CHAPTER

Unified part-patch segmentation of mesh shapes using surface skeletons

4

Joost Koehoorn*, Cong Feng*, Jacek Kustra†, Andrei Jalba‡, Alexandru Telea*

Institute Johann Bernoulli, University of Groningen, Groningen, The Netherlands Philips Research, Eindhoven, The Netherlands† Department of Mathematics and Computer Science, Technical University Eindhoven, The Netherlands‡*

Contents

4.1 Introduction 90
4.2 Related Work 91
4.2.1 Skeletonization 91
4.2.2 Shape Segmentation 93
4.2.2.1 Part-Based Segmentation 93
4.2.2.2 Patch-Based Segmentation 95
4.2.3 Summary of Challenges 95
4.3 Method 96
4.3.1 Preliminaries 96
4.3.2 Regularized Surface Skeleton Computation 97
4.3.3 Cut-Space Computation 99
4.3.4 Cut-Space Partitioning 100
4.3.4.1 Histogram Valley Detection 101
4.3.4.2 Histogram-Based Cut Space Partitioning 102
4.3.5 Partitioning the Full Surface Skeleton 102
4.3.6 Partition Projection to Surface 104
4.3.7 Part-Based Partition Refinement 105
4.3.8 Unified (Part and Patch) Segmentation 107
4.3.8.1 Patch-Type Segmentation Using Surface Skeletons 108
4.3.8.2 Unification Desirable Properties 110
4.3.8.3 Unification Method 110
4.4 Results 112
4.5 Discussion 115
4.6 Conclusion 119
References 120

Skeletonization. DOI: 10.1016/B978-0-08-101291-8.00005-5

4.1 INTRODUCTION

Shape segmentation is an important problem in many application domains such as computer-aided design, computer graphics, scientific visualization, and medical imaging. Informally put, shape segmentation aims at partitioning a given shape into several components or segments that capture application-specific part–whole relations as well as possible. Segmentation enables several shape processing and shape analysis tasks such as editing and content creation, identifying important features that occur in a large dataset, and shape matching, retrieval, and registration [46,3]. Segmentation methods can be roughly classified into *part-based* methods, which aim to split articulated shapes into the components that would be perceived as naturally distinct by humans [40], and *patch-based* methods, which aim to find quasi-flat components separated by edges on synthetic faceted models [41].

Shapes are typically encoded following a boundary representation or a volume representation. Boundary (explicit) representations capture the surface that partitions the shape interior from the surrounding exterior space, using various sampling and reconstruction schemes, e.g. polygonal meshes or point clouds [7]. Volume (implicit) representations, such as voxel models, store a densely sampled labeling of the space in which shapes are embedded, marking points as interior vs exterior [5]. Both representations have their advantages and limitations: Voxel volumes are easy to create and manipulate, but can be very expensive when high resolution is needed; surface meshes can efficiently model high-resolution shapes, but mainly support operations focusing on the shape surface, rather than its volumetric structure.

Skeletal representations are a third way to represent shapes. Informally put, skeletons jointly capture the geometry, symmetry, and topology of a shape in compact ways. 3D shapes admit two types of skeletons: *Surface* skeletons are the locus of centers of maximally inscribed spheres in the shape and as such capture geometry, symmetry, and topology. They generalize to 3D the well-known concept of a 2D symmetry axis [6]. *Curve* skeletons are one-dimensional structures locally centered in the shape that mainly capture a shape's part–whole structure. Skeletons combine the compactness of boundary representations with the ability to model and reason about volumetric properties. Additionally, they provide explicit and efficient access to a shape's symmetry structure and part–whole properties. As such, skeletal representations are a valuable tool to design shape segmentation methods [52].

Many skeleton-based segmentation methods have been proposed [52]. However, most of these methods use only the topological information encoded by *curve* skeletons. As such, they target mostly part-based segmentation of organic articulated shapes. Producing patch-based segmentations of faceted synthetic shapes or, more generally, mixed part-patch segmentations of shapes that fall between the two categories (articulated, faceted) is rarely handled [41,9,26].

