stop early enough, otherwise we may disconnect the skeleton, thereby altering the homotopy property.

Many thinning methods exist which enforce the above desirable criteria in various ways. Tools from mathematical morphology [145] were among the first used to compute curve skeletons by thinning. The residue of openings, based on Lantuéjoul's formula [92], usually leads to disconnected skeleton branches. Methods based on homotopic thinning transformations yield thin and connected skeletons [13, 107, 119, 120]. This is typically done by using suitable local filters, or *templates*, that check whether the removal of a voxel does not change the thinned shape's topology. Such templates are typically very small in size, *e.g.* 3^n up to 5^n voxels. Centeredness can be helped by removing boundary voxels in the order given by their distance-to-boundary value, *i.e.*, in terms of the distance transform $DT_{\partial\Omega}$ [4, 126, 171].

Thinning approaches present several advantages based on their algorithmic simplicity: They are relatively simple to implement and are relatively fast when compared to alternative methods (further explored below): Although *naive* thinning algorithms visit the volume at each iteration, resulting in expensive implementations, approaches based on sorted queues cut the timings to $O(N)$ where N represents the number of voxels. Also, they can readily accommodate the computation of all three skeleton types discussed so far (2D medial axes, 3D surface skeletons, and 3D curve skeletons), by suitable choices of the local thinning filters being used. However, they have a key challenge: The skeleton $S_{\partial\Omega}$ is represented on the same volumetric grid as the input shape Ω. As such, there are situations when this grid is unable to capture locations that are at equal distance from two boundary points. For instance, consider a simple example of a 2D axis-aligned rectangle of width equal to an even pixel count: There is no pixel-grid location which is at *equal* distance from both the left and right vertical rectangle edges. As such, thinning methods typically have to somehow relax the skeleton centeredness (thus, relax the definition in Equation 2.2), or alternatively compute skeletons which are not pixel or voxel thin. As a consequence of this, skeletons computed by thinning can be sensitive to isometric transformations such as rotation, translation or scaling.

2.3.1.2 *Field-based skeletonization*

Field-based skeletonization methods find $S_{\partial\Omega}$ as the singularities of a given scalar or vector field $\mathbf{f} : \mathbb{R}^n$ defined over the shape Ω. As such, they work by first computing a (typically volumetrically-sampled) representation of $\mathbf{f}$ over Ω, and next detecting the desired singular points to form the skeleton. The main differences between sub-methods in this class amount to the specific choice of field and type of singularity to compute.

The arguably simplest field-based method detects $S_{\partial\Omega}$ as the ridge points of the distance transform scalar field $DT_{\partial\Omega}$. These correspond, indeed, to centers of maximally inscribed disks in the shape. Intuitively, if we consider the $n = 2$ case, and visualize the graph of $DT_{\partial\Omega}$ as a height plot, then $S_{\partial\Omega}$ corresponds to the ridges of this graph; these are precisely the locations where fire fronts originating at different boundary points would meet, following the grassfire analogy. One advantage of this method is that computing $DT_{\partial\Omega}$ can be done very efficiently in both 2D or 3D. For this, we can use *e.g.* the fast marching method (FMM) [146], which finds $DT_{\partial\Omega}$ as the solution of the Eikonal equation $\|\nabla DT_{\partial\Omega}\| = 1$ with boundary conditions $DT_{\partial\Omega}(\mathbf{x} \in \partial\Omega) = 0$. The complexity of this method is roughly $O(\|\Omega\| \log \|\partial\Omega\|)$, where $\|\cdot\|$ indicates number of voxels in the respective structure. Distance transforms can be also computed with $O(\|\Omega\|$ cost [106]. Such algorithms can be efficiently parallelized on graphics processing units (GPUs) [20, 40, 40, 166]. However, detecting ridges of $DT_{\partial\Omega}$ can be a very sensitive process. In the best case, fronts meeting at such ridges come from opposite directions, so the ridge angle (90 degrees) is easy to detect, *e.g.* by using an edge detector on the distance image. In a similar setting, Stolpner *et al.* find skeleton voxels as the points where the gradient of the shape's distance transform is multi-valued [164, 165]. However, in all the above approaches, we can see that fronts can meet at arbitrarily low angles along a so-called *ligature* skeleton branch, yielding thus very low ridge angles which are hard to find [57, 78]. Missing such points will thus disconnect the detected skeleton into multiple fragments.

One solution to this problem is to explicitly find the skeleton as the locus of points where the feature transform (Equation 2.3) has more than a single boundary-point as value. As discussed above for thinning methods, this should be done with care, since we are essentially testing distance-to-boundary equality to capture a singularity (the skeleton) on a fixed grid. One way to relax the issues induced by the fixed grid is to compute so-called *tolerance-based* distance and/or feature transforms [128]. These amount to replacing the strict equality relations present in the right hand sides of Equations 2.2 and 2.3 by distance comparisons within the range of a (small) user-given tolerance value $\tau > 0$. Tolerance-based feature transforms are used to compute surface skeletons as voxels in Ω that have at least two different feature-points on $\partial\Omega$ [135]. A different approach that detects skeletons by using interval arithmetic to evaluate Equation 2.2 by the so-called integer medial axis (IMA) [66]. The IMA can be efficiently computed with $O(\|\Omega\|)$ cost. A third solution is to use a different field than the distance transform, so that singularities of that field can be found more robustly. Many such fields have been proposed, *e.g.* second-order moments [140] or divergence-based detectors [139, 155]. While such methods can alleviate some of the difficulties of using the distance transform as 'raw' detector, they also use relatively smoother fields, and introduce the question of finding the right threshold that detects skeleton points, which make voxel-precise skeleton localization harder. To tackle this issue, one can detect a conservative skeleton detection [168]: first, we find a superset of points $S_{\partial\Omega} \subset X$ that contains the skeleton and in the same time is reasonably close to the exact skeleton location; next, the exact skeleton is found by homotopical thinning of X, using one of the thinning methods discussed earlier.

Overall, field-based methods are less sensitive to local decisions than comparable thinning methods, and have a comparable computational complexity. Also, depending on the field being used, they are less sensitive to isometric transformations than thinning methods. However, they suffer from the same centeredness issues implied by the fixed discretization grid. They also introduce issues related to detection sensitiveness to noise and/or small details. The latter type of issues is discussed separately in the context of regularization (Section 2.3.4).

2.3.2 *Boundary sampling methods*

In contrast to the volumetric approaches discussed in Section 2.3.1, boundary-sampling approaches represent the input shape Ω by a sampling of its boundary $\partial\Omega$. This follows the techniques discussed earlier in Section 2.1.2 for representing contours in 2D and surfaces in 3D: The boundary $\partial\Omega$ is represented either as a point cloud (possibly including per-point surface-normal information), or, more generally, as a 3D unstructured mesh. The resulting skeleton $S_{\partial\Omega}$ can be represented, just as in the case of volumetric methods discussed earlier, by a boundary sampling (thus, using the same representation as for the input shape), or by a volumetric model (thus, using a different representation than the input shape). If both the input shape and its skeleton are represented as a boundary sampling, this leads to two important advantages (as opposed to volumetric-only or volumetric-boundary mixed representations): (1) The *memory* requirements of such a representation are significantly lower than for any other representation (see [159]; (2) Space can be freely sampled, *i.e.*, we can place sample points for *both* the input surface and resulting skeleton at any desired location and/or with any desired density (up to the machine precision of floating-point numbers).

We overview next several subclasses of skeletonization methods based on boundary-sampling representations.

2.3.2.1 *Voronoi-related methods*

The methods in this class are based on the key observation that, given a set of points $\mathbf{x}_i \in \mathbb{R}^n$, which are supposed to (densely) sample a shape boundary $\partial\Omega \subset \mathbb{R}^n$, the skeleton $S_{\partial\Omega}$ is a subset of the n-dimensional Voronoi diagram of the points $\{\mathbf{x}_i\}$. Thus, to compute skeletons, we need to (a) compute a Voronoi diagram, and (b) have a procedure to select relevant subsets of this Voronoi diagram.

For (a), many methods exist for both the 2D and 3D cases, see *e.g.* [11, 153]. These methods require a mesh version of the input shape and deliver the Voronoi diagram thereof also as a mesh, thus, fall within the class of boundary-sampling methods discussed here. For (b), one of the best known approaches is the *power crust* method [2], which finds skeletons as the subset of Voronoi-diagram elements which correspond to balls located (deeply) inside the surface described by $\{\mathbf{x}_i\}$. In 2D, a method to compute medial axes from Voronoi diagrams was presented by Ogniewicz and Kubler [118] – this method is discussed in more detail in the context of skeleton regularization in Section 2.3.4. Other related methods use edge collapses [95], starting from a mesh segmentation [76], or sphere sweeping [112].

Voronoi-based methods have the large appeal of being able to use existing formalisms (and technology) to compute skeletons atop of Voronoi diagrams. Several works have shown that, as the sampling density of $\partial\Omega$ increases, the resulting skeletons converge to the true skeletons of $\partial\Omega$ [2, 144]. Separately, the centeredness problems discussed in the context of volumetric methods are largely inexistent for Voronoi-based methods, since we can place both input-surface points and output-skeleton points at any location in $\mathbb{R}^3$ up to machine precision. However, Voronoi-based methods also come with several challenges. First, computing an accurate Voronoi diagram for a large and complex 3D shape is challenging both from computational and implementation viewpoints. Computational geometry algorithms involved are far from trivial. To give just a simple example, the C code of the 2D Voronoi algorithm in [153] is about 14K lines, whereas the code required to implement 2D IMA [66] is under 500 lines. Limit cases, *e.g.* involving nearly-identical or nearly collinear points can cause significant algorithmic trouble. Separately, such algorithms are far harder to parallelize than volumetric algorithms discussed earlier [167].

2.3.2.2 *Point-cloud approaches*

An alternative proposed to tackle the computational and implementation complexity of Voronoi-based approaches is to directly apply the skeleton definition (Equation 2.2) to the point cloud representing the sampling of the input surface $\partial\Omega$. Putting it simply: If we are able to determine the centers $\mathbf{x}_i$ of maximally inscribed balls in the volume Ω, we have been able to compute a point-sampling of the skeleton $S_{\partial\Omega}$. If this can be done (a) avoiding the heavy implementation costs of computing complex Voronoi diagrams and next selecting suitable subsets thereof, and (b) keeping the sampling advantages of boundary-based representations, then we have simple, efficient, and accurate methods to compute skeletons. Additionally, if we represent $\partial\Omega$ by a simple point cloud (rather than a mesh), then many subsequent implementation complexities would arguably disappear.

Interestingly, there are very few methods that we are aware of in this class. One salient method in this class was presented by Ma *et al.* [101]. Given an oriented point-cloud sampling of $\partial\Omega$, they compute, for each input point, a maximally inscribed ball. Key to this computation is the efficient search of a maximally inscribed ball tangent at the search point, which translates to an efficient nearest-neighbor search on the input point cloud.

One major advantage of this computational scheme is the *per-point* independence of each medial ball computation, meaning that medial points can be computed in parallel. This makes the proposed method highly parallelizable. A parallelization solution involving GPUs is presented. Yet, the per-input-point costs for determining maximally inscribed balls can largely differ. This in turn causes serious performance issues in a SIMD (single instruction, multiple datastream) parallelism model, such as the one offered by typical modern GPU programming platforms (OpenCL or CUDA). These issues are well known under the name of 'thread divergence' in GPU programming.

A second issue with point-cloud approaches is that they provide only an unstructured point sampling of $S_{\partial\Omega}$. While this representation may be sufficient for performing certain tasks, such as shape metrology or visual reconstruction (as we shall next see in Chapter 3), performing more complex analyses or operations that involve reasoning about the skeleton-point connectivities is not possible using just a point cloud. We examine the issue of recovering such missing information in Chapters 3 and 6.

2.3.3 *Curve skeleton methods*

As outlined in Section 2.2.1.2, 3D curve skeletons do not have an universally accepted definition. Therefore, many algorithms exist for computing them. Interestingly, each such algorithm *implicitly* provides its own definition of what a curve skeleton is. Therefore, the task of deciding what is the 'right' curve skeleton for a given 3D shape (and next, how to compute it) is far from trivial.

Early methods compute curve skeletons by thinning, or eroding, a voxel representation of the input shape Ω in the order of its distance transform, until a connected 1D voxel curve is left [10, 119]. Thinning can also be used to compute so-called meso-skeletons, *i.e.* a mix of surface skeletons and curve skeletons [98]. Curve skeletons can also be computed as an intersection of 2D medial axes computed from axis-aligned slices of a 3D voxel shape – thereby reducing the problem of curve skeletonization to a 2D mexial axis extraction [182]. In a volumetric setting, other methods involve finding and connecting the local maxima of the input shape's distance transform, with explicit restrictions that the resulting object should be a curve [14, 192].

For mesh-based models, alternative techniques collapse the input mesh describing $\partial\Omega$ along its surface normals under various constraints required to maintain its quality [7]. The result captures a point-sampling of the curve skeleton of Ω. Hassouna *et al.* present a variational technique which extracts the curve skeleton by tracking salient nodes on the input shape Ω in a volumetric cost field that encodes centrality [64]. As such, this method can be seen as a hybrid between volumetric methods (used to compute the centrality metric) and boundary-representation methods (used to describe the structure of the resulting curve skeleton). Tagliassacchi *et al.* compute curve skeletons as centers of point-cloud projections on a cut plane found by optimizing for circularity [172]. Similarly, the ROSA method finds curve-skeleton points as the centers of local point-cloud projections of $\partial\Omega$ under a constraint for circularity optimization [172]. An extensive comparison of curve skeletonization methods is presented in [158].

Curve skeletons can also be extracted by collapsing a previously computed surface-skeleton towards its 'center' using different variants of mean curvature flow [19, 174, 181]. This class of methods builds upon the assumptions that (a) the curve skeleton is a subset of the surface skeleton; and (b) the curve skeleton can be seen as a'local center' of the surface skeleton with respect to a distance metric between a surface-skeleton point and the surface-skeleton boundary $\partial S_{\partial\Omega}$. As such, these approaches see the computation of curve skeletons as a 'recursive' skeletonization operation: First, we compute the surface skeleton from the input shape Ω; next, we compute the curve skeleton as the skeleton of the surface-skeletion (under the same distance metrics, albeit defined over a different space). The main strength of these approaches is precisely their recursive approach: Given a boundary, space, and distance metric, skeleton definitions are the same (for surface and curve skeletons). The main practical problem of these approaches is implementation complexity: To reason about *e.g* the curve skeleton of a shape, we need to (a) have a robust and efficient way to compute the boundary $\partial S_{\partial\Omega}$ of a surface skeleton (*i.e.*, the union of curves of A_3 points described in Section 2.2.2); and (b) have a robust and efficient way to compute shortest paths (geodesics) over $S_{\partial\Omega}$ from any internal point to its boundary.

Curve skeletons can also be defined *directly* based on a representation of $\partial\Omega$. A pioneering method in this direction is presented in [37]. As outlined in Section 2.2.1.2, they define the curve skeleton of a shape $\partial\Omega$ as the locus of points in $S_{\partial\Omega}$ which maximize the length of the medial geodesic function (MGF), defined as the length of the longest shortest-path between any two feature points. The MGF metric is next evaluated using a mixed volumetric and boundary-sampling approach to yield curve skeletons. Based on this idea, an improvement is proposed in [135], in a pure volumetric setting, to compute curve skeletons as the locus of surface-skeleton points that admit two equal-length longest shortest-paths between all pairs of their feature points. Both above approaches are remarkable as they (a) compute curve skeletons which visually agree with curve-skeletons computed by completely different methods; (b) provide a formal definition of curve skeletons, rather than saying that curve skeletons are the output of a given (iterative) algorithm; but (b) cannot provide a formal reason of why the commonly used geodesic metric is a 'good' criterion for detecting curve skeletons.

2.3.4 *Regularization Methods*

We have seen that both volumetric and boundary-representation methods (for both the input shape Ω or its skeleton $S_{\partial\Omega}$) are inherently sensitive, up to various degrees, to sampling resolution. As such, the practical question of skeleton-computation stability arises.

We can refine this question on both theoretical and practical grounds. On the theoretical ground, assume we have to shapes $\partial\Omega_1$ and $\partial\Omega_2$ which are very similar. More formally, we can quantify the shape similarity using *e.g.* the Haussdorff distance between the two shapes [11]. The question is then: Would similar shapes (under the above-defined distance metric) yield similar skeletons (under the same distance metric)?

The answer is, unfortunately, negative. Consider the operator, or function, $\mathscr{S}$ that, given a shape $\partial\Omega$ computes its skeleton $S_{\partial\Omega}$. We can easily argue that the function $\mathscr{S}$ is not continuous in the definition of Cauchy (or Weierstrass) continuity, also called the $\varepsilon - \delta$ criterion. Indeed, if $\mathscr{S}$ were continuous at *any* point $\mathbf{c}$ in the definition domain (thus, any possible shape), then we would have that for any real value $\varepsilon > 0$, however small, there exists some real value $\delta > 0$ such that for all $\mathbf{x}$ in the domain of $\mathscr{S}$ (thus, shapes) with $\|\mathbf{x} - \mathbf{c}\| < \delta$, the value of $\mathscr{S}(\mathbf{x})$ satisfies $\|\mathscr{S}(\mathbf{x}) - \mathscr{S}(\mathbf{c})\| < \varepsilon$. Here, $\| \cdot \|$ denotes distance between two shapes embedded in the same space, *e.g.*, the Haussdorff distance. The above $\varepsilon - \delta$ assumption is, however, not happening over the space of considered shapes. Consider, for instance, a perfectly 2D rectangle $\mathbf{c}$. Consider now an infinitesimally small ‘bump’ added to any of the rectangle edges – that is, shifting the edge outwards with a small distance. Obviously, the Haussdorff distance $\varepsilon = \|\mathbf{x} - \mathbf{c}'\|$ between the ‘pure’ rectangle $\mathbf{c}$ and ‘perturbed’ rectangle $\mathbf{c}'$ can be made as small as desired. However, such a (small) bump on $\mathbf{c}'$ will cause the appearance of a skeleton branch in $\mathscr{S}(\mathbf{c}')$ whose length is *not* a function of the bump size ε – indeed, the length of this branch, or in other words the value $\delta = \|\mathscr{S}(\mathbf{c}') - \mathscr{S}(\mathbf{c})\|$, is *not* in any way bounded by ε. Thus, the function $\mathscr{S}$ is not Cauchy continuous. Note that this fact is not dependent on sampling issue related to the shape or the skeleton, such as sampling density or sampling model (volumetric or boundary-based).

This inherent discontinuity of the skeletonization function, or operator $\mathscr{S}$, is often seen in the literature under various (empirical) names such as sensitivity to noise, generation of ‘spurious’ artifacts, or instability. Beyond the theoretical discontinuity issue pointed above, this phenomenon causes relevant practical problems: For instance, we cannot guarantee that we obtain ‘similar’ skeletons (under a Haussdorff distance metric or similar) even if our input shapes are similar. Therefore, subsequent uses of skeletons, *e.g.* in shape processing or matching applications, are difficult at best.

This long-known issue in skeletonization has been addressed by a so-called *regularization* process. In detail, the aim is to *either* pre-process the shapes to be skeletonized, *or* post-process their skeletons, in a way that Cauchy-Weierstrass continuity is ensured – thus, practically, we can ensure that small changes of a shape result to small changes of its computed skeleton.

Several such regularization method exist, as follows:

2.3.4.1 *Input preprocessing*

The intuition behind these methods is that skeleton branches (details) are inherently created by curvature maxima on the input shape boundary $\partial\Omega$ (see Section 2.2.2. As such, if we are able to decrease the curvature range of $\partial\Omega$, we will in turn decrease the variability (and in the end, discontinuity) of the skeletons of Ω. This can be done by applying a wide range of smoothing filters on $\partial\Omega$, see *e.g* [176]. The problem with this approach is that we need to carefully control the filter parameters to make sure that 'relevant' shape details are kept, whereas 'irrelevant' details are removed. While we could do this from the strict perspective of the shape $\partial\Omega$, is is not evident how smoothing operations on this shape will affect its skeleton.

A different approach to input preprocessing is to apply a *global* operation to all points of $\partial\Omega$, in such a way that this operation is guaranteed to remove unwanted skeleton details. Such an operation is proposed by the so-called scale axis transform [59, 108]. Intuitively, this operation inflates the boundary $\partial\Omega$ isotropically in normal-to-boundary directions, thereby making small-scale boundary perturbations either smooth out or disappear.This is indeed expected, since this inflation is very similar to computing level-sets (isocontours) of the boundary's distance transform $DT_{\partial\Omega}$ at increasingly higher isovalues. As the isovalue increases, such contours are increasingly closer to a ball, thus to a shape of constant curvature, or a shape whose skeleton is simpler (in the limit, the skeleton of a perfect ball is a single point). While this method works very well in practice (see *e.g.* [108]), the setting of the inflation factor is not evident for an end-user. That is, it is not obvious how much to inflate a given shape if we want a certain amount of its skeletal detail to be removed. Moreover, by changing the input shape this way, we cannot guarantee that certain important shape properties, such as topology, are preserved.

2.3.4.2 *Local Metrics*

In contrast to input preprocessing, local metrics take a different approach: The input shape Ω is kept as is, and its skeleton $S_{\partial\Omega}$ is computed by any desired skeletonization method. Next, a regularized, or simplified, skeleton $S^{\tau}_{\partial\Omega}$ is computed, subject to a user-supplied simplification parameter τ.

The key aspect of local simplification metrics, *i.e.* functions that compute $S^{\tau}_{\partial\Omega}$ from $S_{\partial\Omega}$ subject to a given τ, is that they act locally. That is, the value of $S^{\tau}_{\partial\Omega}(\mathbf{x})$ depends only on the immediate neighborhood of the point $\mathbf{x}$ of the skeleton. Local measures include the angle between the feature points and distance-to-boundary [2, 50], divergence-based [16, 155] and first-order moments [140]. Leymarie and Kimia topologically simplify point-cloud skeletons to capture Y-intersection curves and skeleton sheet boundaries in medial scaffolds, and next apply similar local skeleton detectors [93].

The main advantage of such regularization approaches is their simplicity and computational efficiency: Indeed, if the cost of regularizing a point $\mathbf{x}$ depends only on a fixed-size spatial neighborhood of $\mathbf{x}$, it means that the cost of regularizing an entire skeleton $S_{\partial\Omega}$ is linear in the size (*e.g.*, number of sample points) of this skeleton. Moreover, the regularization procedure can work, ideally, in parallel, since every point $\mathbf{x} \in S_{\partial\Omega}$ is treated independently.

However, such metrics have the fundamental problem of not being able to distinguish between locally-identical, yet globally-different, contexts (see *e.g.* [135], Figure 1). As such, thresholding local metrics can disconnect skeletons; reconnection needs extra work [103, 122, 155, 168]. This also makes skeleton simplification, also called skeleton pruning, and makes pruning, less intuitive [148].

2.3.4.3 *Global Metrics*

In contrast to the local regularization metrics discussed earlier, *global* regularization metrics pose the problem differently: Given a point $\mathbf{x} \in S_{\partial\Omega}$ on an unsimplified, raw, skeleton, how can we measure the *overall* importance of $\mathbf{x}$ to the shape Ω?

A key development in this direction has been the work of Dey *et. al* [37], similar to earlier work of [125]. Although the key motivation behind this work has been the exact definition of a *curve* skeleton in three dimensions, their results can (and have been) used to globally regularize skeletons. In detail, given a ‘raw’ skeleton point $\mathbf{x} \in S_{\partial\Omega}$, the MGF metric assigns to $\mathbf{x}$ the longest shortest-path distance, over $\partial\Omega$, between the any two feature points of $\mathbf{x}$. As shown by numerous works [37, 129–131, 135], the MGF metric assigns a ‘natural’ importance which is lower for skeleton points $\mathbf{x}$ created by minute local variations of the shape surface $\partial\Omega$, and larger for more important variations. More interestingly, for the vast majority of shapes, the MGF measure seems to increase *monotonically* from the skeleton boundary $\partial S_{\partial\Omega}$ to the surface-skeleton center. The observation of this phenomenon has been consistently verified in practice on a large number of different shapes (*e.g.* [37, 131, 134, 135]. As such, the MGF metric allows an easy way to compute a so-called *multiscale* skeleton representation $S^{\tau}_{\partial\Omega}$, simply by upper-thresholding all points of the full skeleton $S_{\partial\Omega}$ whose MGF value exceeds τ. If the MGF metric is indeed monotonically increasing from $\partial S_{\partial\Omega}$ to the center of $S_{\partial\Omega}$, for any shape Ω, this means that the above thresholding will deliver increasingly smaller and simpler, and in the same time connected, skeletons. Interestingly, the same MGF metric has been also used to extract multiscale *curve* skeletons [135].

An important analogy with the MGF metric can be found in the 2D case. Here, much earlier than studying the 3D case, a very similar regularization principle has been proposed for 2D medial axes. Interestingly, the same 2D regularization principle has been proposed independently by several researchers for both voumetric-sampled or boundary-sampled shape descriptions [33, 118, 183]. This principle can be compactly described as follows: Consider a 2D skeleton point $\mathbf{x}$. By definition, $\mathbf{x}$ will have at least two feature points on the boundary $\partial\Omega$ of its corresponding input shape. The shortest distance (along the boundary $\partial\Omega$) between such feature points gives a natural and intuitive 'importance metric' or value for the skeleton point $\mathbf{x}$. By upper-thresholding this importance metric by some user define value τ, we can obtain a 2D skeleton where all boundary details *longer* than τ are captured. As such, reconstructing the input shape from the simplified skeleton $S^{\tau}_{\partial\Omega}$ guarantees that we obtain a shape Ω^{tau} where all details of Ω whose length was smaller than τ have been replaced by circle arcs. For the 2D case, the monotonicity of the importance metric (from the 2D skeleton tips to its center) can be easily proven. As such, thresholding this metric delivers nested and connected skeletons. Intuitively, the above metric can be thought as assigning to each point $\mathbf{x} \in \Omega$ the amount of mass, or boundary length, that would collapse to $\mathbf{x}$ if advected in a flow field determined by $\nabla DT_{\partial\Omega}$. The same intuition is mentioned – though not formally argued – behind the 3D MGF metric by Reniers [135]. Variations of this so-called 'collapsed boundary length' metric are known, *e.g.* [177]. These variations ensure that simplified skeletons accurately capture spatially important 2D boundary corners, while neglecting unimportant ones. Using the shape-from-skeleton reconstruction process outlined in Section 2.2.1, these techniques are competitive alternatives to anisotropic smoothing methods which keep 'salient' shape features sharp, while fully removing less salient features. While this so-called saliency metric (essentially the 2D MGF metric divided by the value of distance-to-boundary) works convincingly well in 2D, no equivalent is known for 3D shapes.

2.4 CONCLUSIONS

In this chapter, we have presented a (necessarily limited) overview of the theory and current practice regarding computing 2D and 3D medial descriptors. As we have seen, computing (and using) such descriptors in practice is challenged by both theoretical and practical issues, On the theoretical side, we have the major issue of global (*monotonic*) regularization, *i.e.* the computation of a multiscale of medial descriptors that can guarantee the incremental capture of details of an input shape subject to a user-defined metric and metric threshold value. On the practical side, we see challenges in the fast and robust computation of medial surfaces (and, partially, the computation of curve skeletons) for large and complex 3D shapes. Additionally, we see challenges, but also interesting possibilities, in computing derived skeletal properties that help us to easily and robustly analyze and process such 3D shapes. Having now set the context, and having outlined the above-mentioned issues, the remainder of this thesis will focus on addressing these issues.

3 EXTRACTION OF MEDIAL DESCRIPTORS FROM POLYGONAL SHAPES

I do not fear computers. I fear the lack of them.

Isaac Asimov

3.1 INTRODUCTION

Given the potential of medial representations for several shape analysis operations (Section 1.4.2), the extraction of medial descriptors has been a research field on its own ever since the introduction of such descriptors by Blum [15].

The medial-representation extraction operation, also called *skeletonization*, is the starting point for any subsequent utilization of medial descriptors in shape analysis and processing operations. Thus, to be useful and usable, a skeletonization algorithm has to comply with a number of requirements (see also Sections 1.4.3 and 2.2.1).

As we have seen in Section 2.3, many skeletonization methods exist that comply, in various ways and up to various extents, with our complete set of desirable properties. Globally put, we have seen that several methods exist that cover well skeletonization for 2D shapes. As such, we next will focus our attention on 3D skeletonization only.

Let us now consider skeletonization from the perspective of a *client*, or user, application, rather than a *provider* algorithm. From this viewpoint, the user requires that a skeletonization method (as proposed by a given algorithm) is *useful* and *usable* for the specific goals that the user-application has. This strengthens but also extends the internal requirements that a skeletonization algorithm has (*e.g.* centeredness, thinness, homotopy with the input shape, connectedness, rotational invariance)

This chapter is based on the following papers:

1. 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

2. J. Kustra, A. Jalba, and A. Telea. Probabilistic View-based Curve Skeleton Computation on the GPU. In *8th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications, VISAPP 13*, 2013

with a number of external requirements We identify below three such important additional requirements:

Target operations: These requirements define the class of operations that our computed skeletons should enable us to perform on the input shape. For instance, if we are only interested in shape genus identification, then computing a 3D curve skeleton is sufficient. Curve skeletons are also sufficient for a range of shape matching applications, or for applications where we only treat tubular shapes such as blood vessels. However, applications such as reconstruction of general shapes, edge detection, or patch-based segmentation require the computation of the more complex 3D surface skeleton.

Shape properties: These requirements define which types of properties of the input shape should be captured by the computed skeletons, and which can (or even should) be neglected. For instance, consider a shape whose surface has details occurring at various geometric scales, such as bumps, creases, or indentations. Some of these details may be important to our further shape processing, while some others may be considered less important or even, in the extreme case, irrelevant (*e.g.* noise). Separately, consider how a skeletal representation is influenced by the sampling decisions taken to represent the input shape (*e.g.*, sampling-point density or sampling noise, see Section 2). In most real-world applications, we will encounter both above concerns: Shapes may have irrelevant details *and* will be sampled within some resolution and/or accuracy limits. As such, a skeletonization method should provide effective and easy-to-use regularization means to let users control how such aspects influence (or not) the resulting skeletons (Section 2.3.4).

Computational resources: Recent advances in modeling tools and scanning technology made it possible to create high-detail representations of shapes having *e.g.* millions of surface points (for boundary representations) or billions of voxels (for volumetric representations). If we want to use skeletons in real-world applications, we should be able to compute such skeletons efficiently for such shapes. Here, efficiency regards both computational costs (time required to extract a skeleton given a certain amount of computing power) and representation costs (memory required to implement skeletonization algorithms). The large majority of 3D skeletonization algorithms being proposed so far follows a relatively simple single-instruction, single-datastream (SISD) [47], or 'serial', computation model. However, even for algorithms that have an optimal linear complexity in the size of the input shape (*e.g.* [66], the SISD model does not scale well to cope with the sizes of current 3D shape models. In

We can see the internal *vs* external requirements of a skeletonization method as being roughly analogous to the well-known classification of software-engineering requirements into functional *vs* non-functional.

recent years, an increase of parallel computer architectures in personal computers, in terms of powerful graphics processing units (GPUs), has made the development of parallel algorithms significantly more attractive, with many applications in the shape processing domain too [53, 75]. The single-instruction multiple-datastream (SIMD) computation model offered by GPUs has led to an entire new application field called general-purpose computing on graphics processing units (GPGPU) [52]. Parallel programming languages, such as Nvidia's CUDA [117], made the development of GPGPU applications significantly cheaper and easier for mainstream programmers. However, so far only very few skeletonization algorithms have exploited GPGPU capabilities. This is due to the inherent sequential nature of certain skeletonization algorithms, but also to the complexity of some other algorithms, which makes their conversion to parallel code difficult.

Summarizing the above, we believe that the efficient (parallel) and scalable computation of 3D surface skeletons from large and complex shapes, with easy-to-use regularization, is still an open (and important) issue to make such skeletons useful and usable in practical applications. Zooming in on the scalability requirement, we next choose to focus on *boundary* representations (for both the input shapes and their surface skeletons), as such representations are significantly more scalable in terms of memory than volumetric ones (Section 2.1.2). Given this refined context, let us note that we are not the only ones having been interested in the above-described skeletonization problem. Recently, Ma *et al.* proposed arguably the fastest method to extract surface skeletons from oriented point clouds [101]. Although their GPU-based method is arguably the fastest available medial surface extractor, its usability has been limited. As the authors note, their method produces medial point clouds as an output, which causes difficulties for applications which require a compact surface-skeleton description or a curve skeleton. Moreover, their method does not offer a robust way for regularizing generic surface skeletons without creating disconnections.

Following the above directions, this chapter introduces several skeletonization methods targeted at boundary sampling representations (point cloud and polygonal meshes). Our algorithms tackle the combined set of requirements regarding computational complexity, scalability, and regularization discussed earlier. For this purpose, we introduce first a general 3D surface skeletonization and regularization method (Section 3.2). Our method allows the accurate computation of point-cloud or meshed skeletons from 3D point clouds. If input shape connectivity is also provided (in the form of a surface mesh), we show how to efficiently and robustly regularize these skeletons (Section 3.3). Our entire proposal is parallelized, using either CPU or GPU multithreading. Overall, our method enables the computation of regularized surface skeletons from large shapes (over 10^6 sample points) a matter of seconds or less on a modern GPU – this being a small fraction of the time required for similar computations by existing skeletonization methods. Additionally, we show how we can reconstruct accurate representations of the input 3D shapes using only the surface skeleton point clouds, by using a novel GPU image-based method (Section 3.4). Finally, we show how to extend our method to extract 3D curve skeletons as a subset of the related 3D skeletons (Section 3.6). The above steps are depicted in Figure 3.1 (A-E). The last step (E) of this pipeline, *i.e.*, the reconstruction of separate polygonal surfaces for the various manifolds of a 3D surface skeleton, is explored in more detail next in Chapter 4.

Separately from the above directions (that work fully in 3D space), we also present an approach for the extraction of 3D curve skeletons that uses a set of 2D silhouettes of a 3D shape, rather than a full 3D shape representation (Section 3.7). Besides proposing an efficient GPU implementation, this approach also further explores a number of unknown relationships between 3D curve skeletons and 2D medial axes of corresponding shape projections.

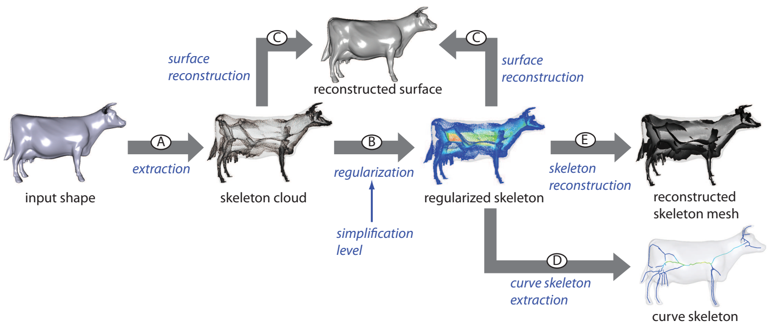


Figure 3.1: 3D skeletonization pipeline: (A) skeleton-cloud extraction, (B) regularization, (C) shape reconstruction, (D) curve-skeleton extraction, and (E) surface-skeleton mesh reconstruction.

From a practical perspective, the work presented in this chapter shows how one can easily and efficiently compute regularized 3D surface and/or curve skeletons from large and complex 3D shapes. As such, the methods and algorithms presented here serve as a basis for the subsequent chapters in this thesis, where methods to refine such 3D skeletons and use them for shape processing applications are discussed (Chapters 4, 5 and 6).

3.2 MEDIAL SURFACE COMPUTATION FROM LARGE POLYGONAL SHAPES

3.2.1 *Surface skeleton extraction*

We extract a skeleton point-cloud from an oriented point-cloud model $C = \{(\mathbf{p}_i, \mathbf{n}_i\}$ of the input shape $\partial\Omega$. At its core, our method is based on the ball shrinking algorithm of Ma *et al.* [101], which works as follows. For each point $\mathbf{p} \in C$, a (large) ball $B(\mathbf{s}, r_0)|\mathbf{s} = -r_0\mathbf{n} + \mathbf{p}$, tangent at $\mathbf{p}$, is created. By definition, $\mathbf{f}_1 \equiv \mathbf{p}$ is the first feature point of $\mathbf{s}$. The algorithm shrinks B by searching the closest feature point $\mathbf{f}_2$ to $\mathbf{s}$ and iteratively moving $\mathbf{s}$ so that B passes through $\mathbf{f}_1$ and $\mathbf{f}_2$ until $\mathbf{s}$ converges. At that moment, B is maximally inscribed, and its center $\mathbf{s}$ yields a new skeleton point with inscribed radius $r_{ins} = \|\mathbf{s} - \mathbf{f}_1\| = \|\mathbf{s} - \mathbf{f}_2\|$. We next propose two performance improvements for this algorithm.

3.2.2 *Numerical Parallelization of Skeleton computation*

In their work, Ma *et al.* first proposed an efficient sequential CPU implementation. Key to this is a heuristic that sets the initial radius r_0 for a ball $B(\mathbf{s}, r_0)$ being shrunk to the radius of a skeleton point already found for a surface point $\tilde{\mathbf{p}}$ close to $\mathbf{p}$. This largely reduces the number of shrinking steps ([101], Sec. 3). However, this requires *sequential* processing of the cloud C in a global distance-based ordering, computed by in-order visiting a *kd* tree containing C.

This idea is paralellized as follows. A single global value r_0 is used, initially set to 2. Next, C is divided into N equal-sized chunks (without any point ordering) and processed one chunk per thread. When a thread finds a new value r_{ins}, r_0 is set to $(r_0 + r_{ins})/2$, *i.e.* adapt r_0 in a moving-average fashion. Additionally, the shrinking of the ball is stopped when two consecutive center positions differ by a value less than an user-specified value τ. Finally, an *approximate* nearest-neighbour (NN) scheme is used based on *kd* trees [114] to search for the closest point $\mathbf{f}_2$ to $\mathbf{s}$ with precision ε.

Table 1 shows our skeleton extraction timings t and average number k of *kd*-tree searches per point for different τ and ε values, on several models, on a 4-core 2.4 GHz CPU with $N = 8$ threads. The first three models are also used by Ma *et al.* Smaller τ values need more iterations; larger τ values yield less accurate skeletons quicker. It can be observed that k is quite stable for all models. For a skeleton centeredness precision $\tau = 10^{-3}$, the proposed method is roughly four times faster than the sequential method of Ma *et al.* which use also a 2.4 GHz CPU. Given our 4-core CPU, this implies a very efficient parallelization. Relaxing ε slightly accelerates this search (Tab. 1, column $\varepsilon = 10^{-3}$) with little accuracy loss.

Table 1: Performance of our skeleton extraction algorithm on both CPU and GPU (see Sec. 3.2.1).

Model	Points	Points	**CPU timings** (t seconds, k iterations)						**GPU timings**	
	(surface)	(skeleton)	$\tau = 10^{-4}, \varepsilon = 0$		$\tau = 10^{-3}, \varepsilon = 0$		$\tau = \varepsilon = 10^{-3}$	Ma *et al.*	Our method	Ma *et al.*
			k	t	k	t	t	[101]	$\tau = 10^{-3}, \varepsilon = 0$	[101]
Armadillo	106289	106289	5	1.29	3	0.87	0.87	4.20	0.21	0.39
Dragon	437645	436439	5	6.53	2	5.46	5.44	20.23	1.21	2.64
Horse	48485	48485	5	0.77	3	0.58	0.58	1.62	0.09	0.29
Cow	185882	185703	5	3.63	3	2.82	2.77		0.64	
Bird	46866	46862	5	0.44	3	0.35	0.34		0.03	
Horse 2	193934	193887	5	7.51	3	5.77	5.77		1.32	
Asiandragon	231574	230964	4	2.83	2	2.19	2.19		0.22	
Asiandragon 2	954027	951010	5	[19.03]	3	[14.41]	[13.67]		1.57	
Hand	197245	196920	4	3.32	2	2.50	2.50		0.51	
Elephant	50485	50422	5	0.62	3	0.48	0.48		0.05	
Buddha	543652	541762	4	6.76	3	5.47	5.37		1.42	
Mouse	403652	402301	6	5.61	3	3.07	3.02		0.34	
Pig	225282	224539	5	7.48	3	4.21	4.17		1.47	
Armadillo 2	172974	172955	6	3.29	3	1.86	1.86		0.47	
Rabbit	124998	123811	6	1.31	3	0.85	0.83		0.06	

Ball-shrinking yields a surface-skeleton cloud $S = \{(\mathbf{s}, \mathbf{f}_1, \mathbf{f}_2)_i\}$ with two feature points $\mathbf{f}_1$, $\mathbf{f}_2$ per skeleton point $\mathbf{s}$ (see Figure 3.4). The local density of S is proportional with the input cloud density times the shape curvature [154]. Thus, we get more skeleton points where the surface changes rapidly and/or is finely sampled.

Since polygonal surface flat patches can be represented using only the polygon vertices, the skeleton computation would only compute the points corresponding to the the polygon vertices. To obtain skeleton points corresponding to the *inside* of the polygons, these should be re-sampled on the surface, and the surface is recomputed. In practice, several surface re-sampling techniques are readily available such as the Yams package [51]. This allows a control over the skeleton density by sub-sampling the input surface. Similarly, for less points, we uniformly sub-sample S by removing all points closer to each skeleton point than some distance δ. Figure 3.3 shows the skeleton of a cow model for four different δ values.

At a high level, similarly to Leymarie *et al.* [93], we regard each input point $\mathbf{f_1}$ as a skeleton 'generator', try to pair it with another point $\mathbf{f}_2$, and test maximality of the resulting balls. However, while [93] explicitly compute pairs $(\mathbf{f}_1, \mathbf{f}_2)$ *and* checks for ball maximality using search strategies based on visibility constraints, we implicitly compute the pairs *while* shrinking balls.

