university of groningen

faculty of mathematics and natural sciences

MASTER'S THESIS

Shape Segmentation for Mesh-Based 3D Models: Unifying Part- and Patch-type Approaches

Joost Koehoorn

first supervisor	Prof. dr. A.C. Telea
second supervisor	dr. J.L. Kustra *Philips Research*

November 2016

Computational Science & Visualization
Mathematics and Natural Sciences
University of Groningen

Joost Koehoorn: *Shape Segmentation for Mesh-Based 3D Models*, Unifying Part- and Patch-type Approaches, © 2016

SUPERVISORS:
Prof. dr. Alexandru C. Telea
dr. Jacek L. Kustra, *Philips Research*

LOCATION:
Groningen

November 2016

*“The only way to do great work
is to love what you do.”*

— Steve Jobs

ABSTRACT

Shape segmentation algorithms can be categorized into two main categories, namely part-based segmentation and patch-based segmentation. The former identifies the structural parts of a shape, much like a human would visually classify parts, and is most suitable for natural shapes, whereas patch-based segmentation concerns itself with partitioning a shape into its quasi-flat areas which best suits man-made objects. This dichotomy poses an issue with being able to automatically segment any input shape, as it is unknown which method to use.

Our contribution is two-fold: we first present a new part-based segmentation method to operate on mesh shapes, derived from cut-space segmentation as proposed by Feng *et al* in 2015. Secondly, we propose a heuristic to automatically select what segmentation is most appropriate without introducing undesirable over-segmentation. To accomplish this we first validate each part of the part-based segmentation based on several structural properties a part must satisfy, then determine for each remaining parts what patches should be introduced.

We tested our approach on over fifty various kinds of shapes and obtained promising results. We show some of the results and also discuss shapes that are problematic with our algorithm. The proposed method and results have been accepted for publication as a book chapter.

SAMENVATTING

Het segmenteren van vormen kan worden onderverdeeld in twee categorieën, namelijk onderdeel-gebaseerd (*part-based*) en vlak-gebaseerd (*patch-based*). De eerste variant herkent de verschillende structurele onderdelen van een vorm, gelijk aan hoe mensen dat perceptueel zouden doen, en is toepasselijk voor natuurlijke vormen. De *patch-based* variant daarintegen herkent de platte vlakken van een vorm, een eigenschap die veel voorkomt bij door mensen gemaakte objecten. Deze tweedeling vormt een probleem voor het automatiseren van een arbitraire vorm, aangezien het onbekend is om wat voor type vorm het gaat.

In deze thesis presenteren we allereerst een nieuwe methode voor *part-based* segmentatie op meshes, op basis van een *cut-space* techniek, in 2015 geïntroduceert door Feng *et al.* Vervolgens tonen we een methode om de twee verschillende vormen van segmentatie samen te brengen, zodat automatisch de best passende segmentatie wordt berekend. Dit is bereikt door eerst voor de gevonden onderdelen te verifiëren of aan bepaalde eigenschappen wordt gedaan, waarna wordt bepaald welke van de vlakken in het resultaat moeten worden betrokken.

We hebben onze aanpak getest op meer dan vijftig uiteenlopende vormen en hierbij goede resultaten gezien. Een aantal van de resultaten worden gepresenteerd en besproken, waarbij ook een aantal vormen worden getoond die problematisch blijken. Onze methode en de resultaten ervan zijn geaccepteerd voor publicatie als onderdeel van een boek.

ACKNOWLEDGMENTS

Hereby I would like to take the opportunity to thank the people that have helped me complete this work. First of all, I am sincerely grateful for the countless hours of support and discussions I had with my supervisors dr. Jacek Kustra and dr. Alex Telea. Their insights and guidance have helped tremendously during this research project.

Furthermore, I would like to express my gratitude to my parents, family, and friends, to whom I could always turn to. Your continued belief and support means a lot to me.

DANKWOORD

Hierbij wil ik van de gelegenheid gebruik maken om de mensen te bedanken die hebben bijgedragen aan het realiseren van deze thesis. Allereerst ben ik dankbaar voor de vele uren aan ondersteuning en discussies met dr. Jacek Kustra en dr. Alex Telea. Hun inzichten en leidraad hebben enorm bijgedragen aan het afronden van dit onderzoek.

Daarnaast wil ik graag mijn ouders, familie, en vrienden bedanken, welke altijd voor mij klaar hebben gestaan. Jullie oponhoudelijke geloof en steun betekent veel voor mij.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

MA	Medial Axis
DT	Distance Transform
FT	Feature Transform
MGF	Medial Geodesic Function
SSG	Shortest Straightest Geodesic
SDF	Shape Diameter Function
MDS	Multi-Dimensional Scaling

1 INTRODUCTION

The human ability to identify, compare and reason about shapes is exceptionally sophisticated. Everything we see around us can be described in terms of shapes, for which we immediately have a feeling of their topological structure and resemblance to other shapes, only from perceiving their surface.

This ability comes as natural to us, however it is far from trivial to teach a computer the same abilities. To a computer, a shape is nothing more than a set of values, structured in a certain manner such that we may reconstruct the visual representation from it using computer graphics techniques. To better understand how hard a computer's task of understanding and, more importantly, reasoning about such a set of values is, imagine being given a page only consisting of numbers and then having to tell what object it describes, if any.

For a computer to do something meaningful with a set of values, it has to be aware of how such a set of values is used to represent a certain shape. We will next see what *shape representations* are commonly used in computers.

1.1 SHAPE REPRESENTATIONS

Several possibilities exist for the representation of a shape. First, we can make the distinction between *implicit* and *explicit* shape representations. The first category uses analytical expressions to implicitly describe a shape from a set of parameters [BBB+97]. As an example, a sphere in three-dimensional space is obtained by the expression $(x-x_0)^2+(y-y_0)^2+(z-z_0)^2 = r^2$ where (x_0, y_0, z_0) and r are its parameters to define the sphere's center and radius, respectively. A benefit of this representation is that the shape is defined with infinite precision and with little memory requirements, however it is hardly used in practice as composing complex shapes implicitly is a hard problem in itself. Moreover, such representations are impractical for many processing purposes for the very reason the shape itself is only implicitly known.

In the explicit category we can further distinguish *boundary* and *volumetric* representations. The first is commonly used when we are only interested in the boundary of a shape, or when data on the inside of the shape is unavailable. In computer graphics applications this is the prevalent representation, in which a point-sampling of the surface is used where the points are connected by triangles, generally referred to as a polygonal mesh [BKP+10]. Without the connectivity information we refer to such representation as an unstructured point cloud, optionally with orientation information when a surface normal is associated with each point.

Volumetric representations contain information of both the surface and shape interior. Typically, a rectilinear grid is used to store a uniform sampling of a multi-dimensional space, storing for each grid cell if it is inside or outside the shape [BBB+97]. Optionally, a grid cell may be assigned additional values to indicate all kinds of extra features. Grid cells are referred to as *pixels* for two-dimensional images, and *voxels* in three-dimensional volumes. From now on we will use the term voxel for grid cells as we refer to them in the context of three-dimensional shapes. Some processing tasks are easier to accomplish using this representation compared to point clouds and meshes, mostly because a voxel's neighborhood is uniform and always well-defined—a pixel's 4 or 8-connected neighborhood *versus* a voxel's 6 or 26-connected neighborhood—which is useful in many shape processing applications. Major drawbacks exist however, in terms of accuracy and memory requirements. The accuracy is inherently limited by the grid's resolution, as values can only be assigned per voxel which are of a certain size. Consequently, to represent fine levels of detail we need very high resolution grids, which then become unfeasible in terms of processing power and memory requirements.

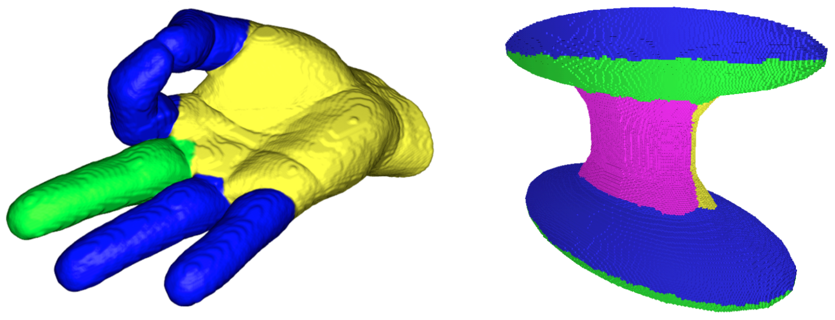
Point clouds and meshes do not suffer from such issues as each point can be positioned anywhere up to machine precision. Moreover, these representations have an additional advantage in terms of processing power over the volumetric grid-based approach, as they require much less data to represent the same information. As an alternative shape representation to the ones discussed above we will next look into the Medial Axis and its applications.

1.2 SKELETONS

An alternative way to represent a shape is by only describing its main structure, or *skeleton* [Blu67]. This representation differs from the methods discussed above, in that it does not store boundary points explicitly but instead captures a shape's boundary by means of its topology and distances to the boundary. The skeleton of two-dimensional shapes consists of 1D curves, whereas three-dimensional shapes produce a collection of 2D manifolds, referred to as a *surface skeleton*. Considering the complexities that come with processing a structure of 2D manifolds researchers have come up with methods to extract 1D *curve skeletons* from three-dimensional shapes, thereby vastly reducing the skeleton's complexity. This family of skeletons is often used in shape analysis tasks as it succinctly describes the topology of a shape [TDS+16]. Due to the reduction in complexity however, the original shape cannot be retrieved in full from a curve skeleton. Moreover, the computation of curve skeletons allows for various interpretations, as no universally accepted definition exists.

1.3 SHAPE SEGMENTATION

One interesting task for a computer is to extract the individual components of a given shape, as such segmentations have a broad array of applications ranging from shape analysis and computer vision to compression and efficient collision detection. In 2004, Shamir introduced the distinction between two kinds of segmentations, part-based and patch-based segmentations [Sha04]. *Part-based* segmentation methods try to identify segments that a human would also intuitively perceive as the distinct segments of an object, of which an example is shown in Figure 1.1a. Such segmentations have applications *e.g.* in shape matching, where individual parts can be better matched to other objects than the whole shape, and object morphing, in which each part can be transformed individually from the rest of the shape.



(a) Part-based segmentation [RT08a] (b) Patch-based segmentation [RT08b]

FIGURE 1.1 Categorization of segmentation methods, showing the results for two volumetric datasets.

On the other hand, *patch-based* segmentations recognize quasi-flat areas, separated by high curvature creases, depicted in Figure 1.1b. Extracting such areas is useful for mesh simplification applications, as a patch is quasi-flat it can be represented using a relatively small number of triangles. Stemming from this is the possibility to use patches to optimize collision detection algorithms, by first performing a collision test using a patch as the bounding box of a larger number of triangles [GWH01]. If the bounding box itself tests negatively for a collision it is known that neither of its included triangles will collide, therefore lifting the need to continue collision testing on a finer scale. Strictly speaking this is true for any subset of the mesh, however patches lend themselves well for this application as a patch generally comprises a small volume—due to its quasi-flat nature—therefore being more likely to test negatively in collision tests, providing a better means of optimization compared to *e.g.* a part that is used as bounding box.

Part-based Segmentations

Computing *part*-based segmentations can be done using a multitude of methods. For instance, one may identify the various parts in a shape by calculating the curve skeleton and analyzing its junctions, as a subset of these corresponds with the distinct parts of the shape [RT07].

Given that curve skeletons are not a complete representation of a surface, they may not be appropriate for any kind of shape and shape processing operation. Therefore, an attempt has been made to derive a part-based segmentation from the surface skeleton [FJT15b]. For each skeleton point, a shortest geodesic around the boundary connecting both feature points is calculated to obtain the shape's *cut-space*, after which the skeleton points are categorized based on their geodesic's length. Finally, this is used to derive shape parts from, where the geodesics are used as smooth segment borders. Feng's results have shown that such segmentation approach derived from the surface skeleton is indeed feasible for part-based segmentations, however its use of a volumetric voxel representation limits its usability and scalability.

Patch-based Segmentations

For *patch*-based approaches, shape surface curvature may be computed to identify creases and extract patches from that information. Such methods however are highly sensitive to noise which limits their usability, and furthermore may not have the desirable property of being pose-invariant. An alternative technique for detecting creases is by using the surface skeleton of a shape, of which some skeleton points correspond with the high curvature ridges [RT08b, KJT16] Simply put, by detecting these points and back-projecting them against the surface, we have marked every crease and may obtain the patches by connected component analysis. Using the surface skeleton is advantageous as skeleton points can be assigned an importance measure resulting in a multiscale skeleton. Such multiscale skeletons can then be used at a certain scale such that details which are due to noise are ignored, giving an effective way to work around noise.

1.4 RESEARCH GOALS

The main goal of this research is to investigate how part- and patch-based segmentation approaches for three-dimensional shapes may be unified into a single combination of both methods. Such a method may have the benefit of producing good results for all kinds of shapes, where the current segmentation methods only produce expected results for certain kinds of shapes. Therefore, the **first research question** is as follows:

1. *How can we design a generalized part/patch-based segmentation algorithm?*

To avoid the inevitable downsides of volumetric methods—their limited resolution and high memory consumption—the whole processing pipeline must only use point cloud data or polygonal meshes. As such, in order to experiment with how cut-space segmentation can be applied for mesh shapes we formulate the **second research question**:

2. *Can we use cut-space segmentation for mesh models and how does it compare to Feng* et al*'s voxel-based technique? [FJT15b]*

To answer these questions, we will look into existing shape segmentation algorithms and how we may define a heuristic to unify them. Also, we will look into redesigning the cut-space derived part-based segmentation method [FJT15b] to work with meshes and see how it compares to other methods. This has several challenges, as a shape's segmentation is not well-defined and its preferred segmentation may depend on the application at hand.

1.5 THESIS OUTLINE

In this section we briefly discuss the structure of this thesis, as follows.

First, Chapter 2 gives an overview of related work, starting with discussing the Medial Axis in its various forms and its applications thereof. Next, we look into a broad range of segmentation approaches from earlier work.

Next, Chapter 3 shows how we re-designed Feng *et al*'s cut-space segmentation approach to work with 3D meshes, reviewing all deviations from the original method that are necessary to obtain proper results. The chapter is concluded with validation procedures and ways of visually improving the segmentation results.

In Chapter 4 we go into detail about the desirable properties of a unifying part- and patch-based segmentation method. We show how an existing patch-based segmentation method is adapted to integrate in a unification framework.

Having described our unified segmentation pipeline in full, Chapter 5 switches to a more practical context. We give details about the technologies that were used along with implementation details.

Chapter 6 continues with showing the results and limitations for various kinds of shapes, also in comparison to related work. Furthermore, we review common values for some of the parameters involved, and show how they may affect the results.

Finally, Chapter 7 presents a discussion on the research's conclusions alongside possible directions for future work.

2 RELATED WORK

In this chapter we first go into detail about skeletons as they play a central role throughout this thesis. Section 2.1 gives the mathematical definition and skeleton notation, followed by a skeleton classification scheme. Then, in Section 2.2 a non-exhaustive overview of related work in shape segmentation is presented.

2.1 MEDIAL AXIS

Another name for the skeleton of a shape is *Medial Axis* (MA), introduced in 1967 by Blum [Blu67]. This representation does not store boundary points explicitly but captures a shape's boundary by means of maximally inscribed disks (in $\mathbb{R}^2$) or spheres (in $\mathbb{R}^3$), such that the union of all such spheres produces the shape boundary. Blum observed how most points inside of a shape have a unique minimal distance to the boundary, *i.e.* there is only a single closest point. For points $\mathbf{x}_i$ on the MA, however, there exist at least two such points at equal distance $\rho_i \in \mathbb{R}$. Therefore, the MA is a shape descriptor that captures the symmetry in a shape, by means of the set of spheres $\{(\mathbf{x}_i, \rho_i)\}$.

As is briefly mentioned in the introduction, the skeleton of two-dimensional shapes consists of a set of 1D curves. By using the same definition for three-dimensional shapes, a set of 2D manifolds is obtained. Generally speaking, the skeleton of an n-dimensional shape is of one dimension less with respect to the dimension of the shape. Considering the complexities that come with processing a structure of 2D manifolds however, researchers have come up with methods to extract 1D *curve skeletons* from three-dimensional shapes, thereby vastly reducing the skeleton's complexity. Due to this reduction in complexity not all shape information is retained; hence only an approximation of the original shape can be reconstructed. Curve skeletons lend themselves well as a topological descriptor of a shape and are therefore used in many shape processing tasks, *e.g.* segmentation, animation and medical imaging tasks such as MRI and CT scans [CSM07].

2.1.1 *Definitions*

Given a shape $\Omega \in \mathbb{R}^{n\in\{2,3\}}$ with boundary $\partial\Omega$ we first define the *distance transform* (DT) $DT_{\partial\Omega} : \Omega \to \mathbb{R}^+$ that determines the distance to the closest boundary point for any given point in the shape:

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

The notation $\|\cdot\|$ stands for Euclidean distance and this particular DT is therefore called *Euclidean distance transform*. We next use the DT in the definition of a skeleton $S_{\partial\Omega}$:

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

Intuitively, $\mathbf{f}_1$ and $\mathbf{f}_2$ are two of the contact points with $\partial\Omega$ of the maximally inscribed sphere in Ω centered around $\mathbf{x}$ which are known as *feature transform* (FT) points [ST04]. The feature transform itself thus assigns to any point in Ω its closest points on $\partial\Omega$, defined as:

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

2.1.2 *Skeleton Classification*

In order to understand the structure of a skeleton and aiding the reasoning about skeletal points, Giblin and Kimia introduced a classification scheme based on the order of contact of maximally inscribed spheres centered at the skeletal points with the shape boundary $\partial\Omega$ [GK04]. They denote a medial point as A_k^n, with k indicating the aforementioned contact order and n the number of different k-fold tangencies.

The medial axis consists primarily of several manifolds, except for spheres or tubular shapes. Such manifolds are commonly referred to as medial sheets and consist of only A_1^3 points, *i.e.* the points having exactly two feature points. On the boundaries of these manifolds we find medial curves, of which we may differentiate between two types. First we can identify A_1^3 points, which each have exactly three distinct feature points on $\partial\Omega$. Such curves occur at the intersection of three medial sheets and together form the so-called Y-intersection curve, or Y-network. The second type of medial curves are A_3 points, which only have a single contact point and are located on the 'open' sides of medial sheets, thus represent the skeleton boundary. Such points map to the surface curvature maxima of $\partial\Omega$, which we will see later is key in computing a patch-based segmentation.

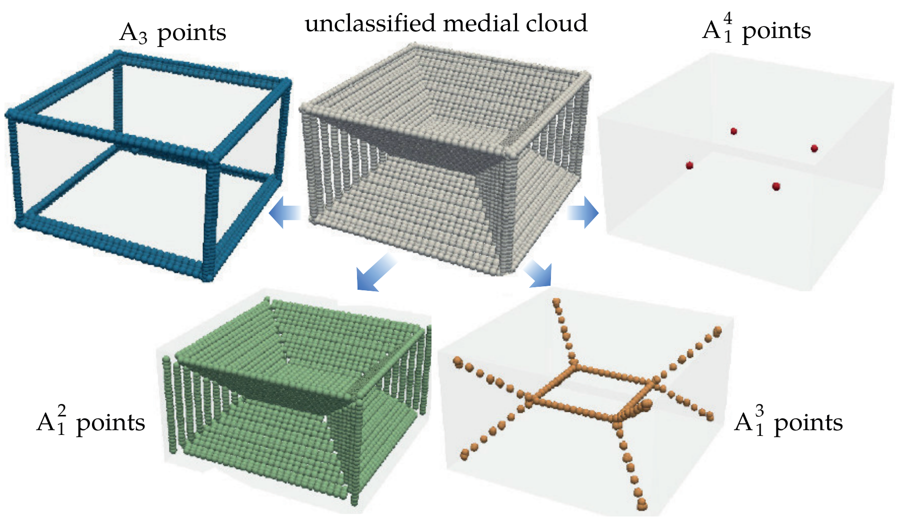


FIGURE 2.1 Classification of skeleton points [KJT16].

Finally, the end points of curves may be individually categorized in two classes. The end points of A_1^3 curves are the A_1^4 points each having four contact points on $\partial\Omega$. Such points correspond with the *internal* corners of the skeleton manifolds, *i.e.* the points along the boundary-curve of such a manifold. Lastly, we may identify A_1A_3 points which are the end points of A_3 curves, or the intersection of A_3 and A_1^3 curves. These points correspond with corners on $\partial\Omega$ where multiple surface edges meet and therefore may be considered *external* corners.