Recently, surface skeletons have been used to produce part-based segmentations of 3D shapes [17,16]. A key to this idea is the construction of a so-called *cut space* containing a large number of well-designed cuts that partition the shape in ways similar to how a human would cut it. By analyzing this space a small number of cuts

is retained to yield the final shape segmentation. Although the method was shown to deliver good results, it has several limitations. First, it only produces part-based segmentations although it uses the surface skeleton that fully describes any type of shape (articulated or faceted). Secondly, it only handles voxel representations and as such is very expensive, or even prohibitive to use, for high-resolution models.

In this chapter, we extend the skeleton cut-space segmentation proposal in [16] with the following main contributions:

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

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

The remainder of this chapter is structured as follows. Section 4.2 reviews related work in part-based and patch-based shape segmentation, with a focus on skeleton-based methods. It also introduces the cut-space idea in [16]. Section 4.3 details our method, explaining the changes and enhancements proposed to the original cut-space segmentation. Section 4.4 presents several results of our method and compares these with related skeleton-based segmentation methods. Section 4.5 discusses our proposal. Section 4.6 concludes the chapter.

4.2 RELATED WORK

Given that our focus is skeleton-based segmentation methods, we proceed by introducing necessary background on skeletonization (Section 4.2.1). Next, we overview several part- and patch-based skeleton-based segmentation methods, outlining their advantages and limitations (Section 4.2.2).

4.2.1 SKELETONIZATION

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

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

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

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

In the above, $\mathbf{f}_1$ and $\mathbf{f}_2$ are two of the contact points with $\partial\Omega$ of the maximally inscribed sphere in Ω centered at $\mathbf{x}$ and of radius $DT(\mathbf{x})$. Such points are also known as *feature points* [21,52], whereas the vectors $\mathbf{f}_i - \mathbf{x}$ are known as *spoke vectors* [49]. These definitions are captured by the so-called *feature transform*

$$FT_{\partial\Omega}(\mathbf{x} \in \Omega) = \arg\min_{\mathbf{y} \in \partial\Omega} \|\mathbf{x} - \mathbf{y}\|, \tag{4.3}$$

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

Surface skeletons of 3D shapes implied by the definition in Eq. (4.2) consist of a set manifolds with boundaries, or skeletal sheets, that meet along so-called Y-intersection curves [14]. The pair $(S_{\partial\Omega}, DT_{\partial\Omega}|S_{\partial\Omega})$ is called the medial axis transform (MAT) of Ω [49]. The MAT can be used to fully reconstruct a shape, e.g., by computing the union of balls centered at points on $S_{\partial\Omega}$ and having as radii the values of $DT_{\partial\Omega}$ at the respective points. As such, the MAT provides a third type of shape representation, along boundary and volumetric ones.

Skeleton points can be classified by the order of tangency of their maximally inscribed spheres with $\partial\Omega$ [19]. This classification enables several applications such as robust edge detection on 3D surfaces [39,26], finding Y-intersection curves for patch-based segmentation [30,41], and surface reconstruction from point clouds [9]. Until recently, surface skeletons have been hard to compute for large and complex shapes [43,21]. Recent methods significantly alleviated such issues for both voxel [1, 24] and mesh [34,23] representations.

Besides surface skeletons, 3D shapes admit also *curve* skeletons. These are generically defined as one-dimensional (curve) structures that are locally centered within the shape [12,52]. Their lower dimensionality implies a simpler structure (branches meeting at junctions), which makes them easier to use for part-based segmentation [15,40,4,47]. Also, curve skeletons are easier to compute for large and complex 3D shapes as compared to surface skeletons, both for mesh representations [4,51] and voxel representations [1,24]. However, in contrast to the MAT, curve skeletons do not fully represent shapes, except when these have locally circular symmetry. In the following, we will focus exclusively on surface skeletons since the use or curve skeletons in shape segmentation is well covered in the literature (see also Section 4.2.2.1).