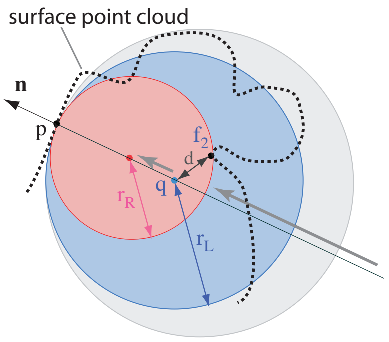


Figure 3.2: Ball shrinking algorithm. Gray arrows indicate how balls shrink (from gray to blue to pink).

3.2.3 *Graphics Processing Unit (GPU) parallelization*

For an efficient transfer of ball shrinking to GPUs, we need (a) an efficient GPU nearest-neighbor (NN) search and (b) an effective load balancing between threads.

To these ends, Ma *et al.* proposed a mixed CPU-GPU approach [101]. For NN search, they use the GPU algorithm in [198]. For load balancing, they work as follows: Iterations are done in parallel on the GPU; after

Algorithm 1: Skeleton extraction algorithm (τ=centeredness; ε=nearest-neighbor precision, see Sec. 3.2.3). The first loop tries to approximate a skeleton closest point to $\mathbf{p}$ on the normal n by repeatedly replacing a candidate point $\mathbf{s} = \mathbf{p} - r\,\mathbf{n}$ by a better approximation $\mathbf{s} = \mathbf{p} - r'\ \mathbf{n}$ such that $\|\mathbf{f}_2 - \mathbf{s}'\| = \|\mathbf{p} - \mathbf{s}'\|$.. The second loop refines the skeleton point position using a bisection search approach.

```
FindSkeletonPoint(Point $\mathbf{p}$, Point $\mathbf{s}$, Point $\mathbf{f}_1$, Point $\mathbf{f}_2$)
Data: $\mathbf{p}$: input surface point with normal $\mathbf{n}$.
Result: $\mathbf{s}, \mathbf{f}_1, \mathbf{f}_2$: output skeleton point $\mathbf{s}$ with features $\mathbf{f}_1, \mathbf{f}_2$.
1  $r_L \leftarrow 2; r_R \leftarrow r_L$                           // initialize search radius
2  $\mathbf{f}_1 \leftarrow \mathbf{p}$                                  // initialize first feature point
3  while (true) do                            // shrink ball until $\|\mathbf{p}-\mathbf{s}\| \in [r_L, r_R]$
4      $\mathbf{s} \leftarrow \mathbf{p} - r_L\,\mathbf{n}$                            // compute ball center
5      $\mathbf{f}_2 \leftarrow search(\mathbf{s}, \varepsilon\ r_L)$
6      if $(|r_L - \|\mathbf{f}_2 - \mathbf{s}\|| < \tau)$ then return;                // found $\mathbf{f}_2$, done
7      if $(\mathbf{f}_2 = \mathbf{p})$ then break;                           // ball is too small
8      $r_R \leftarrow r_L$                                          // shrink the ball
9      $r_L \leftarrow -\frac{\|\mathbf{f}_2-\mathbf{p}\|^2}{2\ \mathbf{n}\cdot(\mathbf{f}_2-\mathbf{p})}$
10 while $(r_R - r_L > \tau)$ do                  // bisection search to find exact radius
11     $r \leftarrow (r_R + r_L)/2$
12     $dr \leftarrow r_R - r_L$
13     $\mathbf{s} \leftarrow \mathbf{p} - r\,\mathbf{n}$
14     $\mathbf{f}_2 \leftarrow search(\mathbf{s}, \varepsilon\ dr)$
15     if $(\mathbf{f}_2 = \mathbf{p})$ then $r_L \leftarrow r$;
16     else $r_R \leftarrow r$;
17 return
```

each iteration, threads ask the CPU whether each ball needs more iterations. If so, the CPU invokes the GPU kernel for the next iteration only for the not-yet-converged balls. As Ma *et al.* mention, this achieves good performance, but cannot use the initial radius heuristic, as that heuristic was designed for a sequential algorithm.

Our GPU proposal directly parallelizes our CPU algorithm with one thread per skeleton point. Since our initial radius heuristic works in a parallel setting, we transfer it directly to the GPU.

In contrast to Ma *et al.*, we use a GPU-only load balancing scheme. This poses some subtle constraints on the choice of the NN technique used. For this, we investigated several options. Garcia *et al.* showed a GPU brute-force NN algorithm [53], which turned to be 20 to 30 times slower than our CPU KD-tree search [114]. Cayton's NN algorithm covers the input set of N points by a randomly-distributed set of $m \ll N$ overlapping balls which are searched in parallel on the GPU [22]. However, this method cannot do several NN searches in parallel, which we need to parallelize ball extraction. Left-balanced GPU KD-trees also perform poorly. Such trees are rather deep (maximum depth $D = \log N$ for N

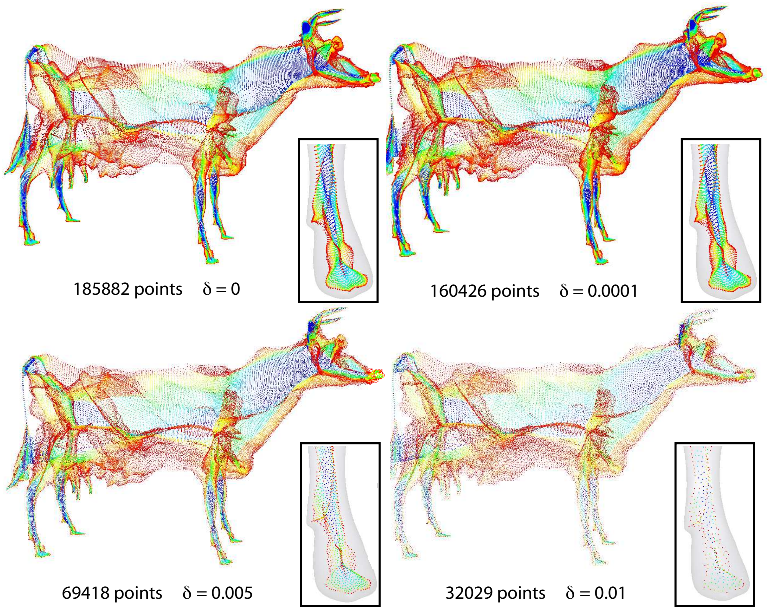


Figure 3.3: Uniform skeleton sampling for different δ values (see Sec. 3.2.1). Colors show angles of feature vectors (θ-SMA detector).

input points). Also, the unfavourable distribution of query points (potential skeleton points having at least two shape points at equal distances) causes many tree node visits during a NN search, *i.e.* many random, uncached, memory accesses, which seriously degrade GPU performance.

For our special context, a KD-tree with alternating splitting dimensions and median-based pivoting proved best. We limit the tree depth to a small value (10 to 12). Leaf nodes store more than one point and are linearly searched. Linear search performs very well on the GPU [53] (more cache hits and coalesced memory accesses), yielding a better overall NN speed.

Table 1 shows the speed of our GPU method on an Nvidia GTX 280. Since the GPU NN search is exact, these timings should be compared with CPU timings for $\varepsilon = 0$. Our method is 4 to 10 times faster than its CPU variant. Compared to the GPU method of Ma *et al.*, we are about 3 times faster, as we do not need CPU-GPU synchronization, and also as we use an initial radius heuristic, which the GPU method of Ma *et al.* does not support.

3.3 MEDIAL SURFACE REGULARIZATION

Skeleton regularization assigns an importance value $\rho : S \rightarrow \mathbb{R}^+$ to skeleton points (Sec. 2.3.4). If ρ monotonically increases from the skeleton boundary inwards, thresholding ρ yields a connected skeleton which captures shape details at a given scale. Such measures are proposed in [37, 125, 135]: for a skeleton point $\mathbf{s}$ with feature points $\mathbf{f}_1$, $\mathbf{f}_2$, $\rho(\mathbf{s})$ is the length of the shortest path γ on $\partial\Omega$ between $\mathbf{f}_1$ and $\mathbf{f}_2$. Although this approach has been successfuly demonstrated on a wide variety of input shapes, the measure monotonicity is not guaranteed. Computing such paths can be done using Dijkstra's algorithm [135]; computing the distance map DT_{f_1} of $\mathbf{f}_1$ by the Fast Marching Method (FMM) and then tracing γ in $-\nabla DT_{f_1}$ from $\mathbf{f}_2$ [121], or hybrid search techniques [170, 190]. However, such methods are very slow, as we shall see next.

3.3.1 *Shortest and straightest geodesics*

We compute the skeleton importance ρ using an approximation of *straightest geodesics* on polyhedral surfaces [69, 124], which generalize straight lines to arbitrary manifolds. Given a point $\mathbf{p} \in \partial\Omega$ and a tangent vector $\mathbf{v} \in T_p$ at $\mathbf{p}$, the (discrete) straightest geodesic γ_S is the unique solution of the (discrete) initial-value problem $\gamma_S(0) = \mathbf{p}$, $\gamma_S'(0) = \mathbf{v}$ [124]. We extend this to define the (discrete) approximate geodesic γ_{se} between two points $\mathbf{s}, \mathbf{e} \in \partial\Omega$, over tangent vectors $\mathbf{v}_i \in T_s$ at $\mathbf{s}$, as the solution of the discrete boundary-value problem

$$\begin{aligned} &\gamma_{S,i}(0) = \mathbf{s},\ \gamma_{S,i}'(0) = \mathbf{v}_i \\ &\gamma_{S,i}(\|\gamma_{S,i}\|) = \mathbf{e} \\ &\gamma_{se} = \underset{i}{\operatorname{argmin}} \|\gamma_{S,i}\|, \end{aligned} \tag{3.1}$$

where $\|\gamma_{S,i}\|$ is the length of the discrete geodesic $\gamma_{S,i}$. Thus, γ_{se} is the shortest among all straightest geodesics between $\mathbf{s}$ and $\mathbf{e}$. Solving Eqns. 3.1 is not easy. Speed-wise, the cost is proportional to the number of tangent directions $\mathbf{v}_i$ considered. Also, current algorithms for computing straightest geodesics [69, 124] estimate γ' by evaluating the (discrete) Gaussian curvature at the mesh vertices visited while tracing. The tangent vectors to γ may change directions especially when this curvature is not exactly 2π, so geodesics may not reach their endpoints $\mathbf{e}$, which is critical for our goal. Our proposed approach, which we refer to as *Shortest Straightest Geodesics* (SSG) adresses these concerns, by extending the *straightest geodesics* with a heuristic which guarantees that it passes through the two skeleton feature points $\mathbf{f}_1, \mathbf{f}_2$. Next, we further elaborate on this heuristic and its numerical implementation details.

3.3.2 *Efficient Straightest Geodesic computation*

For a skeleton point $\mathbf{s}$, we trace M straightest geodesics $\gamma_{S,i}$, $1 \leq i \leq M$ in parallel on the CPU or GPU between the feature points $\mathbf{f}_1$ and $\mathbf{f}_2$ of $\mathbf{s}$ on $\partial\Omega$, with starting angles $\alpha_i = (2\pi i)/M$ uniformly spread around the vector $\mathbf{f} = \mathbf{f}_1 - \mathbf{f}_2$ at $\mathbf{f}_1$. For each direction $\mathbf{v}_i$, we intersect the edges of the mesh faces visited while tracing by the plane with normal $\mathbf{n}_i = \mathbf{f} \times \mathbf{v}_i$. We next set $\rho(\mathbf{s}) = \min_i \|\gamma_{S,i}\|$ *i.e.* the length of the SSG between $\mathbf{f}_1$ and $\mathbf{f}_2$.

We speed up tracing by early termination: we stop tracing a path if its length exceeds the current $\rho(\mathbf{s})$. For a closed mesh, we consider both paths from $\mathbf{f}_1$ to $\mathbf{f}_2$, given by the mesh-plane intersection (closed) curve. When one of these paths is computed, we stop tracing the other path if its current length exceeds the first path's length. For a computed γ_{se}, we also store its tangent vectors $\mathbf{t}_s$ and $\mathbf{t}_e$ in $\mathbf{f}_1$ and $\mathbf{f}_2$ respectively, and use $\mathbf{t}_s$ as starting direction when tracing SSGs for the next skeleton point. Since neighbour skeleton points usually have similar geodesics [37, 129], early termination occurs sooner, which further speeds up tracing.

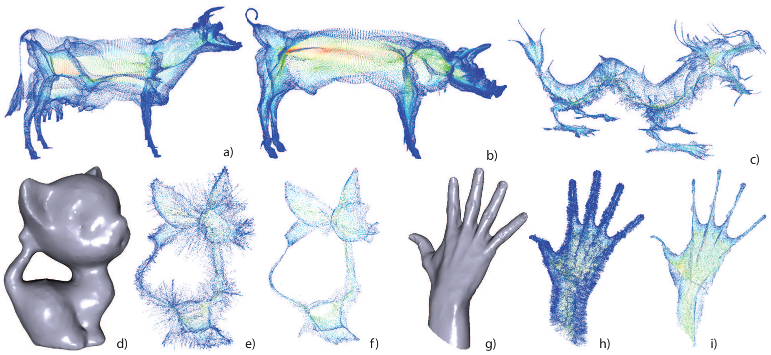


Figure 3.4: Skeleton cloud regularization by geodesic importance. Red points are the most important. Blue points correspond to small surface features (see Sec. 3.3).

As shown in Figure 3.4, ρ monotonically increases from the skeleton boundary (blue) inwards (red). Thresholding ρ removes skeleton points given by small shape details (Figure 3.4 e,f,h,i). Such details can be surface noise (Figure 3.4 d), but also appear in locally-tubular shapes (Figure 3.4 f). In contrast, thresholding non-monotonic metrics such as θ-SMA (Figure 3.3) can lead to the removal of points representing larger shape features and a disconnection of the skeleton at low threshold values.

3.3.3 *Performance and accuracy of SSG tracing*

Table 2 shows the speed of our SSG method on a GTX 280 card *vs* a 2.8 GHz 4-core PC for $M = 20$ directions, one thread per SSG, and the speed-up due to the heuristics in Sec. 3.3.2.

We further discuss our method and compare it with related methods from the perspective of computational speed, accuracy of the computed geodesics, and memory requirements.

Table 2: Timings for computing the geodesic importance.

Model	Timing CPU (seconds)	Timing GPU (seconds)	Optimization speed-up (%)
Cow	61.7	18.2	67.4
Bird	10.3	3.2	33.4
Horse	78.9	13.6	57.4
Asiandragon	61.5	13.8	60.0
Asiandragon 2	541.2	115.1	57.8
Hand	62.2	3.1	48.1
Elephant	7.1	3.3	57.1
Buddha	327.6	43.4	62.3
Mouse	210.3	52.1	69.0
Dragon	198.3	30.4	57.2
Pig	96.9	24.0	64.5
Armadillo	63.4	11.2	59.3
Rabbit	76.4	10.8	49.2

3.3.3.1 *Speed*

We compared our GPU SSG to FMM [121], the Dijkstra algorithm with A^* heuristics [135], the Surazhky-approximate (SA) and Surazhky-exact (SE) geodesic tracing [170], and CPU SSG (Sec. 3.3.2). Of these, only SA and Dijkstra were used to regularize skeletons [37, 135]. GPU SSG is 3 to 10 times faster than CPU SSG (higher speed-ups for larger models, Tab. 2); 10 times faster than Dijkstra; 100 times faster than FMM; 500 times faster than SA; and *thousands* of times faster than SE. This is not surprising: For a n-vertex mesh, Dijkstra is $O(n^2 \log n)$; FMM computes one distance field per vertex ($O(n \log n)$) and traces a geodesic in this field ($O(\sqrt{n})$, proportional to the shape diameter), yielding a complexity of $O(n^2 \log n + n^{3/2})$. SA has the same complexity as FMM. SSG traces all geodesics for a point in parallel. As we have more GPU cores than M directions, GPU SSG is $O(n^{3/2})$.

3.3.3.2 *Accuracy*

All above methods, except SE, compute approximate geodesics. The starting angle sampling M (Sec. 3.3.2) means that SSG may miss very narrow surface dents falling between two consecutive paths $\gamma_{S,i}$ and $\gamma_{S,i+1}$, *i.e.* may overestimate SSG length. Overestimation is not an issue for skeleton regularization: long geodesics yield anyway high-importance points which are to be kept (red points, Figure 3.4). Short geodesics, caused by surface noise which we want to eliminate by importance thresholding (blue points, Figure 3.4), allow only much narrower concavities to fall between them, and thus are less affected by length overestimation.

Table 3: Accuracy and timing comparison for geodesic tracing methods (FMM, SE, and SSG for different numbers of directions M).

Method	Relative error (%)		Timing (seconds)	
	Bird	Pig	Bird	Pig
Surazhsky-exact	0	0	11209	426.2
FMM	1.87	0.85	339.6	20.9
SSG ($M = 5$)	1.90	0.36	0.28	0.05
SSG ($M = 10$)	0.36	0.07	0.56	0.08
SSG ($M = 20$)	0.14	0.02	1.04	0.14
SSG ($M = 30$)	0.08	0.009	1.54	0.22
SSG ($M = 50$)	0.04	$8 \cdot 10^{-4}$	2.43	0.32
SSG ($M = 100$)	0.02	$2 \cdot 10^{-4}$	4.83	0.62
SSG ($M = 500$)	0.01	$4 \cdot 10^{-5}$	23.24	2.95

Figure 3.5 and Table 3 show the median relative error of SSG geodesic-lengths (for different M values), FMM, and SE. We used down-sampled versions of *bird* and *pig* (11718 and 3522 points, respectively), since SE is extremely slow. SSG is more accurate than FMM for $M > 10$, both for median and maximum errors. For $M = 100$, SSG practically gets exact geodesics at a tiny cost *vs* SE. Comparing median errors with SA, SSG is more accurate for $M \geq 30$, see Tab. 1 in [170] where an upper relative error bound of 0.1% is used, which means SSG with $M = 10$ directions for *bird* and $M = 30$ directions for *pig*. As mentioned, the cost of SA is the same as FMM, *i.e.* hundreds of times slower than SSG.

3.3.3.3 *Memory*

For a *single* geodesic tracing on a n vertex model, Dijkstra with A^* is $O(n)$; FMM is $O(n \log n)$; Surazhky *et al.* is $O(n^{3/2})$. SSG is $O(M)$. Verma and Snoeyink improved upon Surazhky *et al.* by combining Dijkstra with A^* with the original method [190]. This reduces the memory cost to $O(n)$, yields a speed-up of 8, but overestimates geodesic lengths by 10% on average. Concluding, our GPU SSG method is a good trade-off: It is nearly as accurate as exact geodesics, and hundreds of times faster than approximate geodesic methods.

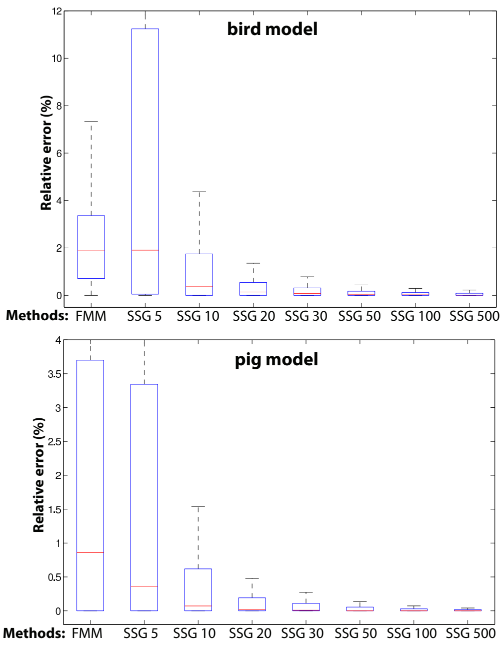


Figure 3.5: Accuracy comparison: FMM *vs* SSG geodesic tracing (see Sec. 3.3.3).

3.4 IMAGE-BASED SURFACE RECONSTRUCTION

We now present an efficient and simple algorithm for reconstruction of a model from its skeleton cloud (Figure 3.6): For each skeleton point $\mathbf{s}_i$ with radius r_i, we build a viewplane-aligned quad Q, or billboard, of world-space edge size 2. We texture Q scaled to $(r_i, r_i, 1)$ with a $D \times D$ texture whose texels $T(u,v)$ encode both depth and shading. If fixed-point texturing is available, T uses a 32-bit RGBA format: The first 3 bytes of $T(u,v)$ encode the height at $(u/D, v/D)$ of a ball of radius 1 centered at $(1/2, 1/2, 0)$. The fourth byte (A) encodes the ball color computed, *e.g.*, with Phong shading. If floating-point textures are available, we encode the height and shading in the L and A channels of a luminance-alpha texture, which leaves two channels for future potential use. The texture size D is set to 512, which yields highly accurate shading and height encoding for very large balls. For maximal speed, we store T on $log_2(D)$ hand-built mip-map levels.

We render the quads by a simple ARB fragment program shader (7 operations only) which gets the z_{NDC} normalized device coordinate (NDC) of the current skeleton point $\mathbf{s}_i$ via the current color. The shader computes the final NDC depth $z_{NDC} + h$ at the current pixel from the incoming z_{NDC}

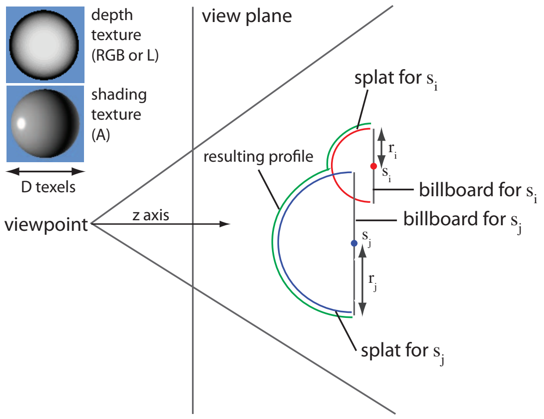


Figure 3.6: Skeleton splatting for surface image-based reconstruction.

and ball height h (from the current texel). If $z+h$ passes the depth test, the current texel's shading value is copied to the fragment output.

Our method renders shaded models directly from skeleton clouds of 500K points at 15 frames/second on a GT 330M card. If lighting changes, we only recompute one shading texture. The x, y splat sizes are pixel-accurate (by OpenGL scaling); depth values are either 24-bit fixed-point of floating point. This delivers images nearly identical to the original mesh (Figure 3.7).

Progressive rendering, like the one proposed by the Qsplat method [141], can be easily done by our method by drawing billboards sorted by a skeleton importance metric, *e.g.* ball radius or our geodesic metric in Sec. 3.3. If we use the geodesic metric, this always produces compact shapes, which pure surface splatting like Qsplat cannot guarantee. Progressive rendering is a useful tool for time-constrained application contexts where one wants to obtain an as accurate as possible representation (view) of a shape given a (limited) amount of time, and next refine this representation gradually as more time becomes available. Although we have not explored this direction further, our image-based shape representation using skeleton clouds should be easily able to support such scenarios, *e.g.* in contexts where one wants to render a large and/or complex 3D shape on a low-speed device (such as a smartphone) or render a shape that is incrementally made available to the renderer (such as rendering in a web browser where the shape is serially transmitted as a skeleton cloud).

Union of balls (UoB): Our image-based skeleton splatting delivers the same result as an UoB model, *e.g.*. [108, 165]. Our splatting could be used as drop-in shape reconstruction for any method that delivers an MST point cloud. As we shall see in Sec. 3.5, our method is roughly one to two orders of magnitude faster than [108] and over two orders of magnitude faster than [165].

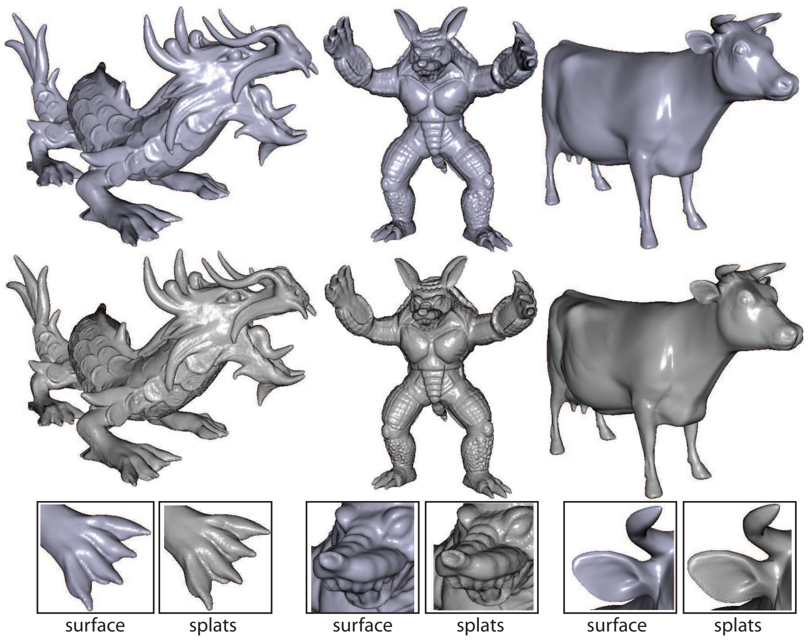


Figure 3.7: Comparison of surface rendering (top row) and skeleton image-based surface reconstruction (middle row). Insets show details.

3.4.1 *Surface skeleton reconstruction*

3D skeletons contain many self-intersecting, closely-spaced, manifolds. Hence, point cloud reconstruction methods for locally smooth and/or non-intersecting and/or watertight surfaces cannot be used [35, 38, 77]. Reconstruction of open, non-manifold, and self-intersecting surfaces [23, 193] is relatively slow and non-trivial. We present next a method for *visually* reconstructing *skeleton* surfaces from point clouds based on specific skeleton properties. The presented method – Delaunay based reconstruction – is suitable for rendering and visualization purposes of the medial surface. However, the generated surface is complex to use for shape analysis purposes due to the lack of clean separation between the medial surface manifolds. Chapters 4 and 6 address this topic in more detail, and provide more advanced methods for separating the manifolds and performing a clean pre-manifold surface reconstruction which is directly usable for medial surface computation purposes.
Delaunay based reconstruction: For each triangle $F = \{\mathbf{f}_i\} \subset \partial\Omega$, we use the inverse of the FT computed at skeleton extraction (Sec. 3.2.1) to gather all skeleton points $S(F)$ having $\mathbf{f}_i$ as feature points. Next, we project all points in $S(F)$ on the plane of F, triangulate these 2D projections [153], and use the resulting mesh patch to connect the points in $S(F)$. The reason for 'lifting' the connectivity from 2D into 3D is

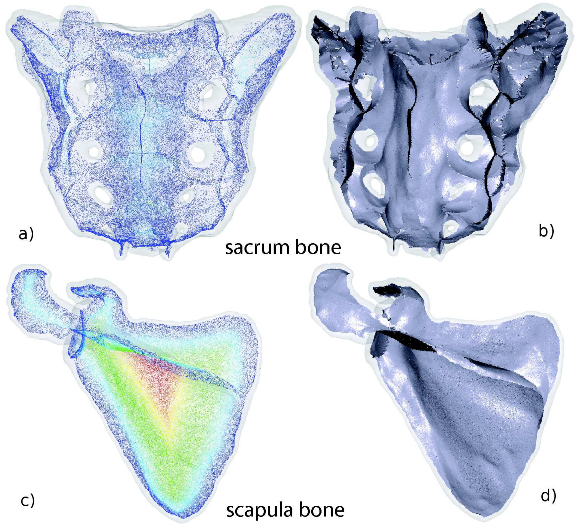


Figure 3.8: Anatomic shapes: point clouds (a,c) and surface skeletons (b,d).

that locally planar surface patches (*i.e.*, triangles F) create, by definition, locally-planar skeleton patches (*i.e.*, triangulation of $S(F)$). This creates duplicate skeleton-mesh triangles, since close model faces have common skeleton points in convex areas. To remove these, we mark all model vertices which map via the FT only to *internal* triangles, *i.e.*, which do not have edges on the boundary of a Delaunay triangulation [153], and skip faces having only marked vertices. The method is $O(N)$ for N skeleton points, since we triangulate small point sets $S(F)$ of size $O(1)$. This takes under 3 seconds for all shapes in this chapter. We use only *local* information (skeleton points of a small surface neighborhood), so we can do out-of-core reconstruction of large skeletons, like [23]. A similar surface reconstruction method for point-clouds describing a surface in 3D space was presented in [27]. In contrast to their context, we do not use principal component analysis (PCA) to determine the projection plane, but obtain this plane by using the feature transform. Additionally, we use this method to reconstruct a polygonal description of the surface skeleton (of an input shape), rather than the polygonal description of the surface of that shape (given a point cloud sampling thereof).

Figures 3.8 and 3.9 show our reconstruction results. All small details (cow eyes, hooves, horns, and pig snout) create skeletal manifolds. Noisy skeletons have no 'stitches' between close ligature sheets (Figure 3.9 m,o,q). It is challenging to reconstruct such manifolds *only* from point clouds – ligatures match surface concavity pairs [122], so their cloud density can be arbitrarily small even for densely sampled models (Sec. 3.2.1). The inverse FT links ligatures to the input surface and thus reconstructs them well. Simplified skeleton meshes are easily created by

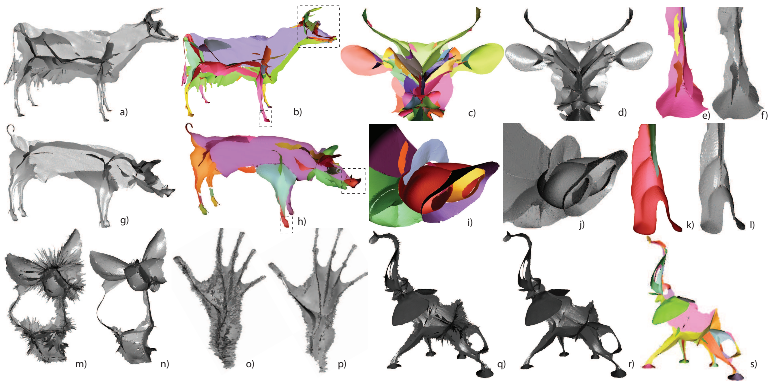


Figure 3.9: Delaunay method (a,g; details d,f,j,l; simplified clouds (m-r) and per-manifold method (b,h,s; details c,e,i,k) for skeleton reconstruction (Sec. 3.4.1).

filtering low-importance points, compare Figs. 3.9 n,p,r to the raw skeletons in Figs. 3.9 m,o,q.

3.5 MEDIAL SURFACE COMPARISON

We next qualitatively compare our point-cloud surface skeletons (PCS) with the discrete scale axis (DSA) method [108], one of the best methods for extracting detailed surface skeletons (Figure 3.10). For similar skeleton simplification levels, PCS and DSA create similar skeletons (Figure 3.10 g-h,m-n). Yet, differences exist (Figure 3.10, red marks). These have two causes: geometry (different skeleton points found) and topology (same skeleton points found but connected differently). Geometry differences imply topology differences. Note how DSA found many skeleton points outside the hand model (Figure 3.10 e-f) and connected these to points inside the hand. In the cow model (Figure 3.10 c-d), skeleton points are largely identical, but DSA wrongly connects the tail and rump skeletons. Such issues, due to strong model concavities, are noted in [108].

DSA can skip large parts of the skeleton periphery, see *e.g.* the pig and cow spine and belly and elephant spine (Figure 3.10 b,d,j). These parts, found by our PCS, are *ligature* sheets between the core skeleton and faraway skeleton points in shallow surface cusps [122]. The reason hereof is that PCS and DSA use different skeleton *scale* metrics: PCS uses geodesic importance (a global metric); DSA computes simplified skeletons by up-scaling the input shape (a local operation). DSA also creates many small holes in skeletal sheets, see Figure 3.10 l. These artefacts (for a genus 0 shape) are likely due to the numerical degeneracies listed in [108].

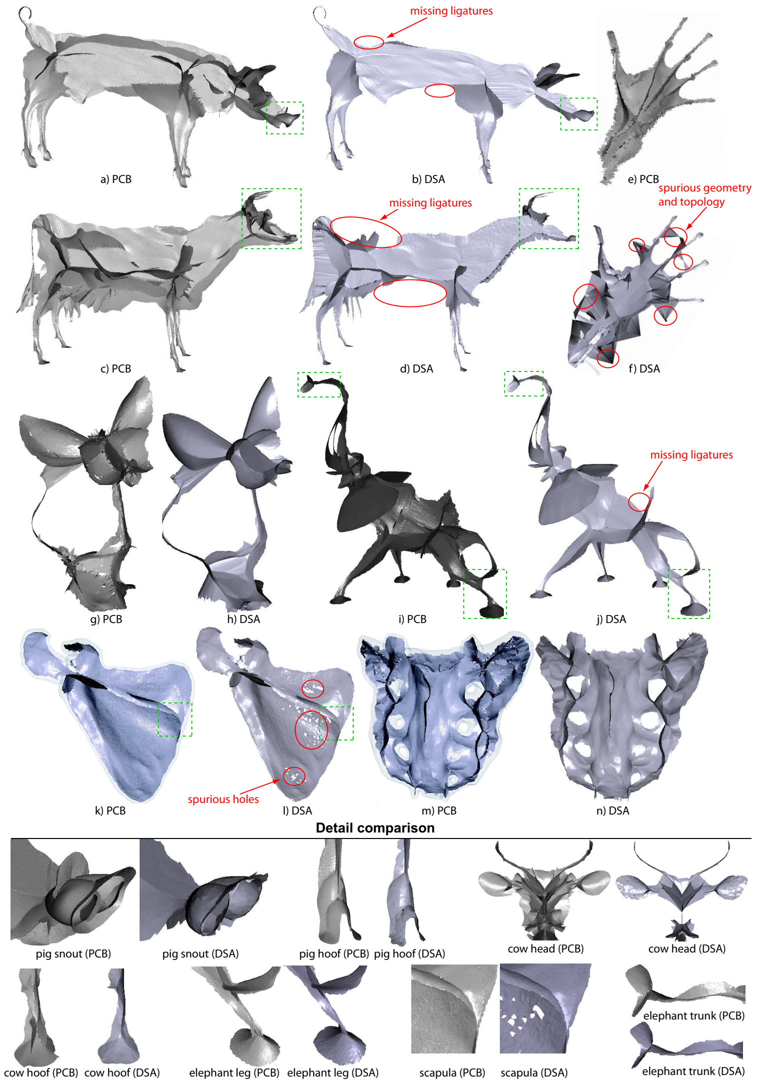


Figure 3.10: Comparison of PCS and DSA methods (Sec. 3.5). Skeleton parts wrongly added/missed by DSA are shown in red. Green-marked details are shown in the insets below.

Table 4: Timing comparison of PCS and DSA skeletonization methods.

Model	PCS (our method)		DSA(Miklos *et al.* [108])		
	time (sec.)	points	time (sec.) $\delta = 0.007$	time (sec.) $\delta = 0.005$	points
pig	27.9	224539	485	975	145153
cow	19.4	185703	448	919	64240
bird	3.3	46862	201	421	104140
horse	17.5	193887	369	744	63297
asiandragon	14.9	230964	n/a	n/a	n/a
hand	4.8	196920	378	778	93301
elephant	3.5	50422	505	1031	166089
armadillo	13.2	172955	426	853	105167

Table 4 shows timing and size statistics for PCS and DSA. For PCS, we used an accuracy $\tau = 10^{-3}$ (Sec. 3.2.1), and also added regularization time. For DSA, we used approximation thresholds δ for the mesh to union-of-balls (UoB) conversion of 0.007 and 0.005 ([108], Sec. 2). Note that $\tau = 10^{-3}$ for PCS is similar to $\delta = 10^{-3}$ for DSA as for skeleton accuracy. PCS is up to 100 times faster than DSA. More accurate UoB settings (smaller δ) make DSA much slower: 45 minutes for a 313K-point skeleton of a 177K-triangle shape at $\delta = 0.002$ [108]. We could not test such settings as $\delta \leq 0.003$ made DSA crash on our models. The same was true for the *asiandragon* model, $\delta = 0.007$. If we replace our SSG regularization by simpler metrics *e.g.* θ-SMA, PCS becomes much faster, basically identical to the timings in Tab. 1.

PCS and DSA create skeletons of different sizes. PCS creates one skeleton point per input point (Sec. 3.2.1). DSA uses a Voronoi diagram, which has a different point count. Yet, as Figure 3.19 shows, the skeletal *detail* created by PCS is similar to DSA.

We also compared PCS with the method of Stolpner *et al.* [165] which approximate medial axes as UoB point clouds. On a 3.4 GHz PC, PCS is over 100 times faster (Tab. 4 *vs* Tab. 1 [165]).

3.5.1 *Direct and inverse correspondence*

In our method, each computed medial point cloud has a direct correspondence to two surface points, by construction. This relationship is maintained by the feature points $(\mathbf{f}_1, \mathbf{f}_2)$. This way, from each point $\mathbf{x} \in S_{\partial\Omega}$ of the surface skeleton of an input shape Ω, we can go back to two of its closest points $\mathbf{f}_1 \in \partial\Omega, \mathbf{f}_2 \in \partial\Omega$. This is essentially a sampled version of the feature transform $FT_{\partial\Omega}$ (Equation 2.3). Of course, this representation is limited: For medial points located on the surface-skeleton boundary (A_3 points, see Section 2.2.2), or medial points located on the intersection of several medial sheets (A_1^3 points), we should have more than two feature points. For implementation simplicity reasons (having to do with the difficulty of maintaining variable-sized arrays in CUDA), we only record two such points per surface-skeleton point. We discuss next in Chapter 6 how this limited medial information can be refined to correctly classify medial points.

Separately, for each surface point $\mathbf{x} \in \partial\Omega$, we record all skeleton points $\mathbf{y} \in S_{\partial\Omega}$ that have $\mathbf{x}$ as feature point. This essentially delivers us the inverse link (from input shape to its surface skeleton), or the inverse $FT_{\partial\Omega}^{-1}$. In contrast to our implementation decision for the direct feature transform $FT_{\partial\Omega}$, we allow $FT_{\partial\Omega}^{-1}$ to have a variable (unbounded) number of points. We chose to do this since we compute $FT_{\partial\Omega}^{-1}$ on the CPU and only after the full surface-skeleton has been extracted (on the GPU). As such, storing the variable-length arrays inherent to the structure $FT_{\partial\Omega}^{-1}$ is easy to do.

Overall, the above two feature transforms $FT_{\partial\Omega}$ and $FT_{\partial\Omega}^{-1}$ allow us to establish a bidirectional mapping between $\partial\Omega$ and $S_{\partial\Omega}$. In turn, this lets us use the best suitable space (shape or its skeleton) to compute desired properties, and next map these accordingly to the other representation or space. Figure 3.11 demonstrates this mapping between the two spaces. Here, we compute, for each medial point $\mathbf{x} \in S_{\partial\Omega}$, the local shape thickness, *i.e.* the distance from $\mathbf{x}$ to any of its feature points. Next, we use the feature transform $FT_{\partial\Omega}$ to 'map' this thickness field to the input shape surface $\partial\Omega$. This way, we obtain an easy-to-interpret image that shows us locally thin shape parts (in red) as opposed to locally-thick shape parts (in green, Figure 3.11 d). This surface-to-skeleton (or inverse) mapping is next extensively used in this thesis (Chapters 5 and 6).

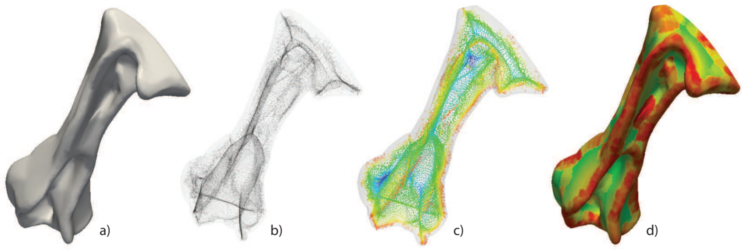


Figure 3.11: Surface-skeleton to shape mapping: Using the feature transform $FT_{\partial\Omega}$, values computed on the medial surface $S_{\partial\Omega}$ can be transferred to the surface. **a)** Polygonal surface (gyrus bone); **b)** medial point cloud; **c)** point cloud with per-point thickness values; **d)** Shape surface showing local thickness.

3.6 CURVE SKELETON EXTRACTION FROM THE SURFACE SKELETON

In contrast to *surface* skeletons, which are 2D manifolds which contain the loci of maximally- inscribed balls in a shape [154], *curve* skeletons are 1D curves which are locally centered in the shape [31]. Compared to surface skeletons, curve skeletons can only capture the main topologic properties of a shape, and much less of its geometric properties. As such, they are less well suited for tasks such as detailed surface simplification or accurate reconstruction. However, they are in general simpler to compute, and well suited for many applications in computer vision, path planning, robotics, shape matching, and computer animation.

The previous sections focused on the extraction and regularization of the medial surface from large polygonal shape representations. This section focuses on the computation of curve skeletons for the same type of shapes. Unlike several proposed methods to directly extract a curve skeleton from an input voxel model, mesh or dense point cloud [7, 19, 37, 101, 135, 172, 174], we propose here to extract the curve skeleton as a subset of the surface skeleton. By this, we leverage the accurate and fast computation of surface skeletons proposed in the previous sections. Moreover, we will show that computing curve skeletons from surface skeletons guarantees several desirable properties, such as centeredness and robustness to noise.

The overall idea for extracting curve skeletons is straightforward: Since the curve skeleton is centered in the shape, a metric defined on the surface skeleton and which monotonically increases from the surface-skeleton boundary towards its center, can be used to compute a curve skeleton subset from the original medial surface.

Curve skeleton points have two or more shortest straight geodesics between their feature points [37, 135]. The above definition for curve skeleton points can be quite easily applied to detect such points for voxel-based models, as shown in [135]. For surface skeletons represented as point clouds, as in our case, the above criterion cannot be immediately used, mainly due to the typically non-uniform density of skeleton point clouds. Therefore, we propose to extract curve skeletons by a method akin to the technique of Siddiqi *et al.* [155, 156]. Our proposal is based on the following observation: In a small vicinity $\mathcal{N}$ of a curve-skeleton point, geodesic tangents, mapped to $\mathcal{N}$, abruptly change directions (Fig. 3.12). Given this observation, we extract curve-skeleton points by looking for high-response points of weight-averaged tangent directions in vicinities around each surface-skeleton point, by a three-step method: (i) find candidates close to the curve skeleton; (ii) filter and regularize candidates; and (iii) reconstruct the curve skeleton. These steps are presented next.