2.2 SHAPE SEGMENTATION

Segmenting a shape $\Omega \in \mathbb{R}^3$ means to obtain a partition of disjunct components C_i on the surface, such that $\cup_i C_i = \partial\Omega$ and $\forall i \neq j$: $C_i \cap C_j = \emptyset$. These definitions hold true for both part-type and patch-type segmentations, the difference between the two types being the choice of what properties a segment should have, *i.e.* where segment borders are to be inserted.

When computing the segmentation of a shape several properties are important for a good segmentation result. For example, the segmentation should be robust to noise, *i.e.* introducing boundary noise should not influence the resulting segmentation. Furthermore, the segmentation should be pose, scale and rotation invariant such that the segmentation stays the same as long as the shape's topology is not altered. Lastly, the segmentation's level of detail should be customizable by the end user, if so desired.

In the following subsections both part-type and patch-type shape segmentation approaches are discussed and an overview of various methods is provided. Please note here that although it is common in literature to interchangeably refer to a single unit in a segmentation as both *segments* and *parts*, given the context of this research however this would cause ambiguity as we discriminate between part-segments and patch-segments. Therefore, throughout this thesis we use *part* to mean a segment as a result of part-type segmentation, and *patch* to indicate a segment from patch-type segmentation. When the term *segment* is used, the statement is not restricted to a single type of segmentation but applies to both part-type and patch-type segmentation.

2.2.1 *Part-type Segmentation*

Part-type segmentation is most appropriate for natural shapes such as animals, and has the goal of finding how the shape could be split up into separate parts, corresponding with what the human brain would typically perceive as a part. This directly leads to a challenge in that what exactly constitutes a part may be subjective and is therefore ill-defined, although a part is typically considered to consist of an elongated structure that sticks out of a shape's rump, to represent a part-whole relation. Furthermore, one aspect of parts is that a part is separated from the rump by a region of negative curvature, which is known as the *minima rule* from [BHS89, HR84]. We will next discuss a non-exhaustive list of methods for computing a part-based segmentation.

Plane Sweeping

In 2001, Li *et al* proposed a method that sweeps the curve skeleton of a mesh using a plane, then identify large variations in the intersecting area of the plane [LWTH01], as illustrated in Figure 2.2. Such variations are likely to represent part-whole transitions, and therefore may be used to derive segment borders from by computing the third order derivative. Unfortunately only a limited set of results is presented in the paper—which arguably suffer from slight over-segmentation—so we cannot accurately describe the characteristics of this method. Moreover, given that the paper is over a decade old, its performance metrics cannot accurately be compared with more recent work.

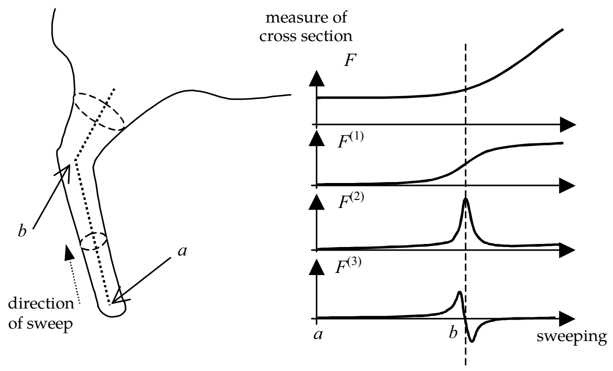


FIGURE 2.2 Plane sweeping may be used to detect parts [LWTH01]. The graphs show up to the third order derivatives from which the transition points (here b) are derived.

Fuzzy Clustering

In order to overcome jaggy boundaries in the segmentation, Katz and Tal proposed a fuzzy clustering approach for mesh shapes in 2003 [KT03]. Fuzzy in this regards means that the shape boundary is assigned with probabilities of belonging to a certain segment, then iteratively optimizing the fuzzy areas to obtain a desirable segment boundary that properly follows the shape's curves. Furthermore, their segmentation is obtained from a hierarchical clustering of the shape. This results in a multiscale segmentation such that the number of segments may be refined by selecting a different clustering level.

Their results show promising results, notably the multiscale processing produce logical segmentations with segment borders corresponding closely with the shape's curvature. In terms of performance this method was quite expensive, where shape simplification and voxelization was applied for some stages of the pipeline in order to accelerate the computation.

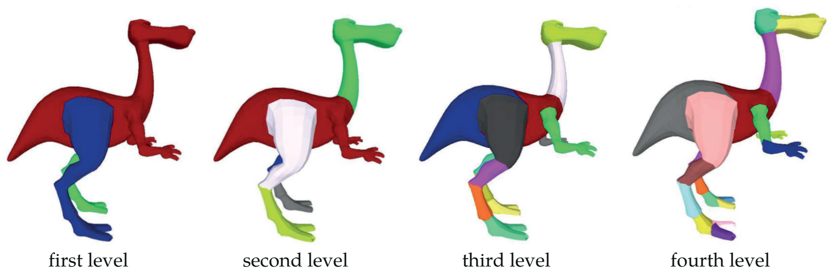


FIGURE 2.3 Part-type segmentation using fuzzy clustering [KT03], showcasing its ability to compute a sensible multiscale segmentation.

The idea of assigning segmentation labels in a fuzzy manner has seen additional usages, *i.e.* an interactive shape editing tool that computes feature classifiers on the shape boundary [CGR+04] and the method by Katz *et al* as discussing next.

Feature Point & Core Extraction

Only two years after fuzzy clustering was proposed, Katz *et al* presented a new method for mesh-based part-type segmentation by means of *Feature Point & Core Extraction* [KLT05]. Even though this method still uses some sort of fuzzy clustering, its initial computation of parts differs greatly from the original fuzzy clustering algorithm. By applying a *Multi-Dimensional Scaling* (MDS) technique a pose-invariant representation is obtained, from which *prominent feature points* are computed, see Figure 2.4. In this context, feature points are not to be confused with Medial Axis feature points but simply represent points of particular interest, each corresponding with a part in the final segmentation. A novel technique for extracting the core component is presented which is used to determine the segments from.

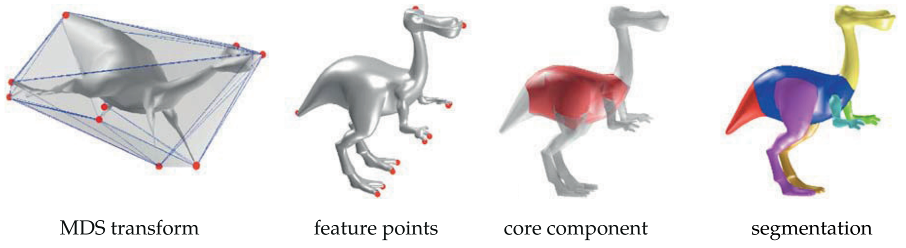


FIGURE 2.4 Part-type segmentation using feature point & core extraction [KLT05].

The paper shows good looking results on a set of natural shapes, *i.e.* the kind of shapes for which part-type based segmentation works well. It shows improved results upon their 2003 method in terms of pose-invariance and especially proportion-invariance. Running times of the full pipeline vary from sub-minute figures for small and medium sized shapes and increases to several minutes for larger models.

Reeb Graphs

Tierny *et al* use Reeb graphs as topological descriptor of a mesh shape, from which a hierarchical segmentation is computed [TVD07]. Given that the Reeb graph is a topology descriptor much like curve skeletons, it can be categorized in the same category. Surface parts are also derived from the graph's junctions and through simplification of the graph, a multiscale segmentation is computed as shown in Figure 2.5. The paper shows that this method provides good results that are pose-invariant and robust to noise. In terms of performance this method performs well, with timing figures ranging from seconds to a few minutes.

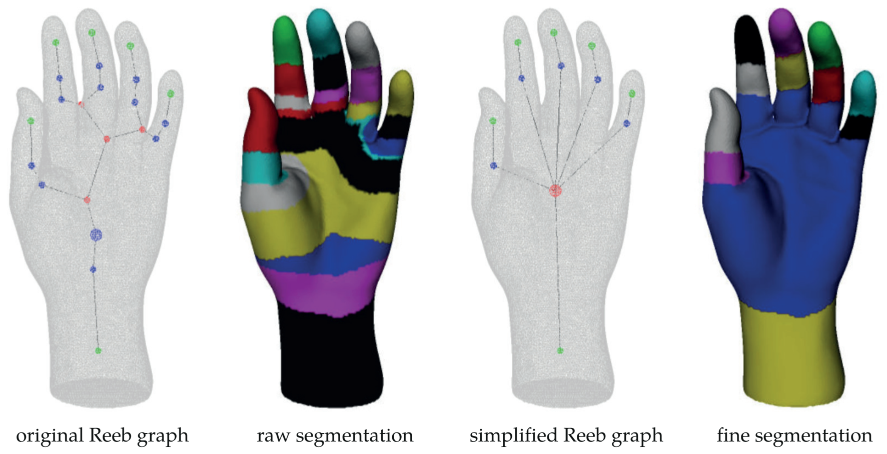


FIGURE 2.5 Part-type segmentation using Reeb graphs [TVD07]. By simplification of the Reeb graph a multiscale segmentation is obtained.

Skeleton Junction Analysis

As was briefly discussed in the introduction, part-type segmentation may be obtained from analyzing the junctions of a curve skeleton. First proposed in 2007, Reniers *et al* partition a shape into parts by finding all junctions in the curve skeleton, then computing Jordan curves on the boundary that serve as segment borders [RT07]. The idea here is that a junction in the curve skeleton corresponds with a protruding feature on the boundary, that is to be considered a part.

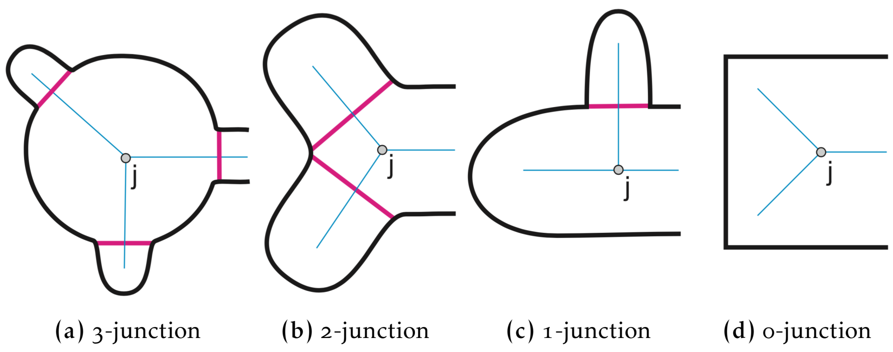


FIGURE 2.6 Overview of all four junction types, where magenta represent desirable part-cuts and blue the curve skeleton. Figure from [RT08a].

The method described in [RT07] however suffers from over-segmentation, as for certain junctions not all connected components as induced by the Jordan curves should be considered as a separate part. Hence, in 2008 Reniers *et al* proposed a modified algorithm that was able to circumvent this problem [RT08a]. By categorizing each junction into four types, as shown in Figure 2.6, the number of parts that stem from a junction is known, which can then be used to obtain a correct segmentation from.

A further refinement to this idea was proposed by Serino *et al* in 2011 [SdBA11] In a voxel-based curve skeleton they detect three kinds of skeletal parts, *i.e.* simple curves, complex sets and single points, which are projected onto the shape surface to partition it into parts. This part-based segmentation contains of a main rump, called *kernel* with attached to it are its protruding parts, called *simple regions* and *bumps*. By recognizing the shape from a single rump, their approach suffers less from oversegmentation and may produce better part boundaries. In 2014, Serino *et al* present an elegant solution to the problem of having too many skeleton junctions, and therefore identifying too many parts [SAdB14]. By comparing the local shape thickness, *i.e.* distance transform value, against the inter-junction distances they detect spurious junctions, using which they determine which junctions to actually derive segment from.

Shape Diameter Function

Shapira *et al* take a different approach to segmenting a mesh, where they segment a shape directly from its boundary [SSC08]. In 2008, they introduced the *Shape Diameter Function* (SDF) which is an estimation of the shape's diameter at every boundary point. For each vertex of the mesh several rays are projected into the shape in various directions, according to a cone-like pattern around the inversed normal, and for each ray the first intersection with the shape boundary is computed to obtain the length of the ray. Next, the SDF is computed as being the average of all ray lengths. To become pose-invariant and robust to noise, the SDF gets smoothed by applying a small number of bilateral filtering iterations on a small neighborhood around each vertex.

Next, a segmentation of the shape is obtained by clustering the vertices having similar SDF values. Under the assumption that areas of similar diameter are likely to represent a certain part, much like the assumption used with the cut-space approach, this yields a part-based segmentation of the shape. This method gives good results for natural shapes as shown in the paper, but has limitations on non-cylindrical parts of objects. In terms of performance the method performs similar to earlier discussed methods, ranging from seconds for small shapes to several minutes for large meshes.

Randomized Cuts

An entirely different approach for part-type segmentation is presented by Golovinskiy in 2008 [GF08]. From the observation that segmentations may differ greatly between algorithms and parameter settings, although still often producing segment borders in the same edges they see opportunity in sampling multiple randomized segmentations so that the *most probable* segmentation may be derived. Initially they compute multiple segmentations using existing algorithms such as k-means clustering, then compute a partition function that outputs the probability that an edge represents a segment border.

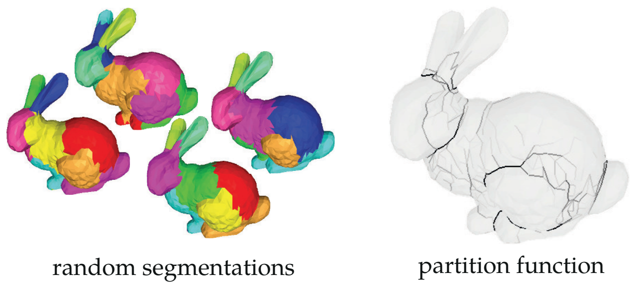


FIGURE 2.7 Part-type segmentation using randomized cuts [GF08]. The partition function shows the likelihood of representing a segment border as computed from the four segmentations in the left figure.

The paper shows that on average, 400 random segmentations are necessary to obtain a stable partition function. Considering that shape segmentation becomes more expensive as the number of mesh faces increases, this method is quickly restricted by shape size. For small shapes with only 4,000 faces the full pipeline already takes several minutes to complete, which is much slower compared to earlier methods. Although the results are visually similar to *e.g.* SDF segmented shapes, the low resolution and therefore rough segment borders diminish the results.

Cut-Space Segmentation

Where most earlier methods rely on using a topological descriptor of a shape, *e.g.* the curve skeleton, the recently proposed method by Feng *et al* is designed to derive a part-type segmentation from the surface skeleton [FJT15b]. This is beneficial, as surface skeletons capture the full details of a shape in contrast to curve skeletons, such that any amount of detail may potentially be taken into account which is impossible when using curve skeletons. For each surface skeleton point, they compute the shortest path (geodesic) all across the surface between the two feature points. These paths may all be considered as *cuts*, *i.e.* places where segment borders may occur, hence the collection of all cuts is called a *cut-space*. The cut-space segmentation approach in this regard is similar to plane sweeping 2.2.1 but uses a different model for computing plane intersections, *i.e.* by means of shortest geodesics.

Feng *et al* next derive parts from the cut-space by grouping similar cuts together. Initially this was done by partitioning cut-lengths [FJT15b] and later improved by using hierarchical clustering of cuts based on cut-length and spatial features [FJT15a]. Having computed a partition of the cut-space, the shortest cut among the set of cuts having a neighboring cut in a different partition, is used as segment border in the segmentation of the shape surface.

Their results show that using a surface skeleton is a viable option for part-type segmentation. Considering however that their method depends on voxel shapes it can only process shapes of limited resolution in reasonable time.

2.2.2 *Patch-type Segmentation*

The goal of patch-type segmentation methods is to identify quasi-flat areas of a shape. These kind of methods are best suited for mechanical, human designed objects as they are often constructed using rectangular cuboids. As was the case with part-type segmentation, there exists no exact definition of what constitutes a patch as it depends on the application at hand. As such, we need to visually reason about the 'correctness' of a given segmentation result. Here we will give a brief overview of some techniques that have been proposed.

Fitting Primitives

One way of finding patches on shape boundaries was proposed in 2006 by Attene *et al* [AFS06]. The method is based on Hierarchical Face Clustering [GWH01], a technique that iteratively collapses edges of the dual graph of a mesh in order to merge faces. By assigning a cost function to the graph's edges the edge with lowest cost is removed, merging together the vertices—and therefore faces—it connects. Attene *et al* propose as cost function the minimum error of fitting a plane, sphere and cylinder through two adjacent faces. For typical patch-type shapes the quasi-flat areas are best fit by a single plane and would therefore be segmented in a single cluster.

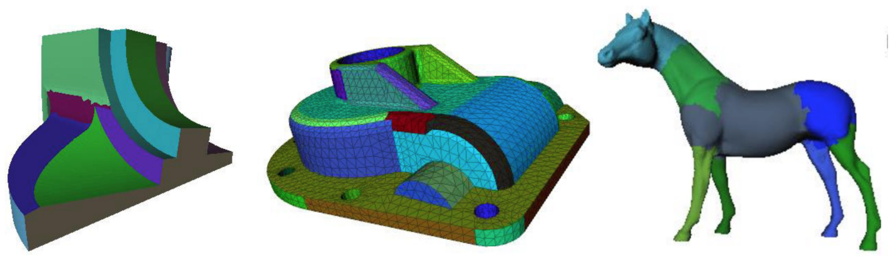


FIGURE 2.8 Patch-type segmentation by fitting primitives [AFS06]. Typical patch-type shapes as in (a) show good results, whereas natural shapes as in (b) are suboptimal when compared to dedicated part-type segmentation methods.

Unfortunately the paper only provides a limited set of results. From these, we may conclude that typical patch-type shapes, as in Figure 2.8a, are segmented well, as well as being robust to noise. We note however that they also present the segmentation of a horse, for which a reasonable, but suboptimal part-type segmentation is computed, see Figure 2.8b. Moreover, the technique is not pose-invariant by nature so we argue that the method is not suitable for robust part-based segmentation. Attene *et al* give in their paper very fast computation speeds, where the hierarchical clustering only takes 22 seconds for a large mesh of 200K faces, not even on modern hardware.

Soft Edge Detection

By definition, quasi-flat areas are separated by areas of high curvature, or *soft edges*. Consequently, by performing curvature analysis on the shape boundary such separation areas may be identified, from which the quasi-flat areas follow. There have been numerous curvature analysis algorithms proposed in the past, of which we discuss two technique that both use surface skeletons. In 2008 Reniers *et al* propose a patch-type segmentation method for voxel shapes [RT08b]. They observe how the boundaries of simplified fore- and background skeletons map to soft convex/concave edges on the shape's boundary, thereby identifying the areas where patches are to be separated.

Although Reniers *et al*'s approach produces good-looking, robust to noise results, it comes with a disadvantage in terms of performance. Given that a voxel-representation is used, only shapes of limited resolution can be processed in reasonable times. For example, a shape of about 250^3 voxels takes just under 10 minutes to compute.

Back in 2014, a new proposal for patch-type segmentation using surface skeletons has been presented by Kustra *et al* that works on

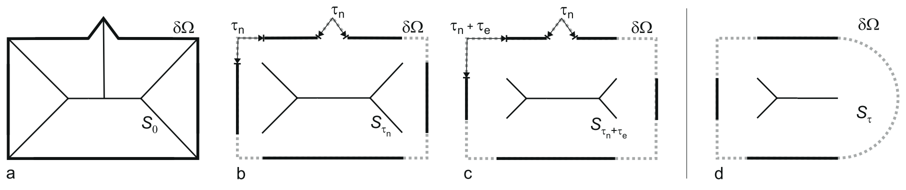


FIGURE 2.9 Patch-type segmentation using skeleton-based edge detection [RT08b]. By simplification of the skeleton (**b**) and distinction between *edges* vs. *noise* (**c**) actual edges are identified.

mesh shapes [KJT16]. Their paper details a novel framework for processing surface skeletons, with one of its capabilities is to classify each skeletal point as discussed in Section 2.1.2. Recall that A_3 points correspond with the skeleton boundary and therefore represent soft edges. By projecting the A_3 points to the surface boundary thick lines are obtained, such that the connected components are the resulting patch-type segmentation. The paper shows that the method produces accurate patches, but may suffer from over-segmentation and jaggy segment borders.