Skeletons are well known to be unstable to small-scale noise on the input shape surface $\partial\Omega$ [49,52]. As such, several so-called regularization methods have been proposed. *Local* methods use information such as the angle between feature vectors [18, 21], distance transform values, or divergence of the distance transform gradient [48] to prune skeletal points caused by noise. *Global* measures approximate the amount of

boundary that "collapses" to or corresponds to a skeletal point. This approximation can be done by computing the length of the geodesic path between the feature points of a skeleton point (the so-called medial geodesic function or MGF [15,43,23]) or by explicitly simulating the advection of mass from $\partial\Omega$ onto S along the gradient of the distance transform [24]. Important skeleton points are next defined to correspond to larger parts of the input boundary. Thresholding global importance measures can deliver a so-called multiscale skeleton, which reflects the input shape at a user-chosen level of detail [52].

4.2.2 SHAPE SEGMENTATION

Let $\Omega \in \mathbb{R}^3$ be a three-dimensional shape with boundary $\partial\Omega$. Segmenting Ω typically amounts to computing a so-called partition $\mathcal{C}$ of Ω into components C_i so that $\bigcup_i C_i = \Omega$ and $C_i \cap C_j = \varnothing, \forall i, j, i \neq j$. In other words, the set $\mathcal{C} = \{C_i\}$ consists of disjoint components that fully cover Ω. We next denote the borders of these segments by ∂C_i.

As mentioned in Section 4.1, segmentation methods can be classified into part-based and patch-based. The difference between the two classes amounts to different constraints put on the segments C_i. Regardless of the method type, however, segmentation can be seen as a combination of two key decisions: (1) finding *where* to cut a shape or where to place the segment borders ∂C_i; and (2) finding *how* to cut, or which properties the segment borders should respect. We next discuss part- and patch-based segmentation methods using skeletons from this perspective. In the following, we denote by C_i^{part} segments resulting from a part-based segmentation $\mathcal{C}^{part}$ and by C_i^{patch} segments resulting from a patch-based segmentation $\mathcal{C}^{patch}$. The notations $\mathcal{C}$ and C_i are used for the unified part-patch segmentation and respectively its segments that we aim to compute.

4.2.2.1 *Part-Based Segmentation*

As already stated, most part-based segmentations focus on natural articulated shapes, such as humanoids, animals, plants, or other objects showing a clear part–whole hierarchical structure. Parts are typically defined as elongated regions of a shape that significantly "stick out" of the shape rump. Such parts are separated from the rump by a negative curvature region, a principle also known as the "minima rule" in cognitive theory [22,8]. Such parts are easily detectable using curve skeletons since they correspond roughly one-to-one to the curve skeleton terminal branches [13]. Separately from the above heuristic that tells where to place cuts, part-based segmentation methods exploit other perceptual principles to constrain the cut shapes, such as the "short cut rule," which states that a cut should be as short (and wiggle free) as possible [50]. Lee et al. segment mesh models using the minima rule and optimizing for short cuts using snake models [28]. Additionally, a "part salience rule," which captures how much a part sticks out of the shape rump, can be used to limit oversegmentation [29].

Several part-based segmentation methods use curve skeletons in their design. Li et al. sweep the curve skeleton with a plane to cut the shape and keep those cuts