3.6.1 *Detecting candidate curve skeleton points*

For each surface skeleton point $\mathbf{s}_i$, we compute the average (projected) tangent direction of its SSG path, *i.e.*,

$$\begin{aligned}
\mathbf{t}'_{s,i} &= \mathbf{t}_{s,i} - \mathbf{F}_i(\mathbf{F}_i \cdot \mathbf{t}_{s,i}) \\
\mathbf{t}'_{e,i} &= \mathbf{t}_{e,i} - \mathbf{F}_i(\mathbf{F}_i \cdot \mathbf{t}_{e,i}) \\
\mathbf{t}_i &= \frac{\mathbf{t}'_{s,i} + \mathbf{t}'_{e,i}}{\|\mathbf{t}'_{s,i} + \mathbf{t}'_{e,i}\|}
\end{aligned} \tag{3.2}$$

where $\mathbf{t}_{s,i}$, $\mathbf{t}_{e,i}$ are the tangent vectors at its feature points ($\mathbf{f}_1$, $\mathbf{f}_2$), $\mathbf{F}_i$ is defined as $\frac{(\mathbf{f}_1-\mathbf{f}_2)}{\|\mathbf{f}_1-\mathbf{f}_2\|}$, *i.e.*, the normalized feature vector of $\mathbf{s}_i$. Projection improves the detector reliability (see below) close to Y-intersection curves, *i.e.*, where several skeletal manifolds intersect [34]. Tangent vectors $\mathbf{t}_i$ span a vector field $\mathbf{T}$ over the surface skeleton (Figure 3.12).

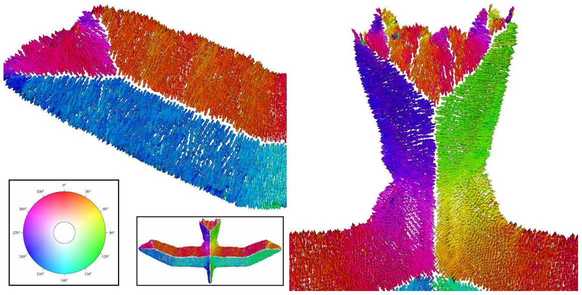


Figure 3.12: Tangent vector field $\mathbf{T}$ (shown with directional color-coding).

One can evaluate the divergence of $\mathbf{T}$ or its more numerically-stable flux [155]. Points close to the curve skeleton have large positive flux/-divergence values. However, along with these, this approach may yield also points close to the Y-intersection curves.

We find candidate curve-skeleton points differently. For each skeleton point $\mathbf{s}_i$, let $\mathcal{N}_i$ be its set of neighbour skeleton points (we use 10 nearest-neighbours in practice). We measure the likelihood of $\mathbf{s}_i$ to be a curve-skeleton point as

$$I(\mathbf{s}_i) = \rho_i - \rho_i \left\| \frac{\sum_{j\in\mathcal{N}_i} w_{ij}\mathbf{t}_j}{\sum_{j\in\mathcal{N}_i} w_{ij}} \right\|, \quad w_{ij} = |\mathbf{F}_i \cdot \mathbf{F}_j|\, e^{-\frac{(\rho_i-\rho_j)^2}{2\sigma_1^2} - \frac{\|\mathbf{s}_i-\mathbf{s}_j\|^2}{2\sigma^2}}$$

with $\rho_i \equiv \rho(\mathbf{s}_i)$. The importance $I_i \equiv I(\mathbf{s}_i)$ averages tangent vectors in $\mathcal{N}_i$ with weights given by a 2D Gaussian kernel and σ_1, σ_2 set to the median of the distances $\|\mathbf{s}_i - \mathbf{s}_j\|_{j\in\mathcal{N}_i}$. The weight w_{ij} lowers the impact of tangent vectors $\mathbf{t}_j$ of skeleton points $\mathbf{s}_j$ which: (i) have feature vectors not parallel to $\mathbf{f}_i$, (ii) have geodesics of different lengths (following [37, 135]), and (iii) are far from $\mathbf{s}_i$. Points close to Y-intersection curves meet conditions (i) and (ii), so they contribute weakly to I. In contrast, points close to the curve skeleton yield large weights, and have tangent vectors pointing outwards in all directions, see Figure 3.12. Such points have large I values, so we find the set $\mathscr{C}$ of candidate curve-skeleton points by thresholding I at a small value $T_I > 0$.

3.6.2 *Regularization of candidate curve-skeleton points*

We assign an importance μ_i to each point $\mathbf{s}_i \in \mathscr{C}$ to prune spurious curve skeleton details. We set μ_i to the smallest surface area between two SSGs of *nearby* skeleton points. This is similar to the metric in [135] which was computed by a flood-fill on a voxel surface. Our case is more complicated as we have two curves on an unstructured *mesh*. We efficiently *approximate* this area using only the angle between the two geodesics and the lengths of a few additional straightest geodesics, as follows.

For each candidate $\mathbf{s}_i \in \mathscr{C}$, we find a neighbour $j^\star \in \mathscr{M}_i$, with $\mathscr{M}_i$ a neighbourhood of $\mathbf{s}_i$ (10 nearest-neighbours), for which

$$J_j = (1 + \mathbf{t}_i \cdot \mathbf{t}_j)\, e^{-\frac{(I_i - I_j)^2}{2\sigma_1^2} - \frac{\|\mathbf{s}_i - \mathbf{s}_j\|^2}{2\sigma_2^2}} \tag{3.3}$$

is minimal, *i.e.*, $j^\star = \operatorname{argmin}_{j \in \mathscr{M}_i} J_j$. Since $\mathbf{s}_i$ and $\mathbf{s}_{j^\star}$ are spatially close, we assume that their feature points coincide, and use $\mathbf{s}_i$ for these. Let $\theta = \angle(\mathbf{t}_i, \mathbf{t}_{j^\star}) \in [0, \pi]$ be the angle between the tangent vectors of $\mathbf{s}_i$ and $\mathbf{s}_{j^\star}$. The pair of SSGs $\gamma_{se,i}$ and $\gamma_{se,j^\star}$ between the feature points $\mathbf{f}_{1,i}$ and $\mathbf{f}_{2,i}$ divide the surface $\partial\Omega$ in two parts, the smallest area of which we want to estimate. For this, we trace P straightest geodesics $\gamma_{S,i,k}$, $1 \leq k \leq P$ from $\mathbf{f}_{1,i}$ to $\mathbf{f}_{2,i}$ on $\partial\Omega$, with uniformly spread starting angles $\alpha_k = 2\pi k/P$ around the vector $\mathbf{f}_i = \mathbf{f}_{1,i} - \mathbf{f}_{2,i}$ at $\mathbf{f}_{1,i}$. Assuming that each geodesic is half of an *ellipse*, with minor axis $\mathbf{f}_i$, the ellipse radii a_k and $b_k \geq a_k$ are given by a simple approximation formula for an ellipse perimeter, *i.e.*, $a_k = \|\mathbf{f}_i\|/2$ and $b_k = (2\,\|\gamma_{S,i,k}\| - \pi\, a_k)/2$. Next, we approximate $\partial\Omega$ between two consecutive geodesics by an *oblate spheroid* with radii a_k and $c_k = (b_k + b_{k+1})/2$, so its area is that of an oblate spheroid wedge with angle $\beta = 2\pi/P$, *i.e.*,

$$S_k = \beta\, c_k^2 + \frac{\beta\, a_k^2}{2e} \ln \frac{1+e}{1-e}, \tag{3.4}$$

with $e = \sqrt{1 - a_k^2/c_k^2}$. Assuming that the starting direction $\alpha_0 = 0$ corresponds to the tangent of $\gamma_{se,i}$, we compute μ_i as

$$\mu_i = \min\left(\sum_{k, \alpha_k < \theta} S_k, \sum_{k, \alpha_k \geq \theta} S_k \right). \tag{3.5}$$

Thresholding μ removes short curve-skeleton branches to yield the final candidate set $\mathscr{C}'$. We use more paths $P = 50$ than for the surface skeleton metric ($M = 20$, Sec. 3.3) to limit area estimation errors. σ_1 and σ_2 are set to the median distance in $\mathscr{M}_i$.

3.6.3 *Curve skeleton reconstruction*

To get the final curve skeleton CS, we connect points in $\mathscr{C}'$ by line segments, by adapting the ball-pivoting method [12]. We start from the point with largest importance μ, find its neighbour in $\mathscr{C}'$ within a radius r with largest μ value, and add a new line segment to CS. Next, we try to extend CS by searching neighbours of its end vertices $\mathbf{e}_i$. To become a new end vertex, a point $\mathbf{x}$ must (i) be within distance r from an $\mathbf{e}_i$; and (ii) the segment $(\mathbf{x}, \mathbf{e}_i)$ must be well-aligned with the curve tangent. When CS cannot be extended, we backtrack and try to extend from vertices of previous segments. This captures the curve skeleton branching.

3.6.4 *Comparison*

Regularization (Sec. 3.6.2) is the costliest step of our curve-skeleton extraction. $\mathscr{C}$ is a small subset of the surface skeleton, but tracing $P = 50$ straightest geodesics $\gamma_{S,i,k}$ per point $\mathbf{s}_i \in \mathscr{C}$ is still expensive, as no early termination can be used (Sec. 3.3). Still, we only need to compute geodesic lengths, which results in a very efficient CUDA mapping (see Tab. 5).

Table 5: Curve-skeleton extraction timings.

Model	Detect (seconds)	Regularize (seconds)	Reconstruct (seconds)	Total (seconds)
Cow	3.1	3.5	0.8	7.4
Bird	0.5	1.2	0.1	1.8
Horse	2.5	2.6	0.2	5.3
Asiandragon	2.6	3.7	1.7	8.0
Hand	1.3	1.9	0.6	3.8
Elephant	0.5	1.5	0.5	2.5
Buddha	3.3	5.3	0.8	9.4
Mouse	3.0	4.6	0.9	8.5
Dragon	3.5	4.7	0.9	9.1
Pig	2.5	4.0	0.8	7.3
Armadillo	1.8	1.5	0.6	3.9
Rabbit	0.5	1.1	0.5	2.1

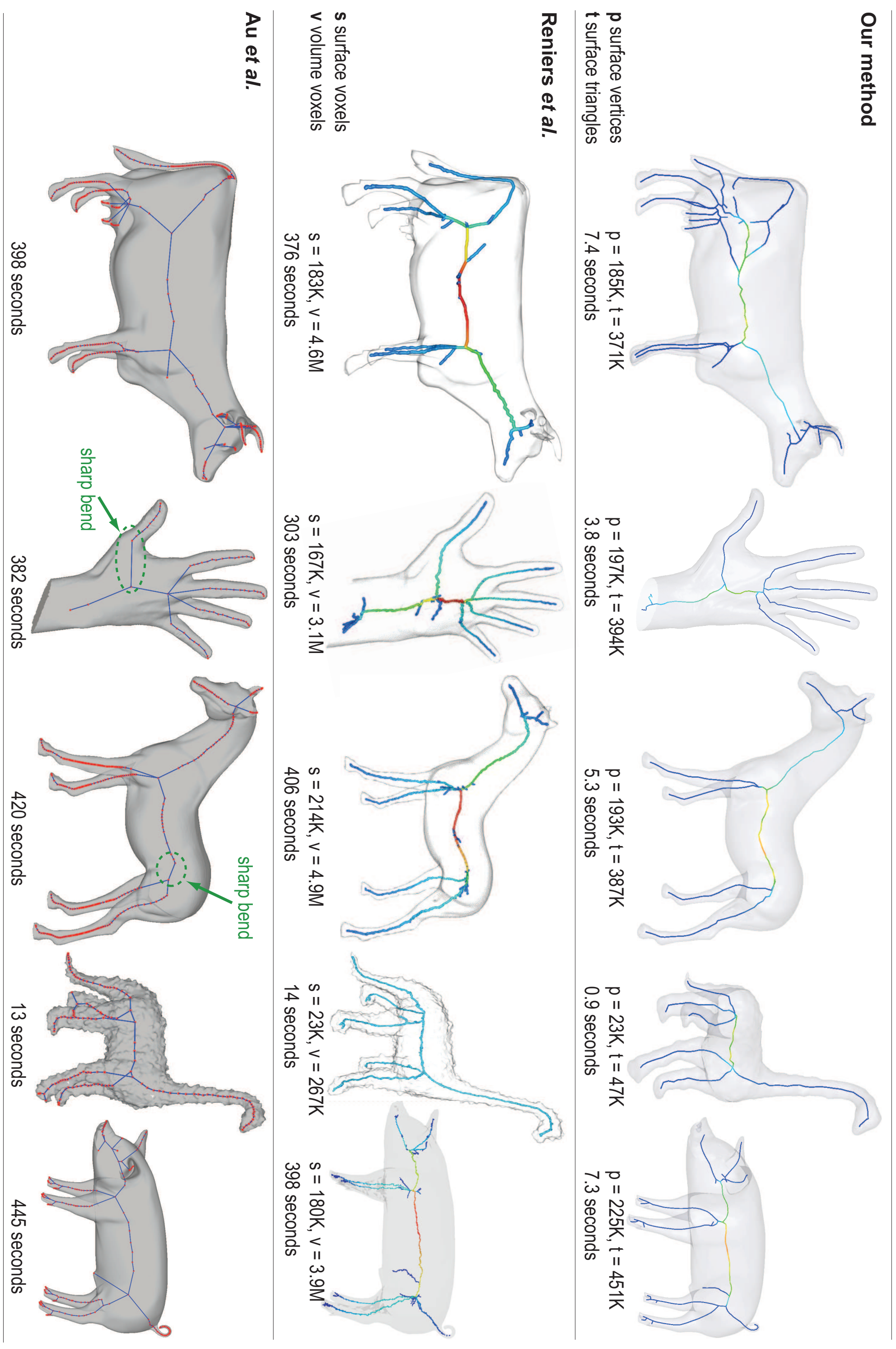


Figure 3.13: Curve skeleton extraction: our method (top row), voxel-based method [135] (middle row), and mesh collapse method [7] (bottom row).

Figure 3.13 compares our results with Reniers *et al.* [135] and with of Au *et al.* who compute curve-skeletons by shape collapse via Laplacian smoothing [7]. Although our point count is smaller than, or at most equal to, the surface voxel count of Reniers *et al.*, we find more skeleton branches, *e.g.* the cow udder and horns. Our skeletons, unlike Au *et al.*, do not have artificial straight-line branches and sharp bends. Au *et al.* added these in a 'surgery' step to connect disjoint skeleton parts (green areas, hand and horse model, Figure 3.13 bottom). The skeletons of Au *et al.* extend deeper into surface cusps, *e.g.* the pig's hooves and snout. Such branches are shortened by our regularization (Sec. 3.6.2). Our method is on average 50 times faster than Au *et al.* and over one order of magnitude faster than Reniers *et al.* Interestingly, the costs of the latter two are similar, since both methods 'visit' the entire input volume: Au *et al.* while collapsing the mesh, and Reniers *et al.* while computing its voxel skeleton detectors.

3.7 CURVE SKELETON EXTRACTION FROM PROJECTIONS

As we have seen in Section 3.6, curve skeletons can be extracted from corresponding surface skeletons. Intuitively, we can say that curve skeletons can be seen as the local centers of such surface skeletons, or 'skeletons of skeletons'. In other words, points $\mathbf{x}$ on a curve skeleton can be seen as maximizing the sum of distances from $\mathbf{x}$ to the surface $\partial\Omega$ of the input shape [181]. If this observation holds, then we can next think about the relationship of curve skeletons to the 2D skeletons of 2D *projections* of the shape Ω.

In this context, Livesu *et al.* recently observed that the 2D projections of the curve-skeleton of Ω (on any projection plane π) are very close to the 2D skeletons of the projections of Ω (on the same plane π). Following this observation, they proposed to extract the curve skeleton of Ω from a (large) set of 2D *views* of Ω [99]. Key to this computation is the extraction of a volume which encodes, at each 3D point inside the shape, the probability that the curve-skeleton passes through that point. The curve-skeleton is extracted from this volume using a set of post-processing techniques. This approach has the major advantage that it requires only a set of 2D views of the input shape, so it can be used when one does not have a complete 3D shape model. However, this method strongly depends on the quality of the skeletal probability volume. Overall, this type of approach follows earlier work that uses a set of 2D projections of a 3D volumetric dataset to reconstruct various aspects of the 3D dataset, such as the well-known Radon transform used in medical imaging to reconstruct a 3D scalar volume from 2D projections thereof. As far as we are aware of, the method of Livesu *et al.* is the first time when a similar principle was applied to the extraction of 3D curve skeletons.

We present here an extension of the view-based approach of Livesu *et al.*, with the following contributions. First, we propose a different way for computing the curve-skeleton probability and representing it as a dense point cloud. On the one hand, this eliminates a major part of the original proposal's false positives (*i.e.*, locations where a curve-skeleton point is suggested, but no such point actually exists), which makes our probability better suited for further skeleton extraction. On the other hand, our point-cloud model eliminates the need for a costly voxel representation. Secondly, we propose a fast GPU implementation of the point cloud computation which also delivers the skeleton probability with higher accuracy than the original method. We demonstrate our technique on several complex 3D models.

3.7.1 *View-based curve skeletonization*

In contrast to the object-space curve skeletonization approaches discussed in Section 3.6, which require a full 3D model of the input shape W, Livesu *et al.* proposes a different approach: Noting that the 2D projection of a 3D curve skeleton is close to the 2D skeleton of the projection, or view, of an input 3D shape, they extract curve skeletons by merging 2D skeletal information obtained from several views of the input shape Ω. Given two such views C_i and $C_i^\perp$, whose up-vectors are parallel and lines of sight are orthogonal, the silhouettes B_i and $B_i^\perp$ of Ω are first computed by orthographic projection of the input shape. Secondly, the 2D skeletons $S_{\partial B_i}$ and $S_{\partial B_i^\perp}$ of these silhouettes are computed. Next, stereo vision is used to reconstruct the 3D skeleton: Point pairs $p \in S_{B_i}$ and $p^\perp \in S_{B_i^\perp}$ are found by scanning each epipolar line, and then back-projected into 3D to yield a potential curve-skeleton point $\mathbf{x}$. The points $\mathbf{x}$ found in this way are accumulated into a so-called *probability volume* $\mathscr{V} \subset \mathbb{R}^3$, which gives, at each spatial point, the likelihood to have a curve-skeleton passing through that point.

The above method has several advantages compared to earlier object-based techniques. First, it can be used directly on shape *views*, rather than 3D shape models, which makes it suitable for any model which can be rendered in a 2D view, regardless of its representation (*e.g.* polygons, splats, points, lines, or textures). Moreover, the method does not require a full and consistent description of the 3D shape – a set of 2D *views* of that shape is sufficient. Finally, the method can be easily parallelized, as view pairs are treated independently. However, this method fundamentally relies on the fast computation of a *good* probability volume which contains a correct estimation of the curve skeleton location. This poses the following requirements:

1. a **reliable** and **accurate** stereo vision correspondence matching, *i.e.* finding the correct pairs of points $(p \in S_{\partial B_i}, p^\perp \in S_{\partial B_i^\perp})$ which

Here and in the remainder of this section, we denote by italics (*e.g.*, p) the 2D projection of a 3D point $\mathbf{p}$ in a 2D view C.

represent the projection of the same curve skeleton point in the view-pair $(C_i, C_i^\perp)$;

2. an **accurate** and **efficient** representation of the probability volume $\mathscr{V}$ for further processing.

Requirement (1) is not considered by Livesu *et al.*, where *all* possible point-pairs along an epipolar line are back-projected. This generates, as we shall see in Sec. 3.7.4.2, a large amount of noise in the probability volume $\mathscr{V}$. Removing this noise requires four relatively complex post-processing steps in the original proposal. Secondly, the probability volume $\mathscr{V}$ is represented as a voxel grid. This makes the method unnecessarily inaccurate, relatively slow and hard to parallelize, and requires large amounts of memory, thus contradicts requirement (2).

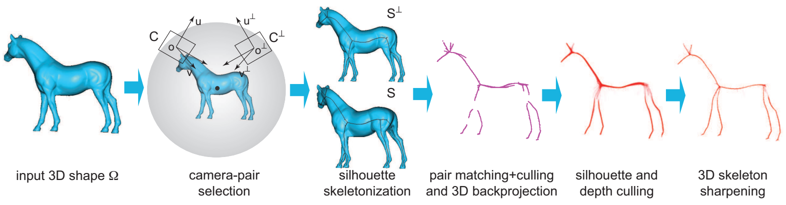

input 3D shape Ω
camera-pair selection
C
u
o
u⊥
C⊥
o⊥
v
S⊥
S
silhouette skeletonization
pair matching+culling and 3D backprojection
silhouette and depth culling
3D skeleton sharpening

Figure 3.14: Curve-skeleton probability computational pipeline.

In the following, we present several enhancements to the original idea of Livesu *et al.* that make view-based skeleton extraction compatible with requirements (1) and (2). This allows us to extract a high-accuracy probability volume for further usage in curve skeleton computation or direct visualization.

3.7.2 *Accurate probability volume computation*

Our proposal has three steps (see also Figure 3.14). First, we extract regularized and subpixel-accuracy 2D skeletons from several views of the input shape (Sec. 3.7.3). Next, we use additional view-based information to infer a conservative set of correspondences between points in such 2D skeleton pairs, back-project these in 3D, and record the obtained points as a point cloud (Sec. 3.7.4). Finally, we apply an additional sharpening step on the 3D point cloud, which directly delivers a highly accurate curve-skeleton probability (Sec. 3.7.5).

3.7.3 *Robust 2D skeletonization*

Given a shape Ω and orthographic camera specification $C = (\mathbf{o}, \mathbf{v}, \mathbf{u})$ described by its origin $\mathbf{o}$, view direction $\mathbf{v}$, and up-vector $\mathbf{u}$, we start by computing the silhouette B of Ω by rendering the shape on the camera's

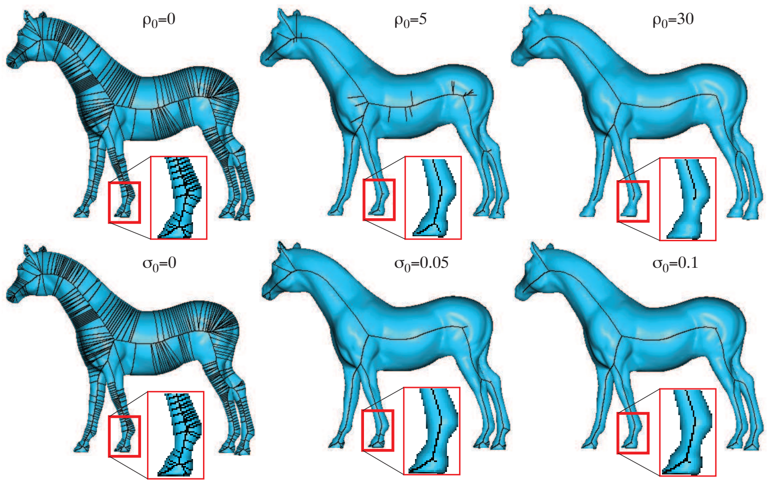


Figure 3.15: Skeleton regularization. Top row: Importance-based method [183] for three different threshold values ρ_0. Bottom row: Salience-based method [177] for three different threshold values σ_0.

view plane $(\mathbf{u}, \mathbf{v} \times \mathbf{u})$. Next, we compute the so-called salience 2D skeleton of B using the technique presented in [177]. The *salience* of a point $\mathbf{p} \in B$ is defined as

$$\sigma(\mathbf{p}) = \frac{\rho(\mathbf{p})}{DT_{\partial B}(\mathbf{p})} \tag{3.6}$$

Here, $DT_{\partial B}(\mathbf{p})$ is the 2D distance transform of the silhouette boundary ∂B and $\rho(\mathbf{p})$ is the so-called skeleton *importance*

$$\rho(\mathbf{p}) = \max_{\mathbf{f}_1, \mathbf{f}_2 \in FT_{\partial B}(\mathbf{p})} \|\gamma_{\mathbf{f}_1\mathbf{f}_2}\| \tag{3.7}$$

where $FT_{\partial B}$ is the feature transform of the boundary ∂B, and $\gamma_{\mathbf{ab}}$ is the compact boundary fragment between two points $\mathbf{a}$ and $\mathbf{b}$ on ∂B. The importance ρ increases monotonically from the endpoints (tips) of the skeleton towards its center. $\rho(\mathbf{p})$ associates, to each skeleton point $\mathbf{p}$, the length of the longest boundary arc (in pixels) subtended by its feature points. Upper thresholding ρ with a value ρ_0 will thus remove both skeleton branches created by small boundary wiggles *and* end-parts of important skeleton branches caused *e.g.* by boundary corners. Figure 3.15 (top row) shows this effect for a silhouette B of a horse model. For $\rho_0 = 1$, we get the full 2D skeleton, which contains many spurious branches. For $\rho_0 = 5$, we get the desired skeleton detail at the legs and head, but still have several spurious branches around the rump and neck. For $\rho_0 = 30$, we eliminate all spurious branches, but also loose relevant portions of branches corresponding to the legs. This is undesired, since, as we shall later see, we need the important branches at their full length to reconstruct a curve-skeleton reaching into all shape protrusions.

In contrast, the salience metric σ (Eqn. 3.6) delivers a better result. As shown in [177], σ is high along the most important, or salient, skeleton branches, and low elsewhere. Hence, we can threshold σ to obtain the skeleton

$$S_{\partial B} = \{\mathbf{p} \in B | \sigma(\mathbf{p}) > \sigma_0\}. \tag{3.8}$$

Equation 3.8 delivers a clean, regularized, skeleton whose spurious branches are eliminated, and whose important branches extend all the way into the shape's protrusions. Figure 3.15 (bottom row) shows the saliency-based regularization. For $\sigma_0 = 0$, we obtain the same full skeleton as for $\rho_0 = 1$. Increasing σ_0 over a value of 0.05 practically removes all spurious branches, but keeps the important ones un-pruned (see zoom-ins). As such, we use the value $\sigma_0 = 0.05$ further in our pipeline.

We further enhance the precision of the computed skeleton by using the subpixel technique presented in [166]. As such, skeleton points are stored as 2D floating-point coordinates rather than integers. This will be important when performing the 3D stereo reconstruction (Sec. 3.7.4.2).

3.7.4 *Accurate correspondence matching*

We find potential 3D curve-skeleton points along the same key idea of [99]: Given a camera $C = (\mathbf{o}, \mathbf{v}, \mathbf{u})$, where $\mathbf{v}$ points towards the object's origin, we construct a pair-camera $C^\perp = (\mathbf{o}^\perp, \mathbf{v}^\perp, \mathbf{u}^\perp)$ which also points at the origin and so that the two up-vectors $\mathbf{u}$ and $\mathbf{u}^\perp$ are parallel. In this case, projected points p in C correspond to projected points $p^\perp$ in $C^\perp$ located on the same horizontal scanline. Given such a point-pair $(p, p^\perp)$, the generated 3D point $\mathbf{x}$ is computed by triangulation, *i.e.* by solving

$$\mathbf{x} = \mathbf{p} + k\mathbf{v} = \mathbf{p}^\perp + k^\perp \mathbf{v}^\perp \tag{3.9}$$

where $\mathbf{p}$ and $\mathbf{p}^\perp$ are the 3D locations, in their respective view planes C and $C^\perp$, corresponding to p and $p^\perp$ respectively, and k and $k^\perp$ are the distances between $\mathbf{x}$ and the view planes of C and $C^\perp$. Note that $\mathbf{p}$ and $\mathbf{p}^\perp$ can be immediately computed as we know the positions of p and $p^\perp$ and the cameras' positions, orientations, and near plane locations.

3.7.4.1 *Correspondence Problem*

As well known in stereo vision, the success of applying Eqn. 3.9 is fundamentally conditioned by having the *correct* 2D points p and $p^\perp$ paired in the two cameras. Let us analyse this issue in our context: Consider that a scan-line y intersects a 2D skeleton shape in m points on the average. Hence, we have m^2 possible point-pairs. These will generate m^2

points in the 3D reconstruction, whereas in reality there are only at most m such points – that is, if no occlusion is present. The excess of $m^2 - m$ points are false positives. Given N such camera-pairs placed uniformly around the object in order to reconstruct its 3D curve skeleton, and considering a camera viewplane of $P \times P$ pixels, we have in the worst case $O(N(m^2 - m)P)$ false-positive points in the curve skeleton. The ratio of false-to-true positives is thus $\Pi = O\left(N(m^2 - m)P)/(NmP)\right) = O(m)$. In our measurements for a wide set of shapes, we noticed that $m = 5$ on the average. Concretely, at an image resolution of $P^2 = 1024^2$ pixels, and using the setting $N = 21$ from Livesu *et al.*, we thus get over 400K false-positive points generated in excess of the $NPm \simeq 100K$ true-positive skeleton points.

The above false-to-true-positive ratio Π is a conservative estimate: Given a rigid shape Ω, the 2D skeleton of its silhouette can change considerably as the silhouette changes, even when no self occlusions occur. This, and additional self-occlusion effects, reduce the true-positive count and thus increases Π. This ultimately creates substantial noise in the curve-skeleton probability estimation, and thus makes an accurate curve skeleton extraction more complex.

3.7.4.2 *Pair-culling heuristic*

We reduce the false-to-true-positive ratio Π by using additional information present in our cameras, as follows. Consider a point p on a scanline L in C and all points $L^\perp = \{p_i^\perp\}$ on the same scanline in $C^\perp$ (see Figure 3.16 e). The 3D reconstructions of all pairs $(p, p_i^\perp)$ lie along the line $\mathbf{p} + k\mathbf{v}$ (Eqn. 3.9). Hence, if we had an estimate of the depth k_{est} between the correct reconstruction and the viewplane of C, we could select the best pair $p_{est}^\perp$ for p as

$$p_{est}^\perp = \underset{p_{est}^\perp \in L^\perp}{\arg\min} |k_{est} - k| \tag{3.10}$$

i.e. the point in $C^\perp$ which yields, together with p, a depth closest to our estimate. We estimate k_{est} as follows: When we draw the shape in C, we also compute its nearest and furthest depth buffers $\mathscr{Z}_n$ and $\mathscr{Z}_f$, by rendering the shape twice using the OpenGL *GL_LESS* and *GL_GREATER* depth-comparison functions respectively. Next, for each point p in the viewplane of C, we set $k_{est} = \frac{1}{2}\left[\mathscr{Z}_n(p) + \mathscr{Z}_f(p)\right]$ (see Figure 3.16 e).

It is essential to note that our heuristic for k_{est} is *not* an attempt to find the exact value of the depth k. Indeed, if we could do this, we would not need to apply Eqn. 3.10, as we could perform the 3D backprojection using a single view. We use k_{est} only as a way to select the most likely point-pair for 3D reconstruction. This is argumented as follows: First, we note that the value k for the correct point-pair must reside between $\mathscr{Z}_n(p)$ and $\mathscr{Z}_f(p)$ - indeed, the reconstructed 3D point $\mathbf{x}$ must be inside the object's hull. Secondly, the curve skeleton is roughly situated in the (local) mid-

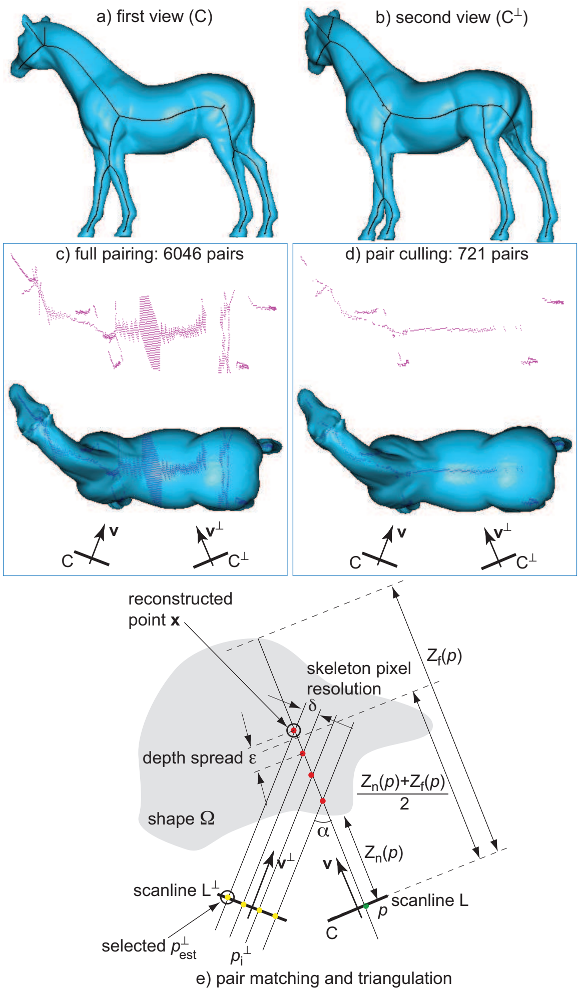

a) first view (C)
b) second view ($C^\perp$)
c) full pairing: 6046 pairs
d) pair culling: 721 pairs
$\mathbf{v}$
$\mathbf{v}^\perp$
C
$C^\perp$
reconstructed point $\mathbf{x}$
skeleton pixel resolution
δ
$Z_f(p)$
depth spread ε
$\frac{Z_n(p)+Z_f(p)}{2}$
shape Ω
α
$Z_n(p)$
scanline $L^\perp$
scanline L
p
selected $p^\perp_{est}$
$p^\perp_i$
e) pair matching and triangulation

Figure 3.16: Correspondence matching for curve-skeleton reconstruction. A camera pair (a,b). Reconstructed 3D points when using full pairing (c) and when using our depth-based pairing (d). Depth-based pairing and triangulation (e)

dle of the object, thus its depth is close to k_{est}. Thirdly, we note that, when the angle between the cameras' vectors $\alpha = \angle(\mathbf{v}, \mathbf{v}^{\perp})$ decreases, then the depths k_i yielded by Eqn. 3.10 for a set of scanline-points $p_i^{\perp} \in L^{\perp}$ get further apart. In detail, if the distance between two neighbour pixels in the scanline $L^{\perp}$ is δ, the distance between their reconstructions using the same point p in the other scanline L is $\varepsilon = \delta / \sin(\alpha)$, see Figure 3.16 c. Hence, if we use a small α (under 90^{deg}), we get fewer depths k_i close to k_{est}, so we decrease the probability that selecting the point whose depth is closest to k_{est} (Eqn. 3.10) will yield an incorrect point-pair for the 3D reconstruction. In contrast, Livesu *et al.* use $\alpha = 90°$, as this slightly simplifies Eqn. 3.9. Given that low α values reduce the likelihood to obtain false pairs using our depth heuristic, we prefer this, and set $\alpha = 20°$.

Figure 3.16 shows the results of using our depth-based pairing heuristic. Images (a) and (b) show the two skeletons $S_{\partial B}$ and $S_{\partial B^{\perp}}$ corresponding to the two cameras C and $C^{\perp}$ respectively. The brute-force many-to-many correspondence pairing yields 6046 three-dimensional points. As visible in Figure 3.16 c, these points are spread uniformly in depth along the view directions of the two cameras. This is expected, since 2D skeleton pixels are equally spaced in the image plane. For clarity, we displayed here only those points which pass the silhouette and depth-culling, *i.e.* which are inside the object from *any* considered view (see further Sec. 3.7.5). The displayed points in Figure 3.16 c are thus *final* points in the curve-skeleton probability volume delivered by many-to-many matching.

Figure 3.16 d shows the reconstructed 3D points when we use our depth-based pair-culling. Since we now only have one-to-one pairs, we obtain much less points (721 *vs* 6046, see the explanations in Sec. 3.7.4.1). Moreover, these points are located very close to the actual curve skeleton, as shown by the top view of the model.

Given our conservative point-pair selection, as shown in Figure 3.17, we generate much fewer curve-skeleton points than if using many-to-many pairing. Although this is highly desirable for obtaining an accurate (false-positive-free) curve skeleton, it also means that the curve skeleton will be sparser than when using all possible pairs. To counteract this, we simply use more view pairs N. In practice, setting $N \simeq 500$ yields sufficiently dense curve skeletons (see results in Sec. 3.8). An additional advantage of using more views is that we do not need to carefully select the optimal views for stereo reconstruction, in contrast to the original method, where such views are obtained by performing a principal component analysis (PCA) on both the 3D shape and its 2D projections.

We further reduce the number of tested point-pairs (Eqn. 3.10) by scanning L from left to right (for p) and $L^{\perp}$ from right to left (for $p^{\perp}$), As such, 3D points are generated in increasing order of their depth k, so $|k_{est} - k|$ first decreases, then increases. Hence, we stop the scan as soon as $|k_{est} - k|$ increases, which gives an additional speed improvement.

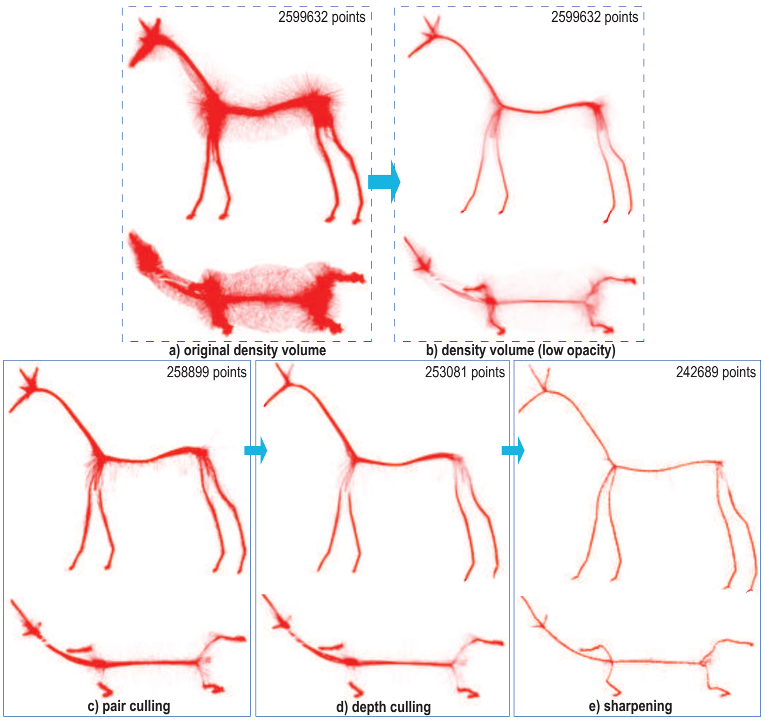


Figure 3.17: Curve-skeleton probability point-cloud. (a) original method [99]. (b) Cloud in (a) displayed with lower opacity. (c) Effect of depth-based pairing. (d) Effect of depth culling. (e) Effect of sharpening (see Sec. 3.7.5).

3.7.5 *Probability sharpening*

We collect the 3D points $\mathbf{x}$ (Eqn. 3.9) found by the depth-based correspondence matching for a given camera-pair $(C, C^{\perp})$ in an unstructured point cloud $\mathscr{CS}$. As C rotates around the input shape, we keep testing that the projections x of the accumulated points $\mathbf{x} \in \mathscr{CS}$ fall inside the silhouette B in C, as well as within C's depth range $[\mathscr{Z}_n(x), \mathscr{Z}_f(x)]$. Points which do not pass these tests are eliminated from $\mathscr{CS}$. The depth test explained above comes atop the silhouette test which was already proposed by Livesu *et al.*. Whereas the silhouette test constrains $\mathscr{CS}$ to fall within the visual hull of our input shape, we constrain $\mathscr{CS}$ even further, namely to fall within the exact shape. The difference is relevant for objects with cavities, whose visual hull is larger than the object itself.

However closer to the true curve-skeleton than the results presented by Livesu *et al.*, our $\mathscr{CS}$ still shows some spread around the location of the true curve skeleton. This is due to two factors. First, consider the inherent variability of 2D skeletons in views of a 3D object: The 2D skeleton is locally centered with respect to the silhouette (or projection) of a 3D object Ω. In areas where Ω has circular symmetry, the 2D skeleton of the projected 3D object is indeed identical to the 2D projection of the true 3D curve skeleton, *i.e.*, skeletonization and projection are commutative. However, this is not true in general for shapes with other cross-sections. Moreover, self-occlusions, in the case of concave objects, will generate 2D skeletons which have little in common with the projection of the curve skeleton. It is important to stress that this is *not* a problem caused by wrong correspondence matching. Secondly, using a small α angle between the camera pairs (Sec. 3.7.4.2), coupled with the inherent resolution limitations of the image-based skeletons, introduces some depth estimation errors which show up as spatial noise in the curve skeleton.

We further improve the sharpness of $\mathscr{CS}$ as follows. For each camera C which generates a silhouette B, we move the points $\mathbf{x} \in \mathscr{CS}$ parallel to the view plane of C with a step equal to $\nabla DT_{\partial B}$. Since $\nabla DT_{\partial B}$ points towards $S_{\partial B}$, this moves the curve-skeleton points towards the 2D skeleton $S_{\partial B}$. Note that, in general, $\nabla DT_{\partial B}$ is not zero along $S_{\partial B}$ [183]. Hence, to prevent points to drift along $S_{\partial B}$, and thus create gaps in the curve skeleton, we disallow moving points $\mathbf{x}$ which already project on $S_{\partial B}$. Note also that this advection never moves points outside Ω, since $S_{\partial B}$ is always inside any silhouette B of Ω. Since the above process is done for all the viewpoints C, the curve skeleton gets influenced by *all* the considered views. This is an important difference with respect to Livesu *et al.*, where a 3D skeleton point is determined only by *two* views.

Figure 3.17 shows the effect of our three improvement steps: depth-based pairing (Sec. 3.7.4.2), depth culling, and sharpening. All images show 3D point clouds rendered with alpha blending. Figure 3.17 a shows the cloud $\mathscr{CS}$ computed following the original method of Livesu *et al.*, that is, with many-to-many correspondence matching along scanlines

(Sec. 3.7.4.1). Clearly, this cloud contains a huge amount of points not even close to the actual curve skeleton. If we decrease the alpha value in the visualization, we see that this cloud, indeed, has a higher density along the curve-skeleton (Figure 3.17 b). We see here also that naive thresholding of the density, which is quite similar to decreasing the alpha value to obtain Figure 3.17 b from Figure 3.17 a, creates problems: If a too low threshold is used, the skeleton still stays thick; if a too high threshold is used, the skeleton risks disconnections (see white gaps in the neck region in Figure 3.17 b).