3 CUT-SPACE SEGMENTATION

Using the cut-space for shape segmentation is a recent technique pioneered by Feng, Jalba, and Telea [FJT15b]. As their pipeline was based on a volumetric voxel-based shape representation, their proposal needs to be redesigned to work with meshes or on point cloud data. In the next section we first go into detail about Feng *et al*'s method, after which we continue on how we designed a cut-space segmentation method for mesh-based models. First, an overview of our full pipeline is presented upfront in Figure 3.2. Then, in Section 3.3 we explain how the cut-space is computed, followed by how segments are derived from the cuts. Then, Sections 3.5 and 3.6 show how the cut-space segmentation information is transferred to the shape surface. Next, we introduce several filters to improve and validate the resulting segmentation in Section 3.7. Finally, two techniques for improving the rendering of the results are discussed in Sections 3.8 and 3.9.

3.1 VOXEL BASED CUT-SPACE SEGMENTATION

Using the surface skeleton, one can derive a part-type segmentation of a shape Ω by computing the shortest geodesics through the feature points $\mathbf{f}_1$ with $\mathbf{f}_2$ for each skeletal point, producing so-called *cuts* of which a set is referred to as a *cut-space*. The full pipeline of this process is given in Figure 3.1. Using the cut-space, shape segments $\mathcal{S}_i$ are identified by analyzing and categorizing the cuts based on their length. Next, segment borders $\mathcal{B}_i$ are selected as being cuts $c(\mathbf{x})$ having cut $c(\mathbf{y})$ among its 26-connected neighbors in a different subset $\mathcal{S}_{j \neq i}$. The set $\{\mathcal{B}_i\}$ of the cuts is used to translate from the cut-space, *i.e.* skeleton domain, back to the shape surface, by finding the connected components of $\partial\Omega$ being separated by the border-cuts in $\{\mathcal{B}_i\}$.

For each skeletal point, a cut needs to be traced connecting $\mathbf{f}_1$ with $\mathbf{f}_2$, creating a cross-section of the shape. Feng *et al* designed an algorithm to trace cuts as a combination of at least three cut parts γ_i, briefly as follows. First, a shortest path γ_1 between $\mathbf{f}_1$ and $\mathbf{f}_2$ is constructed using Dijkstra's shortest path algorithm. Given γ_1, a point $\mathbf{o}$ on the opposite of the shape is computed by finding the middle point $\mathbf{m}$ on γ_1, then finding where a ray through $\mathbf{m}$ with direction $\mathbf{x}-\mathbf{m}$ exits shape

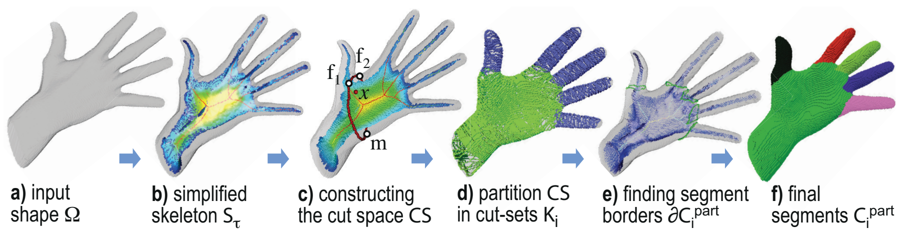

f1
f2
x
m
a) input shape Ω
b) simplified skeleton S_τ
c) constructing the cut space CS
d) partition CS in cut-sets K_i
e) finding segment borders ∂C_i^part
f) final segments C_i^part

FIGURE 3.1 The pipeline as used by Feng *et al* [FJT15a, FJT15b].

Ω. Next the two shortest paths γ_2 between $\mathbf{f}_1$ and $\mathbf{o}$, and γ_3 between $\mathbf{f}_2$ and $\mathbf{o}$ are computed such that a full cut is given by $\gamma_1 \cup \gamma_2 \cup \gamma_3$.

Using this method of constructing a cut, the following beneficial properties are satisfied by all cuts:

1. *Tight:* cuts need to be geodesics, *i.e.* shortest paths on $\partial\Omega$.
2. *Smooth:* cuts shall not contain sharp angles.
3. *Self intersection free:* Cuts may never intersect with themselves.
4. *Locally orthogonal* to the symmetry axis: a cut needs to be locally orthogonal to the curve skeleton, *i.e.* the symmetry axis.
5. *Closed:* no gaps occur in the cut, it is circularly connected.

Feng *et al* computed a partitioning of the cut-space by using a histogram [FJT15b]. They collect all cut-lengths into a histogram, then analyze the histogram for its valleys which are used as thresholds. Using these thresholds the cut-space is partitioned into clusters of similar cut-length, therefore likely to correspond with a part.

3.2 PIPELINE OVERVIEW

To aid in putting the upcoming sections into perspective, an overview of our cut-space segmentation method is presented upfront in Figure 3.2. When compared to Figure 3.1 some differences are introduced in order to effectively compute a part-based segmentation from the cut-space for mesh-based shapes.

Throughout the remainder of this chapter we will include references to the discussed stage in the above pipeline figure.

3.3 COMPUTING THE CUT-SPACE USING MESHES

Since the cut-space $\mathcal{S}$ is defined from a surface skeleton, the first step is to compute such a skeleton. For this we use Jalba *et al*'s GPU optimized skeletonization method [JKT13], which is, to date, the

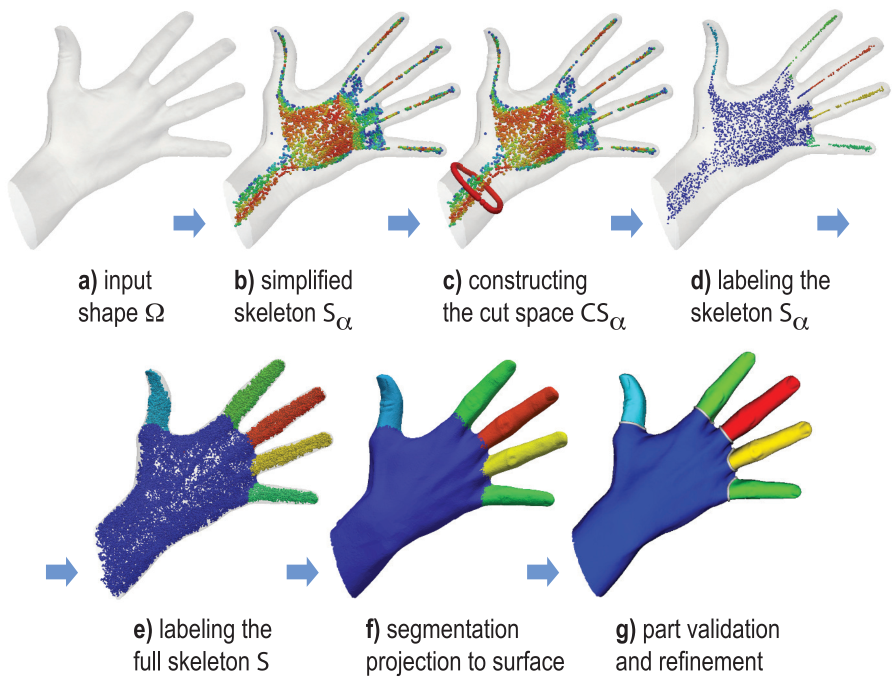

a) input shape Ω
b) simplified skeleton S_α
c) constructing the cut space CS_α
d) labeling the skeleton S_α
e) labeling the full skeleton S
f) segmentation projection to surface
g) part validation and refinement

FIGURE 3.2 Our proposed pipeline to compute part-based segmentations using surface skeletons.

fastest available framework we are aware of. The resulting surface skeleton is an unconnected point cloud $S = \{\mathbf{x}_i\}$ with skeleton points $\mathbf{x}_i$. In contrast, the voxel skeletonization method that is used by Feng *et al* outputs a voxel-based skeleton. The point-cloud skeleton representation introduces several challenges that are addressed in the next subsections.

We note that since this skeletonization method requires a mesh representation as input, our pipeline would also require using meshes as input shape representation. Considering the various methods to construct a mesh from point cloud data, *e.g.* ball-pivoting [BMR+99], Delaunay triangulation, and α-shapes [EKS06], it would still be possible to process pure point cloud datasets by first applying a conversion method to obtain a mesh. Throughout this thesis we therefore assume a mesh is available.

3.3.1 *Skeleton Regularization*

Similar to Feng *et al*, the surface skeleton S needs to be regularized to avoid creating cuts from unimportant skeletal points due to small-scale noise on $\partial\Omega$, refer to stage (b) in Section 3.2. For this we may want to use the MGF (*Medial Geodesic Function*) importance metric—proposed by Jalba *et al* [JKT13] together with the skeletonization method we use—considering that the MGF metric is analogous to the collapse metric that is used by Feng *et al*.

When actually comparing the collapse metric against the MGF variant, we observe a significant difference near the shape's curve skeleton, see Figure 3.3a. The collapse metric attains much higher values throughout this area than its surrounding surface skeleton points. Therefore, Feng *et al*'s thresholding of the collapse metric results in skeleton points that are due to noise to be removed, while preserving the essential curve-skeleton points near the curve skeleton.

When computed on a point-cloud skeleton however the MGF metric does not have this property, see Figure 3.3b. The collapse metric is not only monotonically increasing from the outside to the inside of the shape, as is the MGF, but is also increasing towards the shape center where most boundary mass collapses onto, a property that does not hold with shortest geodesics. As a consequence, thresholding this field may cause thin areas of the shape to be removed which is undesired.

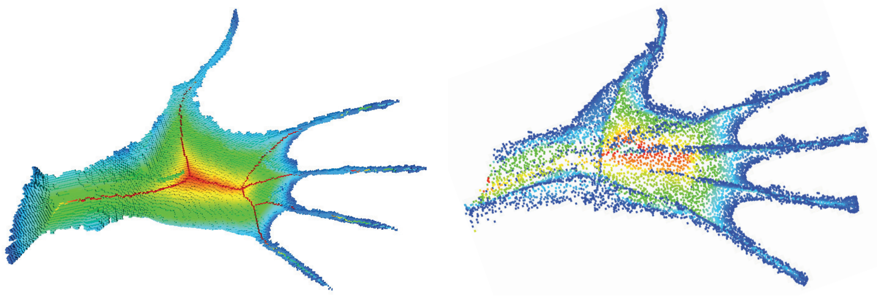
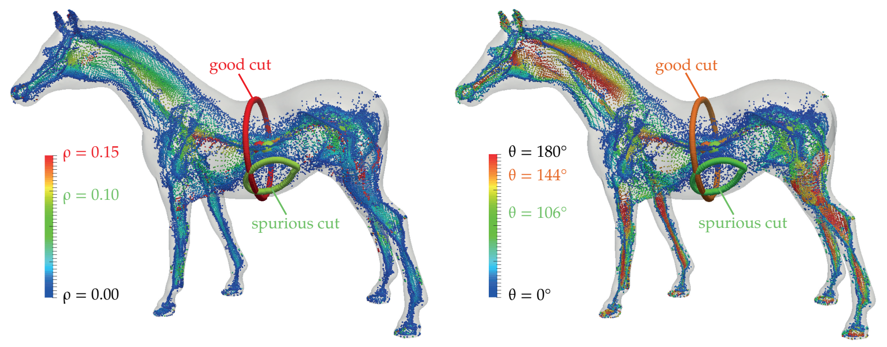

(**a**) Collapse metric, voxel skeleton (**b**) MGF, point-cloud skeleton

FIGURE 3.3 The collapse metric for voxel-based shapes (**a**) is much higher in points close to the shape's curve skeleton, a property that is not seen in the MGF metric (**b**).

The problem with thresholding the MGF is illustrated in Figure 3.4a. Two cuts have been highlighted, of which only the red cut is desired as it corresponds with the main shape axis, whereas the green cut does not and is therefore undesirable to include in the cut-space. If we were to select a threshold τ slightly larger than the value corresponding with the green cut, *i.e.* $\tau = 0.11$, we would remove 70% of the surface skeleton points, including all skeletal points that represent the ears and legs. Even a conservative threshold $\tau = 0.01$ eliminates detail parts such as the ears, see Figure 3.5a. Consequently, we have shown that we cannot regularize the skeleton using the MGF metric, as it cannot both keep detail parts while also eliminating badly oriented cuts.

In search for another regularization method we note that surface skeleton points close to the curve skeleton have larger angles between



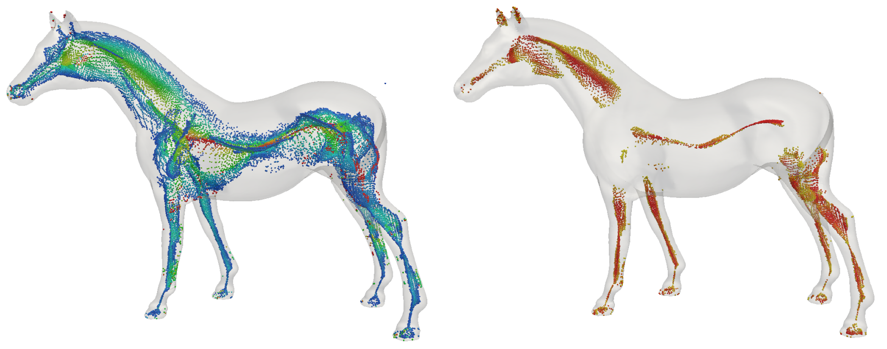
(a) MGF metric (b) θ-SMA metric

FIGURE 3.4 Skeleton regularization using MGF ρ and θ-SMA fields. Using θ-SMA captures the main axis of the shape better, therefore letting the cuts being *locally orthogonal*.

their feature vectors[RvWT08]. As such, we may regularize the surface skeleton to include only points having their FT vectors at a large angle from each other. This method is the well-known local importance method θ-SMA introduced by Foskey *et al* [FLM03]. The metric is shown in Figure 3.4b, which shows consistent high values in all shape parts (rump, legs, muzzle and ears) and consistent low values near the boundary of the surface skeleton.

We therefore define S_α as being a subset of S thresholded on $\theta(\mathbf{x}) = \angle(\mathbf{f}_1, \mathbf{f}_2)$ using a minimum angle $\alpha \in [0, 360]$:

$$S_\alpha = \{\mathbf{x} \in S \mid \theta(\mathbf{x}) > \alpha\} \tag{3.1}$$



(a) Thresholded by $\rho > 0.01$ (b) Thresholded by $\theta > 120°$

FIGURE 3.5 Skeleton regularization using θ-SMA and ρ fields. Using θ-SMA captures the main axis of the shape better, therefore letting the cuts being *locally orthogonal*.

For α we empirically established a value of 120°to give good results for all the shapes we tested. The thresholded skeleton S_α is shown in

Figure 3.5b. Now, undesirable cuts like the green cut in Figure 3.4 get removed while the stable cuts that are *locally orthogonal* to the main axis are retained.

Note that this skeleton is not connected, first because it is just a point cloud, and next due to the removal of low θ-SMA points. This is of no problem however, because we only use S_α for constructing the cut-space.

3.3.2 *Geodesic Computation*

Having reduced the full surface skeleton to a stable set of skeleton points, having arrived at stage (c) in Section 3.2, cuts $c_i = c(\mathbf{x}_i) \in \mathcal{S}$ may be computed. We could copy Feng *et al*'s use of Dijkstra's shortest path algorithm to compute the cuts, where the mesh itself represents the graph on which to compute the paths, as opposed to Feng's approach of deriving a graph from the voxels, with edges defined by a voxel's neighbors. However, an alternative approach to compute shortest geodesics on meshes was proposed recently by Jalba *et al* [JKT13] for their application of skeleton regularization, which adheres to all of the cut requirements as outlined in Section 2.2.1 and does not suffer from any of the discussed drawbacks which may occur using Feng's method. Consequently, this method is also a good candidate for our usage.

A straightest geodesic γ_S is defined as the unique solution of the initial-value problem $\gamma_S(0) = \mathbf{p}, \gamma_S'(0) = \mathbf{v}$, with $\mathbf{p} \in \partial\Omega$ being a point on the shape boundary having tangent vector $\mathbf{v} \in T_p$. Jalba *et al* proposed an extension to define *Shortest Straightest Geodesics* (SSGs) γ_{se} between two points $\mathbf{s}, \mathbf{e} \in \partial\Omega$ to be an accurate approximation of the shortest geodesic from $\mathbf{s}$ to $\mathbf{e}$ [JKT13]. Their method computes multiple straightest geodesics over tangent vectors $\mathbf{v}_i \in T_s$ at $\mathbf{s}$ and then selects the one with shortest length $\|\gamma_{S,i}\|$:

$$\begin{aligned} \gamma_{S,i}(0) = \mathbf{s}, \quad \gamma_{S,i}'(0) = \mathbf{v}_i \\ \gamma_{S,i}(\|\gamma_{S,i}\|) = \mathbf{e} \\ \gamma_{se} = \arg\min_i \left\|\gamma_{S,i}\right\| \end{aligned} \tag{3.2}$$

For the proposed application of computing an importance measure for skeleton regularization purposes, the SSG between feature points $\mathbf{f}_1$ and $\mathbf{f}_2$ is computed. For our purpose of computing shortest cuts however, we require the geodesic to start and end in $\mathbf{f}_1$ and pass $\mathbf{f}_2$ somewhere in-between. As such, we redefine $\gamma_{S,i}$ as follows:

$$\gamma_{S,i}(0) = \mathbf{f}_1, \qquad \exists x \in \mathbb{Z} : \gamma_{S,i}(x) = \mathbf{f}_2, \qquad \gamma_{S,i}(\|\gamma_{S,i}\|) = \mathbf{f}_1 \tag{3.3}$$

To compute γ_{se} using Equations (3.2) and (3.3), we proceed similarly to Jalba *et al* [JKT13]. For each skeleton point $\mathbf{x} \in S$, we trace $M = 30$ straightest geodesics $\gamma_{S,i}, 1 \leq i \leq M$, with starting directions $\mathbf{v}_i$ uniformly distributed in $T_\mathbf{s}$. For each direction $\mathbf{v}_i$, we compute $\gamma_{S,i}$ by intersecting the mesh $\partial\Omega$ with the plane with normal $\mathbf{n}_i = \mathbf{f}_1 \times \mathbf{v}_i$ that passes through $\mathbf{s}$, and select the shortest intersection amongst the directions. In contrast to Jalba *et al*, we continue tracing until having found an intersection with both $\mathbf{f}_2$ and finally arrive back at $\mathbf{f}_1$. Finally, we gather all such SSGs to construct the cut-space

$$\mathcal{S}_\alpha = \{\gamma_{\mathbf{f}_1\mathbf{f}_2} | (\mathbf{f}_1, \mathbf{f}_2) \in FT_{\partial\Omega}|_{S_\alpha}\}, \tag{3.4}$$

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

3.4 CUT-SPACE PARTITIONING

Once the cut-space C_α has been computed, the next step is to identify how to partition the cut-space to obtain the shape's parts. In the pipeline image Section 3.2 this stage is indicated as **(d)**. We use a similar approach as Feng *et al* by means of a histogram of cut lengths, in which peaks indicate cuts with similar lengths that occur often and are therefore likely to represent specific shape parts. With this assumption we may obtain a partitioning of the cut-space by determining all peaks, such that the valleys in between these peaks are used as partition thresholds. Figure 3.6 shows a sample histogram of cut-lengths of a horse shape.

3.4.1 *Histogram Valley Detection*

In order to automatically and robustly detect histogram peaks and valleys, we need to analyze the histogram's bins and their interrelationships. We first apply the mean shift algorithm [CM02] to attenuate the differences between peaks and valleys. Feng *et al* proposed an algorithm that first searches for a bin high enough for it to be considered as peak, then continues searching for the next bin smaller than a certain quantity which is considered a valley. Although this does give an indication of where peaks and valleys are located, it suffers from not finding the valley which is smallest because of its greedy search for valleys. Moreover, what one would consider a valley is dependent on the neighboring bins and we would therefore like to make the search for valleys take into account its surroundings.

To accomplish this, we start searching for a peak with height h_p that represents at least $\lambda_p = H_{\text{peak}} \cdot \|\mathcal{S}_\alpha\|$ cuts. We then continue with subsequent bins and update h_p as even higher peaks are found.