that have important geometric and topological changes indicating a part joining the shape rump [31]. Au et al. compute curve skeletons by iteratively contracting 3D meshes [4]. This enables them to backproject each skeletal point to one or several surface points. Hence, segmenting the curve skeleton into separate branches and next backprojecting each branch enable part-based segmentation. Along the same line, Reniers et al. first define curve skeletons as those points in Ω having at least two equal shortest-paths between their feature points [43] and then use closed loops (cuts) formed by such shortest paths placed at the curve-skeleton junctions to segment a shape [42]. Using shortest paths (geodesics) to construct cuts guarantees that these obey the desirable short cut rule. The method was refined to discard cuts that are far from planar, which reduces unneeded oversegmentation [40]. Serino et al. refine this idea by detecting three kinds of skeletal parts (simple curves, complex sets, and single points), which, by backprojection to the input shape, partition it in parts that protrude from a rump (called simple regions and bumps) and the rump itself (called a kernel) [45]. In comparison to [40], this method can yield better segment boundaries and suffers less from oversegmentation. The method in [45] was subsequently refined to use a computationally efficient and simple to implement curve-skeleton extraction based on the selection of a small number of centers of maximally inscribed balls (so-called anchor points) that guide an iterative voxel removal or thinning process. Apart from shape segmentation, this method can be also used in other contexts where a 3D curve skeleton is required. The problem of oversegmentation due to the potentially large number of curve-skeleton junctions, which lead to skeleton branches that do not map to salient shape parts, also discussed in [40], is elegantly addressed in [44] by so-called "zones of influence," which compare the local shape thickness at junctions with interjunction distances.

Separately, part-based segmentation and skeletonization have been shown to be related operations, which allow computing the latter from the former [32]. Conversely, Shapira et al. [47] note the same relationship but use it to segment a shape by computing a shape-diameter function (SDF) based on the boundary-to-curve-skeleton distance and finding cuts in places where the SDF has sharp variations. Finally, Tierny et al. [55] analyze the Reeb graphs computed for scalar functions defined on the shape surface to yield a hierarchical shape segmentation. Although Reeb graphs are not identical to Euclidean (curve) skeletons, they share their ability to capture topology, and thus such methods can be seen as skeleton-based segmentation techniques.

Overall, curve skeletons have established themselves as good descriptors for producing part-based segmentations for articulated shapes whose parts have a (near) tubular local geometry. Multiscale or hierarchical segmentations can also be easily obtained by considering the curve skeleton hierarchical structure or by pruning less important curve skeleton branches [42]. Since curve skeletons are locally centered in a shape, they yield largely pose-invariant segmentations. Finally, curve skeletons are easy and fast to compute for both voxel and mesh representations (Section 4.2.1).

Recently, *surface* skeletons of voxel models have also been used for part-based segmentation [17]. The key idea is to construct a *cut space* $\mathcal{CS} = \{c_i\}$ that contains a large set of cuts that have suitable properties to act as segment boundaries. This solves

the problem of *how* to cut. Next, a subset of these cuts is selected to become segment borders, thereby solving the problem of *where* to cut. Cut spaces are constructed by building closed loops formed by shortest paths on $\partial\Omega$ between the two feature points of each surface skeleton point. Next, cuts that represent suitable segment boundaries are found by analyzing a histogram of the cut lengths [17] or, alternatively, clustering cuts in terms of length [16]. Related methods of analyzing cut spaces for segmenting shapes have also been proposed, though not using skeletons to construct the cuts; see, e.g., [25,20].

4.2.2.2 Patch-Based Segmentation

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

Most patch-based methods work directly on the shape surface by unsupervised clustering, or grouping, mesh facets found to be similar [35,38,33,10]. Although such methods can produce very good results, they generally are parameter-sensitive. Closer to our interest, surface skeletons have been used for patch-based segmentation. The first such method we are aware of backprojects the (voxel-based) surface skeleton boundaries to the input surface, using the inverse feature transform (Eq. (4.3)) to yield segment boundaries [41]. A different approach is to segment the surface skeleton manifolds using Giblin's skeletal point classification [19] and backproject these to the input surface [9]. All such methods produce good patch segmentations even for shapes having soft edges. Such methods require high-throughput skeletonization tools and a delicate analysis of the resulting surface skeletons to detect boundaries and isolate manifolds.