Eliminating the large amount of false positives from the cloud shown in Figure 3.17 a is very challenging. To do this, Livesu *et al.* apply an involved post-processing pipeline: (1) voxelize the cloud into a voting grid; (2) extract a maximized spanning tree (MST) from the grid; (3) detect and prune perceptually salient tree branches; (4) collapse short branches; (5) recover curve-skeleton loops lost by the MST; and (6) smooth the resulting skeleton; for details we refer to [99]. Although this is possible, as demonstrated by the results of Livesu *et al.*, this post-processing is highly complex, delicate, and time-consuming.

Figure 3.17 c shows our curve-skeleton probability, obtained with the center-based correspondence pair culling (Sec. 3.7.4.2). The point cloud contains now around ten times less points. Also, note that points close to the true curve skeleton have been well detected, *i.e.*, we also have few false negatives. Applying the depth-based culling further removes a small amount of false positives (Figure 3.17 d *vs* Figure 3.17 c). Finally, the density sharpening step effectively attracts the curve-skeleton points towards the local skeleton in each view, so the overall result is a sharpening of the point cloud $\mathcal{CS}$, *i.e.* a point density increase along the true curve skeleton and a density decrease further from the skeleton (Figure 3.17 e).

3.7.6 *Results*

Figure 3.18 shows several results computed with our method. The produced curve-skeleton clouds contain between 100K and 300K points. We render these clouds using small point splats of 2 by 2 pixels, to make them more visible. The key observation is that our skeleton point clouds are *already* very close to the desired 3D location, even in the absence of any cloud post-processing. In contrast, the equivalent point clouds delivered by the method of Livesu *et al.* are much noisier (see example in Figure 3.16 a and related discussion in Sec. 3.7.4.2), and thus require significant post-processing to select the true-positives. Since our point clouds are much sharper, we can directly use them for curve-skeleton visualization, as shown in Figure 3.18. If an explicit line representation of such skeletons is desired, this can be easily obtained by using *e.g.* the curve-skeleton reconstruction algorithm described in [183], Sec. VIII-C. Thin tubular skeleton representations can be obtained by isosurfacing the

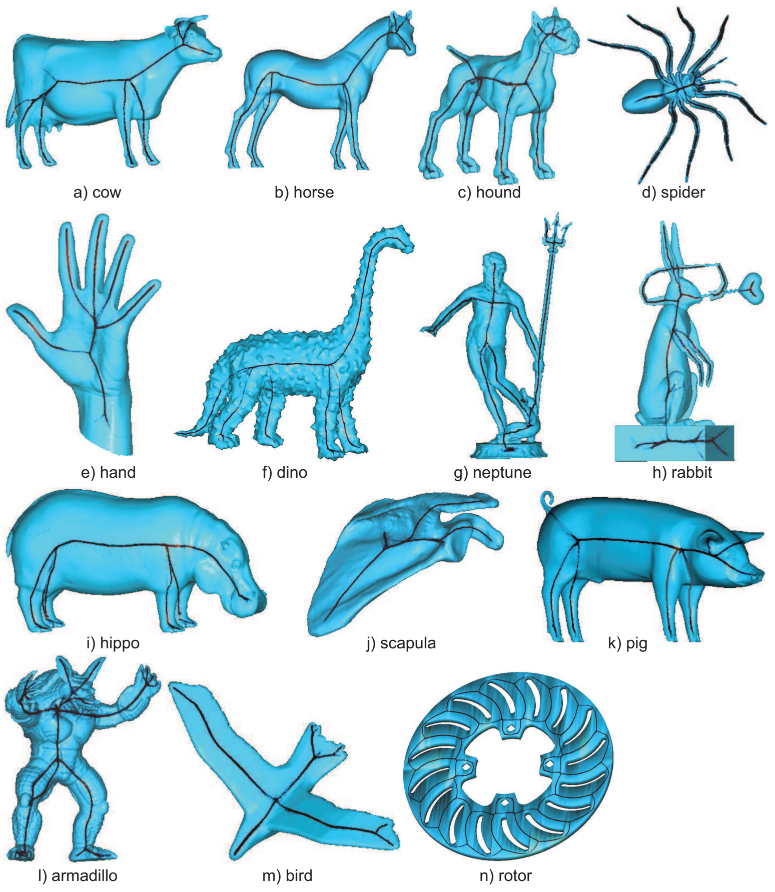


Figure 3.18: Curve-skeleton probability point-clouds for several models (see Sec. 3.8).

density field induced by our 3D point cloud. In this chapter, we refrained from producing such reconstructions, as we want to let our main contribution stand apart – the computation of noise-free, accurate point-cloud representations of the curve skeleton probability.

3.7.6.1 *Comparison*

Figure 3.19 compares our method with several recent curve-skeleton extraction methods. As visible, our curve skeleton has the same overall structure and positioning within the object. However, differences exist. First, our method produces smoother curve skeletons than [37] and [183]. This is due to the density sharpening step, which does not have an equivalent in the latter two methods. Also, [7] requires a so-called connectivity surgery step to repair the curve skeleton after the main Laplacian advection has completed. This necessary step has the undesired by-product of creating straight-line internal skeleton branches (Figure 3.19 d, palm center). Secondly, we correctly find the skeleton's ligature and internal branches. This is also the case for all other methods except [99], where all skeleton branches are merged in a single junction point (Figure 3.19 b). This fact is not surprising, given the branch collapsing post-processing step in the latter method. It is not clear to us why this step is required (or beneficial), as it actually changes the topology of the skeleton, and thus may impair operations such as shape analysis or matching.

3.7.6.2 *Parameters*

All parameters of the method are fixed and independent on the input shape, *i.e.* skeleton saliency threshold $\sigma_0 = 0.05$ (Sec. 3.7.3), number of considered views uniformly distributed around a sphere centered in the object center $N = 500$ (Sec. 3.7.4.2), screen resolution $P^2 = 1024^2$ pixels (Sec. 3.7.4.1), and angle between the camera-pair view vectors $\alpha = 20°$ (Sec. 3.7.4.2). Less views $(N < 500)$ will generate sparser-sampled curve skeletons, as discussed in Sec. 3.7.4.2. Decreasing the pixel resolution generates slightly thicker point distributions in the curve-skeleton cloud. This is expected, since we have less *and* coarser-spaced 2D skeleton pixels, which also implies higher depth estimation errors (Eqn. 3.9). Decreasing α under roughly 5 degrees generates too large inaccuracies in the depth estimation; increasing it over roughly 30 degrees reduces the likelihood of good correspondence pairing; hence, our setting of $\alpha = 20°$.

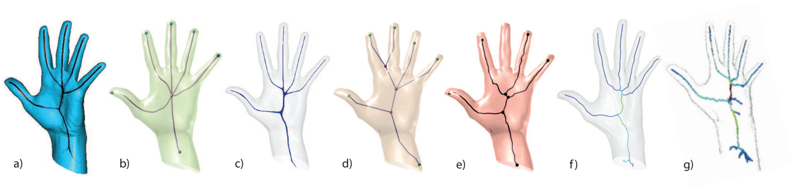


Figure 3.19: Comparison with related methods: (a) our method; (b) [99]; (c) [183]; (d) [7]; (e) [37]; (f) [183]; (g) [135] (see Sec. 3.8)

3.8 DISCUSSION

This chapter dealt with the extraction of medial descriptors (feature points, regularized surface skeletons, and regularized curve skeletons) from polygonal 3D shape representations. Methods for the extraction of surface and curve based medial axis from three dimensional shapes have been presented. We next discuss and outline our key contributions, and thereby set the scope of the extensions and refinements proposed in the next chapters.

3.8.1 *Regularized surface skeleton extraction*

INPUT The presented method works on both uniformly-sampled and non-uniformly sampled mesh models. These meshes can be of any genus (see the rabbit, cat, and dragon models), self-intersecting (cow model), and non-closed (hand model). In particular, the sampling resolution of the extracted surface skeletons follows the local sampling resolution of the input mesh: Highly-sampled mesh areas generate highly-sampled surface skeleton areas, while coarsely-sampled mesh areas generate coarsely-sampled surface skeleton areas. This is a desirable property, since highly-sampled mesh areas typically contain more surface details, which in turn should lead to more surface-skeleton details. Conversely, coarsely-samples mesh areas are typically poor in details, so by producing coarsely-sampled surface skeleton areas for them, we save both computation time and storage space. The proposed skeleton extraction method requires only an oriented point cloud (no connectivity is required). Connectivity data is only used for the computation of the skeleton regularization.

ACCURACY Voxel-based skeletons are limited by the voxel resolution [20, 66, 135]. Like Stolpner *et al.* [165], our skeletons are point clouds close to the true medial axis within a user-prescribed precision in world space. Separately, our image-based shape reconstruction from its (simplified) skeleton is real-time and near-pixel accuracy.

SCALABILITY A 1024^3 distance-and-feature-transform volume needs at least 4 GB RAM [20, 64, 180]. An equivalent mesh, roughly 1 M triangles, needs only 24 MB RAM, which is essential given typical 1 GB GPU RAM limits. Voxelization has also large speed costs and is delicate for certain meshes [43, 50, 116]. Multi-resolution voxel schemes reduce memory costs, but complicate algorithms and reduce GPU speedups. Table 1 (Sec. 3.2.1) ($\tau = 10^{-3}$, $\varepsilon = 0$, equivalent to a 1024^3 volume) shows that our method is over 100 times faster than [17, 37, 93, 108, 135, 165], even without voxelization costs.

GEODESIC COMPUTATION Our GPU computation of shortest, straightest geodesics (SSGs) is over two orders of magnitude faster, and more accurate, than state-of-the-art techniques [121, 170, 190], making global skeleton regularization practical for large models. Our method is directly usable for any application requiring fast, near-exact, geodesics on meshes. As such, this method can be used also for other applications besides computing regularized surface skeletons.

SIMPLICITY Our framework has no complex computational geometry operations or degenerate cases, unlike [108, 135, 165]. Its only user parameters are the skeleton centeredness τ and number of geodesic directions M, explained in Secs. 3.2.1 and 3.3.

3.8.2 *Curve skeleton extraction*

We discuss next our two proposed methods for extracting curve skeletons from surface skeletons (Section 3.6) and from shape projections respectively (Section 3.7).

3.8.2.1 *Curve skeleton extraction from surface skeletons*

Existing *curve* skeleton extractors have widely different speed, accuracy, and curve skeleton definitions [7, 25, 31, 37, 64]. Our curve skeletons cannot replace *all* such methods – as we have noted earlier with several occasions, many curve skeleton methods exist, and most of them use different curve skeleton definitions. Certain of these definitions may (or may not be) optimal for certain applications. However, from a structural perspective, our method is novel in the sense that it offers a fast way to extract curve skeletons from a *surface* skeleton cloud – in other words, we show that curve skeletons can be seen and computed as 'skeletons of skeletons'. The only method that is directly equivalent to our approach that we are aware of is presented in [181]. There, the curve skeleton is computed by iteratively shrinking a surface skeleton point cloud in the local direction given by the bisector of the skeletal feature vectors. In contrast to this iterative approach, our method presented here works in an 'integral', rather than incremental (or differential) manner – we identify curve-skeleton points by computing a cumulative importance metric that discriminates them from regular surface-skeleton points, similar to [135].

Apart from [181], the closest methods to our approach are the medial geodesic function (MGF) [37] and ROSA [172]. Yet, we use a different angle-based criterion than MGF and also than ROSA which computes curve skeletons as centers of point-cloud projections on a cut plane found by optimizing for circularity. Computationally, we are two orders of magnitude faster than ROSA and MGF, and on average 50 times faster than Au *et al.* [7] (Sec. 3.6.3). Additionally, compared to the variational method of Hassouna *et al.* [64], we are 20 times faster (Figure 3.13, Tab. 5 *vs* Figure 10 in [64]).

3.8.2.2 *Curve skeleton extraction from shape projections*

Our projection-based method extends the view-based curve-skeleton extraction of Livesu *et al.* in several directions: (1) Using salience-based skeletons to guarantee preservation of terminal skeleton branches, (2) using depth information to reduce the number of false-positives in the 3D skeleton reconstruction, and (3) sharpening the obtained point-cloud representation to better approximate the 1D singularity locus of the curve skeleton. We trade off speed for accuracy, by generating more conservative skeleton samples and using more viewpoints. However, by using a GPU implementation, we achieve the same speed as the original method, but deliver a much cleaner and sharper 3D skeleton point-cloud approximation. Overall, our method can be used either as a front-end for reconstructing line-based representations of 3D curve skeletons, or for directly rendering such skeletons as unstructured point clouds.

Our image-based method maintains all of the desirable properties of curve skeletons advocated by related work [7, 31, 99, 174, 183]: Our skeletons are **thin** and locally **centered** within the object. Higher-**genus** objects (with tunnels) are handled well (see rabbit and rotor models, Figure 3.18). The method is **robust** against noise, due to the sharpening step (see dino and armadillo models, Figure 3.18). Thin, sharp **detail** protrusions of the models generate curve skeleton branches, as long as these parts project to at least 1 pixel in screen space (see neptune, spider, and rabbit models, Figure 3.18). This is due to the usage of the 2D skeleton saliency metric, which keeps 2D skeleton branches reaching into such salient shape details (Sec. 3.7.3). Input model **resolution**, *e.g.* polygon count, is largely irrelevant to the end result, since 2D skeletons are computed in image space.

In terms of **limitations**, our image-based method cannot recover complete curve skeletons for shape parts which are not visible from any viewpoint, *i.e.*, permanently self-occluded. This is an inherent problem of view-based 3D reconstruction. For such shapes, the object-space skeletonization methods mentioned in Chapter. 3 should be used.

We have implemented our method in C++ with OpenGL and CUDA and tested it on a 2.8 GHz MacBook Pro with an Nvidia GT 330M graphics card. The main effort is spent in computing the regularized 2D salience skeletons (Sec. 3.7.3). We efficiently implemented the computation of $DT_{\partial B}$, $FT_{\partial B}$, ρ, and σ (Eqns. 2.1-3.6) using the method in [20], one of the fastest exact Euclidean distance-and-feature transform techniques in existence (see our publicly available code at [178]). The remaining steps of our pipeline are trivial to parallelize, as points and camera views are treated independently. Overall, our entire pipeline runs roughly at 500 frames/second. Given that we use more views than Livesu *et al.*, *i.e.* roughly 500 *vs* 21, our CUDA-based parallelization is essential, as it allows us to achieve roughly the same timings as this method.

3.9 CONCLUSION

This chapter introduced several methods for extracting medial descriptors from large polygonal input shapes. Both surface and curve skeleton extraction methods were presented. Moreover, regularization techniques for both surface and curve skeletons are presented, based on the geodesic-based metric of [135]. All in all, our methods allow the simple, fast, and accurate computation of multiscale curve and surface skeletons from large and complex 3D shapes represented as polygonal meshes. If regularization is not required, our methods can also handle oriented point-cloud shapes. Separately, we have presented a method for the fast and accurate computation of curve skeletons from multiple 2D views of 3D shapes. Together with our 3D curve-skeletonization method, our view-based method strengthens our overall claim that curve skeletons can be seen as 'skeletons of skeletons' of 3D shapes. We believe that this opens new ways for the theoretical understanding of skeletonization (and the resulting different skeleton types) as a recursive dimensionality-reduction process based on the notion of centrality under a suitable (Euclidean) distance-from-boundary metric.

From a practical perspective, we now have efficient, easy-to-use, and robust *tools* that allow us to easily compute highly-accurate skeleton models of complex shapes. As such, this chapter is the starting point towards further exploration of medial properties. In the next chapters, several methods for subsequent analysis of medial surfaces are presented, based on the 3D skeletons computed by the methods presented here.

4 MULTISCALE MEDIAL CLOUD ANALYSIS METHODS

It's not the tools that you have faith in - tools are just tools. They work, or they don't work. It's people you have faith in or not. Yeah, sure, I'm still optimistic I mean, I get pessimistic sometimes but not for long.

Steve Jobs

4.1 INTRODUCTION

In the previous chapter, we have shown how we can efficiently compute accurate surface and curve skeletons from large 3D oriented point clouds and, subsequently, regularize these if surface connectivity information is available. Specifically, given a raw (oriented) point cloud that describes a (possibly non-watertight) surface, we are able now to extract both its 3D curve skeleton and its more complex 3D surface skeleton. Given also connectivity information for the input point cloud (in the form of a polygonal mesh), we are next able to regularize our 3D curve and surface skeletons, thereby generating a multiscale point classification where removing uninteresting small-scale, or noise-induced, branches is subject to a simple thresholding. The entire method is efficiently parallelized on both the CPU and the GPU, and is robust and easy to use for a wide variety of 3D shapes. However, the application of this skeletonization method in practice can encounter a number of challenges, as follows.

First, let us assume that the input shape we want to skeletonize does not come as a mesh, but an unstructured point cloud. In this case, we cannot apply our geodesic-based regularization (Sec. 3.3), since this technique requires the ability to compute geodesics on a *surface* that approximates the input cloud. One could consider tackling this problem by using, in a preprocessing step, one of the many surface reconstruction tech-

This chapter is based on the following papers:

1. J. Kustra, A. Jalba, and A. Telea. Robust segmentation of multiple intersecting manifolds from unoriented noisy point clouds. *CGF*, 33(1):73–87, 2014

niques in order to extract a 3D meshed surface from the input point cloud. As we shall see, however, there are many cases where this reconstruction is very challenging, *e.g.* in the case of clouds having non-uniform point densities, clouds containing a large amount of positional noise, or clouds that do not locally describe the sampling of a manifold. In all such cases, we cannot directly apply our skeletonization to the input cloud. Moreover, if the extracted 3D mesh does not match certain quality properties (manifold structure, non-degenerate faces, no self-intersections), computing geodesics on such meshes is challenging.

Secondly, as discussed in Sec. 3.8, our skeletonization method produces an unstructured point *cloud* that approximates the surface skeleton, rather than a *surface* representation of the same skeleton. This skeletal point cloud is sufficient for certain applications, such as the image-based shape reconstruction from its surface skeleton discussed in Sec. 3.4. For other applications, however, we need more information such as the separate medial sheets or the point connectivity information. Examples of such applications are shape segmentation methods using the surface skeleton [130], surface classification methods [134], or, more plainly, visualizing the manifold structure of the surface skeleton.

One way to jointly tackle the above two problems is to focus our attention on methods that process both general point clouds (such as those that describe our input 3D shapes) and 3D skeletal point clouds (such as those created by our skeletonization methods presented in Chapter 3). Specifically, we are interested in methods that allow us to recover the *manifold structure* of such point clouds. When applied on our input shapes, this delivers us the connectivity information required for our geodesic-based skeleton regularization. When applied on our output surface skeletons, this delivers us compact skeletal representations which are required by subsequent applications.

In this chapter, we present a general-purpose method that is able to denoise a point cloud, by eliminating its outlier samples, and also segment it into a set of manifolds. As compared to other related methods, our proposal can robustly handle point clouds which represent a collection of non-watertight, self-intersecting, manifolds with boundaries. Using this method, we show how we can denoise point clouds similar to our input shapes to be skeletonized, so that we can next use standard surface-reconstruction methods to obtain our desired input meshes. Next, we show how our method can segment a skeletal point cloud (potentially containing noise points, *i.e.* skeleton fragments corresponding to small-scale input-shape noise or details) into separate manifolds, so that we can next use the same standard surface-reconstruction methods mentioned above to create a 3D meshed representation of our surface skeletons.

4.2 NOISY POINT CLOUD SEGMENTATION INTO MANIFOLDS

4.2.1 *Introduction*

Point cloud models of 3D shapes are generated by many applications such as surface scanning [123], stereo reconstruction [136], shape processing [72, 135], medical imaging [157], and skeletonization (see Chapter 3). Extracting clean 3D surfaces from such point clouds is an important task. Many methods exist for reconstructing an approximation of a continuous 2D surface $\mathscr{S} \subset \mathbb{R}^3$ from a point cloud S. However, most such methods pose constraints on the *manifold* structure of $\mathscr{S}$ and/or structure of the point cloud S, such as the sampling density of S, availability of normals, and presence of watertight manifolds. A more restrictive constraint is that $\mathscr{S}$ contains a single non-self-intersecting manifold $\mathscr{M} \subset \mathbb{R}^3$. Point clouds created by 3D laser scans typically meet the above manifold constraint. Other applications however can generate point clouds which sample intersecting manifolds. Examples hereof are medial shape processing 3, where 3D skeleton manifolds naturally intersect; CAD reverse-engineering, where one aims to separate several intersecting shapes from a single point cloud, *e.g.* in the case where topology information was lost; and dimensionality reduction [161].

Extracting non-manifold and/or (self) intersecting manifold surfaces from unoriented noisy point clouds is very challenging, since reconstruction is ill-posed without prior simplifying assumptions on the manifold structure. As such, few methods that are able to treat such point clouds exist in the literature [23]. To address this, we propose to first segment the cloud into subsets which have manifold properties. Such subsets can next be reconstructed into manifold meshes by many existing surface reconstruction techniques, as a post-processing step. Our contribution in this chapter focuses on point cloud segmentation (as opposed to *e.g.* surface reconstruction from point clouds): Given an unoriented point cloud, we first extract locally quasi-flat point groups. Next, we assemble these groups into larger point sets using a global smoothness criterion. No assumption is made on the manifolds' shapes, sizes, or (self) intersections. Finally, we reconstruct meshed manifolds from each such point set using standard existing surface reconstruction techniques. Manifold smoothness is user controllable via two parameters. Our method is robust against geometric or topological noise, *i.e.* extracts smooth, clean, manifolds embedded into noisy clouds.

The structure of this chapter is as follows. Section 4.2.2 reviews related work. We next detail the three steps of our method: local point classification (Sec. 4.2.3.1), global classification (Sec. 4.2.3.4), and manifold reconstruction (Sec. 4.2.3.8). Section 4.2.4 presents applications in manifold cloud clustering, denoising, and medial surface reconstruction. Section 4.3 discusses our method. Section 4.4 concludes the chapter and outlines future work directions.

4.2.2 *Related work*

Extracting manifold surfaces from noisy 3D point clouds has been researched in several fields, as follows.

4.2.2.1 *Data clustering*

If the cloud is a mix of smooth (intersecting) manifolds with noise, extraction can be seen as a data clustering problem. Clustering methods use assumptions on the underlying data, *e.g.* linear manifolds [63, 191], minimal space between manifolds [21], knowing the manifold count [161], or normal data [127]. Spectral clustering can handle curved manifolds, but its performance is low for intersecting manifolds [61]. K-manifolds [161] uses dimensionality reduction [184] to estimate geodesic distances between points and uses this distance to cluster points via expectation-maximization. K-manifolds can extract curved manifolds, but needs to know the number of clusters, where each point can belong to just one cluster. Since a geodesic distance is used to separate clusters, this approach fails for non-intersecting clusters (see further Sec. 4.2.4.1). Kushnir *et al.* [85] cluster manifolds by finding a minimal normalized-cut in a weighted graph. However, this method is not suitable for manifolds with varying sampling density. Recently, Ioannou *et al.* proposed a point-cloud segmentation method based on a so-called Difference of Normals (DoN) operator [70]. Point normals are estimated at two given spatial scales r_1 and r_2, by fitting a plane on each point's neighbors within the radii r_1 and r_2 respectively. Next, the difference of the two normals $\Delta\mathbf{n}$ is computed. Large differences indicate the existence of shapes in the cloud at a scale between r_1 and r_2. This allows selecting points on shapes at a desired (user-provided) scale, and subsequently clustering these points into groups to find separate objects.

4.2.2.2 *Outlier removal*

Points with statistical properties diverging from the desired smooth manifolds are removed [18, 65, 74, 81, 142]. Such methods do not address point clouds specifically. However, data statistics (distribution, distances, and density) can have a too wide variance for non-uniformly sampled clouds with many noisy outliers, *e.g.* tens of percents of the cloud size. Sotoodech *et al.* [160] remove outliers recursively, in decreasing distance from the cluster center. However, such approaches do not work well for datasets where noise and data overlap or which contain non-intersecting manifolds far away from the largest data density (see further Sec. 4.2.4.2). For such datasets, tensor voting (TV) provides an alternative solution [105]. The main idea is to represent local geometric structure, or features, using *e.g.* second-order symmetric tensors of the covariance matrix created from the neighbors of each point. Each point lying on a (potentially-noisy) manifold propagates its local feature in a small neighborhood of user-specified radius σ, by casting a vote to all nearby points. The neighborhood is determined by a dense so-called 'voting field', aligned with the local point. The vote is a tensor generated according to the local tensor and position in the voting field. All votes are accumulated in order to generate the new local tensor or feature. The final tensor field is then further processed to extract 3D surfaces as iso-surfaces of the extremal directions of the tensor field [105, 105].

4.2.2.3 *Surface reconstruction*

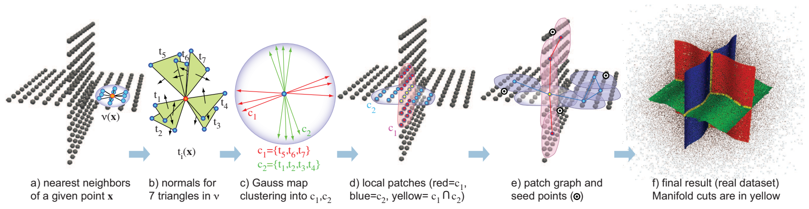
Point-cloud surface reconstruction often uses global priors, *e.g.* surface smoothness, water-tightness, viewpoint-invariance, and topology [2, 12, 29, 38, 39, 46, 72, 77, 149, 173]. For noisy clouds, we distinguish between denoising [189] and surface extraction methods [36, 115, 152]. Chang *et al.* present a comprehensive comparison of the strengths and limitations of 16 surface reconstruction methods [23]. They show that, apart of their own method, all other reviewed methods cannot handle non-manifold surfaces and/or intersecting manifolds. However, the method of Chang *et al.* is much slower, and more complicated, than our proposal (see Sec. 4.3).

4.2.2.4 *Our contribution*

Our goal is to separate manifold points from an input noisy cloud. Our priors are a set of (self-) intersecting manifolds with or without boundaries, such as present in surface-skeleton clouds, among other situations. This is more general than typical priors in data clustering, outlier removal, and surface reconstruction: Unlike manifold clustering, we extract manifolds by labeling points as belonging to several, one, or no manifold (*i.e.*, noise). Unlike outlier removal, we allow a much higher amount and spread of noise. Unlike surface reconstruction, we segment the input cloud into separate manifolds, so we can use standard surface reconstruction techniques onto each such manifold, with no further complications or constraints on the input.

4.2.3 *Method*

Our input is a point cloud $S = \{\mathbf{x}_i\} \subset \mathbb{R}^3$ which represents the sampling of a surface $\mathscr{S} \subset \mathbb{R}^3$ consisting of one or several possibly (self-) intersecting manifolds $\mathscr{M}_i$, embedded into noise. Normals at the points $\mathbf{x}_i$ are not required. We first classify, or label, S into a set of point clouds $M_i \subset S$, such that each M_i is the sampled representation of $\mathscr{M}_i$. The classification handles intersecting manifolds, *i.e.* $M_i \cap M_j$ can be non-void for $i \neq j$. To handle noisy clouds, we allow certain points from S which cannot be classified as belonging to any smooth manifold, to be labeled as noise, and further ignored, *i.e.* $\bigcup_i M_i \subseteq S$.



ν(x)
t5
t6
t7
t1
t4
t2
t3
ti(x)
c1
c2
c1={t5,t6,t7}
c2={t1,t2,t3,t4}
c2
c1
a) nearest neighbors of a given point x
b) normals for 7 triangles in ν
c) Gauss map clustering into c1,c2
d) local patches (red=c1, blue=c2, yellow= c1 ∩ c2)
e) patch graph and seed points (⊙)
f) final result (real dataset) Manifold cuts are in yellow

Figure 4.1: (a-e) Algorithm steps (see Secs. 4.2.3.1-4.2.3.8 for details). (f) Three manifolds are extracted from a noisy point cloud.

Classification has two phases (Figure 4.1). First, for each small spatial neighborhood $\nu(\mathbf{x}_i) \subset S$ of a point $\mathbf{x}_i \in S$, we compute all point sets $\mu^i_j \subset \nu(\mathbf{x}_i)$ which describe quasi-flat manifolds embedded in ν. We call this *local classification* (Sec. 4.2.3.1). Next, we group the point sets μ^i_j into sets M_k which describe large, connected, manifolds $\mathscr{M}_k$. We call this *global classification* (Sec. 4.2.3.4). Finally, we reconstruct piecewise-linear (meshed) representations of the manifolds $\mathscr{M}_k$ from the sets M_k (Sec. 4.2.3.8).

4.2.3.1 *Local classification*

A key property of a 2D manifold is its *local flatness*. To reason about local flatness, we consider each neighborhood $\nu(\mathbf{x}_i)$, $\mathbf{x}_i \in S$. Such neighborhoods can be defined *e.g.* using the k nearest neighbors of $\mathbf{x}_i$ or neighbors within a ball of fixed small radius (Figure 4.1 a), as detailed further in Sec. 4.3.

4.2.3.2 *Local surface estimation*

To determine all manifolds which a neighborhood $\nu(\mathbf{x}_i)$ admits, we first construct all possible flat surfaces which pass through $\mathbf{x}_i$ and two other points $\mathbf{x}_m, \mathbf{x}_n \in \nu(\mathbf{x}_i), i \neq m \neq n$, *i.e.* all triangles $T_\nu = \{(\mathbf{x}_i, \mathbf{x}_m, \mathbf{x}_n)\}$. From these, we exclude degenerate triangles, *i.e.* those which have near-zero area or, in other words, near-collinear or near-identical vertices. The maximum triangle count is given by the number of possible 2-permutations without repetition *i.e.* $\frac{k!}{(k-2)!}$ where $k = |\nu(\mathbf{x}_i)|$ is the neighborhood size. Next, for each neighborhood ν, we gather the unoriented normals $\mathbf{n}_i$ of all triangles $t_i \in T_\nu$ into a normal-set N_ν (Figure 4.1 b).

Consider now the map $\gamma : N_\nu \to \mathbf{S}^2$ where $\mathbf{S}^2$ is the unit sphere in $\mathbb{R}^3$. Here, γ maps from each normal $\mathbf{n} \in N_\nu$ to the sphere point indicated by $\mathbf{n}$, seen as a vector starting at the sphere center. γ is also known as the *Gauss map* of a surface formed by all triangles in T_ν [56]. The peaks of γ coincide with the normal directions of the most salient manifolds, since these are the directions along which many triangles, packed within the small spatial area ν, are oriented. Hence, we can use γ to extract our desired quasi-flat manifolds within ν. We distinguish three cases:

a. If ν samples a quasi-flat surface, all triangles $t_i \in T_\nu$ have similar normals $\mathbf{n}$, so γ has two clear peaks (at $\mathbf{n}$ and $-\mathbf{n}$, respectively);

b. If ν samples the intersection of two or more manifolds, γ has two or more such peak-pairs;

c. If γ does not have clearly separated peaks, but is a rather flat signal, we cannot reliably extract clear quasi-flat manifolds from ν, *i.e.* ν samples a volumetric point distribution rather than a few intersecting quasi-flat surfaces.

All points whose normals fall under such a peak-pair in the Gauss map γ are thus points in ν which belong to the same quasi-flat surface. Hence, if we can reliably find well-separated peaks in γ, we can find the desired separate quasi-flat surfaces in ν. If such peaks are far apart in γ, their corresponding quasi-flat surfaces should *self-intersect* within ν, since ν is small and these surfaces are not parallel. The way we separate far-apart peaks in γ is discussed next in Sec. 4.2.3.3. For illustration hereof, the Gauss map for seven triangles in Figure 4.1 c captures the intersection of two quasi-flat surfaces within the considered neighborhood, one with 3 triangles, the other one with 4 triangles.

4.2.3.3 *Segmenting the Gauss map*

To separate the different quasi-flat surfaces in a neighborhood ν, we cluster the neighborhood's Gauss map γ using the geodesic distance on the sphere $\mathbf{S}^2$ between the map's normals, defined as

$$d(\mathbf{n}_i \in N_\nu, \mathbf{n}_j \in N_\nu) = \arccos|\mathbf{n}_i \cdot \mathbf{n}_j| \in \mathbb{R}^+ \tag{4.1}$$

where $\cdot$ denotes vector dot product. Since we do not know *a priori* the number of peaks in γ, we use a hierarchical bottom-up, full-linkage, agglomerative clustering on the distance matrix induced by d on N_ν [68]. This yields a dendrogram $\mathscr{D}$ whose nodes $n_i = \{t_j \in T_\nu\}$ are sets of triangles having similar normals. We next cut $\mathscr{D}$ at a user-given normal dissimilarity level d_{max}. For each node n_i selected by the cut, we collect all vertices of triangles in n_i to create a so-called quasi-flat surface or *patch* $c_i \in \nu$. We also store in each patch c_i all normals of its triangles. We denote this set of normals by $\mathbf{n}(c_i)$. The threshold d_{max} gives the amount of curvature we allow for our quasi-flat surfaces over ν. Small values of d_{max} yield more, and flatter, patches c_i. Larger values yield fewer, and more curved, patches. For a discussion on parameter setting, see further Sec. 4.3.

Repeating this process, we obtain, for each point $\mathbf{x}_i$ in the input cloud, a set $\pi_i = \{c_j\} \subset \nu(\mathbf{x}_i)$ of quasi-flat patches c_j which pass through $\mathbf{x}_i$. The size of the set π_i tells what type of point $\mathbf{x}_i$ is: If $|\pi_i| = 1$, one patch passes through $\mathbf{x}_i$, so we say that $\mathbf{x}_i$ is a *flat* point. If $|\pi_i| > 1$, more patches pass through $\mathbf{x}_i$, so we consider that $\mathbf{x}_i$ is an *intersection* point of several manifolds or a flat patch surrounded by noise. For example, for a point on the intersection line of the two planes in the cloud in Figure 4.1 d, we get two patches $\pi = \{c_1, c_2\}$. Of course, the notions of flatness and intersection used here are subject to the flatness threshold d_{max} (see also Sec. 4.3). The further distinction between several intersecting manifolds and noise (both occurring when $|\pi_i| > 1$) is not done explicitly by counting the size of the set π_i, but implicitly, by the global process which joins overlapping similar-orientation patches to each other, as described further in Sec. 4.2.3.4.

Our approach is similar to Weber *et al.* [194] where, for each point, all local triangulations containing the current point are projected on a Gauss map, which is next clustered. However, while Weber *et al.* use this local analysis to find sharp edges only, we shall use our local patches to reconstruct manifolds *past* such implicit edges, as described next in Sec. 4.2.3.4.

Conceptually, our quasi-flat patches c_j resemble the idea of splats as used in point-based rendering (PBR), *e.g.* [82, 162]: We also want to approximate the surface (or intersecting surfaces) around each input point cloud $\mathbf{x}_i$ by a local quasi-flat structure, and for this, we use the k nearest neighbors of $\mathbf{x}_i$. Also, both our patches, and PBR splats, overlap their corresponding neighbors, to produce a full coverage of the approximated surface. However, several differences exist. First, PBR splats are typically circular or elliptic in shape, whereas our patches do not have a constrained shape, except for their quasi-planarity given by the Gauss map clustering. Secondly, a patch explicitly stores all normals of its points (used further to merge patches into manifolds, see Sec. 4.2.3.4, whereas a PBR splat typically fits an analytic surface to its sample points and use this surface further. During this merging process, we only enforce *local* quasi-flatness constraints on the resulting manifolds, whereas a PBR splat has typically a *global* smoothness constraint, due to the above-mentioned surface fit. Finally, we allow several patches with significantly different orientations to co-exist at manifold intersection points, whereas PBR splats are typically not used to model such intersecting surfaces. As such, we use here the term *patches* to distinguish these from PBR splats.

4.2.3.4 *Global classification*

To find the desired manifolds M_k from the local patch-sets $\pi_i = \{c_j\}$ computed previously, we assume that each c_j is part of exactly one M_k. We justify this as follows. First, the patches c_j are quasi-flat or lightly curved, by construction – thus, it would make little sense to assign the same quasi-flat structure to two different manifolds. Secondly, any two such patches from the same small neighborhood ν are oriented at strongly different angles, since our clustering threshold d_{max} finds strong, separated, peaks in the Gauss map (see also Sec. 4.3). Thus, when such situations occur, we have two manifolds intersecting in ν, each of the two patches belonging to a different manifold. Thirdly, we assume that all points in any M_k belong to some patch, *i.e.* that our manifolds are a union of patches $\bigcup_k M_k = \bigcup_i \pi_i$.

Assembling patch sets π_i into manifolds M_k is described next.

4.2.3.5 *Patch connectivity graph*

First, we determine how patches in the patch-sets π_i around each point $\mathbf{x}_i$ relate to each other. For this, we construct a patch connectivity graph $G = (V, E = V \times V)$, as follows (Figure 4.1 e). For each patch c_i, we add a graph vertex $u(c_i)$ to V. Given two patches c_α and c_β, we define their surface dissimilarity δ as

$$\delta(c_\alpha, c_\beta) = \min_{\mathbf{n_A} \in \mathbf{n}(c_\alpha),\ \mathbf{n}_B \in \mathbf{n}(c_\beta)} d(\mathbf{n}_A, \mathbf{n}_B) \tag{4.2}$$

with d given by Eqn. 4.1. If two points $\mathbf{x}_i \in S, \mathbf{x}_j \in S$, which belong to the same patch-set, have patches $c_\alpha \in \pi_i, c_\beta \in \pi_j$ whose dissimilarity δ is below a given value δ_{max}, we add an edge to E between $u(c_\alpha)$ and $u(c_\beta)$. Since we compare only points $\mathbf{x}_i, \mathbf{x}_j$ belonging to the same patch-set, δ is evaluated only for close, overlapping patches. Although Eqn. 4.2 is equivalent to a full-linkage between all normals in $\mathbf{n}(c_\alpha)$, $\mathbf{n}(c_\beta)$, we speed up its computation by adding an edge between two graph nodes as soon as we find two normals $\mathbf{n}_A, \mathbf{n}_B$ which are closer than δ_{max}. We store G as a binary adjacency matrix where each entry defines if two patches are connected or not. For a discussion on the setting of δ_{max}, see further Sec. 4.3.

G could be directly used to find points of the different manifolds M_k *e.g.* by computing its connected components via flood filling. However, this only works if manifolds always intersect at non-acute angles, which is not the case for many datasets (see examples in Sec. 4.2.4.2). If manifolds intersect at acute angles, a connected component of G could cover more than a single manifold. Indeed, patches close to manifold intersections and which belong to different manifolds differ too little in terms of normals, and thus get connected when building G. We solve this issue and robustly detect intersecting manifolds from the patch graph G in two steps: seed point detection (Sec. 4.2.3.6) and manifold point labeling (Sec. 4.2.3.7).

4.2.3.6 *Seed point detection*

Seed points $\Omega \subset S$ are cloud points located in quasi-flat areas and far from potential manifold intersections. They are starting points for a flood fill process over G which ultimately delivers our manifold points M_i (Sec. 4.2.3.7). We compute Ω as follows.

1. Mark all cloud points $\mathbf{x}_i \in S$ as unvisited, and set Ω to $\varnothing$.

2. Find an unvisited *flat* point $\mathbf{x}_i$, *i.e.* with $|\pi_i| = 1$ (Sec. 4.2.3.3).

3. Add $\mathbf{x}_i$ to Ω.

4. Mark all unvisited flat neighbors $\{\mathbf{x}_j \in \nu(\mathbf{x}_i) \big| |\pi_j| = 1\}$ of $\mathbf{x}_i$ as visited, using a flood fill.

5. Repeat from step 2 until all points in S are visited.

Ω contains seed points far away from manifold intersections, one seed for each segment of a manifold part delimited by manifold intersection curves and manifold boundaries. For example, for the shape in Figure 4.1 a, we find four seeds, one for each cross arm. The point flatness condition stops the fill, which starts far from intersection areas by construction, to leak from one manifold to another; intersections act as flood barriers, since they have non-flat points. So, we get as many seeds as manifold segments delimited by intersections, *e.g.* four seeds for the shape in Figure 4.1 a.

4.2.3.7 *Manifold labeling*

The seed set Ω (Sec. 4.2.3.6) could be directly used to find manifold connected-components separated by intersection curves. Although such results are useful [188], we aim to find *entire* manifolds *past* intersection curves. For instance, for the shape in Figure 4.1 f, we want to find three intersecting surfaces (red, green and blue in Figure 4.1 f) rather than twelve quarter-surfaces. Since seed points $\mathbf{s}_i \in \Omega$ are flat, their patch-sets have a single patch, *i.e.* $\pi(\mathbf{s}_i) = \{c(\mathbf{s}_i)\}$. We use this observation as follows (see also Figure 4.2 for a 2D sketch): For each $\mathbf{s}_i \in \Omega$, we assign a unique ID to $c(\mathbf{s}_i)$ (step A). Next, we do a flood-fill over patches (step B). For this flood fill, we use the patch connectivity (stored in G) instead of the point neighbors (given by ν) used to find seeds (Sec. 4.2.3.6). Hence, the patch-level flood fill can cross manifold intersections but stays confined to the surface of a *single* manifold.

The patch-level flood fill adds to every patch the IDs of the seed points which flooded through that patch. However, we are interested in having this information at point level, *i.e.*, find which are the seed points which flood through each cloud point. We store this in a compact and fast way by using a binary matrix $B = \{b_{ij}\}$, where each row $1 \leq i \leq |S|$ is a bit-vector whose non-zero values encode the seeds $1 \leq j \leq |\Omega|$ which flooded through point $\mathbf{x}_i$. We finally find the manifolds M_i as all points which share the same row bit-combinations in B. Hence, we detect as many manifolds as different row bit-combinations we have in B.