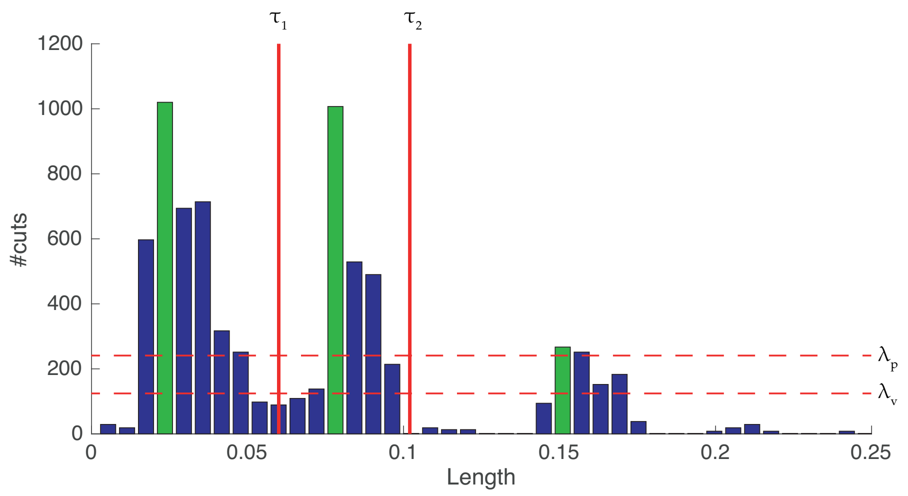


FIGURE 3.6 Histogram of cut-lengths of a horse shape in which two valleys, indicated by red lines, are detected using the algorithm as given in Section 3.4.1. The cuts in range $[0, 0.06)$ represent the horse's legs, range $[0.06, 0.1)$ corresponds with the neck and longer cuts correspond with the torso of the horse.

Using the value of h_p we determine the threshold for valleys to be $\lambda_v = H_{\text{decrease}} \cdot h_p$ such that valley detection becomes dependent on the peak it corresponds with. Once a bin not exceeding the threshold λ_v is encountered, we do not immediately accept it as a valley but instead remember its height h_v.

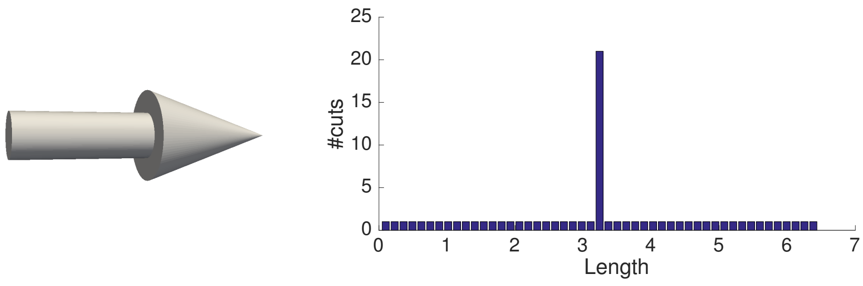
As subsequent bins may be even smaller and therefore preferred as valley, we need to continue searching for bins smaller than h_v and updating h_v accordingly. Once a bin exceeding λ_p is encountered a new peak has been reached, so that we now have two peaks and know which bin in-between is smallest, and therefore our desired valley. The cut-length represented by the i^{th} valley results in threshold τ_i, later used in partitioning $\mathcal{S}_\alpha$.

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

An alternative method of computing thresholds based on a histogram would be to fit $k \in \mathbb{Z}$ Gaussian curves over the histogram, as used by Shapira *et al* for partitioning of their shape diameter function [SSC08]. A problem with this approach is that the algorithm only uses a single parameter k which directly corresponds with $k - 1$ thresholds being found, while the number of thresholds should preferably be determined automatically, and therefore the parameter k is unknown. This problem could potentially be solved by choosing an optimum value for k based on the *gap statistic* [TWH00] but we preferred selecting thresholds using a method that simulates the human approach of detecting peaks and valleys, as described above.

3.4.2 *Histogram Limitations*

We should note that using a histogram for partitioning the cut-space has several limitations. As a histogram is a global aggregation of the cut-space, spatial information is lost which may result in both over-segmentation and under-segmentation. Consider having a cone with in the center of its base having a cylinder mounted on, only halve the radius of that of the cone's base, much like a 3D arrow as shown in Figure 3.7a. For such a shape we would like the segmentation to contain two parts, namely the cone itself and the cylinder. We can however show that this desired result cannot be obtained using the histogram approach, as follows.



(a) Arrow shape in 3D (b) Cut-space histogram for arrow shape

FIGURE 3.7 An arrow shape cannot be segmented into its two parts using a histogram approach.

Given the histogram that would be created for this shape, see Figure 3.7b, our algorithm for determining thresholds will not detect any. Consequently, the full cut-space is assigned a single partition resulting in only a single part, instead of the desired two parts. Hence, under-segmentation can be an issue when using histograms.

We can use the same shape to illustrate how over-segmenting may occur due to an issue we refer to as *threshold aliasing*. Assume that using the histogram in Figure 3.7b, a threshold is inserted just to the right of the spike. Then, the cylinder will be separated from the cone as desired, however the cone itself will also be partitioned into two parts as the cuts close to the cone's apex are assigned the same partition as the cylinder.

Things get even more interesting when a threshold is inserted to the left of the spike, instead of to the right. Now, the cylinder will be in the same partition as the large area of the cone, and only towards the apex will a different partition be assigned. While this does in fact result in the desired number of parts, the parts themselves are incorrect.

Above issues are inherent to using a histogram due to its global nature. An alternative approach is necessary to avoid aforementioned problems and one such alternative is briefly discussed under Future Work in Section 7.4. Despite these issues we still choose to continue using histograms, as the alternative was only proposed when finalizing this work.

3.4.3 *Partitioning Using Thresholds*

Using the set of thresholds $\mathrm{T} = \{\tau_i\}$ that has been determined, we may now partition $\mathcal{S}_\alpha$. A naive method to do so would be to iterate over all cuts and then compare its length with all thresholds, resulting in a complexity in $\mathcal{O}(\|\mathcal{S}_\alpha\| \cdot \|\mathrm{T}\|)$. Although $\|\mathrm{T}\|$ is small, we can reduce the complexity to be only linear in $\|\mathcal{S}_\alpha\|$ by reusing the histogram once more. Recall that the thresholds correspond with histogram bins, therefore we know that a single bin represents only cuts in the same partition. Consequently, we may assign per bin the partition it represents and iterate over all cuts to determine its partition using a constant time lookup.

Having now a partition of the cut-space, note that each partition does not necessarily translate directly into a part, as parts in the same partition may be disconnected. Therefore, the next step is to cluster each partition into its connected components. Whereas Feng *et al* used the neighborhood relation as imposed by the regular grid to determine connectivity, our skeleton lacks such a connectivity relation, being an unconnected point cloud. Therefore, we first define the neighborhood $\mathcal{N}_i$ of a skeleton point $\mathbf{x}_i$ to be the set of skeleton points within a radius r of $\mathbf{x}_i$:

$$\mathcal{N}_i = \mathcal{N}(\mathbf{x}_i) = \left\{\mathbf{x}_j \in S \mid \|\mathbf{x}_i - \mathbf{x}_j\| < r\right\} \tag{3.5}$$

An appropriate choice for r depends on the shape's dimensions. We have established that using 1% of the longest edge of the shape's bounding box works well in general.

Finally, in order to eliminate any noise in the partitioning, *i.e.* areas where cuts within a radius do not agree on an unambiguous part, a mode filter for all cuts c_i is applied to filter out the noise, with the mode being computed over all cuts $c \in \mathcal{N}_i$. Whereas noise reduction applications commonly use a median filter, using the most frequently occurring label is preferred as the part labeling is categorical.

3.5 EXTENDING PARTITION TO FULL SURFACE SKELETON

In Section 3.3.1 we defined the cut-space $\mathcal{S}_\alpha$ to be a subset of the full surface skeleton. Consequently, the partitioning has only been applied on skeleton points that are in $\mathcal{S}_\alpha$, the partition of all other skeleton points is unknown. We require all skeleton points to be assigned a label however, in order to detect where part transitions occur. It is therefore necessary to fill in these points using the established labeling, which corresponds with stage **(e)** in Section 3.2.

Our initial approach to accomplish this was to use a *nearest neighbor* algorithm, to assign for each unknown skeleton point the label of its closest labeled skeleton point. This approach however has several downsides making it unsuitable, which we detail next. First, using the nearest neighbor does not take into account the shape's boundary, which may cause issues as shown in Figure 3.8, a shape of a dog. The surface skeleton points which have not been assigned a label, due to not being part of $\mathcal{S}_\alpha$, are shown in gray, of which a single skeleton point is outlined in red. The arrows show that for the point outlined in red, the closest labeled point is in the ear, whereas the desired labeled point is further away in the neck of the dog. Therefore, the segment of the ear would leak into the neck when using a nearest neighbor approach.

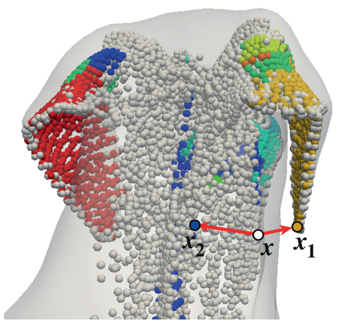


FIGURE 3.8 Using a nearest neighbor approach for assigning labels to gray, unassigned, points results in undesirable results, as the shape boundary is not taken into account.

Yet another deficiency of using a nearest neighbor approach is that narrow parts may percolate into broader parts, as using nearest neighbors implies that part transitions occur halfway between parts and the wider parts have the skeleton points further into the shape, and therefore further away from the shape boundary. An example of this problem is shown in Figure 3.9 of the torso and two front legs of the dog shape, where the torso is much wider than the legs. As can be seen, the skeleton points in the legs reach close to the torso, whereas the torso part's skeleton points are only situated deeply in the torso's center. Now, when using a nearest neighbor approach, many of the skeleton points clearly inside of the torso are still assigned to be part of the leg, as the leg skeleton points are closest.

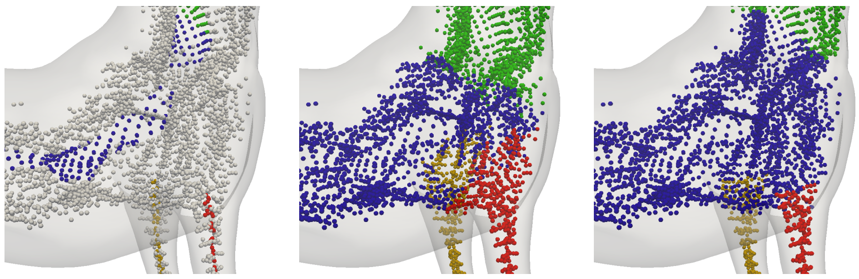

(a) Labeling of $\mathcal{S}_\alpha$ (b) Nearest Neighbor (c) Distance Transform

FIGURE 3.9 Assigning labels from the nearest neighbor causes undesired results as part transitions are then established halfway between parts, causing the legs and neck to be pushed into the torso. Using the Distance Transform avoids this issue.

In conclusion, using the nearest neighbor for assigning labels is not desirable and a different approach is required. To solve the problem of always having part transitions occur halfway in-between the labeled skeleton points, we note that skeleton points should be given some sort of weight to represent their range. By letting a skeleton point's range be the minimum distance to the boundary, we may use the distance transform as defined for the skeleton to represent the weights.

To apply the concept of using the distance transform as weights to assign a part to unlabeled skeleton points, the approach of determining a label for each unlabeled surface point is inverted so that labeled skeleton points $\mathbf{x}_i$ perform a neighbor search in the radius defined by $DT_{\partial\Omega}(\mathbf{x}_i)$ and then propagate its label over to all these neighbors. It is important to note that even already labeled points $\mathbf{x}_j$ are reassigned their label if $DT_{\partial\Omega}(\mathbf{x}_j) < DT_{\partial\Omega}(\mathbf{x}_i)$, such that skeleton points with larger range become dominant. Having the skeleton points p with largest range be dominant may improve the segmentation, as doing so may eliminate another part that would otherwise occupy surface area that is also in reach of p, and therefore should also be considered to be in the same part as p.

Skeleton points that are still unlabeled after this operation are being assigned the label of their nearest neighbor, to ensure all points have been assigned a label. Finally, to find now the connected components in the established partitioning of the cut-space, several flood fills from within each partition, again using the neighborhood relation $\mathcal{N}_i$, are started until all points have been assigned a part label.

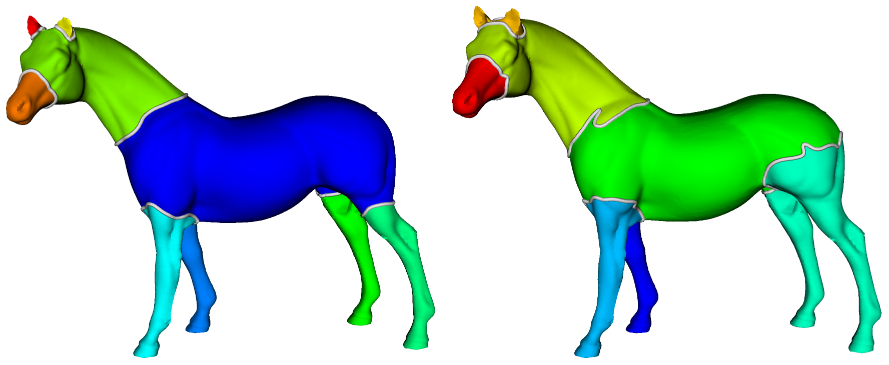

(a) Using Distance Transform (b) Using Nearest Neighbors

FIGURE 3.10 Using the Distance Transform as weights shows a much better final segmentation of the horse than when just using a nearest neighbor approach.

Using this algorithm for filling in all unlabeled skeleton points resolves the issues of the nearest neighbor approach and has a significant effect on the final segmentation, as is shown in Figure 3.10.

3.6 TRANSLATING SKELETON PARTITION TO SURFACE

Having now a part labeling defined on all of the skeleton points, the next step is to translate the segmentation from the skeleton back to the shape surface; see stage (f) in Section 3.2. In Section 3.3 we discussed how Feng *et al* used a set of border candidates $\{\mathcal{B}_i\}$ to select the most appropriate cut from, and then use that cut to represent the actual segment border between two parts. Since cuts are constructed having certain desirable properties, this yields clean segment transitions.

While Feng *et al*'s approach proved feasible for volumetric representations, we encountered several issues with their method when applied on mesh models. To understand these issues, we show next how we redesigned their approach to operate on a point cloud skeleton.

3.6.1 *Selecting Cuts from Border-Sets*

Instead of being able to use the well-defined neighborhood relation as imposed by a volumetric representation, we again need to rely on using $\mathcal{N}_i$ as defined in Equation (3.5). Each point with part label $p = p(\mathbf{x}_i)$ is included in border-set $\left\{\mathcal{B}_{p,q}\right\}$ if $\exists \mathbf{x}_j \in \mathcal{N}_i : p(\mathbf{x}_i) \neq p(\mathbf{x}_j) = q$. Then, analogous to Feng's method, the shortest cut among each $\mathcal{B}_{p,q}$ is selected as most appropriate cut, to be used as a transition between the parts p and q. Now, the issues of this method are twofold, one of which also applies to volumetric shapes as used by Feng *et al.*

The first issue of this approach is that it is not known if a cut actually corresponds with the desired segment border. Moreover, given that the cuts we trace are SSGs in a single plane they are totally straight, which may not fit well with the desired segment border. In Figure 3.11 the shape of a dog shows several situations where an undesirable cut is selected among the candidates in each $\mathcal{B}_{p,q}$, *e.g.* the torso-to-tail border is a very small cut which does not separate the tail from the torso, at all. Now in this example it may even be possible to determine that the cut does not agree with the expected length so that it is filtered out, however we also observed situations where the selected cut was of proper length, while still not representing the actual transition between parts p and q. It is extremely important that proper borders are always selected, as incorrectly chosen cuts directly cause under-segmentation or otherwise totally incorrect results.

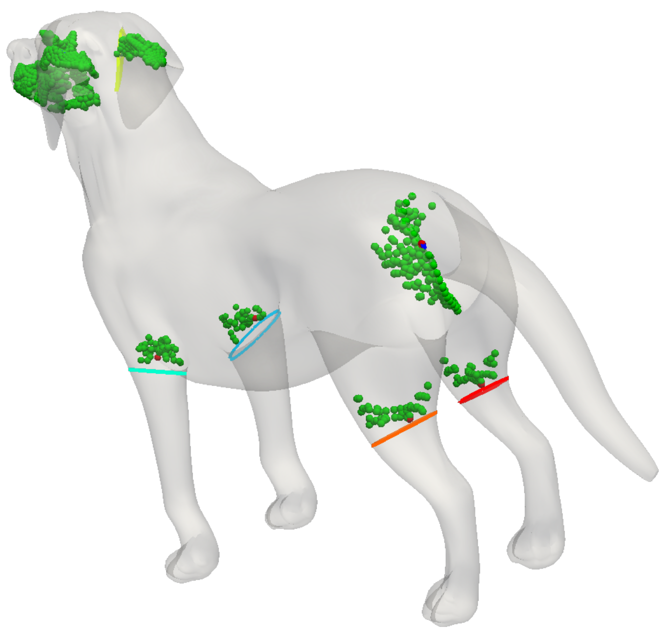


FIGURE 3.11 Selecting the shortest cut (red skeleton points) from border set $\left\{\mathcal{B}_{p,q}\right\}$ (green skeleton points) works well in trivial cases such as the four legs, but fails in more difficult situations, *e.g.* near the tail.

The second issue with Feng's approach is that two adjacent parts p and q having border candidates in $\mathcal{B}_{p,q}$ may require multiple cuts to be selected from the border candidates. Now, one may attempt to partition $\mathcal{B}_{p,q}$ into sets of connected components, for example using hierarchical clustering, so that a border cut may be selected from each component. This however is not sufficient in all cases and may still cause missed segment borders, again resulting in under-segmentation.

An example of a situation that cannot be resolved in this way is shown in Figure 3.12 where the ears are considered to represent a single part and $\mathcal{B}_{p,q}$ cannot be partitioned into multiple components. This results in only a single cut being selected where two cuts are required, therefore only separating one ear from the torso causing under-segmentation of the shape.

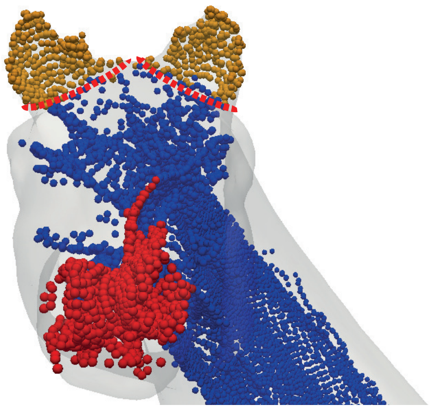
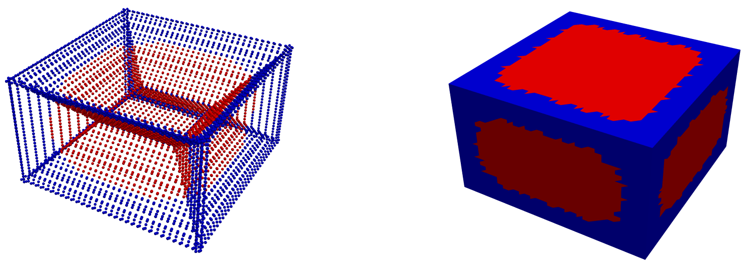


FIGURE 3.12 In order for the horse's ears to be considered as two separate parts, two borders are necessary. As the torso-to-ears border-set consists of only a single connected component, one ear will not be separated from the torso.

Above issues are to such extent that we concluded Feng *et al*'s approach would never work reliably for mesh models. Hence, a different approach is required which is discussed next.

3.6.2 *Using the Feature Transform*

Per the definition of a skeleton, a mapping from skeleton points back to so-called *feature points* on the surface is defined as its feature transform FT, as given in Equation (2.3). For each skeleton point $\mathbf{x}_i$ this transform yields the points on the shape boundary, so all that is necessary to translate the skeleton labeling back to the surface is to copy the skeleton labeling over to the feature points as given by the FT. After having applied this transformation, all still unlabeled surface points get assigned the same part as their nearest neighbor.



(a) Clustered skeleton. (b) Derived parts using FT.

FIGURE 3.13 A single cluster in the skeleton may result in multiple disconnected "parts", which do not actually resemble a part.

In Section 3.5 we discussed how each partition of the skeleton is further split up into its connected components, as a single partition may represent multiple parts. We note however that after having applied the transform from the skeleton to the surface, the labeling that was connected in the skeleton-domain may no longer be connected on the surface, an example of which is shown in Figure 3.13. Therefore, an additional connected component pass needs to be employed on the surface itself.

3.7 PART VALIDATION

Now having a part labeling available on the surface, the connected components represent a supposed part. As has been shown in Figure 3.13 however, it may occur that there exist connected components that do not qualify as a part, according to what humans would perceive as a part. In order to detect such invalid parts and reject them from the segmentation, we must first define the characteristics a part should have for it to be considered an actual part, according to human perception. This stage concludes the pipeline along with the remaining refinements are included in stage **(g)** of Section 3.2.