A simpler patch-based segmentation method using surface skeletons was recently presented in [26]. Briefly put, this method implements the same strategy as [41], i.e., finding segment borders by backprojecting the boundary of the surface skeleton. However, in contrast to [41], this method handles point-cloud skeletons and has a much simpler implementation than [9]. We will use this method further in our unified segmentation pipeline (Section 4.3.8.1) and refer to it as the skeleton boundary backprojection (SBB) method.

Other methods: Outside the class of skeleton-based methods, many other methods exist for shape segmentation. Given our focus on using skeletons for this task, these methods are of lesser interest. For a general survey on 3D shape segmentation, we refer the interested reader to [46,3].

4.2.3 SUMMARY OF CHALLENGES

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

1. be able to efficiently and directly handle high-resolution mesh models since computing skeletons of high-resolution voxel models is prohibitive in terms of memory and time;
2. guarantee smoothness of the resulting segment borders;
3. be able to produce patch-based, part-based, and mixed-type segmentations, thereby handling all types of shapes.

No existing skeleton-based segmentation method complies with all the above. In the remainder of this chapter, we present such a method based on a complete recasting of the cut-space idea in [16] to use mesh models and to handle patch, part, and mixed segmentations.

4.3 METHOD

4.3.1 PRELIMINARIES

To start with, we briefly outline the idea of the cut-space part-based segmentation in [17,16], which we next refer to as the "voxel cut-space segmentation" (VCS) (see also Fig. 4.1, top). Given a voxel shape $\Omega \subset \mathbb{Z}^3$, its surface skeleton S is first computed using the method in [24]. Next, a simplified skeleton S_τ is extracted from S by removing voxels having an importance lower than a small predefined value τ. These are essentially voxels close to the boundary ∂S that correspond to small-scale details on the surface $\partial\Omega$ (Fig. 4.1B, top). This regularization makes sure that cuts (to be computed next) are only created from stable important skeleton parts. In this step, for each voxel $\mathbf{x} \in S_\tau$, a cut $c(\mathbf{x}) \in \partial\Omega$ is computed by tracing three shortest paths $\gamma_1, \gamma_2, \gamma_3$ on $\partial\Omega$ between the feature points $\mathbf{f}_1$ and $\mathbf{f}_2$ of $\mathbf{x}$ (γ_1), $\mathbf{f}_1$ and $\mathbf{m}$ (γ_2), and $\mathbf{f}_2$ and $\mathbf{m}$ (γ_3), where $\mathbf{m}$ is the reflection on $\partial\Omega$ of the midpoint of γ_1 with respect to $\mathbf{x}$; see red curve in Fig. 4.1C, top. Cuts are piecewise-geodesic (thus, smooth and tightly wrapping around Ω), closed, and locally orthogonal to the object symmetry axis (see again Fig. 4.1C, top). These properties ensure that cuts form good candidates for segment boundaries. The cut space $\mathcal{CS} = \{c(\mathbf{x}) | \mathbf{x} \in S_\tau\}$ is next partitioned into several cut-sets $\mathcal{K}_i$ containing similar-length cuts, using either an analysis of the cut-length histogram [17] or hierarchical clustering [16] (Fig. 4.1D, top). Next, the borders ∂C_i^{part} of the segments are computed by searching for cuts that separate the cut-sets $\mathcal{K}_i$ (Fig. 4.1E, top). Finally, the actual segments C_i^{part} are computed by searching for connected components on $\partial\Omega$ separated by the borders ∂C_i^{part} (Fig. 4.1F, top). For full implementation details, we refer to [17,16].

In the remainder of this section, we outline how we adapt the above voxel-based pipeline to work on mesh-based shapes admitting a point-cloud skeleton and also handle mixed part-patch segmentations. Sections 4.3.2–4.3.7 describe our part-based pipeline (steps in Fig. 4.1B–G, bottom). Section 4.3.8 describes the patch-based pipeline and how this is unified with the part-based one (Fig. 4.1H–L, bottom).