Points lying on manifold intersections contain seed-point information from all manifolds intersecting at that location. To add such points to their intersecting manifolds, we visit the k-neighbors of a given non-intersection point $\mathbf{x}_i$, and add to the manifolds of $\mathbf{x}_i$ those which contain the same seed vertices as $\mathbf{x}_i$. The result is a manifold binary matrix

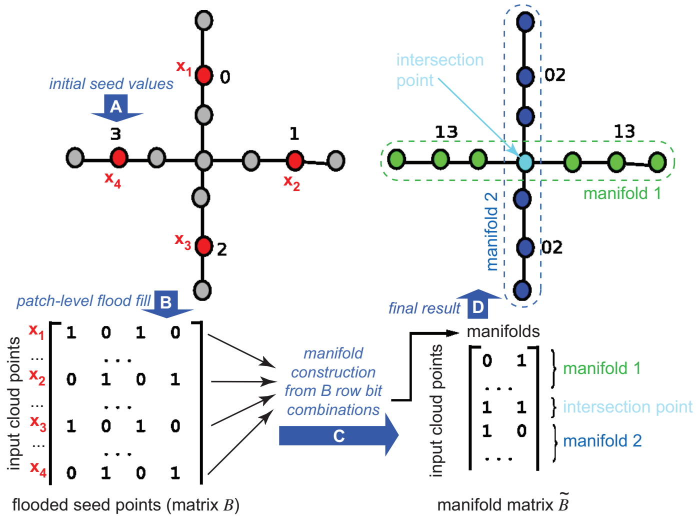


Figure 4.2: Manifold labeling, 2D sketch: (A) assignment of IDs 0..3 to seed points $\mathbf{x}_1..\mathbf{x}_4$; (B) patch-level flood fill yielding matrix B, (C) construction of manifolds, and (D) final manifolds (see Sec. 4.2.3.7). Overall, from the four seed points (marked in red), we extract two manifolds with labels 02 and 13, and one intersection point with label 0123.

$\tilde{B} = \{\tilde{b}_{ij}\}$, where rows $1 \leq i \leq |S|$ correspond to the original cloud points $\mathbf{x}_i \in S$ and columns to manifolds M_j, and $\tilde{b}_{ij} = 1$ indicates that $\mathbf{x}_i \in M_j$ (Figure 4.2, step C). A point can belong to none, one or more manifolds. Points which were not reached from any seed point, and are not seed points themselves, belong to no manifold, *i.e.* are labeled as noise. Points which belong to more manifolds lie on manifold intersection curves, and can be used to explicitly find such curves, see *e.g.* the light-blue point at the intersection in Figure 4.2 (step D), or the three yellow intersection curves in Figure 4.1 f.

Let us now see what happens when two sampled manifolds intersect at a small acute angle α. Figure 4.3 sketches such a situation. If the terminal 'branches' of the two manifolds are far enough from each other, phase A of our algorithm finds here four seed points ($\mathbf{x}_1..\mathbf{x}_4$, Figure 4.3 top). These are labeled with IDs 1..4. Next, the patch-level flood-fill phase propagates these seeds. If α is very small around the intersection point, each ID will 'leak' from its manifold-component to two other components past the intersection point, *e.g.* the top-left ID 1 will flood both the top-right and bottom-right branch. In phase C, we thus find *four* different manifold components, corresponding to the unique labels 123, 124, 134, and 234, as well as one intersection point with label 1234. If α is larger, we have the situation in Figure 4.2, where we find only two manifolds.

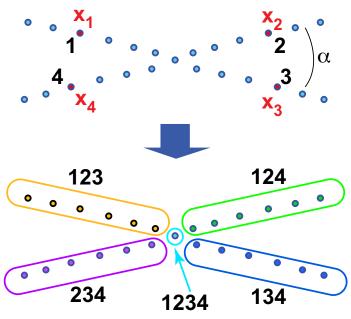


Figure 4.3: Labeling of manifolds intersecting at acute angles.

When α tends to zero, then a single seed point can be found, thus a single manifold is extracted.

4.2.3.8 *Manifold reconstruction*

We have now classified the input point cloud S into a set of (intersecting) point-sets M_k, each one representing a separate manifold. We can now use several existing methods to reconstruct the desired manifold surfaces $\mathscr{M}_k$. A good candidate is the ball pivoting method [12], which can efficiently and effectively reconstruct approximating triangle mesh surfaces from manifold point clouds. Other surface reconstruction methods from 3D point clouds (representing manifolds) can be used as well, as long as these methods can robustly handle non-uniform point densities and manifolds with boundaries, and are computationally attractive (fast). Figure 4.1 f shows the final result on our running example. The three surfaces are correctly extracted from the input cloud. Manifold intersections are marked in yellow. Non-manifold noise points, found by our extraction, are shown black.

Point classification into separate manifolds is essential to reconstruction quality: Feeding an entire, unclassified, cloud to ball pivoting would create ‘stitches’ between points on different manifolds and close to manifold intersections. Other existing point cloud reconstruction methods generate similar artifacts if no prior manifold classification is performed [2, 36, 38, 77], since such methods are not designed for noisy (self) intersecting surfaces (see further examples in Sec. 4.2.4.4).

4.2.4 *Applications*

Having presented our method for segmenting noisy 3D clouds into separate manifolds (and optionally reconstructing meshed versions of these manifolds), we now present several applications of this method.

We start by demonstrating the application of the method for general *manifold clustering*, such as manifolds found in manifold learning applications. We follow by showing how the method can be used for the denoising of point clouds. This can be used as a pre-processing step to apply to the input point cloud prior to skeletonization.

We follow by presenting the application of the method for *medial surface segmentation and reconstruction*. We demonstrate for several complex shapes how the method is capable of clustering the output skeletons from the skeletonization method in 3.

Finally, the last two presented applciation focus on scenarios not necessarily linked to Medial Axis: We demonstrate how the method can be used to perform general *surface segmentation*, along its surface high curvature areas. The last application to be demonstrated, is the usage of our method as a pipeline element for *surface reconstruction and denoising*, when in the presence of noisy input shapes.

4.2.4.1 *Manifold clustering*

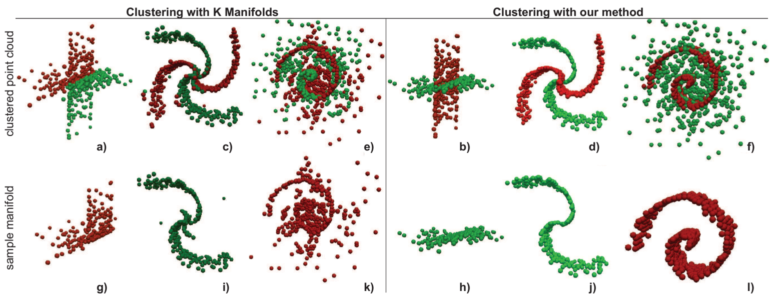


Figure 4.4: Manifold clustering: Our algorithm (right) *vs* K-manifolds (left). Top row: clustering results. Bottom row: One selected manifold from the clustering, displayed separately for illustration purposes. Input clouds are similar to examples in [161].

Manifold clustering of point clouds aims at labeling each point as belonging to a manifold. The labeled cloud can be further applied in manifold learning algorithms [161], shape segmentation, or as an input for surface reconstruction. We compare next our method with the well-known K-manifolds algorithm [161]. As input, we use non-uniformly sampled manifolds embedded in 3D, see Figure 4.4: Two intersecting curved surfaces (a,b), a four-branch spiral (c,d), and a densely-sampled spiral cut by a sparsely-sampled plane (e,f). Since K-manifolds is very slow (several minutes for a few thousand points), we limited our tests to small clouds only. As Figure 4.4 shows, our method finds the several manifolds present in the input robustly and eliminates the surrounding noise. In contrast, K-manifolds does handle well intersecting manifolds (Figure 4.4 a *vs* 4.4 b, 4.4 c *vs* 4.4 d), but separates manifolds from noise less well (Figure 4.4 k *vs* 4.4 l).

4.2.4.2 *Noise removal from (intersecting) manifold clouds*

Noise present in point clouds can occlude the underlying surfaces and limit the success of techniques such as surface reconstruction, clustering, and registration [23]. We evaluate the noise-removal ability of our method for several clouds representing various surfaces. For a fraction $\phi = 40\%$ of the points in each cloud, we add an outlier close to each cloud point $\mathbf{x}$ in a random direction and at a random distance from $\mathbf{x}$ ranging from 0 to 40% of the cloud's size. We next use our classification method (Sec. 4.2.3) to find noisy points, *i.e.*, which do not belong to any manifold. When removing these points, we should recover the initial noiseless cloud (ground truth).

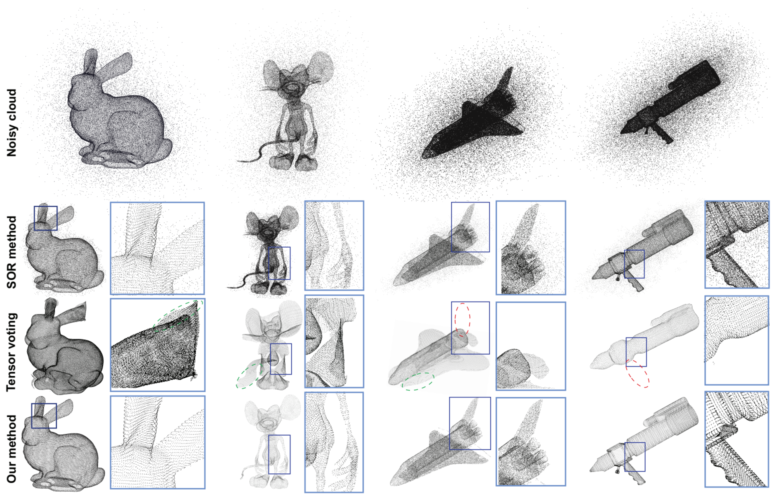


Figure 4.5: Noise removal from manifold clouds. Noisy shapes (top) are denoised with SOR [142], TV [105], and our method. Zoom-ins show details marked in blue. TV: red markers show incorrectly removed points; green markers show not removed noise.

We compared our method with the statistical outlier removal (SOR) of Rusu *et al.* [142] and tensor voting (TV) [105, 105]. Figure 4.5 shows the results. We remove considerably more outliers (noise points) than SOR, *i.e.* produce surfaces which are very close to the original. Our method works well even in spatially complex areas, see *e.g.* the gun handle detail. TV can also effectively remove most noise points. However, in this process, TV tends to create spurious surfaces that connect the original surface with nearby noise points (Figure 4.5, green markers: bunny years, mouse tail connected to body, and thickening of shuttle wings), or it removes details altogether (Figure 4.5, red markers: shuttle tailwing and gun handle). The balance between incorrectly kept noise and incorrectly removed details is strongly influenced by TV's volume sampling resolution [105, 105]: Higher resolutions remove less original non-noise points, but also keep too many noise points. Lower resolutions sub-sample the extracted tensor field. As a consequence, less manifold structures can be traced through noisy areas (so more noise is subsequently removed), but also more original detail points are classified as noise and thus removed.

Figure 4.6 shows the difference between added and removed points, as percentage of the original input point-count, for different values of ϕ ranging from 0 (no noise added) to 40% extra noise points added. Values are averaged for all models in Figure 4.5. We did not include TV in this comparison, since its behavior with respect to keeping incorrect noise points *vs* removing correct original points strongly depends on parameter settings (as explained earlier) and also on the actual input cloud. For $\phi = 0$, both our method and SOR do remove some points of the original model (5% and 15% respectively). Although, ideally, this value should be zero for a point cloud consisting only of manifolds, we remove less such points than SOR, which is desirable. As more noise is added, the ratio of added *vs* removed points stays stable with our method, *i.e.*, we remove only the added noise. In contrast, SOR removes increasingly more points than the added noise, *i.e.* removes more (up to almost 20%) of the original, noise-less, points.

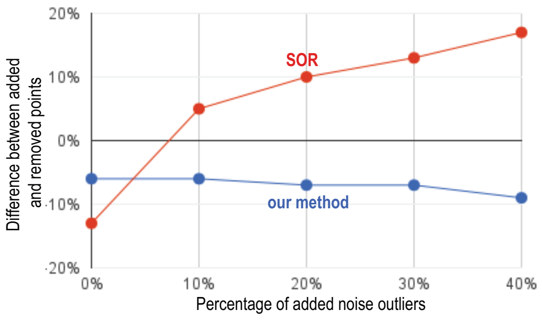


Figure 4.6: Percentage of removed points in excess of added noise points for different noise amounts, averaged for the models in Figure 4.5 Blue line: our method; red line: SOR method [142])

4.2.4.3 *Medial surface segmentation and reconstruction*

Medial surfaces, or skeletons, contain the loci of maximally inscribed balls within a given shape [122, 154]. Such surfaces consist of tens of manifolds of various sizes which meet along a set of Y-intersection curves [23, 34, 93]. Each manifold corresponds to a separate edge-set on the initial surface $\mathscr{S}$. Separating medial surfaces into their corresponding manifolds is useful for applications such as shape classification [23], shape matching [33], and segmentation [130, 132].

In Chapter 3, we have proposed our own method for extracting surface skeletons which, similar to Ma *et al.*, produces point cloud skeletal rep-

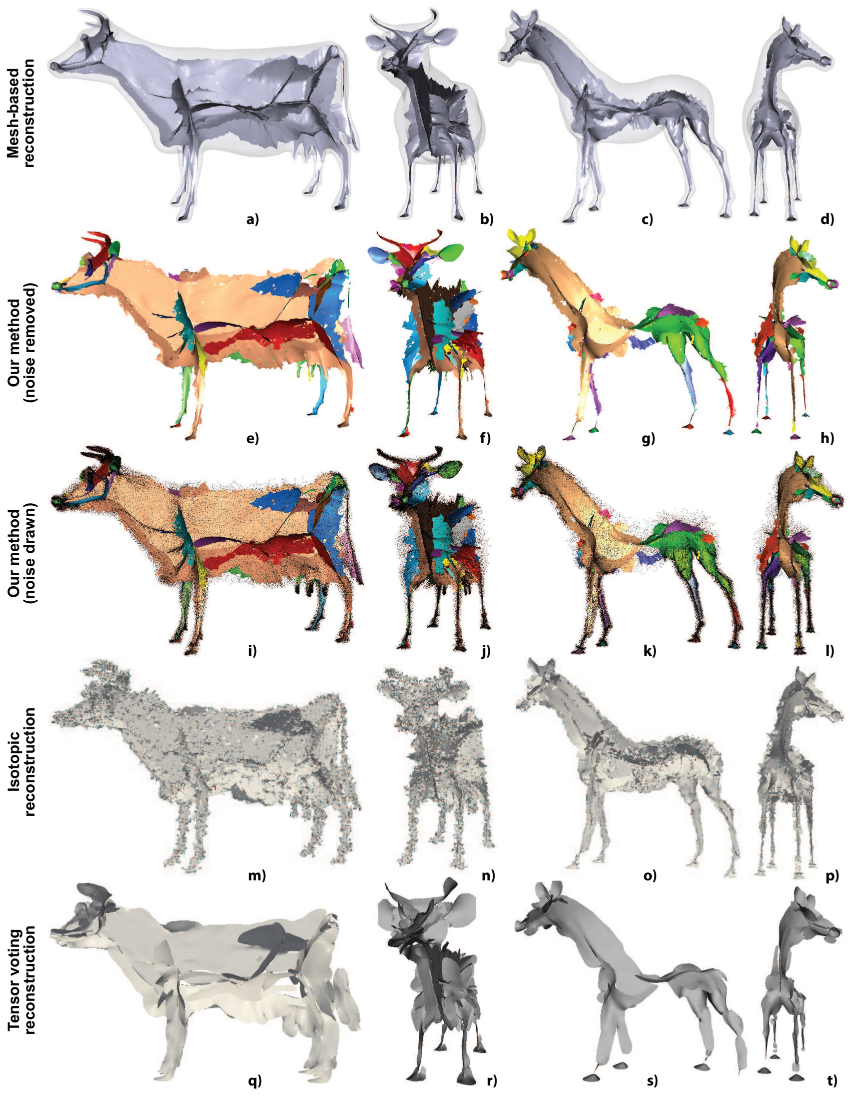


Figure 4.7: Medial surface reconstruction: (a-d) High-accuracy ground truth [181]. (e-h) Our method (noise points removed). (i-l) Our method, (detected noise rendered in black). (m-p) Isotopic reconstruction [39]. (q-t) Tensor voting reconstruction [105].

resentations. However, as stressed by Ma *et al.*, that medial point *clouds* are of limited use since, since for typical applications, more information is required, such as the skeleton surface of the individual manifolds. Creating such representations from medial clouds is *highly* challenging, since typical medial surfaces contain numerous (self-) intersections of very closely spaced, non-watertight, manifolds. Also, small-scale noise on the input shape creates spurious medial sheets, which show up as outlier points in the medial point cloud [101, 108, 154].

In Section 3.4.1, we presented a simple method for reconstructing meshed representations of the surface-skeleton manifolds, using Delaunay triangulations. However, as discussed there, this type of reconstruction produces artifacts such as small-scale holes or stitches for skeletal clouds that have closely-spaced or sparsely-sampled manifolds or contain noise (are not regularized). Moreover, this reconstruction method cannot separate the individual manifolds, but produces a single monolithic skeletal mesh.

Using our manifold extraction method presented in Section 4.2.3, we show next that we can simultaneously

A: eliminate outlier (noise) medial points;

B: obtain a mesh where each medial manifold is separately identified.

Figure 4.7 illustrates this on two medial clouds. Images (e-l) show the skeletal manifolds extracted by our method, colored differently for illustration purposes. As visible, our method captures well the complex medial topology and also robustly finds and eliminates the quite numerous noise points – compare images (i-l) where such points are drawn in black with images (e-h) where we eliminated these points. As mentioned earlier, such noise points are unavoidable when extracting medial axes of discrete objects. Hence, our method has the added value of acting as a filter that generates clean medial surfaces.

For comparison ground-truth, we next used the medial surface reconstruction method presented in Chapter 3.

Our method produces nearly identical medial surfaces with the method presented in Section 3.4.1 (Figure 4.7 a-d *vs* Figure 4.7 e-h). Differences consist in small-scale holes present in our manifolds, which do not exist in the reconstruction as in Section 3.4.1. Upon closer inspection, we see that these holes are due to limitations of the ball pivoting method that we use following our point classification, and not due to the fact that our method incorrectly classifies as noise (and thus removes) manifold points. Hence, we argue that our main goals **A** and **B** are reached. The mesh collapse method (Section 3.4.1) does not produce such small holes in the medial surface reconstruction. This is expected, since this method requires the input to be provided as an oriented *mesh* rather than an unoriented cloud. Moreover, the mesh collapse method needs the feature transform linking this mesh with its medial cloud, the relatively expensive and complicated importance computation for denoising [135] (which adversely affects goal **A**), and delivers a single unstructured medial surface (thus does not satisfy goal **B**). Although we require far less information (meshless unoriented medial clouds), *i.e.* use no knowledge that these points encode a medial surface, we can still extract separate and clean medial manifolds.

4.2.4.4 *Surface segmentation*

Given a point cloud S which samples a surface $\mathscr{S}$ in 3D, our method can segment $\mathscr{S}$ into smooth regions separated by sharp edges. In contrast to many surface segmentation techniques, we do not require $\mathscr{S}$ to be closed, non-intersecting, consisting of a single manifold, have normals, or be a mesh.

Figures 4.10 (d-l) show our cloud segmentation for various 3D point clouds. Zoom-ins show point cloud details to provide insight into the point samples' distribution in various areas. To better visualize the segmented point-sets, we show their reconstructions by ball pivoting. Image (d) shows the segmentation of a cloud from a 3D structured-light scan of a room by a Kinect device. The point sampling is relatively uniform, but noisy. The objects in the room are segmented correctly from each other and the room floor. The back wall (red) is only partially segmented since points in that range, far from the camera, are highly noisy, so they do not create a smooth manifold. In images (e-f), the various parts of the rabbit statue (body, plinth faces, inner ear surfaces, and heart detail) are correctly found. Since the input cloud describes a *hollow* shape, we also get manifolds for the inner surfaces – see the head, body and plinth cavities in the half-opaque rendering in Figure 4.10 f. Very thin, highly-curved, details like the screw connecting the head to the heart shape are marked as noise, as they have a higher local curvature than the imposed flatness d_{max} (Sec. 4.2.3.3). Images (i-j) show the segmentation of a cloud having several tens of intersecting shapes (sails, mast parts, and hull parts). In image (i), we used a low-resolution cloud (38K points). As seen in the zoom-in, fine details such as the masts have extremely few points in the longitudinal direction. Hence, few or no patches are found along the mast structure, just as the screw for the rabbit model, so no manifolds are found there (case **c** of local surface estimation, Sec. 4.2.3.2). Using the higher-resolution cloud (image (j), 74K points) also finds these detail manifolds along the masts. Image (k) shows a complex cloud (125K points) of a CAD model of a car engine with over 100 self-intersecting surfaces and highly non-uniform sampling. Image (l) shows the extracted manifolds. As for the ship, too thin and/or sparsely sampled details are classified as outliers and no manifolds are extracted there.

Figure 4.8 shows the results of the difference-of-normals (DoN) method [70], as implemented in the PCL library [70], applied to two of our point clouds. As the DoN method requires, we first specified the scale range $[r_1, r_2]$ within which features are sought. We did this, by trial and error, so as to obtain results as similar as possible to those produced by our method (Figure 4.10 b,d): Specifically, for the room scene, we want to detect segments corresponding to the various objects positioned on the floor; for the kitten model, we want to detect its medial manifolds. The DoN threshold $\mathbf{\Delta n}$ was set to various values in the range $[0.1, 0.5]$, similar to [70]. The resulting point clusters from DoN are rendered as

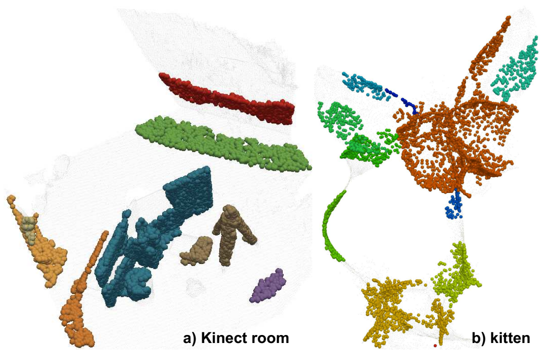


Figure 4.8: Difference-of-normals (DoN) segmentation [70] applied to two point clouds (compare with results in Figure 4.10 b,d).

colored balls, for display purposes. Input points not selected by DoN are rendered light gray. We see that DoN separates reasonably well the small-scale objects from the room floor. Also, a large part of the kitten's medial manifolds are found and separated from each other. However, several issues are visible too. First and foremost, the large manifolds present in the room (floor, walls) are not found – since these are not within the user-selected scale range $[r_1, r_2]$. If we carefully tune *both* r_1 and r_2, we can find parts of these manifolds. However, in the same time, we loose the ability to segment the smaller-scale objects. For the kitten medial cloud, segmenting is even more challenging, since its manifolds are less well separated from each other. In contrast, our method better separates both large and small manifolds, in both the room and kitten point clouds (Figure 4.10 b,d). This is explained by the fact that our method does not search for manifolds at a (user-specified) scale range, but rather tries to construct the largest possible manifolds allowed by the d_{min} and δ_{min} constraints.

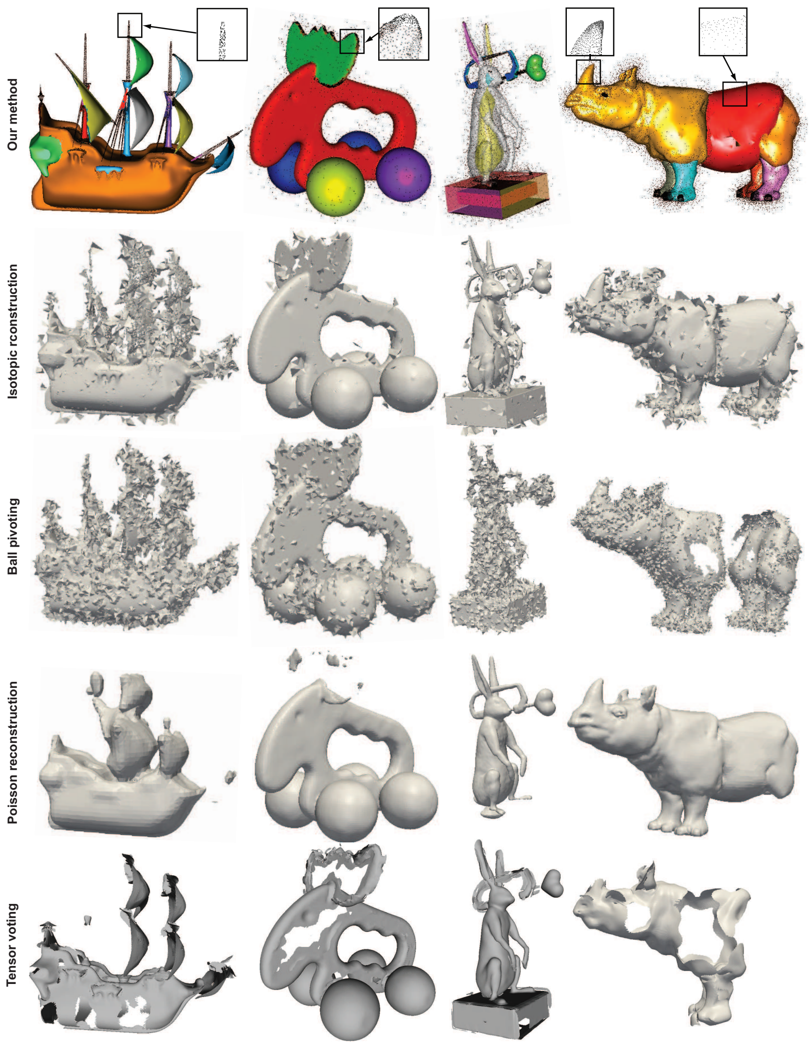


Figure 4.9: Shape reconstruction comparison for noisy point clouds with intersecting manifolds. From top to bottom, rows: our method, isotopic reconstruction [39], ball pivoting [12], Poisson reconstruction [77], and tensor voting [105]. Zoom-ins show point cloud details of selected model areas, for getting insight into the sampling distribution.

4.2.4.5 *Surface reconstruction and denoising*

Figure 4.9 shows a different use-case: We now use our method to reconstruct surfaces formed by several intersecting manifolds embedded into noise. We compare our results with several surface reconstruction methods, which are well known in the literature, easy to use, and their authors provided their implementations: isotopic reconstruction [39], ball pivoting [12], Poisson reconstruction [77], and tensor voting [105, 105].

Our method recovers best the various manifolds embedded into the noisy clouds. Isotopic reconstruction yields the next best results, as it can handle surfaces with boundaries, but still creates many small-scale spurious, non-manifold, surface fragments. Ball pivoting, as expected, cannot handle well dense noise and has problems for highly non-uniform clouds, like the rhino model whose rump has a much lower sampling density than the rest of the model (see zoom-ins in Figure 4.9). However, if ball pivoting is executed *after* our clustering method, most noise points are discarded, given the built-in denoising of our method. This drastically improves the effectiveness of ball pivoting (compare Figure 4.9, first and third rows). Also, since each segmented manifold is smooth, ball-pivoting can be used with a larger rolling-ball radius. This increases the ball pivoting robustness with respect to non-uniform sampling. Comparing the results of ball pivoting with isotopic reconstruction, we see that the latter suffers far less from noise and also does not produce undesired holes. Hence, isotopic reconstruction is a very good candidate to replace ball pivoting in our per-manifold reconstruction following the proposed denoising and classification. Poisson reconstruction, using an octree depth and solver divide value of 10, produces smooth surfaces, but cannot handle well (intersecting) manifolds with boundaries. Finally, tensor voting, used with a volume sampling resolution of 500^3 and point-neighborhood size $\sigma = 15$ (for details, see [105, 105]) yields smooth surfaces, but fails in thin areas where parallel surfaces are close to each other, like the thin muzzle and horns of the elk model.

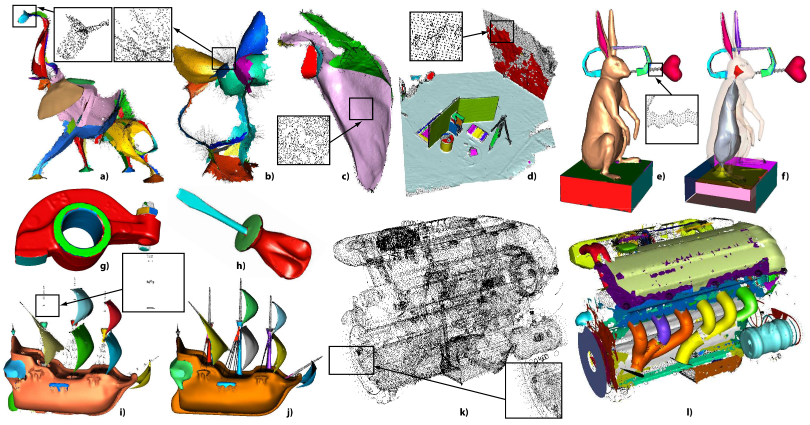


Figure 4.10: Shape segmentation examples. (a-c) Medial surfaces. (d) Structured light acquisition. (e,f) Shape with inner surfaces. (g,h) Mechanical shapes. (i,j,k,l) Mix of different structures with varying sampling density.

We also used the isotopic and tensor voting reconstruction methods to extract manifolds from medial point clouds (see Figure 4.7, two bottom rows). We notice here similar issues as in the examples in Figure 4.9. The isotopic method tends to create small-scale non-manifold noisy details. In contrast, tensor voting creates very smooth surfaces and handles manifold intersection regions very well. However, tensor voting has the tendency to extend the reconstructed manifolds far into the noisy point regions in an anisotropic way, *i.e.* retains noise points which allow a smooth continuation of the medial manifolds but in the same time eliminates noise points from the same areas if these are not aligned with the reconstructed manifolds.

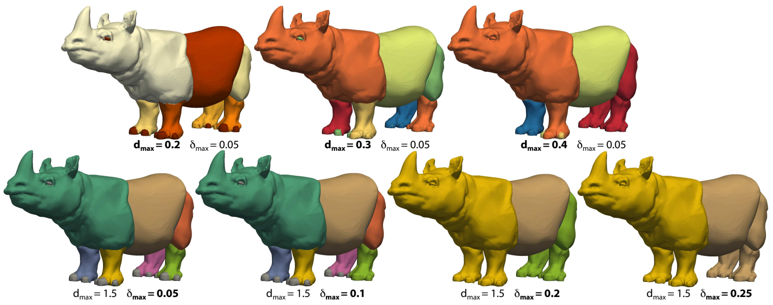

$d_{max} = 0.2$ $\delta_{max} = 0.05$
$d_{max} = 0.3$ $\delta_{max} = 0.05$
$d_{max} = 0.4$ $\delta_{max} = 0.05$
$d_{max} = 1.5$ $\delta_{max} = 0.05$
$d_{max} = 1.5$ $\delta_{max} = 0.1$
$d_{max} = 1.5$ $\delta_{max} = 0.2$
$d_{max} = 1.5$ $\delta_{max} = 0.25$

Figure 4.11: Effect of parameters d_{max} and δ_{max} on manifold extraction results.

4.3 DISCUSSION

We next discuss several relevant aspects of our method for extracting clean manifolds from noisy 3D point clouds.

4.3.0.6 *Generality*

We use a local feature detection for each point-cloud spatial neighborhood, followed by a global flood fill to find individual manifolds. Our approach has two main contributions. First, we extract manifold clouds from large amounts of embedding noise. Next, we segment manifold clouds from a single input cloud. This allows a *direct* reuse of existing surface reconstruction or shape analysis methods for point clouds on complex, multi-manifold, noisy clouds, even when such methods were designed to work only on smooth manifold clouds.

4.3.0.7 *Robustness*

Our method is robust to outlier noise, as shown by several examples (Figs. 4.5, 4.7). This feature is due to the hierarchical clustering of the local normal maps (Sec. 4.2.3.3) and the global clustering of local quasi-flat patches (Sec. 4.2.3.4). The first clustering separates locally relevant patches from noisy outliers, so it acts as a fine-grained noise filter. The second clustering ensures that only similar neighboring patches get grouped into smooth manifolds, so it acts as a coarse-grained noise filter. We pose no constraints on the cloud sampling density, as we detect local flatness using k nearest neighbors, rather than range-search with a user-prescribed radius, such as *e.g.* [70]. The examples in this chapter show that we can handle point clouds with a significant amount of sampling-density variation (see Figs. 4.9 and 4.10). However, we acknowledge our limits: Highly non-uniformly sampled clouds (*e.g.* Figs. 4.10 i,k) will be classified as noise.

4.3.0.8 *Parameters*

Our method has three parameters, as follows. The *neighborhood size* (k nearest neighbours) should be large enough to create triangles around a given point $\mathbf{x}$ to represent all possible surfaces crossing $\mathbf{x}$, but not too large so that meaningless surfaces are created. On all our models, regardless of sampling density, $k \in [7..10]$ provided good results. Setting k too large creates, along with the desired patches (that is, oriented along the sampled manifolds), several spurious patches at various random orientations. However, these are typically much fewer than the desired patches, so their effect gets filtered out by the Gauss map segmentation and subsequent patch-level flood fill steps. The *local flatness* $d_{max} \in [0, \pi/2]$ (Sec. 4.2.3.3), set in this chapter to 0.15, models the compromise between the extracted manifold smoothness, robustness to point-displacement noise, and manifold separation accuracy. Figure 4.11 (top row) illustrates this. Lower d_{max} yield more manifolds, since we allow normals to locally vary less within a manifold. This, for example, separates the toes and eyes detail of the rhino. Larger d_{max} values yield fewer, but potentially more curved, manifolds – the toes and eye get merged with the surrounding points. Increasing d_{max} even further merges the lower and upper leg fragments. For noisy datasets, larger d_{max} values also have the effect of classifying more points as noise, since less curvature is allowed within a manifold. Classifying points as noise prevents them from being treated by the ball pivoting reconstruction which, in turn, creates the small-scale holes mentioned in Sec. 4.2.4.3. However, we argue that such holes are not a classification problem, but a limitation of the postprocessing surface reconstruction method being used: If the aim is to remove such holes *and* still extract smooth manifolds, as controlled by d_{max}, then one should use a surface reconstruction tool that can handle non-uniformly sampled *manifold* clouds. If, however, the aim is to extract less smooth manifolds, then d_{max} should be increased. Another approach would be to set d_{max} adaptively as a function of the neighborhood's point distribution. However, how to do this and still guarantee the desired noise removal and manifold intersection detection is a topic of future research.

The *global flatness*, or patch similarity, $\delta_{max} \in [0, \pi/2]$ (Sec. 4.2.3.5) acts similarly to d_{max}, but at a coarser scale (see Figure 4.11, bottom row). Small δ_{max} values extract relatively flat manifolds, *i.e.* split larger manifolds along their crease lines. Large δ_{max} values extract less, and more curved, manifolds. For the rhino model example, increasing δ_{max} progressively merges all toe details with the legs, and further merges legs with the rump. For the figures shown in this chapter, we used $\delta_{max} \in [0.05, 0.1]$.

4.3.0.9 *Performance*

We implemented our method in C++ using *kd*-trees for nearest-neighbor searches [114]. We easily added CPU parallelization for local classification (Sec. 4.2.3.1) and patch graph building (Sec. 4.2.3.5), since points and patches are treated independently. Table 6 shows timings for a single-threaded *vs* a 4-core 2.8 GHz MacBook 4 GB RAM laptop. Our method scales well with the number of available cores. If desired, a GPU (*e.g.* CUDA) port could be easily done for further speed-ups. Our CPU implementation takes roughly half the time of the GPU surface reconstruction from noisy clouds in [152]. For the same datasets, the tensor voting surface extraction in [105] takes tens of minutes (at a volume resolution 500^3 and $\sigma = 15$). This is not too surprising, given that the tensor voting method is essentially based on several convolution passes of a large 3D tensor volume with a filter of kernel size σ. Decreasing the volume resolution speeds up tensor voting, but makes it unable to capture small-scale manifold details. Finally, the DoN method [70] has similar costs to our method. For example, the kitten skeleton and room models in Figure 4.8 took 4.9 and 15.96 seconds respectively (compare with our timings in Tab. 6).

Computing the patch dissimilarity δ with full-linkage (Eqn. 4.2) is $O(N^2)$ worst-case for a patch with N normals on average. N is a few tens for all tested models. The early termination criterion ($\delta < \delta_{max}$, Sec. 4.2.3.5) makes this cost much lower in practice, roughly $O(N)$ (see also below). We also tested an average-linkage patch dissimilarity, *i.e.*, using the distance between patch average normals, which is $O(N)$. For the models in this chapter, this gave a speed-up of about 20%, with a slight quality decrease – a few small-sized manifolds appear, since averages of two patch normal sets usually differ more than the closest normals of such sets. Given this, we chose to pay the small extra cost of full-linkage for increased manifold quality.

4.3.0.10 *Limitations*

If a neighborhood ν has no apparent 2D manifold structure, but a volumetric or one-dimensional point density, the local Gauss map has no

Model	**Points**	**Manifolds**	**Unclustered points**	**Time (sec.) (1-core CPU)**	**Time (sec.) (4-core CPU)**
Mouse (N)	54829	61	21288	27.33	7.52
Space Shuttle (N)	106580	36	35902	34.68	12.58
Glue gun (N)	128887	124	46379	26.32	7.31
Stanford bunny (N)	42322	4	5976	20.97	4.86
Intersecting planes	30486	3	0	4.89	1.38
Rabbit	124998	40	4012	7.89	2.15
Engine	124481	426	34235	52.33	14.29
Ship	38240	67	9420	13.56	3.72
Ship 2	74573	117	18232	28.02	7.63
Screwdriver	27152	5	673	7.43	1.96
Rockerarm	43213	4	1322	12.12	3.59
Kinect room	135402	19	23733	36.87	9.75
Elephant (MS)	173012	28	16232	56.32	15.51
Cow (MS)	252180	51	22309	89.74	24.04
Scapula (MS)	116930	4	5022	37.28	10.25
Pig (MS)	225281	76	15471	73.38	20.23
Horse (MS)	120442	42	10503	39.65	11.10
Kitten (MS)	43510	23	9503	12.38	3.53

Table 6: Timings for models shown in this chapter (N=model with added noise (see Figure 4.5); MS=medial surface (see Figure 4.7))

clearly separated peaks. In that case, the neighborhood is labeled as noise (see *e.g.* ship's thinnest masts and its ropes, and the rabbit screw in Figure 4.10). If this happens for most neighborhoods, *e.g.* in the case of a surface sampled overall by a thick point cloud, our manifold extraction will fail. This is expected, as our method is designed to find 2D manifolds only. In this respect, tensor voting is more general, as it can extract 2D surfaces, 1D curves, and junction points where several surfaces or curves intersect. However, for the manifold extraction case, both our method and tensor voting share the same limitation: Given a neighborhood where n manifolds intersect, if the sampling rate of these manifolds is too low with respect to n, neither method will be able to reliably separate these manifolds, and both methods will classify the neighborhood as noise. Examples of such configurations are visible for the car engine cloud segmentation (Figure 4.10 k. When the sampling rate is high enough, our method can reliably extract several intersecting manifolds, as illustrated by the medial examples in Figure 4.7.

4.3.0.11 *Comparison*

Many surface reconstruction methods exist, so the comparisons in Sec. 4.2.4 are clearly not exhaustive. Yet, as mentioned, we are not aware of any method that can extract (self) *intersecting* manifolds with *boundaries* from *noisy* clouds, except [23] and [105]. For instance, [12] and [39] can handle boundaries, but are challenged by noise and intersections; [77] can handle noise well, but not manifold boundaries and intersections. [36, 115, 152] can handle noise well, but cannot handle intersections.

In the class of methods that explicitly handle intersections, we are around 5 times faster than [23] (Tab. 6 *vs* Figure 27 in [23]). Note that [23] does not appear to include the medial surface computation cost. If one added that cost, our method is over 15 times faster. Also, [23] is considerably more complex to implement, as it requires a separate robust extraction of 3D medial surfaces from point clouds [93]. Compared to [105], we are over two orders of magnitude faster, and handle better manifolds which are close to each other.

Our method does not aim to replace all general-purpose surface *reconstruction* methods from point clouds. When certain properties hold, such as lack of noise, intersections, or boundaries, then other methods should be used, as outlined above. However, when all these properties lack, *i.e.* we have a noisy cloud describing intersecting manifolds with boundaries, our method can *segment* such clouds into subsets that have manifold properties. Such subsets can be next used for piecewise surface reconstruction (like in the examples shown in this chapter), but also for other, more general, tasks involving 3D point cloud processing, such as denoising, classification, segmentation, and matching.

4.4 CONCLUSIONS

We have presented a method to robustly segment unoriented point clouds into smooth manifolds. We handle clouds with complex combinations of an unknown number of potentially (self) intersecting, open or closed, manifolds embedded into noise. Using a clustering approach, we find the most probable local quasi-flat surface patches passing through each point, and merge these patches to classify the input points into manifolds or noise. Classified per-manifold points are reconstructed into a mesh using out-of-the-box cloud reconstruction methods. Compared to other methods, we allow input points to be classified as belonging to no manifold (*e.g.* noise), one manifold, or being on the intersection of several manifolds. This allows handling highly noisy clouds or point clouds having complex structures. The method allows for an easy parallelization, is simple to use, and has robust default parameter values. We demonstrate our method on several point clouds with use-cases in manifold extraction from embedding noise, regularized medial-surface reconstruction, and point-cloud surface segmentation, and point-cloud reconstruction.

In particular, our method can be effectively used for the separation of a 3D skeletal cloud, obtained by *e.g.* the skeletonization method presented in Chapter 3, into its distinct manifolds. This allows using classical surface reconstruction methods, *e.g.* ball pivoting or similar, to create meshed representations of *each* separate skeletal manifold. Additionally, the denoising properties of our point-cloud clustering method allow removing noisy skeletal points from the skeletal cloud, thereby producing clean surface skeletons. This is an alternative regularization method to the geodesic-based technique presented in Chapter 3, which has the important advantage of removing the constraint that the input shape has to provide connectivity information. Moreover, this type of regularization is simpler to implement than the geodesic-based technique. As such, our manifold segmentation method presented in this chapter has clear advantages for the skeletonization context.