From our cut-space, we observe that actual parts are covered by their associated cuts. On the other hand, the cuts from which non-part like components are derived only partially cover the component, which is an indication that the component does not represent an actual part and shall be rejected. In essence, the surface area inside of a component, as covered by its associated cuts, compared to the surface area outside of that component is an indication of how well the cuts actually agree with the part, and therefore if it should be kept or not.

To estimate the surface area inside and outside of the component, we compute the number of cuts that only cover the component itself, versus the number of cuts that also cover some other components. If less than 80% of all associated cuts cover only the component itself, then the component is rejected and merged with one of its neighbors.

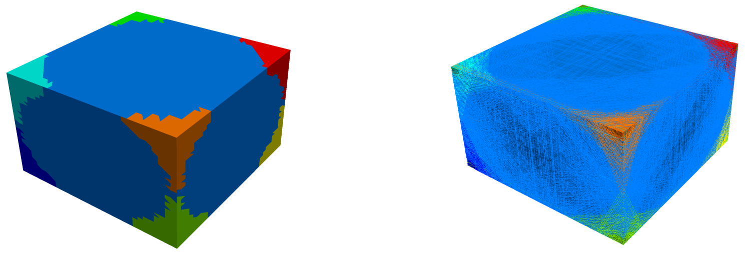

(a) Undesirable corner parts. (b) Accompanying cut-space.

FIGURE 3.14 Even parts that do correspond with their cuts are not necessarily desirable, considering that patches fit much better in this case.

Although above test is able to effectively filter out non-part like components, it is insufficient in situations as shown in Figure 3.14. We would prefer such parts to be disregarded as patch-based segmentation is more sensible in such cases. To also detect these occurrences we observed that for actual parts, their Gauss maps span at least half the map's surface, *i.e.* there exist at least two normals in an angle of at least 180°, or what we refer to as *opposing normals*. Determining if a Gauss map adheres to such a property is not simple and more importantly is it sensitive to noise, as a single normal outlier may incorrectly result in the property to be considered satisfied. To be able to cope better with outliers to be present we compute the angle between all available normals, then take the median angle and require the median to be at least 90°. These validation procedures together have shown to be effective in ensuring the part-based segmentation is meaningful.

3.8 DERIVING CELL LABELS FROM VERTEX LABELS

Having now established and validated the part labels for all vertices, we next need to display the results. Each part label may be associated a color so that all vertices get assigned a color, however rendering this directly does not give desirable results. To actually display the mesh it is necessary to color all of its cells, and a GPU normally fills in cells by interpolating the colors assigned to a cell's vertices. In our case however this approach is invalid, as the associated colors represent categorical data and must therefore not be interpolated. Deriving a single part label per cell from its neighbors is problematic when not all vertices agree on the same part, and simply selecting the mode part yields unsatisfiable results. Therefore, in such cases the cell needs to be split up into sub-cells, each of which then get assigned an unambiguous part label.

We have to distinguish three cases regarding a cell's vertices' labels. Firstly, if all three vertices represent the same part the whole cell can be retained as is and be assigned that part. The second case occurs when only two vertices agree on a part and the third vertex differs, then the cell is divided into two sub-cells as shown in Figure 3.15b. Thirdly, all three vertices may all represent different parts, in which case the cell is split into three sub-cells according to a Voronoi diagram, seen in Figure 3.15c.

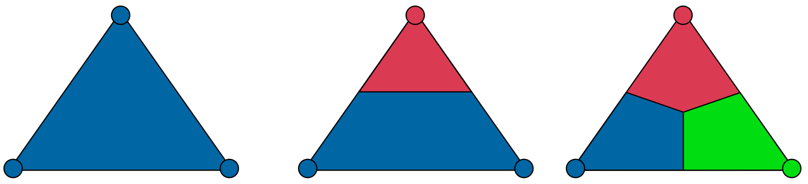

(a) Do not split (b) Halfway split (c) Voronoi split

FIGURE 3.15 The colors associated with a cell's vertices determine how to split a cell in order for unambiguously assigning colors to a cell.

3.9 LAPLACIAN SMOOTHING OF SEGMENT BORDERS

After having applied cell splitting to obtain unambiguous cell labels, we observe that the border between segments is very rough for many shapes as can be seen in Figure 3.16a. Therefore, the desired property *smooth*, as given in Section 2.2.1, is not satisfied so we have to mitigate it. In order to reduce the amount of noise that may be present in a border, each border's points are filtered using three Laplacian smoothing iterations. In each iteration, points are back-projected onto the surface to keep the points located on the shape surface. After having computed the Laplacian smoothed points, the mesh itself is updated to reflect the changes by computing the intersections between the smoothed border and the cells and splitting the cells accordingly, then updating the cell labeling to match. Figure 3.16b shows what the segmentation looks like after the smoothing has been applied.

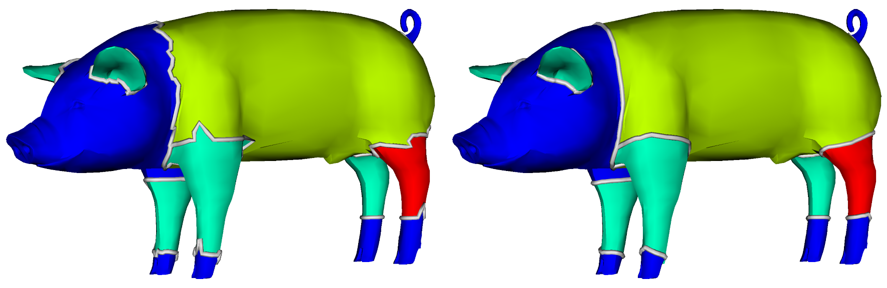

(a) Before smoothing (b) After smoothing

FIGURE 3.16 Laplacian smoothing of segment borders reduces noise significantly.

After having smoothed the segment borders, we end up with a robust segmentation of which the identified parts all meet the requirements.

3.10 SUMMARY

In this chapter we have presented a part-type segmentation method for mesh shapes based on the idea of cut-space segmentation, as introduced for voxel shapes by Feng *et al* [FJT15b]. Several adaptations of their original proposal were required for the cut-space method to work using meshes, but in the end we reach similar results while benefiting from the high resolution that mesh shapes offer. Detailed results of our cut-space segmentation method for mesh shapes are presented in Chapter 6. Moreover, we presented a part verification approach in order to make sure that the results are sensible according to human perception. This property becomes important in the next chapter, as we show there how we introduce patches into the segmentation in order to obtain a unified part-patch segmentation.

4 PART AND PATCH UNIFICATION

In the previous chapter we have discussed a method to compute part-type segmentations. This chapter expands on that work by introducing a novel segmentation model, which is a combination of both part-type and patch-type methods, which we refer to as a *unified* segmentation method. In Section 4.1 we first define the properties such a unification method preferably satisfies, then consider the various strategies we may employ to obtain a unification of both part-type and patch-type segmentations in Section 4.2. Then, the strategy that fits best is further discussed in Sections 4.3 and 4.4.

4.1 PROPERTIES

To be able to select an appropriate strategy we must first define desirable properties, in order to weigh the various strategies against each other and make an advised decision on which strategy to choose. We define the following properties to be of importance for a unification method:

1. *Hybrid:* shapes that contain both part-like areas and areas where patches are a better fit, should incorporate both types as appropriate.
2. *Robust:* the unified segmentation should make sense according to human perception.
3. *Configurable:* the extend of unification should be configurable.
4. *Balanced:* unified segmentations should not be over-segmented due to incorporating both types.

Next, several strategies are discussed with respect to above properties.

4.2 STRATEGIES

One of the most simple approaches to support both part-type and patch-type shapes would be to compute both segmentations separately and then assessing which of the two methods yielded the most appropriate result, then choosing that single segmentation as

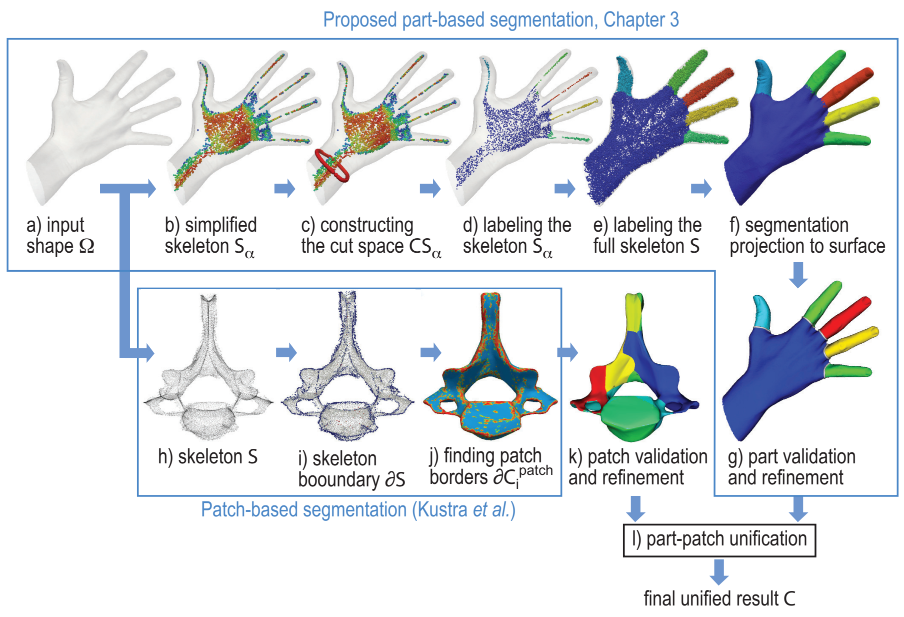

Proposed part-based segmentation, Chapter 3
a) input shape Ω
b) simplified skeleton S_α
c) constructing the cut space CS_α
d) labeling the skeleton S_α
e) labeling the full skeleton S
f) segmentation projection to surface
h) skeleton S
i) skeleton booundary ∂S
j) finding patch borders ∂C_i^{patch}
k) patch validation and refinement
g) part validation and refinement
Patch-based segmentation (Kustra et al.)
l) part-patch unification
final unified result C

FIGURE 4.1 Our proposed pipeline to compute unified part-patch segmentations. **(a-g)** Proposed cut-space segmentation for mesh shapes. **(h-k)** Patch-type segmentation by Kustra *et al* [KJT16], with additional patch refinements. **(l)** Unification of both part- and patch-type segmentations.

final result. Such an approach however would fail to satisfy the properties *hybrid* and *configurable*. Furthermore, in order to satisfy the *robust* property, a stable voting system must be designed which in itself is not trivial. In conclusion, such an approach would fail to meet most of the desired properties and is therefore not considered further.

Instead of using a voting system to choose just a single segmentation outcome, we may overcome the inability of being *hybrid* by selectively combining the results in a meaningful way. By designing heuristics to determine how the segments are chosen it would then be possible to design a method that satisfies all of the desirable properties, as such heuristics could then be tweaked to yield desirable results. Furthermore, such an approach allows for tuning both the part-type and patch-type segmentation separately and one may even use completely different methods for any of the two types. Therefore, we may reuse the cut-space segmentation method as was introduced in Chapter 3 and combine it with an existing patch-type segmentation method.

Pipeline Overview

To put the upcoming forthcoming sections into context, a diagram of the full unification pipeline is presented in Figure 4.1.

In Chapter 2 we have given an overview of several patch-type segmentation approaches, of which the one by Kustra *et al* [KJT16] stands out in particular. Their method has shown to give good results and an additional benefit is that the core of their method is also based on surface skeletons, therefore it fits well into the pipeline we designed in Chapter 3.

4.2.1 *Patch-type Segmentation Using Surface Skeletons*

We briefly discuss Kustra *et al*'s [KJT16] method here, firstly to be able to understand how it works and secondly to show what its results are like for different kinds of shapes. Understanding the characteristics of when and how it works is important for designing merge heuristics, as will be seen in the next section.

As per the definition of a patch, the border between two patches should occur at high surface curvature areas, because such areas interrupt the quasi-flatness of the surface and should therefore be segmented into different patches. Hence, finding the high-curvature areas and inserting patch borders in those areas yields a patch-type segmentation. In the work by Kustra *et al*, high-curvature areas are computed using the surface skeleton as an improvement on earlier work by Reniers *et al* that worked on voxel shapes [RT08b].

Computing Patches

Given the surface skeleton, recall from the skeleton classification in Section 2.1.2 that A_3 points correspond with high positive curvature areas on the surface. Note that the negative curvature areas, or creases, are not represented by A_3 points; they may be found using the complement's shape skeleton, or background skeleton [RT08b]. In their paper, Kustra *et al* show how to classify skeleton points as A_3 by computing the number of clusters in a dilated, or fuzzy, feature set $FT_\tau(\mathbf{x})$, which is different from $FT(\mathbf{x})$ (see Equation (2.3)) in that it also includes all other surface points within a radius of $DT(\mathbf{x}) + \tau$ from skeleton point $\mathbf{x}$ and therefore contains *all* feature points. They then define a cluster $C \subset FT_\tau(\mathbf{x})$ as

$$C = \{\mathbf{f} \in F_\tau(\mathbf{x}) \mid \max_{\mathbf{f}\in C} \min_{\mathbf{g}\neq\mathbf{f}\in C} \|\mathbf{f}-\mathbf{g}\| < \min_{\mathbf{f}\in C} \min_{\mathbf{g}\neq\mathbf{f}\in C} \|\mathbf{f}-\mathbf{g}\|\} \tag{4.1}$$

i.e. all points which are closer to each other than to any point from another cluster [KJT16]. If $\|C\| = 1$ then a skeleton point is classified as being of type A_3.

Using the classified skeleton points, actual high curvature areas on the surface are computed by projecting each A_3 point $\mathbf{x}$ over its extended feature transform $FT_\tau(\mathbf{x})$, *i.e.* they define a set of surface points $E = \{\mathbf{p} \in FT_\tau(\mathbf{x}) \mid \mathbf{x} \in S_\Omega \wedge type(\mathbf{x}) = A_3\}$. Since $FT_\tau(\mathbf{x})$ has been computed to contain *all* surface points the set E results in thick borders on the surface. Then, connected components are computed for surface areas $\partial\Omega \setminus E$, *i.e.* all remaining surface area is segmented by means of connectivity. All points in E get assigned the label of the closest established patch, thereby obtaining a segmentation with patches being a partition of $\partial\Omega$.

Result Characteristics

We show here results of Kustra *et al*'s patch segmentation method to showcase its characteristics, so that we may take these into consideration when next designing the merge heuristics. Figure 4.2a shows an example of where their method gives a very good result on a typical patch-type shape, whereas Figure 4.2b shows a poorly segmented typical part-type shape. The reason for this poor output can be explained, as follows. Finding high-curvature areas is dependent on computing stable A_3 points, which in the case of Figure 4.2b fails due to the cylinder-like shape of many of the horse's parts. This may potentially be mitigated by fine-tuning of the involved parameters, however doing so is a delicate task and may not always result in an improved segmentation.

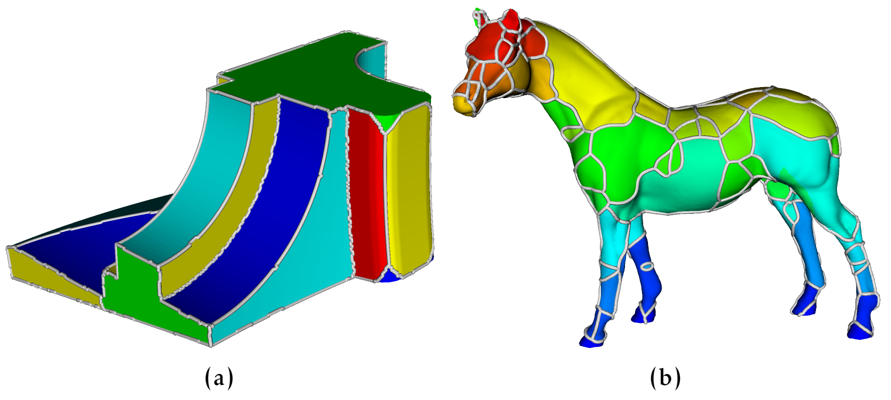


FIGURE 4.2 Showcasing two shapes for which Kustra *et al*'s method works very well on a typical patch-type shape (**a**), but very poorly on a typical part-type shape (**b**).

4.3 MERGE HEURISTICS

In order to combine the parts and patches that are separately computed by cut-space segmentation and Kustra *et al*'s method respectively, we must determine which are most appropriate to use in a unified result. This means that for areas which are quasi-flat or consist of high curvature transitions, it would be preferred to have a patch in such area, whereas part-like areas should preferably be represented by a dedicated segment. To be able to satisfy the property of being robust we must verify that the computed parts and patches themselves are logical according to human perception, such that the final result will also be logical.

Considering that parts should be recognized as dedicated segment, we may simply choose to use all parts from the cut-space segmentation for the unification model. Given that parts computed by the cut-space method have all been validated as described in Section 3.7, they have already been verified to represent an actual part as considered by human perception. Therefore, using all parts *as is* in the unified model is a natural choice.

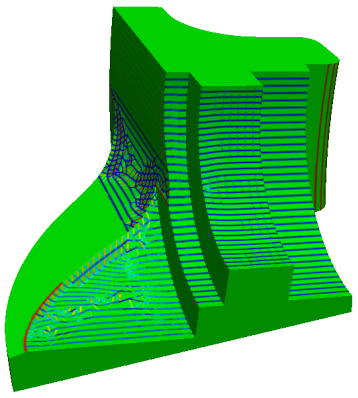
We should next decide on how to introduce patches into the unified segmentation. Considering the characteristics as discussed in Section 4.2.1 we cannot just merge the patches into each part, as doing so would cause over-segmentation given the patch-type result shown in Figure 4.2b. Therefore, we must first ensure patches are validated according to a certain criteria, similar to our approach for part validation. The next sections discuss how to identify *false* patches and how they are filtered out. We will see that doing so will ensure that our method satisfies all four properties.

4.4 PATCH VALIDATION

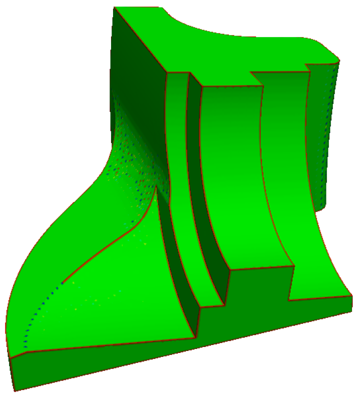
To be able to detect patches that are undesirable, we must establish the criteria that differentiate proper patches from the undesirable patches. When considering the examples shown in Figure 4.2 we note that patch transitions occur in high curvature areas in the case of the desirably segmented fandisk (4.2a), whereas the same is not true for the patches in the horse (4.2b). Therefore, computing the surface curvature in a patch transition may indicate if a patch should be kept or not.

4.4.1 *Computing Surface Curvature*

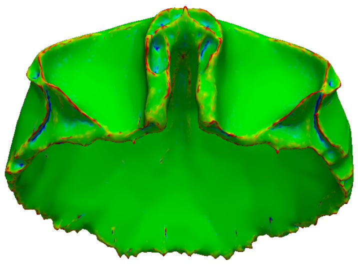
Readily available methods exist for the computation of curvature measures on a shape surface, *i.e. mean curvature* or *Gaussian curvature* as shown in Figure 4.3. As the purpose of computing the surface curvature is to find undesirable patches, we must use a curvature measure that corresponds well with how humans perceive curvature. When considering the examples given in Figure 4.3 it is immediately clear that neither Gaussian nor mean curvature works as desired, *i.e.* in the case of the top row the Gaussian curvature measure is preferred, whereas the bottom row clearly benefits from the mean curvature measure. Therefore, neither of the two curvature measures may be used for our goal.



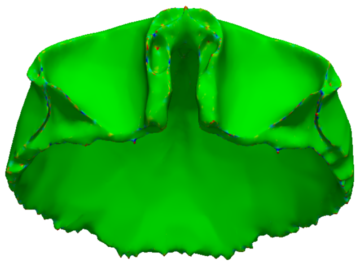
(a) Mean curvature



(b) Gaussian curvature



(c) Mean curvature



(d) Gaussian curvature

FIGURE 4.3 Curvature results for two shapes, computed using ParaView. The top row shows the preferred result using *Gaussian curvature*, whereas for the shape in the bottom row *mean curvature* is preferred. In both cases, the non-preferred method is unusable for our goal.