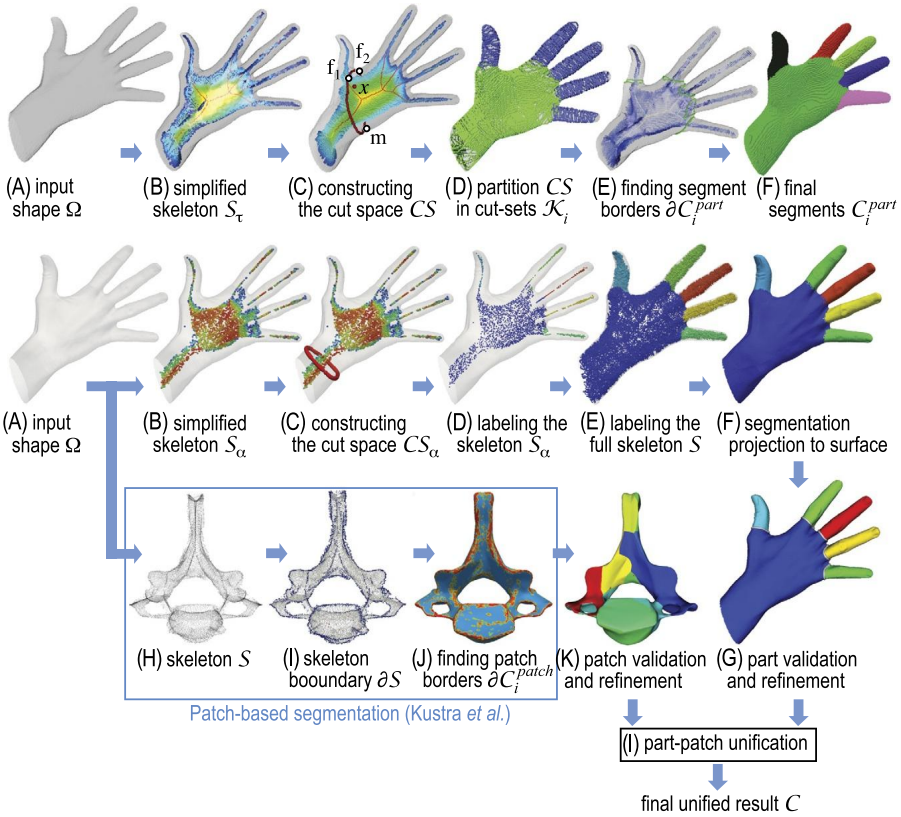

FIGURE 4.1

Top: cut-space segmentation pipeline from [17,16]. Bottom: our proposed pipeline.

4.3.2 REGULARIZED SURFACE SKELETON COMPUTATION

Step 1 of our proposed method parallels step 1 of VCS: We compute a simplified surface skeleton of the input shape. However, we have to skeletonize mesh shapes rather than voxel volumes. For this, we use the technique in [23], which is, to our knowledge, the fastest method to compute surface skeletons of mesh shapes to date. This method outputs a *point-cloud* representation of the skeleton S. Also, per skeleton point, the method computes two feature points, i.e., $\mathbf{f}_1$ and $\mathbf{f}_2$ in Eq. (4.2). In contrast, the voxel skeletonization method in [24] used by VCS outputs a voxel-based skeleton. As we shall see, point-cloud skeleton representations introduce several challenges to be addressed.

Similarly to VCS, we want to regularize S to avoid creating cuts from unimportant skeletal points that are caused by small-scale noise on $\partial\Omega$. At first sight, we could use for this the MGF importance metric delivered by the underlying skeletonization

method [23] (see also Section 4.2.1). The rationale for this would be that the MGF metric is analogous to the collapse metric provided by [24] and used by VCS.