One further application area in which our method could prove useful is CAD reverse engineering, *i.e.*, the recovery of separate 3D parts, or components, from a point cloud where topology information has been lost. A different application area is for other data than point clouds. By replacing the definitions of spatial neighborhood and orientation similarity, we could extract smooth manifold-like structures embedded in other spaces, *e.g.* find bundles or sheets of fibers in tractography datasets or segment multivariate spatial data. Such topics are left open to future research.

5 MEDIAL POINT CLOUD DENSITY ANALYSIS

It is the harmony of the diverse parts, their symmetry, their happy balance; in a word it is all that introduces order, all that gives unity, that permits us to see clearly and to comprehend at once both the ensemble and the details.

— Henri Poincaré

5.1 INTRODUCTION

In Chapter 3, we proposed a general-purpose method for extracting both 3D surface and 3D curve skeletons from complex 3D mesh shapes. Subsequent examples illustrated the method's speed, scalability, robustness to noise, regularization properties, and generality. We also discussed the bidirectional mapping that our surface skeletons establish *vs* the input shape surface, by using the feature transform (Section 3.5.1). In Chapter 4, we showed how we can segment such 3D surface skeletons (represented as point clouds) into separate manifolds, and construct meshed representations for these manifolds. As such, Chapter 4 shows how we can *refine* a raw 3D skeleton description by adding supplementary information to it, such as the classification of individual skeletal points into manifolds, and the addition of skeletal manifold connectivity.

However, the refinement of raw skeletal data presented in Chapter 4 is obviously not the final step one can take to enrich 3D skeletal abstractions. There is more information residing in such raw 3D skeletons,

This chapter is based on the following papers:

1. J. Kustra, A. Jalba, and A. Telea. Shape segmentation using medial point clouds with applications to dental cast analysis. In *Proc. VISAPP*, pages 151–159, 2014
2. J. Kustra, M. de Jager, A. Jalba, and A. Telea. Teeth shape modeling pipeline for oral healthcare appliances development. In *Proc. ICCE*. IEEE, 2014
3. J. Kustra, M. de Jager, A. Jalba, and A. Telea. A medial point cloud based algorithm for dental cast segmentation. In *Proc. ICCE*. IEEE, 2014

such as local point density, distribution of feature points, and distances of skeletal points to the input shape, to name just a few. Such information can be explicitly extracted and exploited to many ends, such as the enabling of additional shape-processing applications using skeletal descriptors. Now that we have seen how to scalably compute accurate 3D surface skeletons from complex shapes (and how to create manifold representations of such skeletons), it is time to focus on these higher-level steps of skeleton-based shape-processing applications.

In this context, this chapter takes a further step for using specific properties of 3D skeletal clouds to enable shape-processing applications. Specifically, we exploit the properties of the relationship linking the sampling *density* of an input shape $\partial\Omega$ with the sampling density of its 3D surface skeleton $S_{\partial\Omega}$. For point-sampled shapes (such as 3D meshes or point clouds), we have briefly outlined, in Chapter 3, that the two densities are related, due to the fact that our skeletonization method produces precisely one skeleton point per input-shape point. Globally, this implies that input-shape zones having higher sampling densities lead to surface-skeletons zones having higher sampling densities; a similar relationship exists between low sampling density zones on the input surface and its surface-skeleton.

However, as we shall next, the sampling density of surface skeletons does not only depend on the (local) sampling density of its input surface, but also on the local *curvature* of this input surface. Specifically, convex input-surface features result into a locally higher skeleton density, and concave surface regions result into a lower relative density of the skeleton respectively. These variations in sampling density can be used as tool for the analysis of specific surface features. This analysis can next lead to efficient algorithms for segmenting 3D surfaces, with concrete applications in a real-world shape analysis problem: the segmentation of teeth from an orthodontic 3D cast. This application also demonstrates the added-value of medial descriptors in real-world contexts where other shape processing algorithms have not been able to produce comparable results.

5.2 APPLICATION CONTEXT

Segmenting 3D surfaces into their natural components has many applications in shape analysis, computer vision, shape compression, and medical imaging. Segmentation requirements strongly depend on the target application, so many segmentation methods exist. At a global level, one can distinguish between patch-type and part-type segmentation methods [150]. *Patch-type* methods use local shape information such as surface curvature to produce quasi-flat segments separated by high-curvature creases or ridges. *Part-type* methods are more semantically-oriented, *i.e.*, try to find segments that a human user would intuitively see as distinct logical shape parts. Such segments are not always separated by high-curvature ridges.

For some shapes, neither patch-type nor patch-type methods do yield the desired result. Consider for example the dental cast model in Figure 5.3 a, where we want to find the separate teeth as individual segments, and also separate them from the gums. Part-based methods fail here, since the teeth are not clear *protrusions* from the shape's rump, as would be, *e.g.*, the limbs sticking out of an articulated body model. Patch-based methods also fail, since ridges separating teeth from each other and from the gums are quite shallow, so local-curvature detectors typically used by patch-based methods will fail to find segment borders. Figure 5.3 b illustrates this by showing the surface's curvature (concave=blue, convex=red).

The main motivation of the work presented in this chapter is the need to segment dental casts, which are tools used to create orthodontic treatment plans for dental patients. Advances in range imaging and 3D scanning allow capturing dental scans directly from a patient [6], and next to digitize such shapes into 3D surface meshes of the teeth-and-gum structure. Digital casts allow the automatic assessment of several orthodontic metrics, such as the arch length discrepancy, and new opportunities towards teeth alignment treatment planning and simulation. However, all such analyses require a prior segmentation of the scan into individual teeth and the gums. Typical dental scans do not exhibit sharp creases between individual teeth (due to scanning resolution limitations or actual teeth touching), nor between teeth and gums. Hence, existing patch-based segmentation cannot be directly used.

Dental cast segmentation has been widely explored recently due to the increasing availability of digital models. However, in this context, manual segmentation is prohibitively slow for current orthodontic practice. Several methods have been proposed instead, ranging from fully automated methods [83] to methods needing minimal user interaction [67, 84, 197]. The method in [83] avoids the 3D mesh processing complexity by first transforming the 3D data into a plan-view range image which is next segmented. Recently, a snake-based approach for teeth segmentation has been proposed [84]. Here, a snake is iteratively fit to the surface curvature until it reaches local minima. However, such methods have problems in low-curvature regions, *i.e.*, where creases separating teeth are shallow. Also, such methods assume a way to find the positions and number of teeth prior to segmentation, *e.g.* using the dental arch metric. In our proposal, we do not rely on such priors.

For the motivating use-case of segmenting dental casts, we present next a new method to compute patch-based segmentations of 3D shapes which do not exhibit strong creases between segments. Instead of using local information such as curvature, we take a *global* approach, based on the shape's surface skeleton. Key to our method is the observation that surface skeletons capture *all* input shape creases, regardless of their sharpness. To compute a high-resolution surface skeleton, we use the GPU-based skeletonization method presented in Chapter 3, which delivers point-cloud skeletons for models of hundreds of thousands of polygons in a few seconds. Next, we regularize the surface skeleton, to eliminate small manifolds that do not correspond to input shape segments which are large enough to be of interest. Next, rather than segmenting the surface (as virtually all patch-based method do), we segment its regularized surface skeleton, by a mean-shift approach. Finally, we project back the found skeletal segments onto the input surface, and use a nearest-neighbour approach to yield a segmentation that entirely covers the input shape (Figure 5.1).

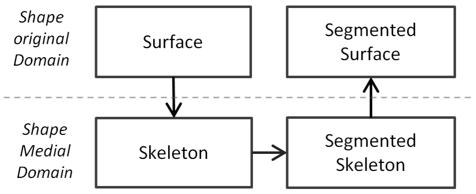


Figure 5.1: Algorithm workflow: A shape is transformed into the medial domain. Its skeleton is next segmented by a mean-shift approach. The segmentation is transferred back to the original shape.

The structure of this chapter is as follows. Section 5.3 details our skeleton-based segmentation method. Section 5.4 presents our segmentation results. Section 5.5 discusses our method and its results. Finally, Section 5.6 concludes the chapter.

5.3 METHOD

Our method first transforms the surface into its medial domain (see Figure 5.3). Specifically, we use the skeletonization method presented in Chapter 3 to compute a simplified, or regularized, skeleton S_τ of the input surface, where the simplification parameter τ takes care of removing noisy skeletal points created by very small surface perturbations. We next exploit skeletal point density properties to perform the segmentation in this domain. Finally, we project the medial segmentation back to the surface. We detail these steps next.

5.3.1 *Surface curvature* vs *skeleton density*

To further segment our skeleton cloud, we use the following observations linking the curvature of a shape $\partial\Omega$ and point density on it surface skeleton $S_{\partial\Omega}$. Consider a densely sampled 3D shape $\partial\Omega$. For a positive-curvature region of $\partial\Omega$ (bump, or convexity), the feature vectors $r\mathbf{n}$, where r is the surface-to-skeleton distance at a surface point $\mathbf{x}$ (equal

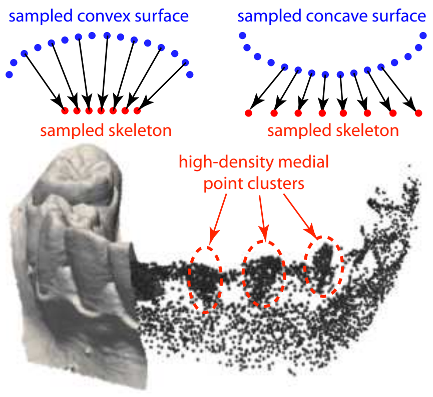


Figure 5.2: Relationship between local surface curvature and medial cloud density. **Top**: Concept sketched in 2D. **Bottom**: High-density point clusters are formed inside positive-curvature 3D surface areas (front teeth).

to the distance transform of $\partial\Omega$ sampled at the respective skeleton point) and $\mathbf{n}$, the normal at the same surface point $\mathbf{x}$, point inwards in a converging fashion. Hence, the density of the skeletal points corresponding to the regions around $\mathbf{x}$ will be higher than the surface sampling density.

Conversely, for a region on $\partial\Omega$ with negative curvature (a crease, or concave, region), the feature vector directions will point inwards in a diverging fashion, so the density of the skeletal points for this region will be lower than the surface sampling density. Figure 5.2 (top) illustrates this in 2D. Figure 5.2 (bottom) shows an actual example for a 3D skeletal cloud computed from a dental cast. We observe that skeletal parts *enclosed* in the front teeth present a high point density, since these teeth are indeed convex shape parts. Note that this relationship of the local sampling densities of the input surface $\partial\Omega$ and surface skeleton $S_{\partial\Omega}$ use the important assumption that we compute exactly one skeleton point per surface point. In this context, our observed sampling-density relationship can be seen as a consequence of the earlier observation of Siddiqi *et al.* that define the surface skeleton of a 3D shape as the locus of points inside the shape having a (high) negative divergence of the boundary's distance-transform gradient [155]. Indeed, areas where this divergence is highly negative indicate shape regions where *many* boundary points 'converge' to the same or near location(s) if they were advected in the distance-transform gradient field. Thus, these are regions where the sampling-density of the surface skeleton would be larger than the input surface sampling density. Of course, the ensuing question is what happens for skeletal regions corresponding to concave boundary fragments. In such regions, the divergence is (formally) still negative, but of much lower absolute value. These regions correspond to so-called skeleton *ligature* branches. As such, these regions are hard to find by simply thresholding the gradient's divergence. Siddiqi *et al.* observed this. To find such regions, they needed to add a homotopy-preserving thinning step to their skeletonization algorithm, thereby ensuring that divergence thresholding would not disconnect the resulting skeletons.

For our current aims, however, the above observations on the divergence of the distance-transform gradient are not critical. Indeed, we do not use such a divergence-based measure to *detect* the surface skeleton. Indeed, our surface skeletons are computed by using the ball shrinking algorithm presented in Chapter 3, which is not sensitive to such local properties. In contrast, our main observation here simply relates the local sampling densities of the two surfaces (input shape and its surface skeleton) in terms of the curvature properties of the input surface. As such, we can next analyze the sampling density of the shape's surface skeleton as an indicator of the convexity of the input shape – thereby providing us with a tool to segment the input shape.

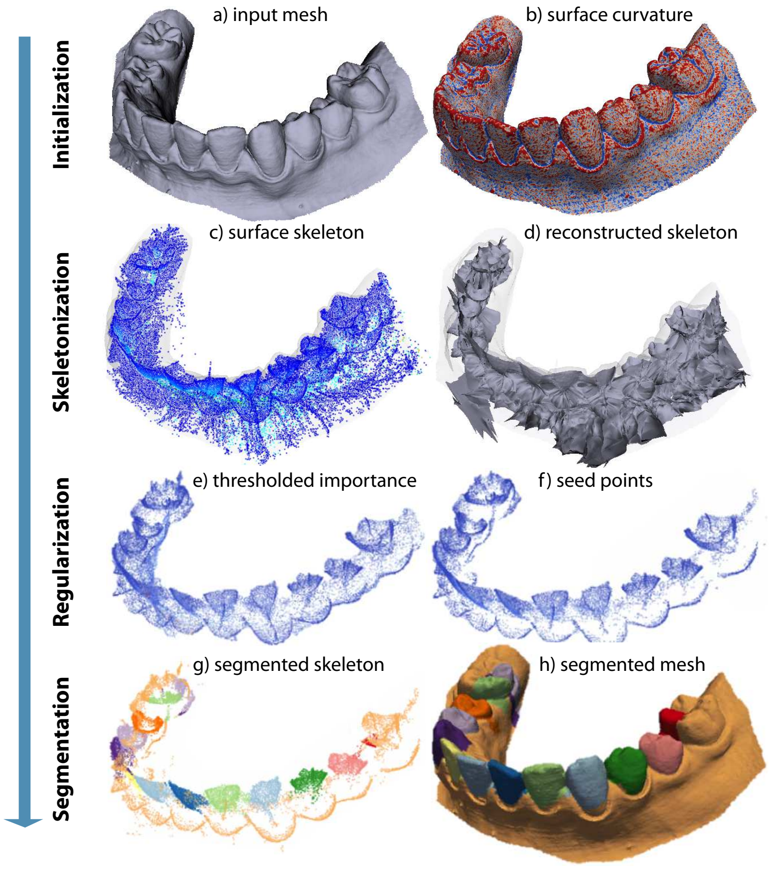


Figure 5.3: Algorithm steps: Input shape (top row). Surface skeletonization (second row). Medial cloud regularization (third row). Medial cloud segmentation and segments' transfer to the input surface (bottom row).

5.3.2 *Mean shift clustering*

We now show how to use the density-related observations in Sec. 5.3.1 to segment the skeleton cloud that captures the surface of an orthodontic scan.

Since the skeletal cloud exhibits strong density variations, it should be possible to segment it into point clusters representing the dense regions based on a method which exploits such density variations. An ideal such method is *mean shift clustering* [30], which we extend to our segmentation needs, as follows. We start by selecting a set of seed points $P \subset S_\tau$ from the simplified skeleton S_τ. The seed point selection is discussed separately in Sec. 5.3.3. Each seed point $\mathbf{x} \in P$ is assigned a unique 'segment id'. For each seed point $\mathbf{x} \in P$, we aim to find its so-called convergence

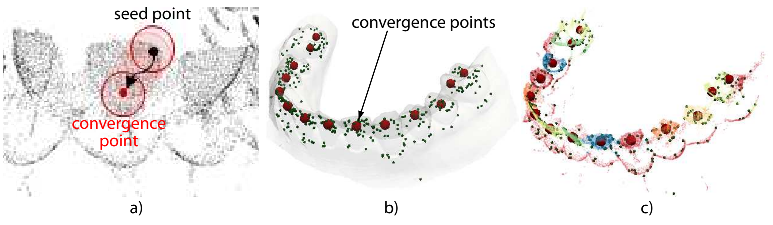


Figure 5.4: Mean shift clustering: (a) A seed point (black dot) is shifted to the centroid of its skeleton-cloud neighborhood density until convergence (red dot). (b) Final convergence points for dental cast. (c) Segment IDs assigned to skeleton points.

point $\mathbf{c}(\mathbf{x}) \in \mathbb{R}^3$. For this, we first find all neighbours $N_{\mathbf{x}}^{\varepsilon} \subset S_\tau$ of $\mathbf{x}$ within a small fixed radius ε and determine the centroid of $N_{\mathbf{x}}^{\varepsilon}$

$$\mathbf{m}(\mathbf{x}) = \frac{\sum\limits_{\mathbf{y}\in N_{\mathbf{x}}^{\varepsilon}} K(\|\mathbf{x}-\mathbf{y}\|)\mathbf{y}}{\sum\limits_{\mathbf{y}\in N_{\mathbf{x}}^{\varepsilon}} K(\|\mathbf{x}-\mathbf{y}\|)} \tag{5.1}$$

where K is a 1D Gaussian kernel

$$K(x) = \frac{1}{\sqrt{2\pi}} e^{\frac{x}{2\sigma^2}} \tag{5.2}$$

following the kernel density estimation idea in [30]. Here, ε is set to a small fraction (about 5%) of the model size. We next iteratively shift the seed points $\mathbf{x}$ to their centroids $\mathbf{m}(\mathbf{x})$ following Eqn. 5.1 (see also Figure 5.4 a) until these stabilize, *i.e.*, move at one iteration less than a small threshold $\lambda = \|\mathbf{m}(\mathbf{x}) - \mathbf{x}\|$, set in practice to 10^{-4}. Also, for each non-seed point (which is *not* shifted), we define a voting weight $v(\mathbf{y})$, initialized to zero at the beginning of the algorithm. At every mean-shift iteration, we add a value $K(\|\mathbf{y}-\mathbf{m}(\mathbf{x})\|)$ to $v(\mathbf{y})$ for each non-seed point $\mathbf{y} \in N_{\mathbf{x}}^{\varepsilon}$, and also add a pointer from $\mathbf{y}$ to $\mathbf{m}(\mathbf{x})$, to indicate that $\mathbf{y}$ was int he neighbourhood of $\mathbf{m}(\mathbf{x})$. When $\mathbf{m}(\mathbf{x})$ has converged, we search its neighbourhood for other existing convergence points $\mathbf{c}'$ than itself. If one exists, we merge the ids of $\mathbf{m}(\mathbf{x})$ and $\mathbf{c}'$. Otherwise, we create a new convergence point $\mathbf{c} = \mathbf{m}(\mathbf{x})$.

At the end of the mean shift, all seed points have thus converged to a set C of convergence points (see Figure 5.4 b). The ids of the points $\mathbf{c} \in C$ give us the final segments. Finally, to assign each non-seed point $\mathbf{y} \in S_\tau$ to a segment, is done by assigning to $\mathbf{y}$ the id of the convergence point that it is linked to and which has the highest amount of votes within the k last iterations (Figure 5.4 c). Different segmentation levels can be achieved by considering the voting of only the last k iterations of the mean shift process. This way, only the areas around the skeleton-cloud density *peaks* are considered. This is illustrated by the dental cast models, where the gum areas remain mostly unsegmented (Figure 5.5 a-e), for which we used a value $k = 20$. In contrast, for the other shapes in Figure 5.5 f-k, the full mean-shift path has been considered for the voting, leading to the full surface being segmented into patches.

5.3.3 *Seed point detection*

We find the initial seed points P used in mean shift clustering by using the specific geometry of our dental casts. We want at least one seed point in each *relevant* segment (that is, inside each high-density cluster such as the ones in Figure 5.2; we do not want seed points outside such segments. We allow more seed points in a segment, so that finding seed points is not parameter-critical. The definition of segment relevance is application-dependent: For our dental cast use-case, we only want the teeth segments and the gum, *i.e.*, we don't want to over-segment the gum. To achieve this, we use as seed points all skeleton points which (a) have a low distance $DT_{\partial\Omega}(\mathbf{s})$ to the original surface $\partial\Omega$ and (b) have a high curvature, computed as the angle between the feature vectors $\mathbf{f}_1 - \mathbf{s}$, $\mathbf{f}_2 - \mathbf{s}$.

5.3.4 *Segmentation transfer to surface*

In the last step, we transfer the skeleton segmentation to the input surface, as follows. For each point $\mathbf{s} \in S_\tau$, we copy the segment ID of $\mathbf{s}$ to its two feature points $\mathbf{f}_1$ and $\mathbf{f}_2$. However, this does not assign a segment ID to *all* points on $\partial\Omega$, since we segmented the simplified skeleton S_τ rather than the full skeleton S_Ω. For all points $\mathbf{p} \in \partial\Omega$ which are not assigned a segment ID, we search the closest surface point $\mathbf{p}' \in \partial\Omega$ which has an ID $ID(\mathbf{p}')$ assigned, and mark $\mathbf{p}$ with the same $ID(\mathbf{p}')$. This fills the gaps between segments on $\partial\Omega$ in distance order, yielding a full, non-overlapping, surface segmentation. In mathematical terms, this can be seen as a nearest-neighbor interpolation of the segment-ID signal on the surface $\partial\Omega$ (that is, an interpolation of the signal ID at any point $\mathbf{x} \in \partial\Omega$ based on shortest geodesic distance over $\partial\Omega$).

5.4 RESULTS

Figure 5.5 shows several results of our skeleton-based segmentation method. Surface segments are colored differently, for illustration. For images (a-e), which show dental cast scans, we see that the method separates very well the incisives, canines, molars and pre-molars, both from each other, and also from the gums. We also see several problems. The molars are over-segmented (Figure 5.5 a,b), and occasionally the gums are also over-segmented (Figure 5.5 c,e). The gums over-segmentation is *not* problematic for the considered orthodontic application, since users are only interested to analyze the teeth, and not the gums. The molars

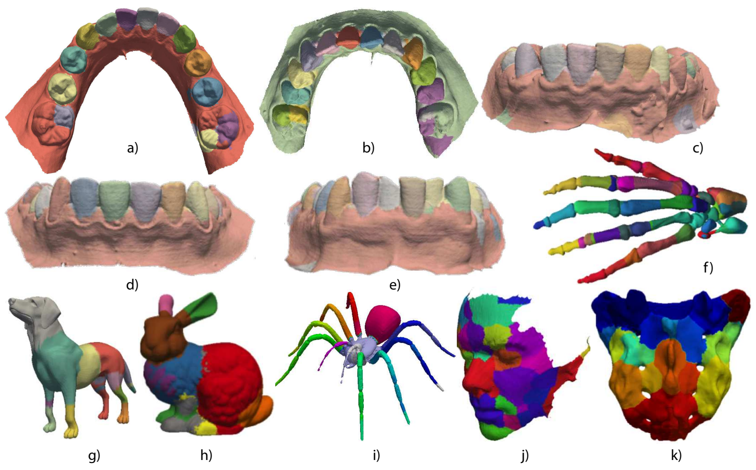


Figure 5.5: Segmentation results for different dental casts (a-e) and other shapes (f-k).

over-segmentation is be explained by the fact that they (a) have a more complex geometry than the other teeth (more internal creases) *and* also that, in our models, the input surface has less points here. Hence, the skeletal manifolds for to these detail-rich areas are too poorly sampled to fully capture their features.

We also segmented other shapes than dental casts, to get more insight in the method's behaviour (Figs. 5.5 f-k). Several observations can be made. For models having convex surface areas separated by well-delimited concave ridges, such as the hand or spider, we get the expected segmentation, just as for the dental casts. For the other models, segments are created *around* the most salient convex bumps of the shape. These segments meet along the model concavities, or creases – see *e.g.* the nose, eyes, and chin of the face model (Figure 5.5 j); bunny years, tail, head, rump, and front paw (Figure 5.5 h); and the convex bone components of the sacrum model (Figure 5.5 k).

5.5 DISCUSSION

Several aspects are relevant for our method.

5.5.0.1 *Simplicity and novelty*

A key asset of our method is its algorithmic simplicity. We use algorithms with proven accuracy, convergence, and complexity properties [30, 73, 135]. Our method is the first we are aware of to use surface skeletons computed on mesh models to segment surfaces. Its only competitor, using more expensive and lower-resolution voxel models,

Model	dental 1	dental 2	dental 3	dog	bunny	bone	hand	spider	face
# pts surf.	119594	127578	82887	18114	34834	41035	327323	29741	35437
# pts skel.	6136	24339	11214	16823	10706	8271	128839	8389	8450
# seed points	339	487	373	336	535	827	644	829	1041
CPU skel. (s)	51.3	48.53	22.87	34.89	10.96	10.26	151.68	5.85	12.8
GPU skel. (s)	11.7	11.0	5.78	3.24	3.6	1.82	31.2	0.95	2.11
CPU segm. (s)	1.95	1.92	1.3	43.07	2.71	40.12	152.02	167.53	11.35
CPU total (s)	53.25	50.45	24.17	77.96	13.67	50.38	303.7	173.38	24.15
GPU total (s)	13.65	12.92	7.08	46.31	6.31	41.94	183.22	168.48	13.46

Table 7: Segmentation timings.

is [130]. Apart from that, we note that our approach is also the first use of mean shift to cluster skeletons.

5.5.0.2 *Robustness*

The method is robust to different surface point-sampling densities, as the segmentation is performed based on the medial surface density properties. We tested our method on several dental cast models with the same parameter settings, and obtained identical results (*cf.* Figs. 5.3, 5.5).

5.5.0.3 *Applicability*

Our method is, by construction, geared towards the segmentation of *convex* surface patches separated by shallow *concave* creases. In this sense, we stress that the skeleton simplification (Eqn. 3.7) *only* eliminates skeleton points corresponding to small-scale surface bumps (convexities), but no skeleton point corresponding to a concavity (crease). This makes our method principially more robust than many other curvature-based segmentation methods. Also, our method can handle non-watertight surfaces (such as our teeth scans, which are not closed at the base, or the face model in Figure 5.5 j) with no problems.

5.5.0.4 *Performance*

We implemented our method in C++ using ANN [114] to find nearest neighbours and the GPU skeletonization presented in Chapter 3. The latter also provides the needed distance transform, feature points, and importance metric used for skeleton regularization. For 3D surface supersampling, we used Yams [51]. Tab. 7 shows timings on a Windows PC at 2.66 GHz with an Nvidia 690 GTX for several shapes. Skeletonization times include regularization (Eqn. 3.7); we show timings for our skeletonization method in Chapter 3 on both GPU and GPU. Segmentation timings include mean shift, segment ID assignment, and transfer on the original surface, all done on the CPU. As expected, GPU skeletonization is much faster than its CPU counterpart. The segmentation cost varies in function of the actual model. For the dental casts, this cost is quite low. Indeed, for these models, we use only the last k iterations (Sec. 5.3.2), whereas for the other models we use the full mean shift path. The segmentation (not optimized, unlike skeletonization) is dominated by the nearest neighbor searches. Such searches can be massively accelerated using GPUs, as shown in [73], so a GPU mean shift implementation should massively accelerate this step.

5.5.0.5 *Dental use-case*

For teeth segmentation, our method can segment all teeth whose medial surfaces converge to a point density maximum. Given the geometry of the incisives, canines, molars and pre-molars, we saw that these present the expected properties, so are robustly segmented. Molars have a slightly more complex geometry, including too shallow separation creases from gums. This creates some challenges (over-segmentation) when using the exact same mean-shift parameters as for the other teeth (see *e.g.* Figure 5.3 a,b). Possible ways to overcome this are super-sampling the input mesh, leading to a surface skeleton with better separated manifolds. However, we stress that, even with such limitations, our method is superior to existing alternatives in the orthodontic industry (see references in Sec. 5.2), as all such methods require a non-trivial amount of user input to produce their segmentations.

5.5.0.6 *Limitations*

As seen in Figure 5.5, our method cannot be directly applied to any 3D shape. The method is geared towards segmenting shapes whose skeletal manifolds exhibit clearly separated high-density branches, each branch corresponding to one surface segment. These are surfaces with convex patches separated by concave ridges. As such, one should not attempt to compare our current method with other general-purpose surface segmentation methods. Still, the main added value of our technique for segmenting general shapes (apart from the segmentation of dental casts) is the proof that point-cloud skeletons *can* be used for segmenting complex 3D surfaces. To our knowledge, this result has not been shown so far in current literature. As such, we hope that future work will show how this result can be extended to segmenting more general surfaces.

5.6 CONCLUSIONS

We have presented a method for segmenting compact convex patches of 3D polygonal surfaces from dental cast scans which are separated by shallow creases. For this, we use several properties of surface skeletons, in particular the sampling relationship between the skeleton and its original surface, and the correspondence relationship between the two surfaces given by the feature transform. To our knowledge, our method is the second existing technique able to use surface skeletons to segment 3D surfaces. In contrast to the first published technique in this area [130], we can directly handle meshed models without a costly voxelization step; we do not require the complex and sensitive detection of skeletal boundaries; and we can treat significantly more complex shapes than the earlier cited method in this class.

We foresee several possible extensions of our method towards a more general-purpose surface segmentation technique. Examples are the incorporation of surface differential properties, captured by the feature transform, in the analysis and segmentation of the surface skeleton, and application-adaptive skeleton simplification metrics that preserve or eliminate specific surface details for the purpose of more versatile segmentation.

Apart from the above application-specific context, this chapter contributes to the overall goal of this thesis by showing how specific skeletal properties (sampling density, in this case) can be used to support real-world applications (shape segmentation, in this case) in a generic manner. This, in turn, contributes to our overall claim that surface skeletons are useful and usable descriptors that support shape processing applications.

6 REFINED ABSTRACTIONS FOR MEDIAL POINT CLOUDS

It is the harmony of the diverse parts, their symmetry, their happy balance; in a word it is all that introduces order, all that gives unity, that permits us to see clearly and to comprehend at once both the ensemble and the details.

— Henri Poincaré

6.1 INTRODUCTION

In the previous chapters, we have presented several methods to extract and regularize medial point clouds, and use such skeletal point clouds to construct the first steps towards more advanced shape processing methods. Specifically, Chapter 4 presented a point cloud processing method capable of identifying manifold structures embedded in noisy point clouds. This enabled us to extract noise-free and separate manifolds from 3D surface skeleton clouds. Chapter 5 extended the idea of refining medial descriptors in the direction of analyzing their density properties to enable shape segmentation applications.

However, the above medial properties (manifold sheets and local density characteristics) are not the only (useful) properties one can extract, and reason about, for surface skeletons. Specifically, surface skeletons have a complex structure, consisting of several so-called medial *sheets*, or manifolds. Manifold points have several *types*, which characterize the kind of surface points they correspond to via the MAT [58]. *Boundaries* of these curves, mapped via the MAT to shape edges [134, 154], can be used for shape segmentation [130]. The curves where sheets meet, also called *Y-intersection* curves or medial scaffold, can be used for shape re-

This chapter is based on the following papers:

1. J. Kustra, A. Jalba, and A. Telea. Computing refined skeletal features from medial point clouds. *Pattern Recognition Letters (Submitted)*, 2014

construction and shape matching [23, 34, 93].The individual *sheets* correspond to separate shape parts, enabling additional shape simplification and segmentation applications.

Although the usage of such features, for both curve and surface skeletons has been demonstrated and applied on voxel-based representations [130, 135], their computation for point-cloud or meshed medial surfaces (such as introduced in Chapter 3) is far from trivial. Extracting features such as endpoints, branches, and junctions from curve skeletons is much easier than for surface skeletons, due to the inherent simpler structure of curve skeletons. Among other aspects, such as a higher computational simplicity and robustness, this makes curve skeletons to be more frequently used in applications, even though they encode less information than surface skeletons. As such, to become useful and effective for a wide range of real-world shape processing applications, recent fast-and-accurate 3D medial surface point-cloud extraction methods, such as our own method presented in Chapter 3, need enhancement in the sense of robustly classifying medial points, computing separate medial sheets, extracting sheet boundaries and Y-intersection curves, and mapping all these higher-level medial features robustly and efficiently to the shape surface.

In this chapter, we show how to efficiently and robustly construct all above features from medial surface point-clouds, by combining several input-shape and medial properties. This turns 3D medial clouds into shape representations which can be directly used for several shape analysis applications.

The structure of this chapter is as follows. Section 6.1.1 reviews the challenges associated with medial surfaces as described in Chapter 3 and motivates the remainder of the chapter. Section 6.2 presents our methods to compute these features from unstructured medial clouds. Section 6.3 shows the use of our medial features for part-based and patch-based shape segmentation and classification, to illustrate the applicability of our feature computation. Section 6.4 discusses our techniques. Section 6.5 concludes the chapter.

6.1.1 *Motivation*

The feature extraction or decomposition of medial surface features is an essential step towards its further usage in applications for shape analysis or classification. Examples of such features are the individual medial sheets [154, 165], Y-intersection curves or sheet boundaries, *i.e.*, the medial scaffold [93]. Applications of such features has been previously demonstrated on finely-sampled voxel skeletons [132], leading to the usage of medial surfaces to create compelling multiscale shape segmentations [130]. Doing all above for medial clouds is, however, far from trivial. In previous chapters, this analysis has been previously attempted by using local medial geometry properties (Chapters 4 and 5). As shown in

Chapter 4, using generic point-cloud segmentation methods for 3D surface skeletons is doable, but quite challenging, since medial surfaces consist of numerous intersecting manifolds with boundaries, which are hard to capture even with very dense point clouds. In addition, the method presented in Chapter 4 requires parameter settings related to local medial geometrical properties, such as the maximum local connectivity angle. These parameters are specific to the point cloud properties, which only become available once the skeleton has been computed. Therefore, the automation or abstraction of shape processing pipelines becomes a non-trivial task for a high variability of input shapes.

Overall, the above observations make us believe that additional higher-level information computed for surface skeletons, that show how these skeletons relate to their input shapes, is useful and necessary for shape analysis applications. Indeed, on the one hand, having such information, beyond simple local connectivity or point-sampling density data, should enable us to reason about shape properties on a higher level. On the other hand, such higher level information may remove the need of setting various parameters and thresholds for the analysis of lower-level skeletal information – the desired information is readily available, and we do not need to 'reverse engineer' it from the actual skeleton geometry or sampling density.

In this chapter, we present several methods that enrich a 3D medial point cloud with several refined abstractions, or features. Rather than using the local medial cloud geometric properties, the relationships between the medial points $\mathbf{x}_i$ feature vectors $\mathbf{f}_1, \mathbf{f}_2$ and their correspondence to surface Ω properties are used. Figure 6.1 overviews our proposal. Given a meshed surface, we first compute its 'raw' skeleton point cloud. Next, we classify medial points following [58]. This classification further enables us to extract separate medial point-sets corresponding to medial sheets (and subsequently detect the skeletal Y-network). Separately, all above features (medial point types and sheets) enable us to support several practical applications: simple and efficient medial point-cloud regularization, medial sheet reconstruction, and part-based and patch-based shape segmentation. Computing and using these refined skeletal features is discussed next.

6.2 COMPUTING REFINED MEDIAL FEATURES

To address the interpretation challenges outlined in Sec. 6.1.1, we next present several new methods for computing the above-mentioned higher-level features from skeletal point clouds. We start by showing how to robustly classify medial points following [58], find skeletal boundaries and Y-curves, and robustly regularize the medial surface (Sec. 6.2.1). We next use these features to robustly segment the medial surface into separate manifolds (Sec. 6.2.2). As input for all these operations, we only assume a surface skeleton represented by an unstructured and unoriented

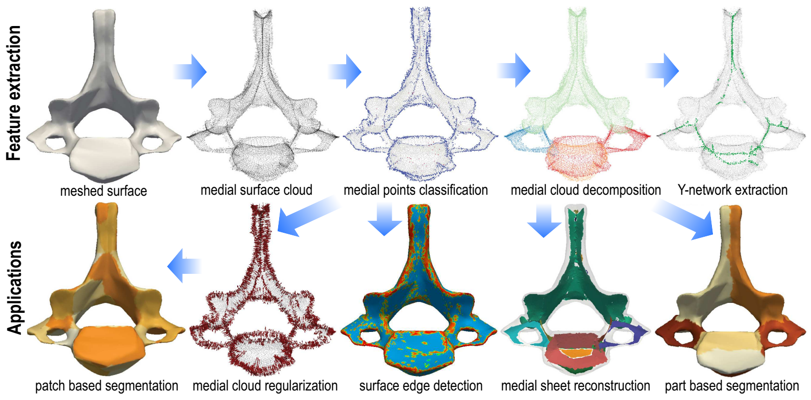


Figure 6.1: Refined skeletal features computed from medial point clouds (top row) and subsequently enabled applications (bottom row).

point-cloud having exactly one skeleton point per surface point and exactly two feature-points per skeleton point, as computed *e.g.* by our skeletonization method presented in Chapter 3.

6.2.1 *Medial points classification*

6.2.1.1 *Estimating the feature transform*

To classify unstructured medial clouds following [58], we first need to estimate the feature transform $FT(\mathbf{x} \in S_{\partial\Omega})$ (Eqn. 2.3). As explained earlier, the entire FT is not directly available in most skeletonization methods; in particular, our point-cloud methods (Chapter 3) only compute two feature points per skeleton point (Section 3.5.1). To find all feature points, we proceed as follows. Let $DT_{\mathbf{x}}$ be next a shorthand for the input shape's distance transform $DT_{\partial\Omega}(\mathbf{x})$. For each skeleton point $\mathbf{x} \in S_{\partial\Omega}$, we first find the closest points $FT_\tau(\mathbf{x}) \subset \partial\Omega$ in a radius $DT_{\mathbf{x}} + \tau$, where τ is defined as a fraction of $\varepsilon\rho_{\partial\Omega}(\mathbf{x})$, where $\rho_{\partial\Omega}(\mathbf{x})$ is the average point density on $\partial\Omega$ in a small neighborhood around $\{\mathbf{f}_1(\mathbf{x})\} \cup \{\mathbf{f}_2(\mathbf{x})\}$, and ε is a small constant set to 0.1. The slightly increased radius determines that the set $F_\tau(\mathbf{x})$ will conservatively contain *all* feature points of $\mathbf{x}$, *i.e.* $FT(\mathbf{x}) \subset FT_\tau(\mathbf{x})$. Setting τ to track the local sampling density of $\partial\Omega$ allows us to conservatively estimate F_τ for non-uniformly sampled meshes without introducing too many false-positives, *i.e.*, minimizing the set $FT_\tau \setminus FT$.

Given the finite tolerance τ and the discrete sampling of $\partial\Omega$, $FT_\tau(\mathbf{x})$ will also contain surface points which are slightly further from $\mathbf{x}$ than feature points; this is especially salient for points $\mathbf{x}$ of type A_3, that map to circular or spherical sectors on $\partial\Omega$ via the feature transform. However, as we shall see next, the conservative estimation of $FT(\mathbf{x})$ given by $FT_\tau(\mathbf{x})$ does not pose any problems to our medial attribute computation.

6.2.1.2 *Classification of medial points*

Since $FT_\tau(\mathbf{x})$ is essentially a dilated, or fuzzy, version of $FT(\mathbf{x})$, it consists of one or several point *clusters* centered around actual feature points. A cluster $C \subset FT_\tau(\mathbf{x})$ can be defined as

$$C = \{\mathbf{f} \in FT_\tau(\mathbf{x}) | \forall \mathbf{y} \in C, \mathbf{z} \in (FT_\tau(\mathbf{x}) \setminus C), \|\mathbf{f}-\mathbf{y}\| < \|\mathbf{f}-\mathbf{z}\|\} \quad (6.1)$$

i.e., all points which are closer to each other than to any point from another cluster.

We observed that the number of these clusters is a good indicator of the type of the medial point $\mathbf{x}$: For A_3 points, there is one such cluster, whose diameter is proportional to $DT_\mathbf{x}$; for A_1^2 points, we find two clusters; and for $A_1^{k,k\geq3}$ points, we find k clusters. To compute k, we cluster the point-set $FT_\tau(\mathbf{x})$ by a single-linkage hierarchical agglomerative method based on the Euclidean distance between the points. Next, we cut the resulting dendrogram, or cluster-tree, at a distance value equal to the average local sampling density $\rho_{\partial\Omega}$. This results in k clusters. The value of k gives us the point type, as explained above.

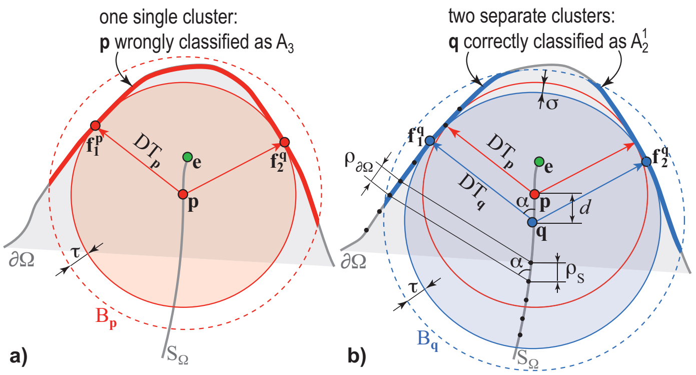


Figure 6.2: Skeleton point classification based on fuzzy FT_τ analysis. The figure is drawn for the case of 2D skeletons, for illustration simplicity.