An intuitive way to assess a shape's curvature would be to look at the shape's normals. Given the vertices $V = \{\mathbf{x}_i, \mathbf{n}_i\}_{i=1}^{N}$, and their normals, of the mesh representation of $\partial\Omega$, and $E = \{\,(x,y)\big|x \in V, y \in V\,\}$ as the mesh edges, we may compute an approximation of the curvature K by taking the maximum angle between all neighbors:

$$\mathrm{K}(\mathbf{x}_i) = \max\Big\{\angle(\mathbf{n}_i, \mathbf{n}_j)\,\Big|\,\mathbf{x}_j \in \mathcal{N}_i\Big\} \tag{4.2}$$

with $\mathcal{N}_i$ being the vertices within a radius r, with the distance $\|\cdots -_{\|} E$ representing the shortest path distance over the edges in E:

$$\mathcal{N}_i = \Big\{\mathbf{x}_j \in V\,\Big|\,\|\mathbf{x}_i - \mathbf{x}_j\|_E < r\Big\} \tag{4.3}$$

A drawback of using this formula however is that it may give unexpected results for corner vertices where the approximated curvature is far smaller than expected, as can be seen in Figure 4.4.

For our purpose of the curvature this poses a problem, given that the corner vertices, shown in red in aforementioned figure, are most likely to be on a patch transition and should therefore produce results that are in line with expectations.

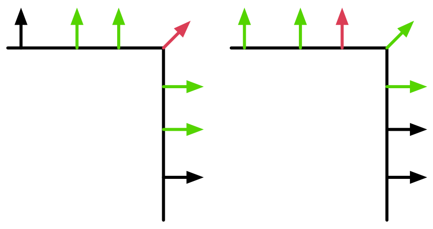

(a) $\alpha = 45°$ (b) $\alpha = 90°$

FIGURE 4.4 The red arrow represents the vertex's normal that is evaluated, the green arrows represent the normals that are compared against. For the corner vertex in (a) we find an angle of 45°, whereas the vertex just offset of the corner (b) unintuitively gets assigned a larger angle of 90°.

To circumvent this issue, we propose an alternative curvature measure that is computed from surface normals. For the proposed computation of $\mathrm{K}(\mathbf{x}_i)$ we first find a set of vertices $\mathcal{N}_i^{\mathrm{K}}$ that are on the border of $\mathcal{N}_i$, as follows:

$$\mathcal{N}_i^{\mathrm{K}} = \left\{ \mathbf{x}_j \in V \setminus \mathcal{N}_i \,\middle|\, \exists \mathbf{x}_k \in \xi_j, \mathcal{N}_i \right\} \tag{4.4}$$

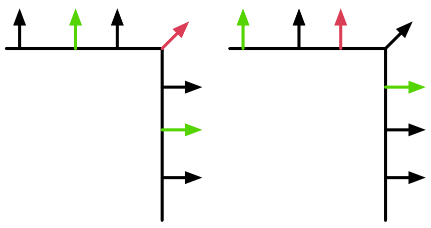
with ξ_i being the connected vertices of $\mathbf{x}_i$:

$$\xi_i = \left\{ \mathbf{x}_j \in V \,\middle|\, \exists (i,j) \in E \right\} \tag{4.5}$$

Then, we approximate the curvature by computing the mean-angle of all combinations of $\mathcal{N}_i^{\mathrm{K}}$:

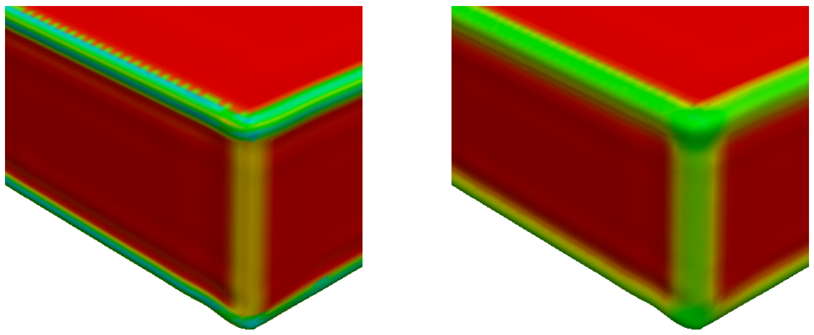
$$\mathrm{K}(\mathbf{x}_i) = \mathrm{mean}\left\{ \angle(\mathbf{n}_i, \mathbf{n}_j) \,\middle|\, (\mathbf{x}_i, \mathbf{x}_j \neq \mathbf{x}_i) \in \mathcal{N}_i^{\mathrm{K}} \times \mathcal{N}_i^{\mathrm{K}} \right\} \tag{4.6}$$

with $\times$ denoting the Cartesian product. Figure 4.5 shows how this proposal finds an expected angle of 90° for the corner vertex. The resulting curvature does not suffer from unexpected drops for corner vertices and therefore becomes more predictable. Figure 4.6 shows the differences on a box shape, where our proposed curvature measure is more uniform and shows no curvature dips.



(a) $\alpha = 90°$ (b) $\alpha = 90°$

FIGURE 4.5 Proposed method of approximating surface curvature, where the green normals are all compared against each other, but not against the red normal. Doing so ensures that an angle of 90° is computed for both vertices.



(a) Naive method: using a direct comparison to neighbors (b) Proposed method: comparing neighbors to each other

FIGURE 4.6 Only comparing a vertex's normal to its neighbors results in lower than expected curvatures in high curvature areas. Our proposed curvature measure solves this issue.

Applying the proposed curvature measure to the shapes for which Gauss and median curvature gave unsatisfactory results, we refer back to Figure 4.3, we now find a result that is suitable for assessing whether or not a patch should be kept, see Figure 4.7. An additional benefit of our proposed curvature approximation is that the radius in which the curvature is evaluated becomes configurable, using an intuitive parameter. Empirical testing has shown that using a radius of 2–5% of the maximum dimension of the shape's bounding box—a common parameter normalization strategy [CGR+04]—gives satisfactory results.

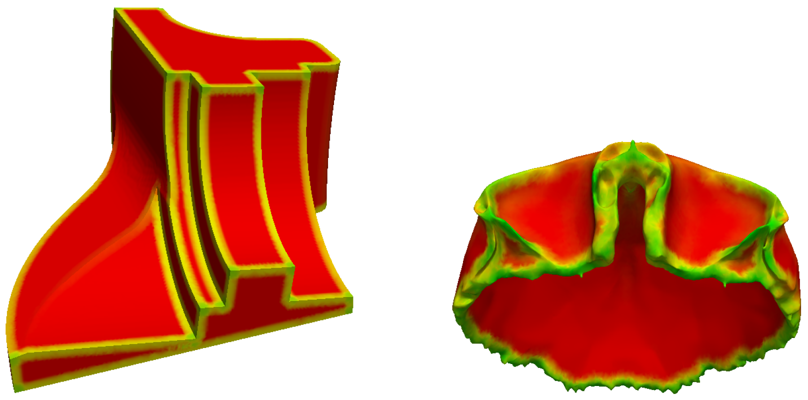

(a) Proposed curvature measure (b) Comparing neighbors to each other

FIGURE 4.7 Showing the curvature approximation as proposed for the shapes in Figure 4.3

4.4.2 *Removing Undesirable Patches*

Using the computed surface curvature field we may next determine which patches are undesirable. Recall that patch borders should occur in high curvature areas, so we propose to evaluate the surface curvature per patch-border as those are the critical areas. Doing so essentially results in determining undesirable borders rather than invalid patches, which directly provides us with the most appropriate neighboring patch to merge with. This is beneficial compared to reassigning the patch-label of all vertices in the invalid patch according to the closest vertex in neighboring patches, as in that case all borders that were imposed by the invalid patch will be shifted inwards of the invalid patch. This is highly undesirable as it may cause patch borders that originally corresponded with a high curvature area to become unaligned.

Define patch $\mathcal{P}_x$ as the set of vertices that have been assigned patch label x:

$$\mathcal{P}_x = \{\mathbf{x}_i \in V \,|\, \text{patch}(\mathbf{x}_i) = x\} \tag{4.7}$$

We next define the border $\mathcal{B}_{x,y}$ that connects patches x and y:

$$\mathcal{B}_{x,y} = \left\{\mathbf{x}_i \in \mathcal{P}_x \,\middle|\, \exists \mathbf{x}_j \in \xi_i, \mathcal{P}_y\right\} \cup \left\{\mathbf{x}_i \in \mathcal{P}_y \,\middle|\, \exists \mathbf{x}_j \in \xi_i, \mathcal{P}_x\right\} \tag{4.8}$$

To determine now if a border should be eliminated we evaluate the median curvature of all vertices in each border $\mathcal{B}_{x,y}$, according to the curvature $\mathrm{K}(\mathbf{x}_i)$ from Equation (4.6):

$$\mathrm{K}'(\mathcal{B}_{x,y}) = \underset{\mathbf{x}_i \in \mathcal{B}_{x,y}}{\text{median}} \{\mathrm{K}(\mathbf{x}_i)\} \tag{4.9}$$

Each border with $\mathrm{K}'(\mathcal{B}_{x,y})$ being smaller than a certain threshold β should be eliminated, resulting in patches x and y being merged: $\mathcal{P}_x \leftarrow \mathcal{P}_x \cup \mathcal{P}_y, \mathcal{P}_y \leftarrow \emptyset$. As merging patches may cause remaining borders to be merged as well, we should eliminate borders using a deterministic ordering as not doing so may give unpredictable, but also undesirable behavior. Therefore, we keep track of a list of borders sorted on K' and iteratively eliminate borders, beginning with lowest median surface curvature K'.

4.5 MERGING SEGMENTATIONS

After removing undesirable patches, we have both a validated part-based and patch-based segmentation. As stated in Section 4.3 we introduce all segments from part-based segmentation into the unified model, as it ensures that all parts will be represented in the final unified segmentation. Next, each part P_x is further partitioned by merging in the patches it contains. Note that a patch $\mathcal{P}_y$ may extend across the boundaries of P_x, *i.e.* it may occur that $\exists \mathbf{x}_i \in \mathcal{P}_y \setminus P_x$ is true. Consequently, unified patches may have become too small—according to a given threshold for the number of patch vertices with respect to the total number of vertices $\|V\|$—and such patches will be replaced by the neighboring patches. This avoids the possibility of having patches smaller than the desired threshold.

4.6 SUMMARY

In this chapter we presented a method of computing a unified part-patch segmentation for mesh shapes, using the cut-space segmentation method as introduced in Chapter 3 combined with the patch-based segmentation method by Kustra *et al.* Because of the validation procedures that are presented for both parts and patches we ensure the validity of segments, such that we may easily combine the two by simply merging them together. Considering the desirable properties as were outlined in Section 4.1 we find that our proposed strategy satisfies all these properties. First, the procedure has shown to effectively prevent over-segmentation and therefore allow for meaningful unified segmentations that match human perception. Second, by allowing for separate configuration of the part-based and patch-type segmentation, we enable tweaking the results according to a specific goal. Finally, because of how patches are added to parts we may also get hybrid segmentations. In Chapter 6 we present the unified segmentation result of various shapes, to see how our proposal works in practice.

5 IMPLEMENTATION

In this chapter, we provide a quick overview of the technologies that were used in Section 5.1, followed by several implementation details such as the algorithms and data structures that were used in Section 5.2. The chapter is concluded in Section 5.3 by a look at the User Interface we implemented.

5.1 TECHNOLOGIES

At the start of the project we inherited the codebase from Kustra *et al*'s earlier research project [KJT16], which contained the implementation and infrastructure for computing SSGs and classification of surface skeleton points. Their project is implemented in C++ and uses the VTK library[1] for several mesh algorithms and data structures. For fast computation of nearest neighbors, the ANN library[2] is used, which supports both k-NN and searching within a radius, using k-D trees internally for efficient spatial searches. For parallelization, both *pthreads* and *OpenMP* have been used. The pipeline has been scripted and parameterized through the use of scripting language Lua.

5.2 IMPLEMENTATION DETAILS

The cut-space segmentation steps **(a-g)** have been implemented from scratch, as detailed next, whereas the patch-type segmentation is computed using the existing implementation from [KJT16]. Finally, the validation and refinement of segments has also been implemented for the purpose of this thesis.

5.2.1 *Cut-Space Segmentation*

For computing the cut-space segmentation, the surface skeleton computation was used as is. The algorithm for SSG tracing was available for computing an importance measure, but has been altered according to the definition given in Equation (3.3) in order to compute

1 The Visualization Toolkit: http://www.vtk.org
2 Approximate Nearest Neighbor: https://www.cs.umd.edu/~mount/ANN/

the cut-space. This is accomplished by changing the stopping criteria in the original code, we now only stop after having reached the initial feature point itself. Note that only the CPU implementation of this method is used, whereas Kustra *et al* also have a GPU implementation available that would offer significant speedup.

For median filtering we use the QuickSelect algorithm[3] for real-valued sets, and a histogram—like in counting sort—for integer-valued sets, in order to compute median values in linear complexity (on average). Depth-first searches, *e.g.* for connected component detection, is implemented using stacks.

Where necessary, special care has been taken for handling duplicated vertices. This usually occurs in meshes as it allows for assigning different normals at the same point, however when considering adjacent vertices and identifying edges it is important for such vertices to be merged together. This has no effect on the output shape because only the parts that rely on this connectivity immediately transfer their results back to the original mesh. De-duplication of vertices is accomplished using VTK mesh cleaning algorithm, after which we create a mapping between the vertices of both meshes, in order to transfer information back-and-forth.

5.2.2 *Patch Validation and Refinement*

Computing the patch-type segmentation in itself has not changed from Kustra *et al*'s implementation that we started with, however additional filtering of the result has been added. For curvature analysis, the connected neighborhood $\mathcal{N}_i$ for vertex $\mathbf{v}_i$, refer back to Equation (4.3), is not computed using a nearest neighbor search as it would not take into account the connectivity of the mesh. Therefore, a breadth-first search algorithm is used that greedily assigns with each neighbor its distance to $\mathbf{v}_i$ over the edges of the mesh. Although this may overestimate the distance due to not necessarily following the shortest paths, the use of a breadth-first approach minimizes the error.

Once the median curvature has been computed in all vertices that are on a patch-border, a border-curvature-matrix is build. This symmetric matrix contains for each combination of adjacent patches the median curvature between the patches. When considering to merge patch $\mathcal{P}_y$ into $\mathcal{P}_x$, first the curvature value $\delta_{x,y}$ stored in the matrix for $(\mathcal{P}_x, \mathcal{P}_y)$ is checked against the threshold β. The merge is committed only if $\delta_{x,y} < \beta$. After merging, the matrix is updated to reflect the fact that $\mathcal{P}_y$ no longer exists, therefore all of $\mathcal{P}_y$'s adjacent patches have to become

3 Fast Median Search C implementation: http://ndevilla.free.fr/median/median/

adjacent to $\mathcal{P}_x$. If such a patch was already adjacent to $\mathcal{P}_x$ we use the maximum curvature value. Doing so ensures that we never merge two patches if they have at least one border that is significant (*i.e.* has a curvature larger than β) such that the significant border will stay intact.

5.2.3 *Part-Patch Unification*

Given the separate part-type and patch-type segmentations, with all segments having been validated, the unification process is fairly simple. For each vertex $\mathbf{v}_i$ we determine its segment unified$(\mathbf{x}_i)$ by a constant factor lookup into a 2D structure: $labels[\text{part}(\mathbf{v}_i)][\text{patch}(\mathbf{v}_i)]$. If either of the keys does not exist, the next available segment label is stored in $labels$.

5.2.4 *Segment Border Smoothing*

After the unified segmentation has been computed, the segment borders are smoothed using Laplacian smoothing, as discussed in Section 3.9. During the transformation from vertex labels to cell labels (see Section 3.8), the newly inserted vertices are linked together and stored per border. These linked lists are then consistently sorted by swapping *previous*/*next* pointers if necessary. With the ordered list of vertices we then perform several Laplacian smoothing iterations, back-projecting the vertices onto the surface boundary in each iteration. An additional constraint on the vertices near the start/end of the border is necessary. Since those vertices would otherwise be pulled inwards—due to the absence of vertices on one side—the endpoints of borders would shift apart from each other, something that is structurally impossible. For this reason, we reduce the pull factor to 0 for the first/last vertex and linearly for subsequent $n-1$ vertices. We found however that only locking the first vertex using $n = 1$ works satisfactory.

After having applied the Laplacian smoothing iterations, we have yet to incorporate the changes into the mesh itself. This is complicated, given that vertices may have moved over the surface's boundary freely. In essence, we need to compute the intersection between the shifted borders and the mesh in order to find the points on existing edges. An algorithm similar to that for cut-space computation has been implemented, but with some additional handling of edge-cases that occur due the changes of direction along the border.

We note that we have looked into a method by Lawonn *et al* [LGRP14] for more robust smoothing of borders on a mesh, unfortunately however the implementation of given method would be complex and the author was not in possession of the source code anymore. Their proposed solution for curve smoothing seems more robust and better customizable.

5.3 GRAPHICAL INTERFACE

We provide a simple Qt application for interacting with the results and being able to quickly switch between the part-type, patch-type and unified segmentation results. It allows for capturing screenshots of all segmentations and coloring options in a single operation, along with the parameters that were used, to easily obtain a set of results from the same orientation.

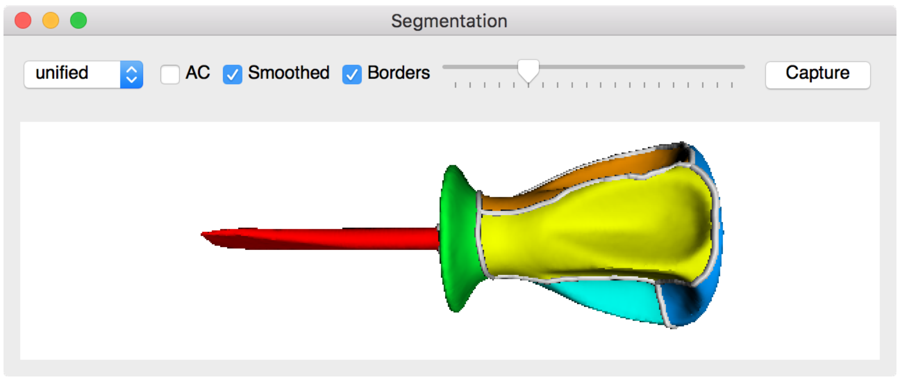


FIGURE 5.1 A user interface is provided to interact with the segmentation results.

6 RESULTS

In this chapter we first show the results of the part-type segmentation in Section 6.1, including the method's limitations and a discussion on how changing parameters affects the resulting segmentation. Next, Section 6.2 is similar in structure but focuses on the results of our unification proposal. Then, Section 6.3 briefly discusses the parameters that are available for tweaking the results. Finally in Section 6.4 a listing of execution times is presented to indicate the performance of the various stages of the pipeline

In order for the results to show the benefits of using a unified segmentation approach, we have selected a broad range of various kinds of shapes. These include some typical natural shapes that are often used for showcasing part-based segmentation methods, but also more challenging anatomical shapes are included. Such shapes have complex geometries and contain both parts and patches and are therefore well suited for our unification approach. Furthermore, by not focusing on certain kinds of shapes we can assess the robustness of our method and see under which circumstances our methods may fail to work as desired.

6.1 CUT-SPACE SEGMENTATION

We first give a comparison of our cut-space results with those of Feng *et al* in Figure 6.1. We find that the results are similar in most cases, with a few notable exceptions. We notice for the cow shape how our method arguably suffers from over-segmentation, which we may attribute to *threshold aliasing* as explained in Section 3.4.2. In the next section we will see some more shapes that show this limitation. Next is the screwdriver for which our method produces a more natural result, although we note Feng *et al* may produce a similar result by changing its threshold selection procedure.