However, upon careful examination, we note several issues. If we compare the MGF and collapse metrics for the same shape (Figs. 4.2A, B), then we see that they are similar, except for points close to the shape curve skeleton, where the collapse metric attains much higher values than over the surrounding surface skeleton (red curves in Fig. 4.2B). Hence, thresholding the collapse metric, as done in VCS, eliminates noisy skeleton-points *and* also preserves the important curve-skeleton points, which capture the shape part–whole structure (Section 4.2.2.1). In contrast, thresholding the MGF metric eliminates noise, but also all skeleton points in thin shape areas, which is undesired. Fig. 4.2C illustrates this: At the current threshold level, the shown surface skeleton contains both important points (such as the one generating the well-oriented red cut) and unimportant points (such as the one generating the green cut). If we thresholded the MGF metric with values larger than the one corresponding to the green cut, i.e., around the value τ shown in the color legend, then we would loose about 70% of the entire skeleton, including the complete legs and ears of the horse model. Even a very conservative setting of the threshold $\tau = 0.01$ immediately eliminates detail parts such as the ears (see Fig. 4.2E). Hence, we cannot use the MGF metric to reliably keep skeletal points that correspond to small shape parts *and* in the same time eliminate spurious skeleton points that generate badly oriented cuts.

To solve this problem, we note that surface-skeleton points close to the curve-skeleton have large angles between their feature vectors [43]. Separately, skeleton points created by small-scale noise on the input surface have low MGF values and thus close feature vectors. Hence, we regularize the point-cloud skeleton using the angle between the feature vectors $\mathbf{f}_1$ and $\mathbf{f}_2$ of a skeletal point, which is a well-known local importance metric [18,21]. We define the simplified skeleton as

$$S_\alpha = \{\mathbf{x} \in S | \angle(\mathbf{f}_1, \mathbf{f}_2) > \alpha\} \tag{4.4}$$

where $\alpha = 120°$ is a fixed preset that delivered good results for all our tested shapes. Fig. 4.2D shows the result: The angle metric consistently gets high values close to the curve-skeleton branches of *all* shape parts (rump, legs, muzzle, ears) and gets low values close to the noisy boundary of the surface skeleton. Hence, if we threshold the surface skeleton above the value α indicated in the figure, then we robustly eliminate spurious cuts like the green one and keep cuts that wrap around the shape local symmetry axis (curve skeleton) like the red one. The resulting simplified skeleton S_α is shown in Fig. 4.2F. Note that this skeleton is not connected, first because it is just a point cloud, and next due to the removal of low-importance points. However, this does not pose any problem further on since we use it only to construct our segmentation cuts.

The proposed angle-based regularization of the surface skeleton is simple to implement (involves a dot product of the normalized feature vectors). Additionally, it eliminates the need to compute the MGF metric for surface skeletons, which is the most expensive part of the skeletonization method for mesh shapes in [23].