Let us justify why k is a good point-type indicator. Figure 6.2 a shows an *incorrect* classification of medial point $\mathbf{p}$ which is on the skeleton S_Ω branch ended by point $\mathbf{e}$. Since the intersection of $\partial\Omega$ with a ball $B_\mathbf{p}$ of radius $DT_\mathbf{p} + \tau$ and center $\mathbf{p}$ (dotted red circle) yields a single cluster (thick red line), $\mathbf{p}$ is incorrectly marked as A_3 rather than as A_1^2. This is caused by (1) the value τ used to compute FT_τ being too large; (2) $\mathbf{p}$ being close to $\mathbf{e}$; and (3) the bump on $\partial\Omega$ corresponding to $\mathbf{e}$ being too shallow. Consider now the minimal distance d_{min} from $\mathbf{p}$ that we have to move on S_Ω away from $\mathbf{e}$ to find a point $\mathbf{q}$ which is *correctly* classified as A_1^2 (Figure 6.2 b). This happens when the intersection of the ball $B_\mathbf{q}$ of radius $DT_\mathbf{q} + \tau$ and center $\mathbf{q}$ (dotted blue circle) yields two disconnected clusters on $\partial\Omega$ (marked thick blue). To find d_{min}, note that the maximal 'inward shift' between the upper parts of $B_\mathbf{p}$ and $B_\mathbf{q}$ equals $\sigma = DT_\mathbf{p} - DT_\mathbf{q} + d_{min}$. To cause the disconnection of the compact cluster in Figure 6.2 a, σ must be larger than the maximal bump height on $\partial\Omega$ that fits in the sphere-shell of thickness τ, *i.e.*, $DT_\mathbf{p} - DT_\mathbf{q} + d_{min} > \tau$. Since $DT_\mathbf{q} - DT_\mathbf{p} = d_{min}\cos\alpha$, where α is the angle between a feature vector and the tangent plane to S_Ω, it follows that

$$d_{min} > \frac{\tau}{1-\cos\alpha}. \tag{6.2}$$

Separately, for a $\partial\Omega$ with local sampling density $\rho_{\partial\Omega}$, the corresponding skeleton sampling density is

$$\rho_S = \frac{\rho}{\sin\alpha}. \tag{6.3}$$

Incorrect classification (Figure 6.2 a) can only occur when $\rho_S < d_{min}$. Indeed, a correct classification should mark only a one sampling-point-thin 'band' of skeleton points as A_3 (surface skeleton boundary). If ρ_S is smaller than the minimal ball-shift d_{min} required to change point type from A_3 to A_1^2, this band gets thicker, which blurs our classification. Substituting our value of $\tau = \varepsilon\rho_{\partial\Omega}$ (Sec. 6.2.1.1) in Eqn. 6.2, it follows that this problem can only appear when $\varepsilon \geq \frac{1-\cos\alpha}{\sin\alpha}$. For our chosen value of $\varepsilon = 0.1$, this implies $\alpha \lesssim 11.4°$. In other words, for any medial sheets except those corresponding to highly obtuse angles on $\partial\Omega$, our method finds skeleton boundaries (A_3 points) which are precisely one sample-point thick.

Following the above, if $k = 1$ or $k = 2$, we can confidently say that to have found an A_3, respectively A_1^2, skeleton point. As k increases, the spatial separation of the clusters decreases too, so k does not reflect accurately the skeleton point type. We have empirically verified that the cluster count k accurately finds A_1^3 up to A_1^4 points for densely-sampled surfaces $\partial\Omega$. A more robust way to find such Y-curve points, that is far less sensitive on the sampling density of $\partial\Omega$, is described further in Sec. 6.2.2.2, based on the segmentation of S_Ω into individual medial sheets.

Related to our work, [134] found A_3 points by computing the set difference between the full medial surface S and a simplified version S_τ of S, where τ is a small fixed value and simplification uses the MGF metric (Sec. 2.3.4). Compared to our approach, their method does not find $A_1^{k,k>1}$ points, and does not give an analysis of how to set parameter values.

6.2.1.3 *Skeleton regularization using A_3 edge filtering*

As outlined in Sec. 2.3.4, the MGF metric [37, 135] provides very good regularization properties such as separating spurious skeleton points

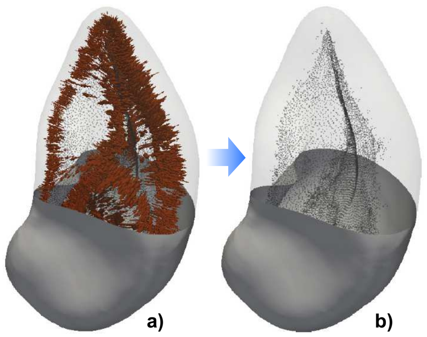


Figure 6.3: Skeleton regularization. (a) Rounded spleen shape with feature vectors shown for A_3 points. (b) Skeleton regularized by filtering A_3 points.

from important ones while maintaining skeleton connectivity. The MGF importance $\rho(\mathbf{x})$ of a medial point $\mathbf{x}$ equals the length of the longest shortest-geodesic on $\partial\Omega$ between *any* two feature points of $FT(\mathbf{x})$. Hence, the MGF requires an accurate computation of the feature transform FT (Eqn. 2.3). As discussed in Sec. 6.2.1.1, we compute a conservative FT_τ which may contain tens of feature points for A_3-type points. Computing shortest-geodesics between all such point-pairs is very expensive. Given this cost, (Chapter 3) and [135] compute the MGF using only two feature points per skeleton point, *i.e.* implicitly consider all medial points to be of type A_1^2. This has two problems. First, the importance ρ for A_3 points will be typically underestimated, since one has no guarantee of finding the *longest* shortest-geodesic connecting any two feature points. This, in turn, creates a relatively jagged appearance of the simplified skeleton. Secondly, computing the MGF is expensive for large models, even when using only two feature points per medial point and highly optimized GPU implementations (Chapter 3).

We propose here an alternative way to regularize medial surfaces by simply filtering A_3 points found by our classification. Figure 6.3 shows this for a shape having highly rounded edges, *i.e.* whose A_3 points have many feature points. This is the kind of shape where the aforementioned problem of the MGF metric occurs. Figure 6.3 a shows the medial cloud with feature vectors (in red) for the A_3 points. Figure 6.3 b shows our regularized skeleton, with all noisy points being removed. Since A_3 points appear only on the medial boundary by definition, our regularization does not create gaps or disconnect the medial surface. Since our method requires only a simple clustering of feature points based on their Euclidean distance, it is considerably faster than the MGF metric (see Sec. 6.4 for details). However, in contrast to the MGF, our method cannot deliver a *multiscale* of progressively simplified skeletons; we can only remove the finest scale of noisy boundary points. As such, our regularization is useful when one needs a clean and detail-rich surface skeleton for further processing, rather than a multiscale skeleton representation.

6.2.2 *Surface skeleton decomposition*

Besides classifying skeleton points, higher level features can be computed. One such feature is the decomposition of the medial surface into separate sheets, used in shape analysis and segmentation tasks [93, 130]. While such decompositions can be computed relatively easy for voxel skeletons [132], this is challenging for medial clouds, especially when these contain a large number of complex sheets (Chapter 4).

We address this task by clustering the medial cloud based on a novel definition of medial sheets that uses the medial cloud properties (Sec. 6.2.2.1). Next, we use this decomposition to robustly find Y-intersection curves where several such sheets meet (Sec. 6.2.2.2). Finally, we use the feature transform to construct compact (meshed) sheet representations (Sec. 6.2.2.3).

6.2.2.1 *Medial sheet computation*

We first define a distance for a pair of two medial points **x** and **y** as

$$\delta(\mathbf{x},\mathbf{y}) = \sum_{\mathbf{a}\in F(\mathbf{x})} \min_{\mathbf{b}\in F(\mathbf{y})} MGF(\mathbf{a},\mathbf{b}), \tag{6.4}$$

where $MGF(\mathbf{a},\mathbf{b})$ is the medial geodesic function, *i.e.* the length of the shortest geodesic on $\partial\Omega$ between feature points **a** and **b** [135]. Next, we define a medial sheet γ as all medial points having a distance δ lower than a threshold τ

$$\gamma = \{\mathbf{x} \in S_\Omega \mid \exists \mathbf{y} \in \gamma, \delta(\mathbf{x},\mathbf{y}) < \tau\}. \tag{6.5}$$

The value of τ is set to a relative *low* geodesic distance in the distance range between two points on $\partial\Omega$. In practice, this resulted in setting it in the range $[3\times\overline{\rho_{\partial\Omega}}, 10\times\overline{\rho_{\partial\Omega}}]$, where $\overline{\rho_{\partial\Omega}}$ is the average surface $\partial\Omega$ sampling density. The rationale behind Eqn. 6.5 is that medial points **x** and **y** which are close *and* belong to the same sheet have small distances (along $\partial\Omega$) between their corresponding feature points. This statement is supported as follows: Since medial sheets are locally smooth and have a low curvature [122], their feature vectors vary smoothly and slowly locally; in turn, this implies that the corresponding feature points vary slowly and smoothly across $\partial\Omega$. Figure 6.4 illustrates this for a 2D shape (for simplicity): Medial points **x** and **y** are on the same sheet, and have small MGF distances between their feature points, thus a small $\delta(\mathbf{x},\mathbf{y})$. In contrast, medial points **w** and **z**, which belong to different sheets, have at least two feature vectors pointing in different directions, thus a large $\delta(\mathbf{x},\mathbf{y})$.

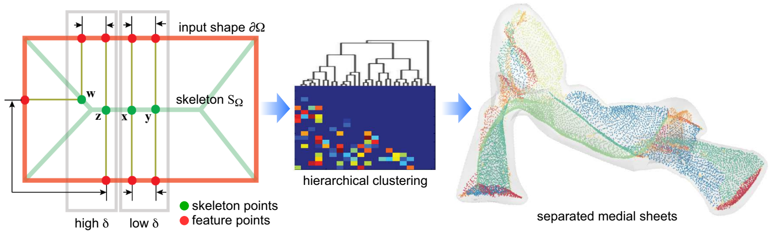


Figure 6.4: Medial sheet computation: a) Distance function δ, illustrated in 2D. b) Sparse distance matrix, used as an input for hierarchical point clustering. c) Medial sheets found for a palatine bone shape (see Sec. 6.2.2.1).

Equations 6.4 and 6.5 define medial sheets without using any medial connectivity information. We can thus use them to segment a medial cloud into its sheets, as follows. First, we define a distance matrix M encoding the distances $\delta(\mathbf{x},\mathbf{y})$ between all medial point pairs. For efficiency, we only compute matrix entries that correspond to δ values below our chosen threshold τ, since the sheet definition (Eqn. 6.5) only requires to know when $\delta < \tau$. To do this, we first notice that medial points $\mathbf{x}$ and $\mathbf{y}$ which are far apart will also have large values of $\delta(\mathbf{x},\mathbf{y})$. As such, for a point $\mathbf{x}$, we only consider in M those entries that correspond to its K nearest neighbours $\mathbf{y}$. Secondly, when computing $\delta(\mathbf{x},\mathbf{y})$, if the length of the geodesic traced on $\partial\Omega$ from $\mathbf{x}$ to $\mathbf{y}$ exceeds τ, we stop tracing it and skip the respective matrix entry. Overall, this turns the computation and storage of M from a quadratic process in the number of medial points into a linear process. Finally, we use M as input for a single-linkage hierarchical clustering [71], which outputs a partition of S_Ω into a set of medial sheets γ_i, so that $\gamma_i \cap \gamma_{j\neq i} = \varnothing$ and $\cup_i \gamma_i = S_\Omega$. Figure 6.4 c illustrates the separated sheets of the medial surface of a palatine bone shape. Same-sheet points are marked by the same color. Although the medial cloud is quite complex, its sheets are cleanly separated. Such sheets can be processed to create compact representations thereof, as discussed next in Sec. 6.2.2.3.

Figure 6.5 compares our method for sheet extraction from a medial cloud with two other methods. Image (a) shows the method of [132], which works in brief as follows: Given a (voxel) medial surface S, its Y-network voxels S_Y are found based on the cardinality of the feature transform for A_1^3 points (Sec. 2.2.2). Next, separate medial sheets are found as being the connected components of the voxel set $S \setminus S_Y$. While this method gives good results, it is quite sensitive to the voxel sampling of the input shape. For instance, the cog wheel detail in Figure 6.5 a (128^3 voxels) shows two separate components c_1 (red) and c_2 (purple), which actually are part of the same sheet. These are wrongly separated since (1) the sampling resolution disconnects the detected medial sheet halfway and (2) sheet detection is based on connected component finding.

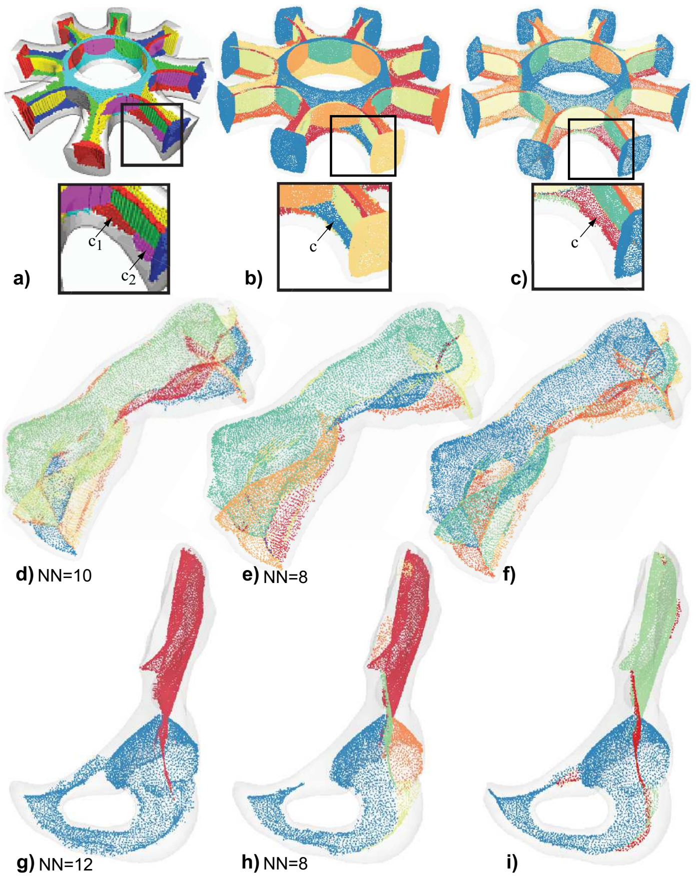


Figure 6.5: Medial sheet extraction: (a) Method of [132]; (b,d,e,g,h) Method of Chapter 4; (c,f,i) our method. For the method in Chapter 4, one of its parameters, the number of nearest neighbours *NN*, is indicated.

Image (b) shows the method presented in Chapter 4. Image (c) shows our method. As visible, both local (Chapter 4) and our presented method correctly detect a single sheet c instead of the two separate fragments c_1 and c_2. Images (d-i) further compare our method with our method presented earlier in Chapter 4 for two shapes and two different values of the nearest-neighbour count *NN* (one of the parameters of the method in Chapter 4). Two observations can be made here. First, we see how the results depend quite strongly on the choice for *NN*. In contrast, the proposed method does not require tuning such a parameter. Secondly, and more importantly, the point-similarity used in Chapter 4 is essentially purely *local*, as it involves only inter-point distances and local flatness of the sheets. In contrast, the proposed method uses a distance function (Eqn. 6.4) which captures *global* shape properties, due to the underlying

MGF function. This makes the sheet computation far less sensitive to shape variations.

6.2.2.2 *Y-intersection curve extraction*

Once the medial sheets are found, the Y-intersection curves can be found as those points $\mathbf{x} \in S_\Omega$ that belong to at least two different sheets. However, performing this test directly on the medial sheet-set is not possible, since our sheets are disjoint, *i.e.* $\gamma_i \cap \gamma_{j \neq i} = \varnothing$. Hence, we find Y-curve points as those skeleton points $\mathbf{x}$ which have at least one k-nearest neighbour belonging to a different sheet than the one containing $\mathbf{x}$. Tuning k allows controlling the thickness of the Y network being computed. Figure 6.6 shows three examples of Y networks, computed for $k = 3$.

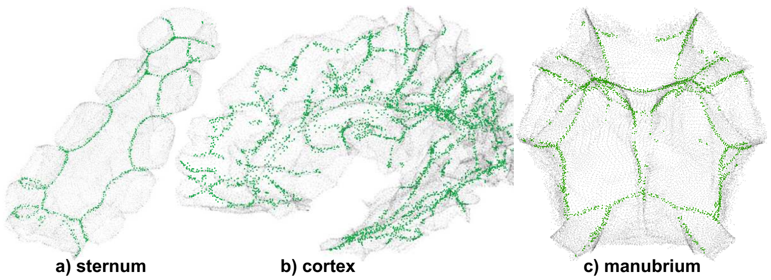


Figure 6.6: Y-network extraction with Y-curve points colored green.

6.2.2.3 *Computing compact medial sheets*

In Sec. 6.2.2.1, we computed medial sheets as unstructured point clouds. Many shape processing operations require the input to be a triangle mesh. We show next how such meshes can be created based on an analysis of the feature vectors $\mathbf{v}_1(\mathbf{x})$ and $\mathbf{v}_2(\mathbf{x})$ of each skeleton point $\mathbf{x}$ (Chapter 3). The key idea is to use the feature vectors to back-project the connectivity information captured by the $\partial\Omega$ mesh onto each sheet γ. The method has two steps, as follows.

Feature vector alignment: The projection

$$P(\gamma) = P_1(\gamma) \cup P_2(\gamma) = \{\mathbf{x} \in \partial\Omega \mid \exists \mathbf{y} \in \gamma, \mathbf{x} \in \{\mathbf{f}_1(\mathbf{y}), \mathbf{f}_2(\mathbf{y})\}\} \quad (6.6)$$

of a sheet γ consists of two triangulated areas $P_1(\gamma)$ and $P_2(\gamma)$ of $\partial\Omega$, one for each side of γ. If we can isolate any of these two areas, we can next simply transfer its connectivity information onto γ to obtain our desired sheet mesh. For this, we reorder, or align, the feature vectors $\mathbf{v}_1(\mathbf{x})$ and $\mathbf{v}_2(\mathbf{x})$ of all sheet points so that all $\mathbf{f}_1$ are included in $P_1(\gamma)$, and all $\mathbf{f}_2$ are included in $P_2(\gamma)$, as follows. First, we select a reference point $\mathbf{x}_{ref} \in \gamma$

and mark it as visited. We next visit all other points $\mathbf{x} \in \gamma$ in order of increasing distance to $\mathbf{x}_{ref}$ and redefine their feature points as

$$\mathbf{f}_i = \underset{\mathbf{f} \in \{\mathbf{f}_1, \mathbf{f}_2\}}{\arg\min} \, MGF(\mathbf{f}, \mathbf{f}_i^{vis}), \quad i \in \{1,2\}, \tag{6.7}$$

where $\mathbf{f}_i^{vis}$ is the closest visited (aligned) feature point to $\mathbf{f}_i$, and mark $\mathbf{f}$ as visited. When all points of γ are visited, all feature vectors $\mathbf{v}_1$ will be on the same side of γ as $\mathbf{v}_1(\mathbf{x}_{ref})$, while all $\mathbf{v}_2$ will be on the other side. We can next find the projection of side $i \in \{1,2\}$ of γ as $P_i(\gamma) = \{\mathbf{x} \in \partial\Omega \mid \exists \mathbf{y} \in \gamma, \mathbf{x} = \mathbf{f}_i(\mathbf{y})\}$.

Connectivity projection: We finally construct a meshed version of γ by simply copying all triangle information from $P_i(\gamma)$ to γ, with i being either 1 or 2 (both sides are equally good). That is, for any triangle $t = \{\mathbf{x}^i\}_{1 \le i \le 3}$ in *e.g.* $P_1(\gamma)$, we construct a triangle $t_\gamma = \{\mathbf{y}^i\}_{1 \le i \le 3}$ where $\mathbf{x}^i = \mathbf{f}_1^i$. Figure 6.7 illustrates the resulting meshed sheets for the surface skeletons of several complex anatomical shapes from the open database in [110], where neighbour sheets have different colors for illustration purposes. Given these meshed sheets, we can now use any polygon-based geometric algorithm to analyse or process them further, *e.g.*, to estimate curvature, areas, elongation, or compute shortest paths or distance fields.

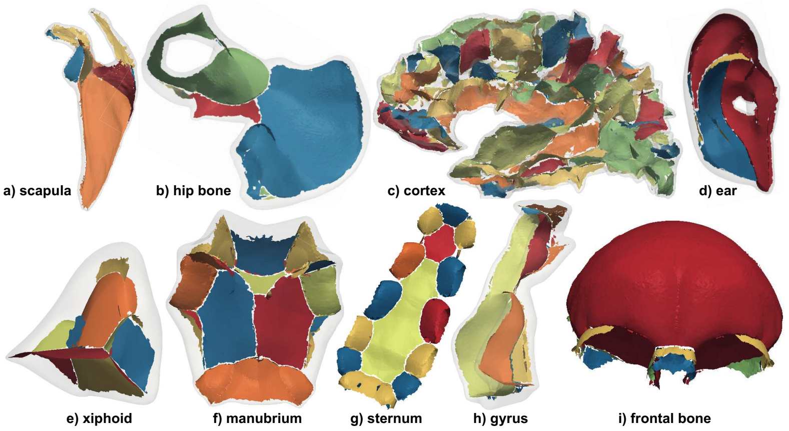


Figure 6.7: Compact medial sheets computed for several anatomical models (Sec. 6.2.2.3).

6.3 APPLICATIONS

We next use our computed medial features (point classification, regularization, and medial surface decomposition into sheets) to support several shape analysis applications. These examples implicitly illustrate the qual-

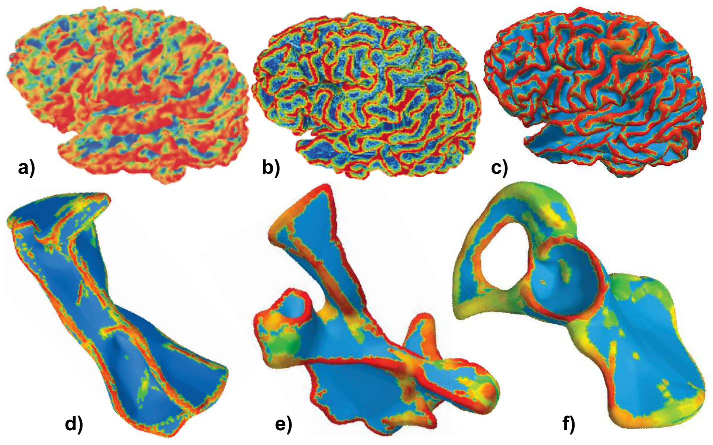


Figure 6.8: Soft edge detection on surfaces using curvature estimation [176] (a); skeleton method of [134] (b); our method (c-f).

ity and robustness of our feature computation methods. Secondly, they show how such features enhance the added-value of surface skeletons by allowing it to support the construction of surface processing tools.

6.3.1 *Surface edge detection*

Finding edges on a surface in 3D space has many applications in segmentation and classification. Most existing edge detectors are based on the surface's curvature tensor [26, 111, 176]. A challenge of such detectors is that they operate at a given scale, *i.e.* find edges of a sharpness range which must be specified. Using skeletons allows finding both sharp and blunt edges: Following the observation that medial surface boundaries (A_3 points) correspond to curvature maxima or edges on the input surface [122], Reniers *et al.* compute surface edges by finding A_3 points as explained in Sec. 6.2.1.2, and next back-projecting these on the input surface by the feature transform [134]. We propose here an alternative approach: For each A_3 skeletal point $\mathbf{x}$, detected as explained in Sec. 6.2.1.2, we find all surface points enclosed in a sphere of radius $DT_{\mathbf{x}} + \tau$, with τ set as explained in Sec. 6.2.1.1, and set each surface point with the smallest $DT_{\mathbf{x}}$ value which enclosed it. Remaining surface points are set to $max(DT_{\mathbf{x}})$. Figure 6.8 (a-c) compares our method with the classical curvature detector of [176] and with [134] for a brain cortex surface. The goal is to find the sulcal brain structures, which correspond to (soft) convex surface edges, an important task in many structural and functional anatomic brain analyses. The presence of sulci is shown using a blue (concave) to red (sharp convex) rainbow colormap, mapping the three studied detectors: mean curvature [176] (Figure 6.8 a), geodesic distance to back-projected A_3 points [134] (Figure 6.8 b), and our sphere radius metric (Figure 6.8 c). Our method achieves a sharper sulci separation than [134], which in turn performs better than [176]. Images (c-f) show our method applied to three additional shapes which exhibit a mix of sharp and blunt edges. As visible, our detector finds both sharp (and thus, thin) and blunt (and thus, thick) edges. The edge sharpness and thickness is also visible in the color mapping.

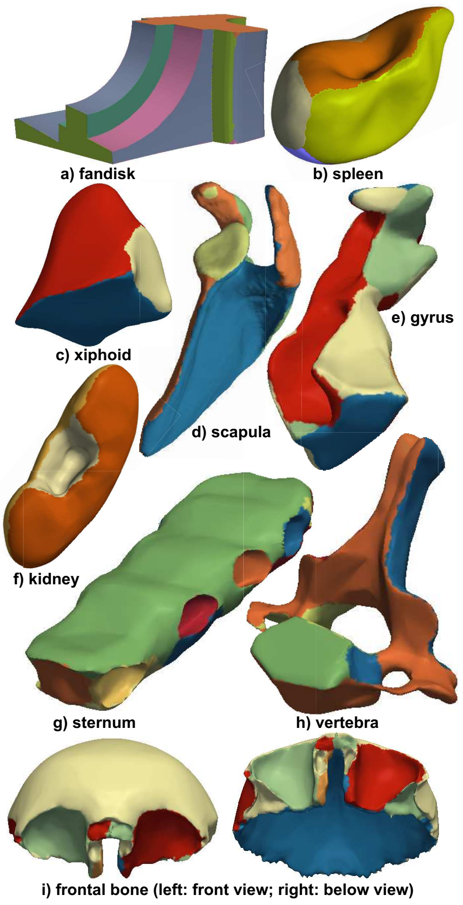


Figure 6.9: Patch based segmentation (Sec. 6.3.2).

6.3.2 *Patch-based segmentation*

Patch-based segmentation (PBS) divides a shape $\partial\Omega$ into *patches*, *i.e.* quasi-flat areas which are separated by sharp creases. Most PBS methods work by clustering surface points using, as similarity metric, the surface curvature or similar quantities [151]. Since medial surfaces fully capture the surface information via the MAT (Chapter. 3), these medial surfaces can be used for PBS. For this, Reniers *et al.* compute soft edges by using the feature transform of low-importance medial-surface points, and next use these thick edges to segment the shape. However, their method needs to handle a large number of special cases (and is thereby quite complicated), and only works for voxel shapes. We propose here a much simpler approach: We project all skeleton-boundary points $\mathbf{p}$ (type A_3) to $\partial\Omega$ via our extended feature transform FT_τ, *i.e.* compute the set $E = \{\mathbf{x} \in FT_\tau(\mathbf{p}) | \mathbf{p} \in S_{\partial\Omega} \wedge type(\mathbf{p}) = A_3\} \subset \partial\Omega$. The set E consists of a thick version of the edges of $\partial\Omega$. Due to the conservativeness of F_τ (Sec. 6.2.1.1), E will contain connected edges, in contrast to *e.g.* a naive thresholding of the curvature of $\partial\Omega$ or other similar local surface classifiers. Hence, we next find patches by simply computing connected components of $\partial\Omega \setminus E$. Finally, we add the points in E to their closest patch, thereby making the resulting patches become a partition of $\partial\Omega$.

Figure 6.9 shows our results, using the same color scheme as Figure 6.7. For models with clear, sharp, edges, we see how patches neatly follow these edges (*e.g.* Figure 6.9 a, rib sockets in Figure 6.9 g, skull concavity in Figure 6.9 i). More importantly, our method handles equally well models with soft edges (Figs. 6.9 b,c,f) and/or mixes of sharp and soft edges (Figs. 6.9 d,g,h).

6.3.3 *Medial sheet mapping segmentation*

In contrast to patch-based segmentation (Sec. 6.3.2), part-based segmentation (pBS) separates a shape $\partial\Omega$ into its meaningful components that are perceived as being the natural 'parts' of the shape [151]. Among the many methods for pBS, *curve* skeletons are often used, as they readily capture the part-in-whole topology of shapes having elongated protrusions. One way to compute a pBS is to find the so-called junction points of curve skeletons (equivalent to Y-curves for surface skeletons), and then cut the shape with curves that go around these points [135]. Such methods are robust and relatively simple to implement, but work well only for shapes with a tubular structure, *i.e.*, which have a meaningful curve skeleton. We propose here to use the surface skeleton for pBS. For this, we compute its medial sheets γ (Sec. 6.2.2.1), and next project these into $\partial\Omega$ using $P(\gamma)$ (Eqn. 6.6). Since all points on $\partial\Omega$ have a skeleton point by construction (Chapter 3), the entire shape is covered by such projections, which give us the 'parts' of the shape. The borders separating two such neighbour parts are nothing but the projections of the Y-curves. Since such curves are smooth [154], and the feature-vector field used for the projection is also smooth (since parallel to $\nabla DT_{\partial\Omega}$ which is divergence-free away from the skeleton [155]), the resulting part borders will also be smooth. Figure 6.10 show several part-based segmentation examples. Although many alternative pBS segmentations are possible, we argue that the identified segments coincide well with what one would regard to be the distinct shape parts. Notably, such segmentations cannot be achieved using a curve-skeleton, since the shown shapes do not have a tubular structure.

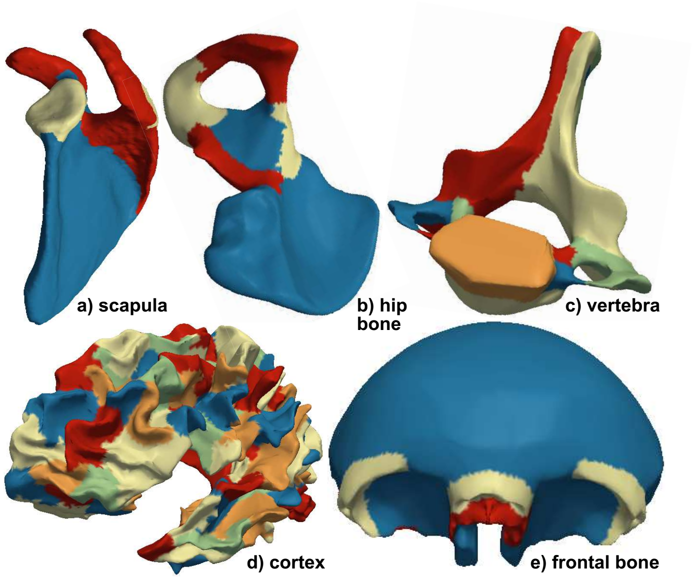


Figure 6.10: Medial sheet mapping segmentation (Sec. 6.3.3).

6.4 DISCUSSION

We discuss next several aspects related to our contribution – showing that we can efficiently and easily compute high-level medial features from large point-cloud skeletons, and that using such features in various applications is a practical proposition.

Generality: As input for all our methods, we require only a raw medial 3D cloud having two feature points per skeleton point. Such point clouds can be very efficiently and easily computed by recent GPU methods [73, 101] or older CPU methods [66], for any type of 3D shape topology or geometry.

Point classification: To our knowledge, our work is the first attempt to compute Giblin's medial point classification [58] for unstructured medial clouds. In addition, using this classification for skeleton regularization (Sec. 6.2.1.3) is considerably simpler to implement, and also much faster, than the alternative MGF metric. Compared to local regularization metrics [122, 155, 168], we do not disconnect the skeleton, and guarantee to remove only a thin layer of boundary points. This gives a simple, fast, and effective way to create clean medial surfaces that preserve relevant skeletal details.

Medial sheet extraction: Separating sheets from a raw medial cloud is a challenging task for which few methods exist, and generic point-cloud clustering tools cannot be easily used (Chapter 4). Our contribution here is the similarity metric (Eqn. 6.4) which combines both local and global shape information. This enables a medial cloud segmentation into sheets which is noise-resistant and has a simple parameter setting. In detail, [88] requires tuning three parameters: The number of nearest neighbours of each skeletal point, the maximal allowed local-flatness of each sheet, and the sheet similarity. In contrast, our method requires a single parameter, the maximum MGF distance between feature-points of two skeleton-points that are on the same sheet (τ in Eqn. 6.5). For all tested shapes, a value of τ equal to four times the local point-density $\rho_{\partial\Omega}$ on the input surface yielded optimal results.

Y-network extraction: Our Y-network extraction finds the points *around* the Y-network of the skeleton. Exact Y-network points are not (by definition) directly extracted by the core skeletonization method we use, since this method always assumes two contact points for each medial point (Eqn. 2.2).

Scalability: We implemented our medial feature computation methods in C++. On an Intel Core i7 3.8 GHz computer, our single-threaded code computes all medial features, except the medial sheets, in a few seconds for all shapes shown in this chapter, which range between 30K and 230K skeleton points. Medial sheet extraction is more costly, as it uses the expensive MGF metric (Eqn. 6.4). On the same platform, using the proposed CPU-based MGF computation (Chapter 3), sheet extraction takes under one minute for all tested shapes. Higher performance can be easily obtained, if desired, *e.g.* using the GPU-based MGF computation from [73].

Limitations: The quality of our medial features highly depends on the quality of the input medial cloud. This depends next on the sampling density of the input shape, since we require only two feature points per medial point (Sec. 6.2).

Applications: For the segmentation and classification applications in Sec. 6.3, we note that better specific techniques (not using medial descriptors) exist. Our sample applications are aimed at showing the possibilities that refined medial features open, as alternatives and in contrast to established approaches, and not as a final answer to the underlying use-cases.

6.5 CONCLUSIONS

We have presented a set of techniques for computing refined medial features from raw medial-surface point clouds. These features include medial point classification, skeleton regularization, Y-network extraction, separating medial sheets, and reconstructing meshed sheets. Such features enrich the abstraction level on which one can reason about medial surfaces, and open ways for constructing new shape processing applications that use medial clouds. We provide, for illustration, sample applications for edge detection and shape segmentation. Overall, our work shows that the more complex surface skeletons can be, technically, used with the same ease and computational efficiency as the simpler, and more frequently used, curve skeletons, without significant additional costs, and with actual significant benefits.

Together with the manifold extraction techniques presented in Chapter 4 and the skeletal density analysis techniques presented in Chapter 5, the techniques presented in this chapter enrich the set of refined medial descriptors that can be easily computed from point-cloud surface skeletons obtained from complex and large 3D models. Along this, we show how these descriptors are useful and usable in supporting a range of surface processing and analysis applications, such as segmentation and classification. All in all, the work in this chapter builds towards our general claim that medial descriptors, complemented by suitable refined descriptors, are efficient and effective tools for shape processing and analysis.

7 DISCUSSION AND CONCLUSIONS

This thesis has presented a framework for three-dimensional shape processing using medial surface representations. Our work addresses the joint challenges posed in computing and interpreting 3D shapes by using medial descriptors, *i.e.* the two following main questions:

- Can 3D medial descriptors (surface and curve skeletons) be computed efficiently, accurately and robustly from large and complex 3D shapes?
- Can such medial descriptors be refined and used to efficiently and effectively support various shape processing applications?

In this chapter, we reflect on these two main questions, and state that they have been both answered positively (up to a sufficient extent). Additionally, we present more detailed information pertaining to the various limitations of our work and also to potential directions for future work.

7.1 COMPUTING 3D SKELETONS

The first part of our investigation (Chapter 3 has focused on designing methods for extracting both curve and surface skeletons of 3D shapes which can be *applied* in practice, *i.e.*, efficiently, effectively, and easily usable on large and complex 3D point-sampled models. We further refined these applicability criteria into a number of desirable sub-criteria:

- *robustness to noise* (in terms of providing a multiscale technique that would separate skeleton branches created by noise and/or small details from those created by important shape parts),
- *scalability* (in terms of size of the input shapes and computational speed),
- *genericity* (in terms of being able to handle complex 3D shapes of various genuses, closed or open, having various sampling densities, and having connectivity information or not), and
- *ease of use* (in terms of number and complexity of the exposed user parameters).

We next discuss our results and findings structured along these criteria.

7.1.1 *Robustness to noise*

From these criteria, robustness to noise has proven the most challenging to satisfy. Based on a study of the literature covering skeletonization of 2D shapes, we have concluded that the so-called boundary-collapse (BC) importance metric, proposed separately and in different contexts by several authors [45, 118, 183] is an elegant, simple to implement, fast to compute, and generic way to rank 2D skeleton points in terms of the amount of input-shape boundary that such skeleton points subtend. A similar criterion was known, at the moment of writing of this thesis, for 3D shapes. Specifically, the medial geodesic function (MGF), proposed by [37] in the context of detecting 3D curve skeletons, was used by [135] in the context of regularizing 3D surface skeletons. An extension of the MGF was also proposed by [135] for regularizing 3D curve skeletons.

However, computing the MGF was shown to be very expensive by both [37] and [135]. Since this metric was shown by previous work to be the best way to regularize 3D skeletons (both curve and surface variants) for basically any kind of shape, our first attempt to regularize 3D skeletons was to re-use, or extend, the simpler BC metric from 2D to 3D shapes. For this, we studied the conjecture proposed by [99] that projections of 3D (curve) skeletons of some 3D shape match the 2D skeletons of the shape's 2D projection. By using *regularized* 2D skeletons of such projections under the BC metric, we thus aimed to reconstruct regularized 3D (curve) skeletons, without the need of computing a 'true' 3D importance metric. The results of this work, discussed in Sec. 3.7, show that it is indeed possible to compute regularized 3D *curve* skeletons of a wide range of shapes by only using what is essentially a 2D regularized skeletonization technique. Further use of a GPU implementation of [183] made us able to compute 3D regularized curve-skeletons at close to interactive frame rates.

However, the above approach cannot compute curve skeletons for all possible 3D shapes, due to inherent occlusions which make parts of the geometry of such shapes not recoverable from any 2D projection. More importantly, this approach cannot compute regularized 3D *surface* skeletons. As such, we reverted our attention to the latter types of skeletons. For these, we decided that the MGF metric proves, to date, the only viable alternative that can regularize any type of 3D shape by gradually eliminating branches (manifolds) caused by small surface details or noise, and keeping the important branches, without changing the topology of the surface skeleton.

7.1.2 *Scalability*

Given the above observation, our next aim has been to make the MGF metric practical for large and complex 3D shapes. The original approaches proposing this metric both used a voxel shape representation [37, 135]. Even assuming a modern powerful workstation, such an approach would be limited in terms of memory scalability to handling volumes of roughly 1024^3 voxels. In turn, this makes the accurate capture of fine details present in many mesh models very challenging. As such, we decided that a mesh-based representation for the *entire* skeletonization pipeline, from the input shape and up to the skeleton representation, is a better solution. In this direction, we studied the performance bottlenecks of the point-cloud-based skeletonization method of [101], and proposed a parallel implementation that removes most, if not all, such bottlenecks (Section 3.2.1). We verified that our parallel implementation, realized both on the CPU (using multi-threading) and on the GPU (using the CUDA computing platform) scales roughly linearly with the number of input points, thus has optimal complexity. Separately, we noted that our GPU skeletonization delivers a performance boost of about two orders of magnitude as compared to state-of-the-art 3D skeletonization methods that use voxel representations. Practically, this means that we are now able to provide surface skeletons of models of up to 1M vertices in sub-second time on a modern GPU. Separately, we note that this performance boost is in line with typical speed-ups delivered by (well designed) CUDA algorithms as opposed to their serial CPU counterparts.

An interesting side-finding of this research was that the key bottleneck in designing parallel mesh-based skeletonization methods is the efficiency of the nearest-neighbor technique used. Although many highly efficient such techniques are known for a CPU single-thread implementation, adapting them to the context of a parallel GPU implementation is challenging, and the obtained performance figures have been seen to vary highly and/or differ from the ones stated by earlier studies in the same field. As such, we believe that working on designing efficient GPU-based nearest-neighbor schemes is an interesting (and important) area for future research.

Atop of the raw skeletal point cloud delivered by our parallel skeletonization, we next studied different ways to efficiently compute the MGF metric, using a parallel implementation. Our proposal, described in Section 3.3.2, proved to be between one and three orders of magnitude faster than known geodesic-tracing algorithms. Concretely, we can compute regularized surface skeletons of large models (hundreds of thousands of vertices) in a few (tens of) seconds on a modern GPU. Additionally, our parallel geodesic technique is not limited or constrained to be used only with skeleton applications. Any computer graphics, or shape processing, application requiring efficient and accurate computation of geodesics on meshed surfaces, either in terms of a given starting point and given initial direction or in terms of the shortest geodesic between two given points, can use our technique directly.

As a final scalability note, we should mention that recent developments in the field of parallel computing with the CUDA platform are highly likely to provide an additional boost to our technique. In practice, we identified that the major bottleneck to our geodesic computation is given by the non-uniform load balancing of the CUDA cores, each which traces one whole geodesic. Late-generation CUDA techniques, including dynamic load balancing and fast atomic operations, could be used to redesign our parallel tracing to achieve even higher performance figures.

7.1.3 *Genericity*

For algorithmic simplicity and ease of implementation on parallel hardware frameworks, two and only two feature points are extracted for each surface point. As pointed out previously, due to the surface sampling limitations, this can lead to noisy skeleton points around smooth rounded edges in the shape. This limitation is overcome in chapter 6, by presenting a method to regularize and eliminate those noisy edges. In this approach all remaining skeletal points and edges are retained, contrary to what a *geodesic* based regularization can provide: The regularization in this later case is uniform, making no distinction of points generated due to sampling on low curvature edges and actual sharp shape edges. Note that the noisy skeleton points generated on smooth edges can have relatively large geodesic distances between their feature points.

The presented extraction method is suitable for computing non uniformly sampled meshes. These meshes are suitable for further analysis, however care must be taken when choosing the correct parameters for the application of several of the presented methods. For instance, the resulting point density differences of the algorithms presented in Chapter 5 become not only due to the boundary curvature properties but also due to its sampling properties.

Non-watertight meshes can be handled by the presented skeletonization method. Since the connectivity information is not actively used for the skeleton computation, only the closest neighbours to the sphere locus are considered. A typical example of open meshes where the method has been demonstrated are the dental casts: These consist of open meshes on the bottom of the shape.

7.2 MANIFOLD EXTRACTION FOR SHAPE ANALYSIS

In Chapter 4, a generic point cloud processing method has been presented to tackle several challenges associated with the extraction of medial surfaces from point cloud boundary representations. With this method, smooth manifolds (which can be embedded into noise) can be uniquely identified. This method can be applied in several stages of shape processing:

SHAPE PRE-PROCESSING FOR SKELETONIZATION As pointed out along this thesis, small variations in the input boundary can lead to large noisy branches in the computed skeleton. In the presence of outlier points, further unpredictable results would be expected. Therefore, the extraction of smooth manifolds enables a more predictable medial surface computation suitable for further shape analysis operations.

MEDIAL SURFACE DECOMPOSITION Medial surfaces consist mainly of (intersecting) manifold surfaces. Finding such manifolds is not easy, and is essential for many subsequent usages of the surface skeleton. Our method can be used for directly extracting medial sheets from unstructured medial clouds. Moreover, our method can remove medial noise from such sheets prior to segmentation, thereby acting as a simpler (and cheaper) regulatization tool.