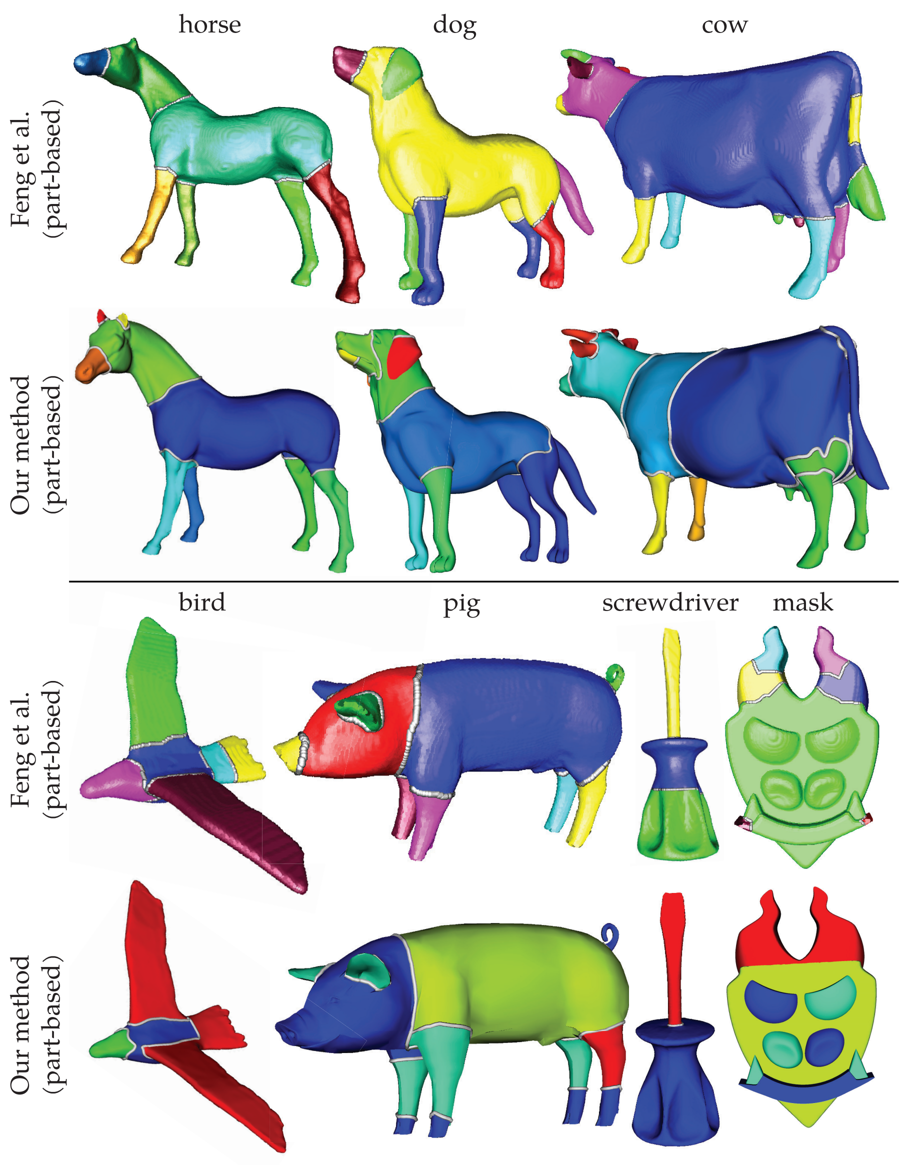


FIGURE 6.1 Comparing our part-based cut-space segmentation results with Feng *et al* segmentation results.

Continuing with Figure 6.2, another notable difference is seen for the scapula and tooth shapes where our mesh based method produces more parts compared to Feng's results. In the case of scapula we may argue that the additional part is undesirable, in which case we may choose to eliminate it by increasing the minimum size criterion. In the case of the tooth shape it really depends on the application which result should be preferred, as both are equally valid. For all other examples we notice only few differences, indicating that our mesh-based translation of Feng's voxel-based method behaves similarly.

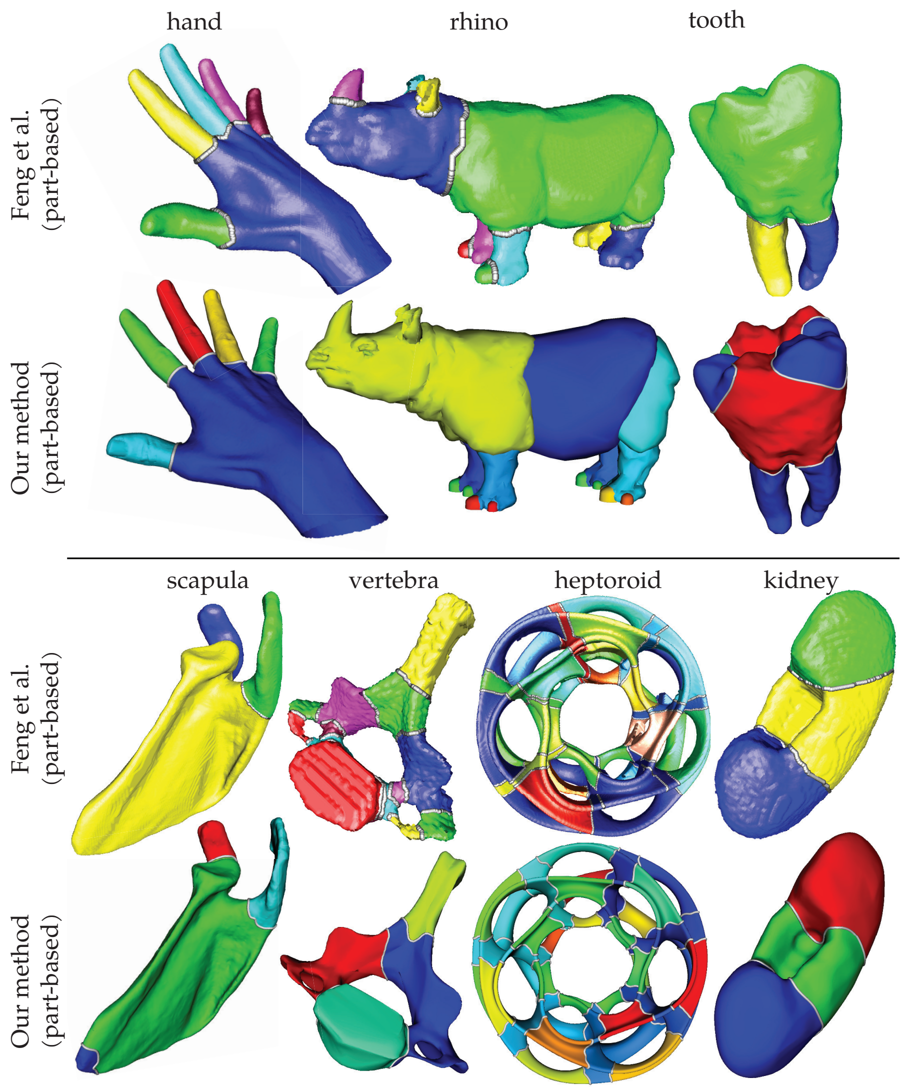


FIGURE 6.2 Comparing our part-pased cut-space segmentation results with Feng *et al* segmentation results.

We include another set of shapes that shows how our method handles non-part-like shapes in Figure 6.3. For each of those shapes we note that our method did result in the whole shape being considered as a single part, which is important for suitable unified segmentations, as we will look at in the next section.

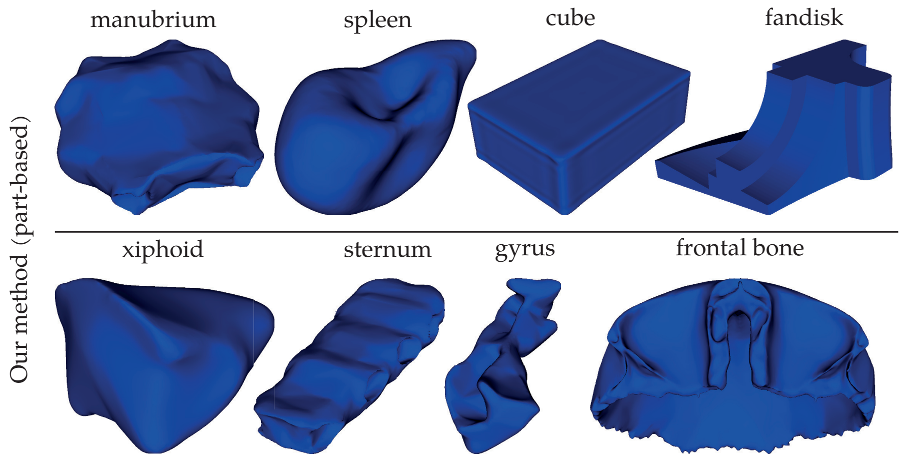


FIGURE 6.3 Patch-like shapes result in no parts to be found.

6.1.1 *Comparison*

To evaluate the results of our proposed part-based segmentation method we include a quick comparison against related work. Figure 6.4 shows the results of the horse shape, an example that is widely used in shape segmentation papers. The figure differentiates between non skeleton-based methods in the top row (**a–d**) and skeleton-based methods in the bottom row (**e–h**). We observe that the skeleton-based methods produce more natural results, considering the segment border quality and over-segmentation that occurs when using non skeleton-based methods. This is due to the skeleton's ability to capture the shape's main axis, around which parts are then identified. Recall from Section 3.3.1 that the θ-SMA measure was used as regularization measure in order to only consider skeleton points that approach this main axis, thereby satisfying the *local orthogonal* property of cuts.

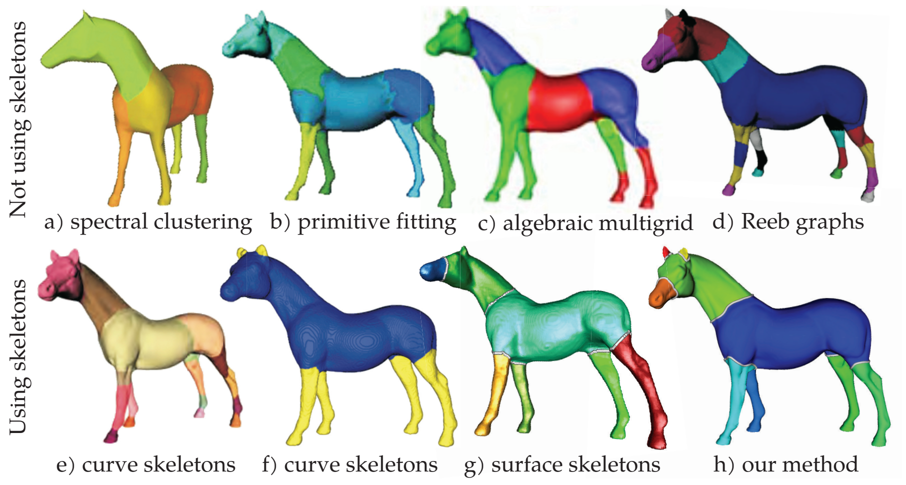


FIGURE 6.4 Comparison of eight part-based segmentation methods. **(a)** Liu and Zhang [LZ04] **(b)** Attene *et al* [AFS06] **(c)** Clarenz *et al* [CGR+04] **(d)** Tierny *et al* [TVD07] **(e)** Lien *et al* [LKA06] **(f)** Reniers *et al* [RT08a] **(g)** Feng *et al* [FJT15a] **(h)** our method.

From Figure 6.4 we may not draw the conclusion that surface skeleton methods—or curve skeletons for that matter—produce generally better results when compared to other approaches, given that only a very restrictive comparison is presented. We only note that our method performs similarly to the curve skeleton methods in **(e–f)**, so we find that using surface skeleton is a viable approach for shape segmentation.

6.1.2 *Limitations*

We show here several examples of how *threshold aliasing*, see Section 3.4.2, may result in undesirable segmentations. This phenomenon occurs due to our usage of histograms to derive thresholds and is not an inherent issue of cut-space segmentation in itself. In Figure 6.5 we show several examples where this phenomenon occurs. The green circles show the segment borders that are desirable, whereas the red circles designate borders that are introduced because they occur at the same threshold-value.

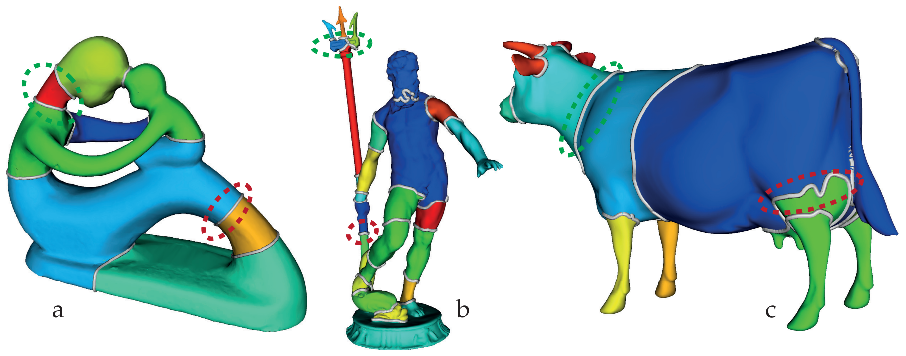


FIGURE 6.5 Threshold aliasing causes over-segmentation.

A solution to this problem may be found in using hierarchical clustering of the cut-space. Feng *et al* have already shown that replacing histogram based clustering with hierarchical clustering works well, while preventing above problems [FJT15a].

6.2 UNIFIED SEGMENTATION

From the shapes in Figure 6.3 it becomes clear that a part-based segmentation method is unsuitable for certain kinds of shapes. In this section we show how such shapes are segmented meaningfully using our unified approach, and show how the part-type shapes are affected.

The results for typical patch-type shapes is shown in Figure 6.6. These results are actually identical to the patch-type results as given by Kustra *et al* [KJT16], because all patches are considered to be valid and therefore no additional filtering has taken place. In Section 6.3 we detail how changing parameters may influence the granularity of patches.

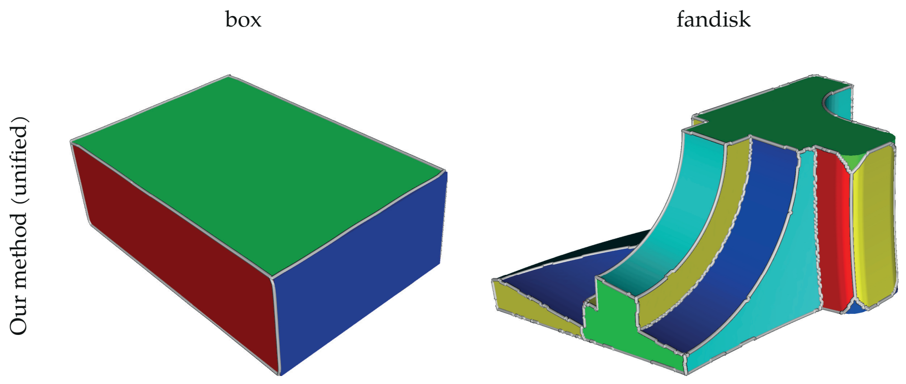


FIGURE 6.6 Segmentation result of typical patch-type shapes.

In Figure 6.7 several shapes are shown for which the unified results include both parts and patches, a characteristic that has not been possible with earlier methods. The screwdriver now includes patches on the handle and the top and bottom of the wings of the bird are considered to be separate patches. The hip bone and scapula shapes are more examples where adding patches to the part-based result gives a richer, but still sensible, segmentation.

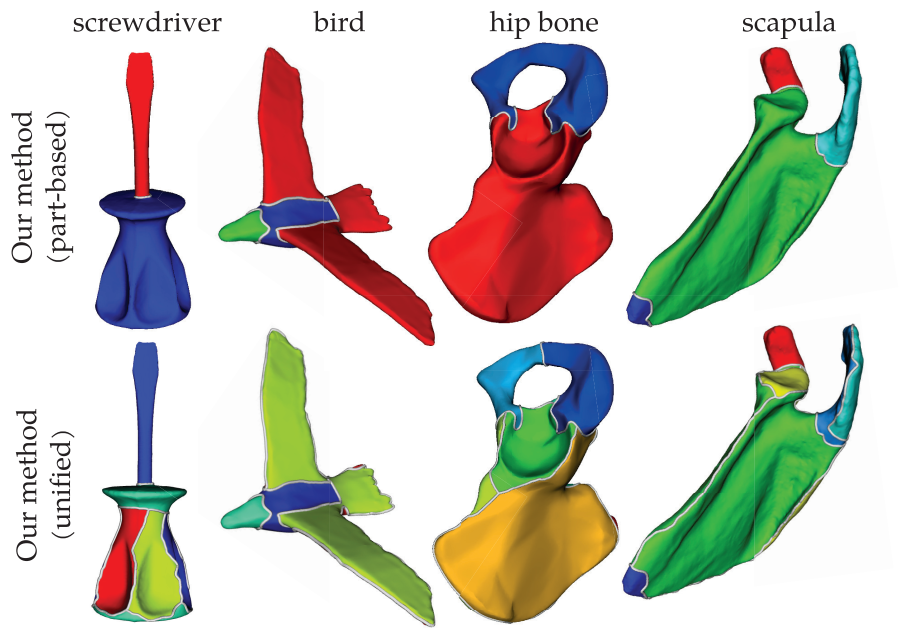


FIGURE 6.7 Several shapes show the benefit of using a unified segmentation method.

Although the patches we find are computed using Kustra *et al*'s method, the derived unified result may show differences due to our additional filtering stages. Figure 6.8 gives a comparison between Kustra *et al*'s raw results versus the unification method as presented here. The *scapula* shape shows how an insignificant border has been detected and removed, resulting in an improved segmentation. The *xiphoid* shape shows well the effect of smoothening segment borders, and finally for the *frontal bone* we notice how our method has reduced the number of patches, due to its detection strategy for insignificant borders.

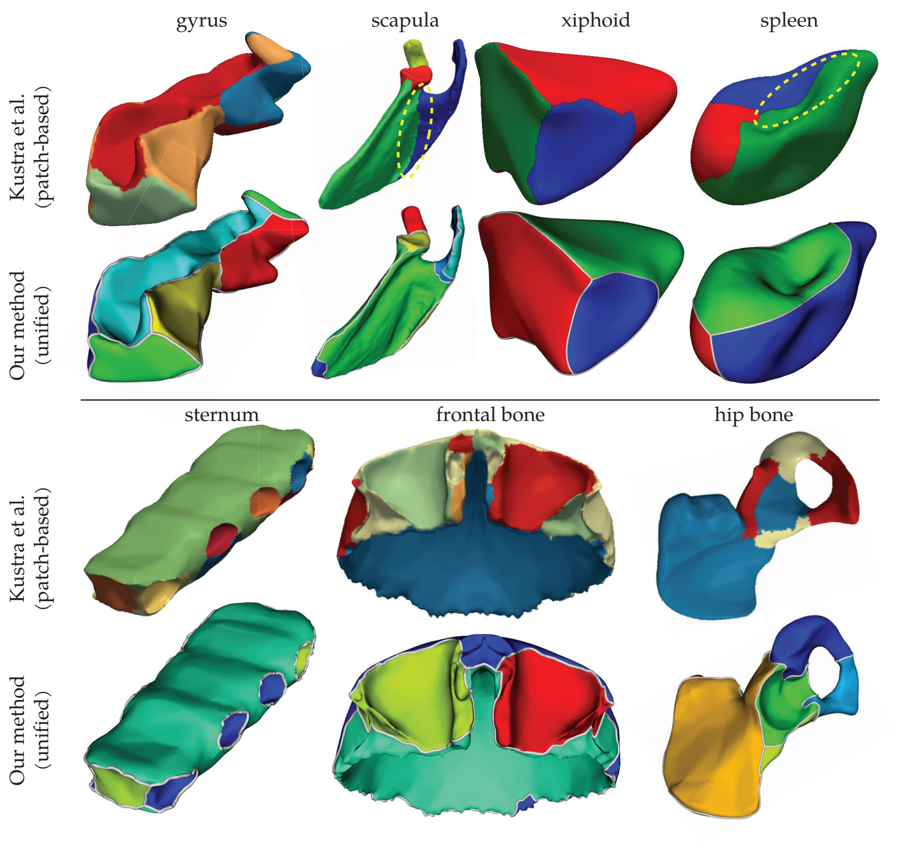


FIGURE 6.8 Comparison with results from Kustra *et al*.

6.2.1 *Limitations*

Although our unification method has given good results for many shapes, it does come with some limitations. The strategy of always copying parts into the unified segmentation assumes that detected parts are always desired in the final result, whereas it may occur that using patches is more appropriate. An example of such a case is presented in Figure 6.9, which shows a similar case as was given before in Figure 3.14. The difference now is that a larger threshold was selected, such that the edges have started to meet. Once this happens, our validation procedures fail and the unification result is far from desired.

A limitation that is inherent to our usage of the point-cloud-skeletonization by Jalba *et al* [JKT13] is that we rely on a regularly and densely sampled input shape, because the surface skeleton would otherwise be of low quality and unsuitable for further processing. By computing a dense supersampling of such shapes it is however possible to mitigate this problem.

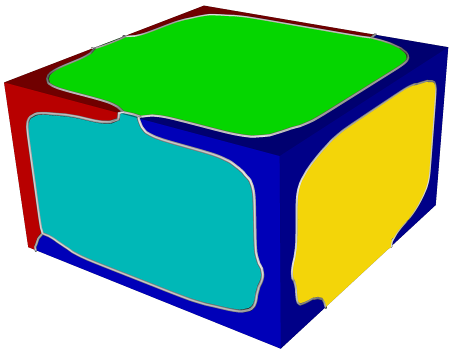


FIGURE 6.9 Undesirable parts may not be detected, in which case our unification approach yields unsatisfiable results.