SHAPE SEGMENTATION Our method can be used for other contexts than the regularization (denoising) and/or segmentation of medial clouds, *e.g.* in the context of denoising and segmentation of generic 3D point clouds. We have shown results that support the claim that our method is faster, of a better quality, and easier to parameterize, as compared to state-of-the-art methods in this field. This result can lead to a new generation of point-cloud processing methods for general 3D shapes.

7.3 FEATURE EXTRACTION FROM MEDIAL SURFACES

Chapters 5 and 6 present several methods that extract so-calle refined medial descriptors from raw medial point clouds. Concrete instances of such descriptors are: classification of medial points into types according to [58], decomposition into medial sheets, medial boundary extraction, Y-network extraction, and analysis of the medial sampling density properties. From a theoretical viewpoint, our work shows that all such descriptors, recognized as being relevant in the medial literature, can be easily and efficiently computed, which is a novel result. Subsequently, we show how the computation of these descriptors enables the implementation of several well-known shape processing applications such as segmentation and classification. This supports our claim that medial descriptors do indeed offer new efficient ways to tackle such applications, along traditional methods.

7.4 LIMITATIONS

As in any explorative research work, several (subtle) limitations exist to our proposals. We next outline the main ones. Firstly, we cannot claim that our methods are usable out-of-the-box for *any* type of 3D point-sampled shape. Extreme conditions such as very low or very high sampling densities or very high amounts of sampling noise will obviously negatively affect our results. However, we should note that such conditions do equally (if not more) affect all skeletonization methods we are aware of.

Secondly, nearly all our applications (with the exceptions shown in Chapter 6) regularize skeletons by the MGF metric. While this metric has indeed desirable properties (keeps the skeleton connected, and has a simple geometric interpretation in terms of feature sizes on the input surface), we do not have a fundamental justification of why other metrics could not be equally good (or better) for regularizing skeletons. The research area of skeleton regularization is, thus, still open to exploration.

Thirdly, in terms of completeness, we should admit that many other more complex shape descriptors exist apart from the ones studied here. In defense of our work, we state that one should first show how the basic descriptors can be computed and used, before one proceeds to the more involved ones. This is the approach we have taken here.

Finally, we should state that many other alternative methods exist for the applications covered in this thesis (shape classification and segmentation). However, our key goal was not to optimize the solution(s) for a particular application, but to show that medial descriptors, long blamed for their lack of effectiveness due to aspects such as computational complexity and instability, *can* be used as easily and efficiently as other methods to the same practical ends. We believe that this is a major contribution of our work, which partially dispells the aforementioned negative claims concerning skeletons, and thereby opens several new research and application directions in this area.

7.5 FUTURE WORK

Although several methods and applications have been presented which potentially enable several novel shape analysis applications, the presented work is far from exhaustive. We foresee several new research directions spawned by our work. Examples thereof cover the usage of skeletons for shape segmentation in wider application fields, *e.g.* full volumetric datasets such as 3D CT and MRI scans; the usage of our skeletonization algorithms for enabling real-time shape matching and/or registration algorithms for time-dependent datasets; the extension of the presented work to time changing shapes; and novel shape representation and/or compression methods using the symmetry information encoded by medial descriptors. The groundwork and tools for these fields have been laid out by this thesis. It is the task of future research to exploit our findings.

BIBLIOGRAPHY

[1] M. Alexa, J. Behr, D. Cohen-Or, S. Fleishman, D. Levin, and C. Silva. Point set surfaces. In *Proc. IEEE Visualization*, pages 21–28, 2001.

[2] N. Amenta, S. Choi, and R. Kolluri. The power crust. In *Proc. SMA*, pages 65–73. ACM, 2001.

[3] N. Amenta, M. Bern, and M. Kamvysselis. A new Voronoi-based surface reconstruction algorithm. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '98, pages 415–421, New York, NY, USA, 1998. ACM. ISBN 0-89791-999-8. DOI 10.1145/280814.280947. URL http://doi.acm.org/10.1145/280814.280947.

[4] C. Arcelli, G. Sanniti di Baja, and L. Serino. Distance-driven skeletonization in voxel images. *IEEE TPAMI*, 33(4):709–720, 2011.

[5] C. Armstrong, T. Tam, D. Robinson, R. McKeag, and M. Price. Automatic generation of well structured meshed using medial axis and surface subdivision. In *Proc. ASME Design Automation*, pages 1–10, 1991.

[6] Atron, Inc. 3D intraoral scanner, 2013. www.a-tron3d.com/en/products/id-3d-intraoral-scanner.html.

[7] O. K. C. Au, C. Tai, H. Chu, D. Cohen-Or, and T. Lee. Skeleton extraction by mesh contraction. In *Proc. ACM SIGGRAPH*, pages 441–449, 2008.

[8] X. Bai. Skeleton-based shape classification using path similarity. *International Journal of Pattern Recognition*, 22(4):733–746, 2008.

[9] X. Bai and L. Latecki. Path similarity skeleton graph matching. *IEEE TPAMI*, 30(7):1282–1292, 2008.

[10] X. Bai, L. Latecki, and W. Y. Liu. Skeleton pruning by contour partitioning with discrete curve evolution. *IEEE TPAMI*, 3(29): 449–462, 2007.

[11] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications*. Springer, 2010.

[12] F. Bernardini, J. Mittleman, H. Rushmeier, C. Silva, and G. Taubin. The ball-pivoting algorithm for surface reconstruction. *IEEE TVCG*, 5(4):349–359, 1999.

[13] S. Beucher. Digital skeletons in Euclidean and geodesic spaces. *Signal Processing*, 38(1):127–141, 1994.

[14] I. Bitter, M. Sato, M. Bender, K. McDonnell, and A. Kaufman. CEASAR: A smooth, accurate and robust centerline extraction algorithm. In *Proc. IEEE Visualization*, pages 45–52, 2000.

[15] H. Blum. A Transformation for Extracting New Descriptors of Shape. In W. Wathen-Dunn, editor, *Models for the Perception of Speech and Visual Form*, pages 362–380. MIT Press, Cambridge, 1967.

[16] S. Bouix, K. Siddiqi, and A. Tannenbaum. Flux driven automatic centerline extraction. *Medical Image Analysis*, 9(3):209–221, 2005.

[17] S. Bouix, K. Siddiqi, A. Tannenbaum, and S. Zucker. Medial axis computation and evolution. In *Statistics and analysis of shape*, chapter 1, pages 1–28. Springer LNCS, 2006.

[18] M. Breunig, H. P. Kriegel, R. Ng, and J. Sander. LOF: Identifying density-based local outliers. In *Proc. SIGMOD*, pages 93–104. ACM, 2000.

[19] J. Cao, A. Tagliasacchi, M. Olson, H. Zhang, and Z. Su. Point cloud skeletons via laplacian-based contraction. In *Proc. IEEE SMI*, pages 187–197, 2010.

[20] T. Cao, K. Tang, A. Mohamed, and T. Tan. Parallel banding algorithm to compute exact distance transform with the GPU. In *Proc. SIGGRAPH I3D Symp.*, pages 134–141, 2010.

[21] W. Cao and R. Haralick. Nonlinear manifold clustering by dimensionality. In *Proc. ICPR*, pages 920–924. IEEE, 2006.

[22] L. Cayton. A nearest neighbor data structure for graphics hardware. In *Proc. ADMS*, pages 192–197, 2010. people.kyb.tuebingen.mpg.de/lcayton.

[23] M. Chang, F. Leymarie, and B. Kimia. Surface reconstruction from point clouds by transforming the medial scaffold. *CVIU*, (113):1130–1146, 2009.

[24] J. Chuand and N. Ahuja. Path planning using the Newtonian potential. In *Proc. IEEE Intl. Conf. on Robotics and Automation*, volume 1, pages 558–563, 1991.

[25] J. Chuang, C. Tsai, and M. Ko. Skeletonization of three-dimensional object using generalized potential field. *IEEE TPAMI*, 22(11):1241–1251, 2000.

[26] U. Clarenz, M. Griebel, M. Schewitzer, and A. Telea. Feature sensitive multiscale editing on surfaces. *Visual Computer*, 20(5): 329–343, 2004.

[27] U. Clarenz, M. Rumpf, and A. Telea. Surface processing methods for point sets using finite elements. *Computers & Graphics*, 28 (6):851–868, 2004.

[28] U. Clarenz, M. Rumpf, and A. Telea. Finite elements on point based surfaces. In *Proc. Symp. on Point Based Graphics*, pages 192–200. Eurographics, 2004.

[29] D. Cohen-Steiner and F. Da. A greedy Delaunay-based surface reconstruction algorithm. *Visual Comput.*, 20(1):4–16, 2004.

[30] D. Comaniciu and P. Meer. Mean shift: A robust approach toward feature space analysis. *IEEE TPAMI*, 24(5):603–619, 2002.

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

[32] N. Cornea, D. Silver, and P. Min. Curve-skeleton properties, applications, and algorithms. *IEEE TVCG*, 13(3):87–95, 2007.

[33] L. Costa and R. Cesar. *Shape analysis and classification*. CRC Press, 2000.

[34] J. Damon. Global medial structure of regions in $\mathbb{R}^3$. *Geometry and Topology*, 10:2385–2429, 2006.

[35] T. Dey and S. Goswami. Provable surface reconstruction from noisy samples. In *Proc. Ann. SCG*, pages 428–438, 2004.

[36] T. Dey and S. Goswami. Provable surface reconstruction from noisy samples. *Int. J. Comput. Geom. Ap.*, 35(1):340–355, 2006.

[37] T. Dey and J. Sun. Defining and computing curve skeletons with medial geodesic functions. In *Proc. SGP*, pages 143–152. IEEE, 2006.

[38] T. Dey and W. Zhao. Approximating the medial axis from the Voronoi diagram with a convergence guarantee. *Algorithmica*, 38: 179–200, 2003.

[39] T. Dey, K. Li, E. Ramos, and R. Wenger. Isotopic reconstruction of surfaces with boundaries. *CGF*, 28(5):1371–1382, 2009.

[40] 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 LNCS, 2006.

[41] H. Du and H. Qin. Medial axis extraction and shape manipulation of solid objects using parabolic PDEs. In *Proc. ACM SMA*, pages 168–176, 2004.

[42] M. van Eede, D. Macrini, A. Telea, and C. Sminchisescu. Canonical skeletons for shape matching. In *Proc. ICPR*, pages 542–550, 2006.

[43] E. Eisemann and X. Decoret. Fast scene voxelization and applications. In *Proc. SIGGRAPH I3D Symp.*, pages 71–78, 2006.

[44] 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.

[45] A. X. Falcão, J. Stolfi, and R. A. Lotufo. The image foresting transform: Theory, algorithms, and applications. *IEEE TPAMI*, 26(1):19–29, 2004.

[46] S. Flöry. Fitting curves and surfaces to point clouds in the presence of obstacles. *CAGD*, 26(2):693–707, 2009.

[47] M. J. Flynn. Some computer organizations and their effectiveness. *IEEE Trans. Comput.*, 21(9):948–960, September 1972. DOI 10.1109/TC.1972.5009071. URL `http://dx.doi.org/10.1109/TC.1972.5009071`.

[48] H. Fogg, C. Armstrong, and R. Robinson. Decomposition of domains into quad blocks using the medial axis transform. In *Proc. 20th International Meshing Roundtable*. Springer, 2011.

[49] S. Foix, G. Alenya, and C. Torras. Lock-in time-of-flight (ToF) cameras: A survey. *IEEE Sensors Journal*, 11:1917–1926, 2011.

[50] M. Foskey, M. Lin, and D. Manocha. Efficient computation of a simplified medial axis. In *Proc. Shape Modeling*, pages 135–142, 2003.

[51] P. Frey. YAMS: a fully automatic adaptive isotropic surface remeshing procedure. tech. rep. 0252, INRIA, Nov. 2001. `www.ann.jussieu.fr/~frey`.

[52] J. Fung and S. Mann. Computer vision signal processing on graphics processing units. In *Acoustics, Speech, and Signal Processing, 2004. Proceedings.(ICASSP'04). IEEE International Conference on*, volume 5, pages V–93. IEEE, 2004.

[53] V. Garcia, E. Debreuve, and M. Barlaud. Fast *k* nearest neighbor search using GPU. In *Proceedings International Workshop on Computer Vision on GPU (CVGPU)*, pages 77–83, 2008.

[54] S. Garrido, L. Moreno, M. Abderrahim, and F. Martin. Path planning for mobile robot navigation using voronoi diagram and fast marching. In *Proc. IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems*, pages 2376–2381, 2006.

[55] S. Garrido, L. Moreno, and D. Blanco. Voronoi diagram and fast marching applied to path planning. In *Proc. IEEE Intl. Conf. on Robotics and Automation*, pages 3049–3054, 2006.

[56] C. F. Gauss and P. Pesic. *General investigations of curved surfaces*. Dover Books on Mathematics. Dover Publ., 2005.

[57] Y. Ge and J. Fitzpatrick. On the generation of skeletons from discrete euclidean distance maps. *IEEE TPAMI*, 18:1055–1066, 1996.

[58] P. Giblin and B. Kimia. A formal classification of 3D medial axis points and their local geometry. *IEEE TPAMI*, 26(2):238–251, 2004.

[59] J. Giesen, B. Miklos, M. Pauly, and C. Wormser. The scale axis transform. In *Proc. Annual Symp. Comp. Geom.*, pages 106–115, 2009.

[60] W. Goh. Strategies for shape matching using skeletons. *CVIU*, 110(3):326–345, 2008.

[61] A. B. Goldberg. Multi-manifold semi-supervised learning. *Proc. AISTATS*, pages 169–176, 2009.

[62] A. Hajdu, C. Giamas, and I. Pitas. Object simplification using a skeleton-based weight function. In *Proc. ISSCS*, pages 1–4, 2007.

[63] R. Haralick and R. Harpaz. Linear manifold clustering. *Machine Learning and Data Mining in Pattern*, pages 132–141, 2005.

[64] M. Hassouna and A. Farag. Variational curve skeletons using gradient vector flow. *IEEE TPAMI*, 31(12):2257–2274, 2009.

[65] D. M. Hawkins. *Identification of Outliers*. Chapman and Hall, London, 1980.

[66] W. Hesselink and J. Roerdink. Euclidean skeletons of digital image and volume data in linear time by the integer medial axis transform. *IEEE TPAMI*, 30(12):2204–2217, 2008.

[67] Y. Hirogaki, T. Sohmura, H. Satoh, J. Takahashi, and K. Takada. Complete 3-d reconstruction of dental cast shape using perceptual grouping. *Medical Imaging, IEEE Transactions on*, 20(10):1093–1101, 2001. DOI 10.1109/42.959306.

[68] M. de Hoon, S. Imoto, J. Nolan, and S. Myiano. Open source clustering software. *Bioinformatics*, 20(9):1453–1454, 2004.

[69] I. Hotz and H. Hagen. Visualizing geodesics. In *Proc. IEEE Visualization*, pages 311–318, 2000.

[70] Y. Ioannou. DoN implementation in the PCL library, 2013. URL `pointclouds.org/documentation/tutorials/don_segmentation.php`.

[71] A. K. Jain and M. N. Murty. Data clustering: A review. *ACM Computing Surveys*, 31(3):264–323, 1999.

[72] A. Jalba and J. Roerdink. Efficient surface reconstruction using Coulomb potentials. *IEEE TVCG*, 13(6):1512–1519, 2007.

[73] 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.

[74] T. Johnson, I. Kwok, and R. Ng. Fast computation of 2-dimensional depth contours. In *Proc. KDDM*, pages 224–228. AAAI Press, 1998.

[75] G. Katz and J. Kider. All-pairs shortest-paths for large graphs on the {GPU}. In *Proc. Graphics Hardware*, pages 208–216, 2008.

[76] S. Katz and A. Tal. Hierarchical mesh decomposition using fuzzy clustering and cuts. *ACM TOG*, 22(3):954–961, 2003.

[77] M. Kazhdan, M. Bolitho, and H. Hoppe. Poisson surface reconstruction. In *Proc. SGP*, pages 61–70, 2006.

[78] R. Kimmel, D. Shaked, N. Kiryati, and A. Bruckstein. Skeletonization via Distance Maps and Level Sets. *Computer Vision and Image Understanding: CVIU*, 62(3):382–391, 1995.

[79] A. Kiraly, K. Helferty, E. Hoffman, and G. McLennan. Three-dimensional path planning for virtual bronchoscopy. *IEEE TMI*, 23(11):1365–1379, 2004.

[80] R. Klette and A. Rosenfeld. *Digital geometry: Geometric methods for digital picture analysis*. Morgan Kaufmann, 2004.

[81] E. M. Knorr, R. T. Ng, and V. Tucakov. Distance-Based Outliers: Algorithms and Applications. *J. VLDB*, 8(3-4), 2000.

[82] J. W. L. Kobbelt. Optimized sub-sampling of point sets for surface splatting. *CGF*, 23(3):643–652, 2004.

[83] T. Kondo, S. Ong, and K. W. C. Foong. Tooth segmentation of dental study models using range images. *IEEE Trans Med Imag*, 23(3):350–362, 2004. DOI 10.1109/TMI.2004.824235.

[84] T. Kronfeld, D. Brunner, and G. Brunnett. Snake-based segmentation of teeth from virtual dental casts. *CAGD*, 7(2):221–233, 2010.

[85] D. Kushnir, M. Galun, and A. Brandt. Fast multiscale clustering and manifold identification. *Patt. Recog.*, 39(10):1876–1891, 2006.

[86] J. Kustra, M. de Jager, A. Jalba, and A. Telea. Teeth shape modeling pipeline for oral healthcare appliances development. In *Proc. ICCE*. IEEE, 2014.

[87] J. Kustra, M. de Jager, A. Jalba, and A. Telea. A medial point cloud based algorithm for dental cast segmentation. In *Proc. ICCE*. IEEE, 2014.

[88] J. Kustra, A. Jalba, and A. Telea. Robust segmentation of multiple intersecting manifolds from unoriented noisy point clouds. *CGF*, 33(1):73–87, 2014.

[89] J. Kustra, A. Jalba, and A. Telea. Shape segmentation using medial point clouds with applications to dental cast analysis. In *Proc. VISAPP*, pages 151–159, 2014.

[90] J. Kustra, A. Jalba, and A. Telea. Computing refined skeletal features from medial point clouds. *Pattern Recognition Letters (Submitted)*, 2014.

[91] J. Kustra, A. Jalba, and A. Telea. Probabilistic View-based Curve Skeleton Computation on the GPU. In *8th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications, VISAPP 13*, 2013.

[92] C. Lantuéjoul. *La squelettisation et son application aux mesures topologiques de mosaiques polycristallines*. PhD thesis, School of Mines, Paris, 1979.

[93] F. Leymarie and B. Kimia. The medial scaffold of 3D unorganized point clouds. *IEEE TVCG*, 29(2):313–330, 2007.

[94] F. Leymarie and M. Levine. Simulating the grassfire transform using an active contour model. *IEEE TPAMI*, 14(1):56–75, jan 1992. DOI 10.1109/34.107013.

[95] X. Li, T. Woon, T. Tan, and Z. Huang. Decomposing polygon meshes for interactive applications. In *Proc. I3D Symp.*, pages 35–42, 2001.

[96] L. Linsen and H. Prautzsch. Global versus local triangulations. In *Proc. Eurographics*, page 257–263, 2001.

[97] B. Liu, A. C. Telea, J. B. Roerdink, G. J. Clapworthy, D. Williams, P. Yang, F. Dong, V. Codreanu, and A. Chiarini. Parallel centerline extraction on the gpu. *Computers &amp; Graphics*, 2014. Accepted.

[98] L. Liu, E. Chambers, D. Letscher, and T. Ju. A simple and robust thinning algorithm on cell complexes. *CGF*, 29(7):2253‚Äì2260, 2010.

[99] M. Livesu, F. Guggeri, and R. Scateni. Reconstructing the curve-skeletons of 3D shapes using the visual hull. *IEEE TVCG*, 18(11): 1891–1901, 2012.

[100] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. *SIGGRAPH Comput. Graph.*, 21(4):163–169, August 1987. DOI 10.1145/37402.37422. URL `http://doi.acm.org/10.1145/37402.37422`.

[101] J. Ma, S. Bae, and S. Choi. 3D medial axis point approximation using nearest neighbors and the normal field. *Vis. Comput.*, 28(1): 7–19, 2012.

[102] D. Macrini, K. Siddiqi, and S. Dickinson. From skeletons to bone graphs: Medial abstraction for object recognition. In *Proc. CVPR*, pages 324–332, 2008.

[103] G. Malandain and S. Fernandez-Vidal. Euclidean skeletons. *Image Vision Comput.*, 16(5):317–327, 1998.

[104] G. F. Marshall. *Handbook of Optical and Laser Scanning*. Marcel Dekker, Inc., 2004.

[105] G. Medioni and C. K. Tang. Tensor voting: Theory and applications. In *Proc. RFIA*, 2000.

[106] A. Meijster, J. B. T. M. Roerdink, and W. H. Hesselink. A general algorithm for computing distance transforms in linear time. In *Mathematical Morphology and its Applications to Image and Signal Processing*, pages 331–340, 2000.

[107] F. Meyer. Skeletons and perceptual graphs. *Signal Processing*, 16 (4):335–363, 1989.

[108] B. Miklos, J. Giesen, and M. Pauly. Discrete scale axis representations for 3D geometry. In *Proc. ACM SIGGRAPH*, pages 394–493, 2010.

[109] B. Mirtich. Using skeletons for nonholonomic path planning among obstacles. In *Proc. IEEE Intl. Conf. on Robotics and Automation*, volume 3, pages 2533–2540, 1992.

[110] N. Mitsuhashi, K. Fujieda, T. Tamura, S. Kawamoto, T. Takagi, and K. Okubo. BodyParts3D: 3D structure database for anatomical concepts. *Nucleic Acids Research*, 37(1):782–785, 2009.

[111] H. Moreton and C. Séquin. Functional optimization for fair surface design. In *Proc. ACM SIGGRAPH*, pages 167–176, 1992.

[112] M. Mortara, G. Patanet, M. Spagnuolo, B. Falcidieno, and J. Rossignac. Plumber: A method for multiscale decomposition of 3D shapes into tubular primitives and bodies. In *Proc. ACM SMA*, pages 339–344, 2004.

[113] B. Mory. *Interactive Segmentation of 3D Medical Images with Implicit Surfaces*. PhD thesis, Ecole Polytechnique Federale de Lausanne, 2011.

[114] D. Mount and S. Arya. Approximate nearest neighbor search software, 2011. www.cs.umd.edu/~mount/ANN.

[115] P. Mullen, F. DeGoes, M. Desbrun, D. Cohen, and P. Alliez. Signing the unsigned: robust surface reconstruction from raw pointsets. *CGF*, 29(5):1733–1741, 2010.

[116] F. Nooruddin and G. Turk. Simplification and repair of polygonal models using volumetric techniques. *IEEE TVCG*, 9(2):191–205, 2003. see also www.cs.princeton.edu/~min/binvox.

[117] NVIDEA. Cuda, 2014. www.nvidea.com/cuda.

[118] R. L. Ogniewicz and O. Kubler. Hierarchic Voronoi skeletons. *Pattern Recognition*, (28):343–359, 1995.

[119] K. Palagyi and A. Kuba. Directional 3D thinning using 8 subiterations. In *Proc. DGCI*, volume 1568, pages 325–336. Springer LNCS, 1999.

[120] P. Peter and M. Breu. Refined homotopic thinning algorithms and quality measures for skeletonisation methods. In *Innovations for Shape Analysis Mathematics and Visualization*, pages 77–92. Springer, 2013.

[121] 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`.

[122] S. Pizer, K. Siddiqi, G. Szekely, J. Damon, and S. Zucker. Multiscale medial loci and their properties. *IJCV*, 55(2-3):155–179, 2003.

[123] M. Pollefeys, D. Nistér, J. M. Frahm, A. Akbarzadeh, P. Mordohai, B. Clipp, C. Engels, D. Gallup, S. J. Kim, P. Merrell, C. Salmi, S. Sinha, B. Talton, L. Wang, Q. Yang, H. Stewénius, R. Yang, G. Welch, and H. Towles. Detailed real-time urban 3D reconstruction from video. *Int. J. Comput. Vision*, 78(2-3):143–167, 2008.

[124] K. Polthier and M. Schmies. Straightest geodesics on polyhedral surfaces. In *ACM SIGGRAPH Courses*, pages 30–38, 2006.

[125] S. Prohaska and H. C. Hege. Fast visualization of plane-like structures in voxel data. In *Proc. IEEE Visualization*, page 29–36, 2002.

[126] C. Pudney. Distance-ordered homotopic thinning: A skeletonization algorithm for 3D digital images. *CVIU*, 72(3):404–413, 1998.

[127] T. Rabbani, F. Van Den Heuvel, and G. Vosselmann. Segmentation of point clouds using smoothness constraint. *Intl. Archives of Photogrammetry Remote Sensing and Spatial Information Sciences*, 36(5):1–6, 2006.

[128] D. Reniers and A. Telea. Tolerance-based feature transforms. In *Advances in Comp. Graphics and Comp. Vision*, pages 187–200. Springer, 2007.

[129] D. Reniers and A. Telea. Part-type segmentation of articulated voxel-shapes using the junction rule. *CGF*, 27(7):1837–1844, 2008.

[130] D. Reniers and A. Telea. Patch-type segmentation of voxel shapes using simplified surface skeletons. *CGF*, 27(7):1954–1962, 2008.

[131] D. Reniers and A. Telea. Robust segmentation of voxel shapes using medial surfaces. In *Proc. SMI*, 2008.

[132] D. Reniers and A. Telea. Segmenting simplified surface skeletons. In *Proc. DGCI*, pages 132–145. Springer, 2008.

[133] D. Reniers and A. Telea. Extreme simplification and rendering of point sets using algebraic multigrid. *Computing and Visualization in Science*, 12(1):9–22, 2009.

[134] D. Reniers, A. Jalba, and A. Telea. Robust classification and analysis of anatomical surfaces using 3D skeletons. In *Proc. VCBM*, pages 61–68. EG Press, 2008.

[135] 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.

[136] T. Ringbeck and B. Hagebeuker. A 3D Time of Flight Camera for Object Detection. *Measurement*, 9:867–879, 2007.

[137] T. Robinson. Automated mixed dimensional modelling with the medial object. In *Proc.* 17th *International Meshing Roundtable*, pages 281–298. Springer, 2008.

[138] P. Rosenthal and L. Linsen. Smooth surface extraction from unstructured point-based volume data using PDEs. *IEEE TVCG*, 14 (6):1531–1546, 2001.

[139] L. Rossi and A. Torsello. An adaptive hierarchical approach to the extraction of high resolution medial surfaces. In *Proc. 3DIMPVT*, pages 371–378, 2012.

[140] M. Rumpf and A. Telea. A continuous skeletonization method based on level sets. In *Proc. VisSym*, pages 151–158, 2002.

[141] S. Rusinkiewicz and M. Levoy. QSplat: A multiresolution point rendering system for large meshes. In *Proc. SIGGRAPH*, pages 230–237, 2000.

[142] R. B. Rusu, Z. C. Marton, N. Blodow, M. Dolha, and M. Beetz. Towards 3D point cloud based object maps for household environments. *Robotics and Autonomous Systems*, 56(11):927–941, 2008.

[143] P. Sampl. Semi-structured mesh generation based on medial axis. In *Proc.* 9th *International Meshing Roundtable*, pages 21–32. Sandia National Laboratories, 2000.

[144] M. Schmitt. Some examples of algorithms analysis in computational geometry by means of mathematical morphological techniques. In J. D. Boissonnat and J. P. Laumond, editors, *Geometry and Robotics*, volume 391 of *Lecture Notes in Computer Science*, pages 225–246. Springer Berlin Heidelberg, 1989. ISBN 978-3-540-51683-5. DOI 10.1007/3-540-51683-2_33. URL http://dx.doi.org/10.1007/3-540-51683-2_33.

[145] J. Serra. *Image Analysis and Mathematical Morphology*. Academic Press, Inc., Orlando, FL, USA, 1983. ISBN 0126372403.

[146] J. Sethian. *Level Set Methods and Fast Marching Methods*. Cambridge Univ. Press, 2002.

[147] J. Shah. Gray skeletons and segmentation of shapes. *CVIU*, 99 (1):96–109, 2005.

[148] D. Shaked and A. Bruckstein. Pruning medial axes. *CVIU*, 69(2): 156–169, 1998.

[149] S. Shalom, A. Shamir, H. Zhang, and D. Cohen-Or. Cone carving for surface reconstruction. *ACM TOG*, 29(6):547–555, 2010.

[150] A. Shamir. A formulation of boundary mesh segmentation. In *Proc.3DPVT*, pages 378–386, 2004.

[151] A. Shamir. A survey on mesh segmentation techniques. *CGF*, 27 (6):1539–1556, 2008.

[152] H. Sheung and C. Wang. Robust mesh reconstruction from unoriented noisy points. In *Proc. SPM*, pages 13–24. ACM, 2009.

[153] J. Shewchuk. Triangle: Engineering a {2D} Quality Mesh Generator and Delaunay Triangulator. In *Applied Computational Geometry: Towards Geometric Engineering*, pages 203–222. Springer LLNC, 1996.

[154] K. Siddiqi and S. Pizer. *Medial Representations: Mathematics, Algorithms and Applications*. Springer, 2009.

[155] K. Siddiqi, S. Bouix, A. Tannenbaum, and S. Zucker. Hamilton-Jacobi skeletons. *IJCV*, 48(3):215–231, 2002.

[156] K. Siddiqi, S. Bouix, A. Tannenbaum, and S. W. Zucker. The hamilton-jacobi skeleton. In *Proc. of the International Conference on Computer Vision - Volume 2*, ICCV '99, pages 828–, Washington, DC, USA, 1999. IEEE Computer Society.

[157] A. Sitek, R. H. Huesman, and G. T. Gullberg. Tomographic reconstruction using an adaptive tetrahedral mesh defined by a point cloud. *IEEE Trans. Med. Imag.*, 25(9):1172–1179, 2006.

[158] A. Sobiecki, H. C. Yasan, A. Jalba, and A. Telea. Qualitative comparison of contraction-based curve skeletonization methods. In *Mathematical Morphology and Its Applications to Signal and Image Processing*, pages 425–439. Springer LNCS, 2013.

[159] A. Sobiecki, A. Jalba, and A. Telea. Comparison of curve and surface skeletonization methods for voxel shapes. *Pattern Recognition Letters*, 47:147–156, 2014.

[160] S. Sotoodeh. Hierarchical clustered outlier detection in laser scanner point clouds. *Intl. Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, 36(3/W52):383–388, 2007.

[161] R. Souvenir and R. Pless. Manifold clustering. In *Proc. ICCV*, pages 648–653, Beijing, China, 2005.

[162] M. B. M. Spernat and L. Kobbelt. Phong splatting. In *Proc. PBG*, pages 25–32, 2004.

[163] J. Stasko. An evaluation of space-filling information visualizations for depicting hierarchical structures. *Intl. J. Human-Computer Studies*, 53(5):663–694, 2000.

[164] S. Stolpner, S. Whitesides, and K. Siddiqi. Sampled medial loci and boundary differential geometry. In *Proc. IEEE 3DIM*, pages 87–95, 2009.

[165] S. Stolpner, S. Whitesides, and K. Siddiqi. Sampled medial loci for 3D shape representation. *CVIU*, 115(5):695–706, 2011.

[166] R. Strzodka and A. Telea. Generalized distance transforms and skeletons in graphics hardware. In *Proc. VisSym*, pages 221–230, 2004.

[167] A. Sud. *Efficient computation of discrete Voronoi diagram and homotopy-preserving simplified medial axis of a 3D polyhedron*. PhD thesis, UNC Chapel Hill, 2006.

[168] A. Sud, M. Foskey, and D. Manocha. Homotopy-preserving medial axis simplification. In *Proc. SPM*, pages 103–110, 2005.

[169] H. Sundar, D. Silver, N. Gagvani, and S. Dickinson. Skeleton based shape matching and retrieval. In *Proc. SMI*, pages 130–138, 2003.

[170] 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.

[171] S. Svensson. Reversible surface skeletons of 3D objects by iterative thinning of distance transforms. In *Proc. Discrete Geometry for Computer Imagery*, pages 400–411. Springer, 2001.

[172] A. Tagliasacchi, H. Zhang, and D. Cohen-Or. Curve skeleton extraction from incomplete point cloud. In *Proc. ACM SIGGRAPH*, pages 541–550, 2009.

[173] A. Tagliasacchi, M. Olson, H. Zhang, G. Hamarneh, and D. Cohen-Or. VASE: Volume-aware surface evolution for surface reconstruction from incomplete point clouds. *CGF*, 30(5):1563–1571, 2011.

[174] A. Tagliasacchi, I. Alhashim, M. Olson, and H. Zhang. Skeletonization by mean curvature flow. In *Proc. Symp. Geom. Proc.*, pages 342–350, 2012.

[175] R. Tam and W. Heidrich. Shape simplification based on the medial axis transform. In *Proc. IEEE Visualization*, pages 63–68, 2003.

[176] G. Taubin. Estimating the tensor of curvature of a surface from a polyhedral approximation. In *Proc. ICCV*, pages 902–907, 1995.

[177] A. Telea. Feature preserving smoothing of shapes using saliency skeletons. In *Visualization in Medicine and Life Sciences*, pages 155–172. Springer, 2012.

[178] A. Telea. GPU skeletonization code, 2012. www.cs.rug.nl/svcg/Shapes/CUDASkel.

[179] A. Telea and O. Ersoy. Image-based edge bundles: Simplified visualization of large graphs. *Comp. Graph. Forum*, 29(3):543–551, 2010.

[180] A. Telea and A. Jalba. Voxel-based assessment of printability of 3D shapes. In *Proc. ISMM*, pages 393–404. Springer LNCS, 2011.

[181] A. Telea and A. Jalba. Computing curve skeletons from medial surfaces of 3d shapes. In *Proc. Theory and Practice of Computer Graphics (TPCG)*, pages 224–232. Eurographics, 2012.

[182] A. Telea and A. Vilanova. A robust level-set algorithm for centerline extraction. In *Proc. Data Visualization (VisSym)*, pages 185–194, 2003.

[183] A. Telea and J. J. van Wijk. An augmented fast marching method for computing skeletons and centerlines. In *Proc. VisSym*, pages 251–259, 2002.

[184] J. B. Tenenbaum, V. de Silva, and J. C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, 2000.

[185] R. Torres and A. Falcão. Contour salience descriptors for effective image retrieval and analysis. *Image and Vision Computing*, 25(1): 3 – 13, 2007. ISSN 0262-8856. SIBGRAPI.

[186] R. Torres, A. Falcão, and L. Costa. A graph-based approach for multiscale shape analysis. *Pattern Recognition*, 37(6):1163 – 1174, 2004. ISSN 0031-3203.

[187] TranscenData. TranscenData Ltd., 2014. www.transcandata.com.

[188] R. Unnikrishnan and M. Hebert. Robust extraction of multiple structures from non-uniformly sampled data. In *Proc. IROS*, pages 1322–1329, 2003.

[189] R. Unnikrishnan and M. Hebert. Denoising manifold and non-manifold point clouds. In *Proc. BMVC*, 2007.

[190] V. Verma and J. Snoeyink. Reducing the memory required to find a geodesic shortest path on a large mesh. In *Proc. ACM GIS*, pages 227–235, 2009.

[191] R. Vidal, Y. Ma, and S. Sastry. Generalized Principal Component Analysis. *IEEE TPAMI*, 27(12):1945–1959, 2005.

[192] M. Wan, F. Dachille, and A. Kaufman. Distance-field based skeletons for virtual navigation. In *Proc. IEEE Visualization*, pages 239–246, 2001.

[193] J. Wang, M. Oliveira, and A. Kaufman. Reconstructing manifold and non-manifold surfaces from point clouds. In *Proc. SMA*, pages 139–147, 2007.

[194] C. Weber, S. Hahmann, and H. Hagen. Sharp feature detection in point clouds. In *Proc. SMA*, pages 175–186, 2010.

[195] H. Xie, J. Wang, J. Hua, H. Qin, and A. Kaufman. Piecewise $\rfloor^1$ continuous surface reconstruction of noisy point cloud via local implicit quadric regression. In *Proc. IEEE Visualization*, pages 198–206, 2003.

[196] J. Xie, P. Heng, and M. Shah. Shape matching and modeling using skeletal context. *Pattern Recognition*, 41:1756–1767, 2008.

[197] M. Zhao, L. Ma, W. Tan, and D. Nie. Interactive tooth segmentation of dental models. In *Proc. EMBS*, pages 654–657, 2005.

[198] H. Zhou, X. Yuan, W. Cui, H. Qu, and B. Chen. Energy-Based Hierarchical Edge Clustering of Graphs. In *Proc. PacificVis*, pages 55–62, 2008.

[199] M. van der Zwan, Y. Meiburg, and A. Telea. A dense medial descriptor for image analysis. In *Proc. VISAPP*, pages 285–293, 2013.

LIST OF PUBLICATIONS

The following publications resulted from the work presented in this thesis:

1. 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

2. J. Kustra, A. Jalba, and A. Telea. Probabilistic View-based Curve Skeleton Computation on the GPU. In *8th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications, VISAPP 13*, 2013

3. J. Kustra, A. Jalba, and A. Telea. Robust segmentation of multiple intersecting manifolds from unoriented noisy point clouds. *CGF*, 33(1):73–87, 2014

4. J. Kustra, A. Jalba, and A. Telea. Shape segmentation using medial point clouds with applications to dental cast analysis. In *Proc. VISAPP*, pages 151–159, 2014

5. J. Kustra, M. de Jager, A. Jalba, and A. Telea. Teeth shape modeling pipeline for oral healthcare appliances development. In *Proc. ICCE*. IEEE, 2014

6. J. Kustra, M. de Jager, A. Jalba, and A. Telea. A medial point cloud based algorithm for dental cast segmentation. In *Proc. ICCE*. IEEE, 2014

7. J. Kustra, A. Jalba, and A. Telea. Computing refined skeletal features from medial point clouds. *Pattern Recognition Letters (Submitted)*, 2014

ACKNOWLEDGEMENTS

One of the qualities which I believe is unique to humans, is the ability to dream. It is probably this ability which propels most mankind great achievements. Education, as process of discovery, is a key catalizer of dreams, expanding the individual minds to wonder further than before.

It would therefore seem obvious any system relying on human power, would have education, and consequently free thinking, as one of its most basic assets. However, as the story of the 2014 Nobel Prize winner showed us, education is not always an intrinsic value but a continuous endeavor with internal and external factors. On my personal level, the process of learning and discovery is a vital experience - at all levels worth aspiring and striving for.

During the endeavor to pursue this PhD, several people have enabled, guided and supported me through these internal and external struggles, contributing to the realization of this ambitious goal.

First and foremost, my supervisor, Alex Telea. Your sharp knowledge on such a broad range of subjects have been a unique source of inspiration for me. You are a mentor and an outstanding leader. You have demonstrated firsthand how to persistently pursue ideas and transform ideas into tangible concepts. Your guidance has enabled me excel myself and to reach several milestones in my life which I had longed for. For everything, I am uniquely grateful.

My co-supervisor, Andrei Jalba: Andrei, your numerical problem solving skills are amazing! It was a great experience to have the opportunity to discuss and work together towards several of the results in this thesis. Your support in this context was fantastic - Thank you so much for it!

The committee members, Prof. A. Falcao, Prof. F. Leymarie and Prof. W. Hesselink, for their very detailed reviews. Your detailed comments and remarks identified several points which have contributed to several improvements in the thesis.

My group leaders who followed me during this track. Peter Stolk, thank you for trusting and enabling me to pursue this goal, especially in a time when this decision was truly a brave one! Guido Volleberg, thank you for all your continous support, trust and recognition! I greatly appreciated your leadership and working together - This strongly contributed to this achievement!

My family. The support I always found on your side to learn and pursue education and interests strongly contibuted to shaping the person I aspired to be. Knowing that you unconditionally are always there for me, is absolutely priceless, for which I am extremely grateful.

My friends. Sharing great, memorable and, (depending on the perception) crazy moments have a rationally unexplainable importance for experiencing life!

My colleagues, with whom I shared interesting discussions and hard working sessions. I appreciate the work, all we learned from each other and most of all, seeing the results of our commitment coming together into meaningful applications.

Finally, my partner in life, Mahsa. Being together brings meaning and hapiness to so many seemingly trivial details. Any obstacle becomes minuscule when compared to the significance of our life. During this adventure, you were the one and only who daily accompanied my joys, disappointments and thrills. Thank you for dreaming with me, let's keep on making our dreams come true!

ABOUT THE AUTHOR

Jacek Kustra was born in Czestochowa, Poland in 1981. He is a Scientist, currently in Philips Research, where he has focused on a wide spectrum of technologies, from Electronics design to advanced algorithm research. His current focus is on innovative healthcare systems for oncology based applications. He followed this PhD track independently and in parallel to his full time job.

The strong interest in science and technology, specifically in electronics and software and algorithms has led him since a very early age to start developing his own software and to actively pursue several professional activities related to software development. This interest, in combination with his adventurous spirit and multicultural background (having been born in Poland, raised in Portugal, working in several countries - and in love with a Persian), provide him with the opportunity to be fluent in several cultures and languages with a strong awareness of global issues. He is eager to actively contribute to solutions with his creative thinking and problem solving skills.

His academic background includes a 5 year degree in Electronics and Telecommunications Engineering in the University of Aveiro in Portugal, where he expanded his knowledge in a variety of fields, ranging from analog and digital electronics, signal and image processing, control systems, ending with a specialization in Medical Electronics and Systems, specifically in medical image reconstruction and visualization. During these studies, he was also a professional Volleyball player in the first portuguese league and worked as a software freelancer for the local Portuguese industry. By following his education with a post-graduate degree in Biomedical Engineering in the University of Aveiro, he brought together his passion for problem solving and technology together with his ambition to contribute to the improvement of people's lives. During these years, he was a lecturer on computer science courses and focused his research on Electroencephalography and functional Magnetic Resonance Imaging applications, contributing with several novel algorithmic approaches.

The eagerness to continuously learn, solve novel problems and apply his skills to meaningful applications, led him to his current professional position and to pursue the personaly very rewarding experience of this PhD degree.