6.3 PARAMETERS

Our method comprises a large number of parameters that may be individually tweaked to obtain desirable results. We present here the few parameters that have the largest influence on the resulting segmentation, alongside a discussion of how changing these parameters affects the final results.

Cut-Space Segmentation

For the cut-space segmentation, the number of segments can be influenced by tweaking the histogram analysis parameters H_{peak} and H_{decrease}. The value of H_{peak} is 0.01 by default, meaning that the minimum number of cuts in a single histogram bin should be at least 1% of the total number of cuts in order to qualify as a histogram peak. Increasing this value may therefore cause less peaks to be found and as such produce fewer segments. For some of the results that were presented earlier we used a slightly higher value of up to 0.01 in order to prevent over-segmentation.

The default value for H_{decrease} was set at 0.25, indicating that a valley bin must not exceed 25% of the number of cuts as represented by the corresponding peak. Hence, increasing this threshold allows for larger bins to be considered as valley, therefore allowing for more thresholds to be introduced. This causes the number of segments to increase. In our experience, both H_{peak} and H_{decrease} are often tweaked in unison in order to obtain the desired segmentation.

Unification

Considering that the unification procedure is just merging the part-based and patch-based results, it does not involve any parameters. We may however influence to what extend patches are introduced into the part-based segmentation by adjusting the neighborhood radius r that is involved in the curvature measure from Section 4.4.1, and by changing the threshold β that is used for controlling the minimally required angle on a patch boundary.

For the neighborhood radius r we selected a default value of 2.5% of the shape's maximum dimension, whereas the threshold β is set to 53° by default. In Figure 6.10 we present a cube with slightly rounded edges, ow which the four edges in the side have a slightly larger radius than the edges towards the top and bottom. From this example it can be seen that we may choose to segment only the bottom and top in their own patch—because they are separated by edges that are slightly more sharp compared to the side edges—or all of the sides separately.

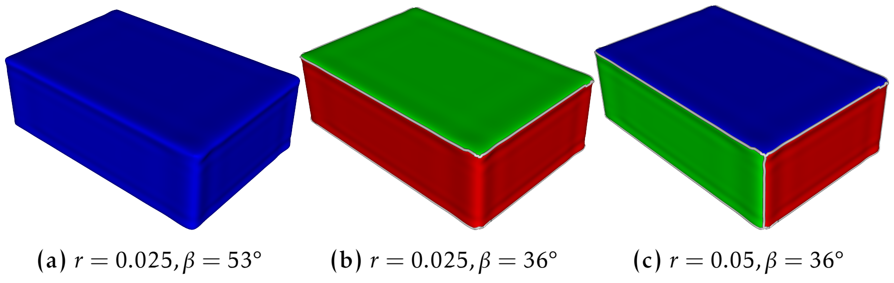

(a) $r = 0.025, \beta = 53°$ (b) $r = 0.025, \beta = 36°$ (c) $r = 0.05, \beta = 36°$

FIGURE 6.10 Tweaking the parameters r and β in unison in order to influence to what extend patches are introduced into the unified segmentation.

Note that we may obtain the result in Figure 6.10b from multiple distinct configurations: either the neighborhood radius r is changed such that the blunt edges do not fit in that neighborhood while the sharper edges do, such that the curvature differs enabling us to discriminate between the two kinds of edges. Analogous, we may choose to only adjust β to be in-between the curvature measures of the two kinds of edges.

6.4 PERFORMANCE

To see how our method performs in terms of processing time, Table 6.1 gives an overview of various shapes and their execution times, broken down into the several stages. These results have been established on a MacBook Pro with an Intel Core i7 4850HQ processor and 16GB of memory. It is clear from this table that performing the cut-space segmentation is rather costly, especially when compared to patch-based segmentation. We note here however that most of the cut-space time is spent on computing the shortest straightest geodesics, for which we currently only use a CPU implementation. A GPU implementation for computation of SSGs has been presented by Jalba *et al* [JKT13] and may show significant performance improvements, as their GPU implementation shows speed-ups of circa 60% when compared to a multi-threaded CPU implementation.

	#vertices	#cuts	t_{skel}	t_{cuts}	t_{patch}	$t_{\text{unification}}$
				(in seconds)		
Bird	47,184	26,485	6.3	262.0	5.3	14.1
Cow	137,862	16,447	23.0	242.3	10.9	63.0
Hand	49,546	2,815	7.3	47.9	4.2	11.1
Heptoroid	79,056	63,876	5.0	539.1	6.3	5.8
Horse	49,749	7,453	5.6	63.2	5.2	6.4
Hound	16,158	364	0.7	5.1	2.0	2.7
Kidney	30,389	9,986	6.5	177.7	4.0	27.0
Pig	4,800	756	0.2	2.0	0.6	0.6
Scapula	117,432	83,340	17.3	2325.7	11.0	26.6
Vertebra	22,789	7,779	0.2	57.4	2.4	2.2

TABLE 6.1 Performance figures of our segmentation pipeline.

6.5 SUMMARY

In this chapter we have presented the results of our cut-space segmentation method for mesh shapes along with our unification method to compute part-patch segmentations. Our cut-space method shows good results which are similar to the results of the original proposal by Feng *et al*. The unification approach shows to effectively segment both typical part and patch shapes without suffering from over-segmentation. The set of parameters allow for tweaking the results while a default set of parameters gives mostly good results. In terms of performance our method is somewhat slow, but we note that several optimization possibilities are still possible.

7 CONCLUSIONS

This chapter brings this thesis to a conclusion. In Section 7.1 we present an answer to the research questions as they were given in Chapter 1, followed in Section 7.2 with a discussion on our findings. Next, we include a reference to a book chapter publication this work has led to in Section 7.3. Lastly, we see the possibility of future work in several directions, of which an overview is given in Section 7.4.

7.1 CONCLUSION

Recall from Chapter 1 that we presented two research questions to be answered by this research project. The **first research question** is as follows:

1. *How can we design a generalized part/patch-based segmentation algorithm?*

In this thesis we presented such an algorithm in Chapter 4 that satisfies the desirable properties we set out, *i.e. hybrid*, *robust*, *configurable* and *balanced*. Although our proposed unification approach in itself boils down to the merging of both a part-type and patch-type segmentation together, it requires that the vailidy of parts and patches has been verified in order to prevent over-segmentation and visually unexpected results. We have shown this strategy to work well using our proposed cut-space algorithm for mesh shapes together with Kustra *et al*'s patch-based segmentation method [KJT16].

We believe however that our proposed algorithm is also valid when other segmentation algorithms are used to compute the two base segmentations, as long as their results have been validated to be believed sensible. Key in this validation process is our proposed usage of a Gauss map to quantify a part's correctness, see Section 3.7. Also for the validation of patches we have presented an effective and robust curvature measure Section 4.4. Both of these validation procedures are essential for allowing any shape to result in a sensible unified segmentation, thereby our proposal is indeed capable as a generalized part/patch-based method.

We continue with presenting an answer to the **second research question**:

2. *Can we use cut-space segmentation for mesh models and how does it compare to Feng* et al*'s voxel-based technique? [FJT15b]*

We focused on answering this question in Chapter 3. A method for computing the cut-space was presented in Section 3.3 that uses SSGs for the computation of cuts. Because of the characteristics of these SSGs and difficulties with selecting the cuts on segment borders, we choose a different approach for segmenting the mesh surface based on the cut-space than was proposed by Feng *et al*, refer back to Section 3.6. With these adaptations of the voxel-based method we have shown that it is indeed possible to use the cut-space segmentation method for mesh shapes.

In the comparison we did against Feng *et al*'s results we found that our method for meshes yields similar results as the voxel-based approach. Because of the usage of meshes however, our method does not have the resolution limitations that are inherent to voxel-based representations.

7.2 DISCUSSION

In this thesis we have explored a method for computing a unified part-patch segmentation for mesh shapes. To our knowledge the proposed framework is the first in its kind, allowing for computing a sensible segmentation that may either describe parts, patches, or possibly both as appropriate for the shape at hand.

In the process of implementing this framework we have also presented an adaptation of cut-space segmentation by Feng *et al* [FJT15b] to work on mesh shapes instead of voxel-based shapes. This kind of segmentation method uses a surface skeleton which is favorable to the usage of curve skeletons, given that surface skeletons are shape descriptors that capture the full shape as opposed to only the main topology of a shape. We found that our mesh-based implementation produces results that are similar to Feng *et al*'s voxel-based results.

In terms of results, we have shown that our proposed method produces good results, also in comparison to related work. Our method produces visually similar results in most cases. We have not been able to do a qualitative comparison given that assessing a segmentation's quality is hard for many shapes, because each segmentation method may have different applications and therefore different characteristics; none of which can be considered the *ground truth*. Another problem is that most related work focuses only on the kind of shapes that it is designed for, *i.e.* part-based segmentation methods only deal with natural shapes and vice-versa for patch-based approaches that only show their capability of extracting patches from *e.g.* a mechanical shape. This leaves a gap in showcasing their ability to be robust against a kind of shape that it is not primarily designed for.

As outlined earlier in this work, we see some limitations of our work. We have presented shapes for which the unified result is not ideal, or cases where cut-space segmentation suffers from *threshold aliasing*. Furthermore, we recognize a limitation in terms of speed which currently is suboptimal, although we could turn to using a GPU implementation for SSG computation to see significant speedup.

Lastly, we have not presented examples that show our method's ability to be pose-, scale- and rotation-invariant, hence we cannot comment on this desirable feature of a segmentation algorithm. The evaluation of the results we did is however typical for these kind of shape segmentation papers.

7.3 PUBLICATION

The results of this thesis have been accepted for publication as a book chapter in *"Skekelonization: Theory, Methods and Applications"* by *Elsevier*, in collaboration with C. Feng, A.C. Jalba and my supervisors J.L. Kustra and A.C. Telea [KFJ+16].

7.4 FUTURE WORK

We see a couple of possible directions this research could be followed up with, as described next.

Clustering-based Partitioning

In a recent extension of Feng *et al*'s initial work on cut-space analysis, in order to solve the inherit problems of using a histogram as discussed in Section 3.4.2, they were able to obtain significantly improved part-based segmentation results by using a hierarchical clustering method to find segments, instead of a histogram [FJT15a]. By defining a dissimilarity function to compare cuts, computed from the cuts' lengths and Euclidean distance between skeleton points, the inherit problems of using a histogram were avoided and in turn resulted in better results. We foresee that using a similar approach would allow for similar improvements, as this step in the pipeline is independent of the shape's representation. Therefore, future work could focus on experimenting with replacing the histogram with clustering-based partitioning.

Unification Using Alternative Segmentation Approaches

In this work we proposed a part-type segmentation approach derived from Feng *et al*'s cut-space segmentation method [FJT15b] and used Kustra *et al*'s patch-type segmentation approach [KJT16] to ultimately compute a unified segmentation using a combination of the two methods. Further research could experiment with using alternative segmentation methods to see how the results compare against our proposal.

Quantifying Segmentation Quality

The results as presented in this thesis were tweaked according to what looked good in the eyes of the author. Given the subjectivity involved, comparing results with different methods also becomes a subjective process. There currently is no way of measuring the quality of a given segmentation results, as no *ground truth* has been established. A database of shapes and their generally accepted desired segmentation would enable objectively comparing different segmentation methods and selecting the best choice among differently configured parameters.

GPU Optimization

Given that the methods described in this thesis are designed for mesh representations, we anticipated to optimize the implementation using GPGPU technologies as meshes are generally very well handled by GPUs. For this work however, the segmentation pipeline is fully implemented on the CPU as several stages contain data dependencies that do not scale well to the architecture of a GPU. Therefore, additional work is required in order to efficiently execute the proposed methods on a GPU.

BIBLIOGRAPHY

[AFS06] M. ATTENE, B. FALCIDIENO, and M. SPAGNUOLO. "Hierarchical mesh segmentation based on fitting primitives". In: *The Visual Computer* 22.3 (2006), pp. 181–193 (16, 17, 57).

[BBB+97] J. BLOOMENTHAL, C. BAJAJ, J. BLINN, M.-P. CANI, A. ROCKWOOD, B. WYVILL, and G. WYVILL. *Introduction to Implicit Surfaces*. Morgan Kaufmann, 1997 (1, 2).

[BHS89] M. BRAUNSTEIN, D. HOFFMAN, and A. SAIDPOUR. "Parts of visual objects: and experimental test of the minima rule". In: *Perception* 18 (1989), pp. 817–826 (10).

[BKP+10] M. BOTSCH, L. KOBBELT, M. PAULY, P. ALLIEZ, and B. LÉVY. *Polygon Mesh Processing*. A K Peters, 2010 (2).

[Blu67] H. BLUM. "A Transformation for Extracting New Descriptors of Shape". In: *Models for the Perception of Speech and Visual Form* (1967), pp. 362–380 (2, 7).

[BMR+99] F. BERNARDINI, J. MITTLEMAN, H. RUSHMEIER, C. SILVA, and G. TAUBIN. "The Ball-Pivoting Algorithm for Surface Reconstruction". In: *Visualization and Computer Graphics* 5.4 (1999), pp. 349–359 (21).

[CGR+04] U. CLARENZ, M. GRIEBEL, M. RUMPF, M. SCHWEITZER, and A. TELEA. "Feature sensitive multiscale editing on surfaces". In: *The Visual Computer* 20.5 (2004), pp. 329–343 (12, 46, 57).

[CM02] D. COMANICIU and P. MEER. "Mean Shift: A Robust Approach Toward Feature Space Analysis". In: *Pattern Analysis and Machine Intelligence* 24.5 (2002), pp. 603–619 (25).

[CSM07] N. D. CORNEA, D. SILVER, and P. MIN. "Curve-Skeleton Properties, Applications, and Algorithms". In: *Visualization and Computer Graphics* 13.3 (2007), pp. 530–548 (7).

[EKS06] H. EDELSBRUNNER, D. KIRKPATRICK, and R. SEIDEL. "On the Shape of a Set of Points in the Plane". In: *IEEE Transactions on Information Theory* 29.4 (2006), pp. 551–559 (21).

[FJT15a] C. FENG, A. C. JALBA, and A. C. TELEA. "Improved Part-Based Segmentation of Voxel Shapes by Skeleton Cut Spaces". In: *Mathematical Morphology – Theory and Applications* 1.1 (2015), pp. 60–78 (16, 20, 57, 58, 67).

[FJT15b] C. FENG, A.C. JALBA, and A.C. TELEA. “Part-Based Segmentation by Skeleton Cut Space Analysis”. In: *Mathematical Morphology and Its Applications to Signal and Image Processing*. 2015, pp. 607–618 (4, 5, 15, 16, 19, 20, 37, 66, 68).

[FLM03] M. FOSKEY, M.C. LIN, and D. MANOCHA. “Efficient Computation of a Simplified Medial Axis”. In: *Solid Modeling and Applications*. ACM, 2003, pp. 96–107 (23).

[GF08] A. GOLOVINSKIY and T. FUNKHOUSER. “Randomized cuts for 3D mesh analysis”. In: vol. 27. 5. 2008, pp. 1–12 (15).

[GK04] P. GIBLIN and B.B. KIMIA. “A formal classification of 3D medial axis points and their local geometry”. In: *Pattern Analysis and Machine Intelligence* 26.2 (2004), pp. 238–251 (8).

[GWH01] M. GARLAND, A. WILLMOTT, and P.S. HECKBERT. “Hierarchical Face Clustering on Polygonal Surfaces”. In: *Symposium on Interactive 3D graphics*. 2001, pp. 49–58 (4, 16).

[HR84] D. HOFFMAN and W. RICHARDS. “Parts of recognition”. In: *Cognition* 18 (1984), pp. 65–96 (10).

[JKT13] A.C. JALBA, J.L. KUSTRA, and A.C. TELEA. “Surface and curve skeletonization of large 3D models on the GPU”. In: vol. 35. 6. 2013, pp. 1495–1508 (20, 21, 24, 25, 60, 63).

[KFJ+16] J. KOEHOORN, C. FENG, A.C. JALBA, J.L. KUSTRA, and A.C. TELEA. “Unified Part-Patch Segmentation of Mesh Shapes using Surface Skeletons”. In: *Skekelonization: Theory, Methods and Applications*. Elsevier, forthcoming 2016 (67).

[KJT16] J.L. KUSTRA, A.C. JALBA, and A.C. TELEA. “Computing Refined Skeletal Features from Medial Point Clouds”. In: *Pattern Recognition Letters* 76.1 (2016), pp. 13–21 (4, 9, 18, 40, 41, 49, 58, 65, 68).

[KLT05] S. KATZ, G. LEIFMAN, and A. TAL. “Mesh segmentation using feature point and core extraction”. In: *The Visual Computer* 21.8 (2005), pp. 649–658 (12).

[KT03] S. KATZ and A. TAL. “Hierarchical mesh decomposition using fuzzy clustering and cuts”. In: 2003, pp. 954–961 (11).

[LGRP14] K. LAWONN, R. GASTEIGER, C. RÖSSL, and B. PREIM. “Adaptive and robust curve smoothing on surface meshes”. In: *Computers & Graphics* 40 (2014), pp. 22–35 (51).

[LKA06] J. LIEN, J. KEYSER, and N. AMATO. “Simultaneous shape decomposition and skeletonization”. In: *Solid and Physical Modeling*. 2006, pp. 219–228 (57).

[**LWTH01**] X. LI, T. W. WOON, T. S. TAN, and Z. HUANG. "Decomposing polygon meshes for interactive applications". In: *Interactive 3D Graphics*. 2001, pp. 35–42 (10, 11).

[**LZ04**] R. LIU and H. ZHANG. "Segmentation of 3D meshes through spectral clustering". In: *Proc. Pacific Graphics*. 2004, pp. 298–305 (57).

[**RT07**] D. RENIERS and A. C. TELEA. "Skeleton-based Hierarchical Shape Segmentation". In: *Shape Modeling and Applications*. IEEE, 2007, pp. 179–188 (4, 13, 14).

[**RT08a**] D. RENIERS and A. C. TELEA. "Part-type Segmentation of Articulated Voxel-Shapes using the Junction Rule". In: *Computer Graphics Forum* 27.7 (2008), pp. 1845–1852 (3, 13, 14, 57).

[**RT08b**] D. RENIERS and A. C. TELEA. "Patch-type Segmentation of Voxel Shapes using Simplified Surface Skeletons". In: *Computer Graphics Forum* 27.7 (2008), pp. 1837–1844 (3, 4, 17, 18, 41).

[**RvWT08**] D. RENIERS, J. J. van WIJK, and A. C. TELEA. "Computing Multiscale Curve and Surface Skeletons of Genus 0 Shapes Using a Global Importance Measure". In: *Visualization and Computer Graphics* 14.2 (2008), pp. 355–368 (23).

[**SAdB14**] L. SERINO, C. ARCELLI, and G. S. di BAJA. "From skeleton branches to object parts". In: *Computer Vision and Image Understanding* 129 (2014), pp. 42–51 (14).

[**SdBA11**] L. SERINO, G. S. di BAJA, and C. ARCELLI. "Using the skeleton for 3D object decomposition". In: *Image Analysis*. 2011, pp. 447–456 (14).

[**Sha04**] A. SHAMIR. "A formulation of boundary mesh segmentation". In: *3D Data Processing, Visualization and Transmission*. IEEE, 2004, pp. 82–89 (3).

[**SSC08**] L. SHAPIRA, A. SHAMIR, and D. COHEN-OR. "Consistent mesh partitioning and skeletonisation using the shape diameter function". In: *The Visual Computer* 24 (2008), pp. 249–259 (14, 26).

[**ST04**] R. STRZODKA and A. TELEA. "Generalized Distance Transforms and Skeletons in Graphics Hardware". In: *Proceedings of the Sixth Joint Eurographics - IEEE TCVG Conference on Visualization*. VISSYM '04. Eurographics Association, 2004, pp. 221–230 (8).

[**TDS+16**] A. TAGLIASACCHI, T. DELAME, M. SPAGNUOLO, N. AMENTA, and A. TELEA. "3D Skeletons: A State-of-the-Art Report". In: *Computer Graphics Forum* 35.2 (2016), pp. 573–597 (3).

[TVD07] J. Tierny, J. Vandeborre, and M. Daoudi. “Topology driven 3D mesh hierarchical segmentation”. In: *Shape Modeling and Applications*. 2007, pp. 215–220 (12, 13, 57).

[TWH00] R. Tibshirani, G. Walther, and T. Hastie. “Estimating the number of clusters in a dataset via the Gap statistic”. In: *Statistical Methodology: Series B* 63.2 (2000), pp. 411–423 (26).