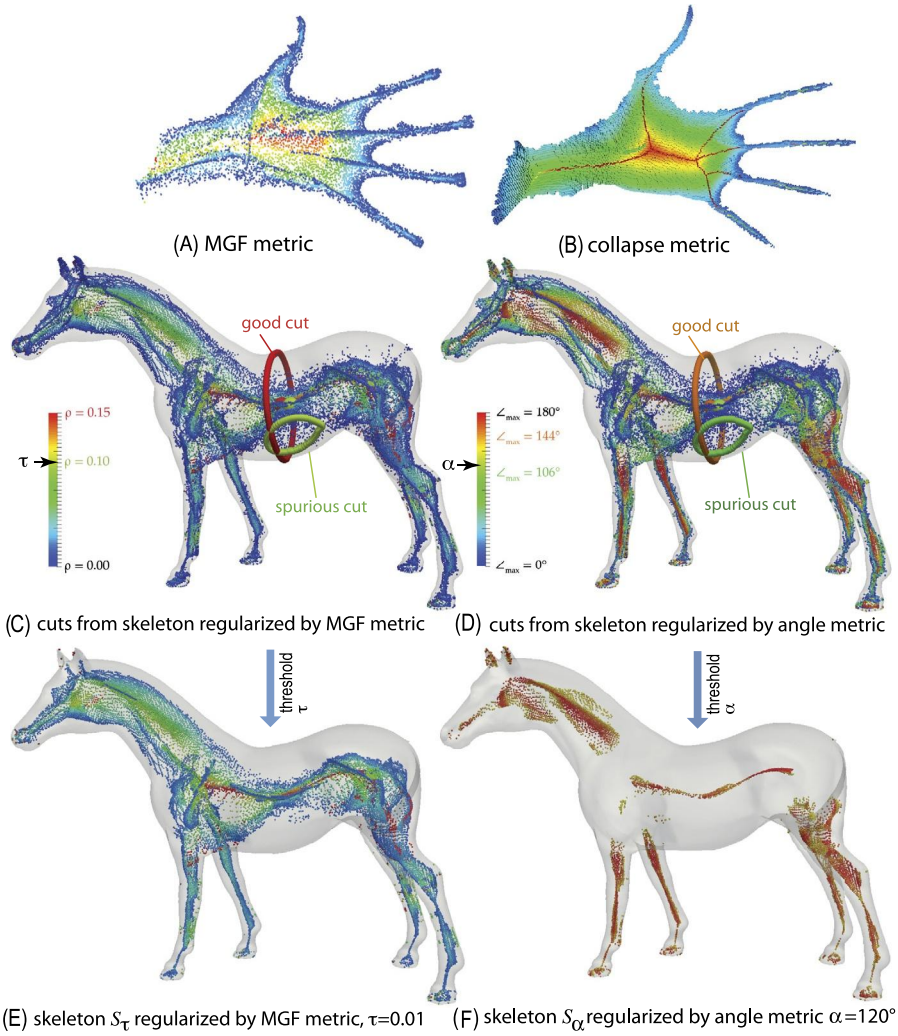

FIGURE 4.2

(A, B) Comparison of MGF and collapse importance metrics. (C) Thresholding the MGF metric keeps skeleton points that generate spurious cuts. (D) Thresholding the angle metric allows robust separating good from spurious cuts. All models are color-coded by the respective importance metrics.

4.3.3 CUT-SPACE COMPUTATION

Step 2 of our method computes the cut-space $\mathcal{CS}_\alpha = \{c(\mathbf{x})|\mathbf{x} \in S_\alpha\}$ from the regularized skeleton S_α. To construct cuts, we cannot use the VCS approach, which employs

Dijkstra's algorithm to connect the feature points $\mathbf{f}_1$ and $\mathbf{f}_2$ of a skeleton point $\mathbf{x}$ by three shortest paths in the adjacency graph implied by the voxel surface $\partial\Omega$ (see again Fig. 4.1C and related explanation). If we did so, e.g., by using the adjacency graph implied by the surface mesh vertices and triangle edges, then we would obtain heavily zig-zagging cuts, which would be useless for our task of inferring segment borders. To create cuts, we propose here to use the geodesic-tracing technique introduced in [23] for the different purpose of evaluating the MGF skeleton-importance metric (see a discussion in Section 4.3.2).

In detail, we proceed as follows. A straightest geodesic (SG) $\gamma_S : \mathbb{R}^+ \rightarrow \partial\Omega$ is defined as the unique solution of the initial-value problem $\gamma_S(0) = \mathbf{p}$, $\gamma_S'(0) = \mathbf{v}$ with $\mathbf{p} \in \partial\Omega$ being a point on the shape boundary having tangent vector $\mathbf{v} \in T_\mathbf{p}$, where $T_\mathbf{p}$ is the plane tangent to $\partial\Omega$ at $\mathbf{p}$. Jalba et al. [23] proposed an extension to define shortest-and-straightest geodesics (SSGs) γ_{se} between two points $\mathbf{s} \in \partial\Omega$ and $\mathbf{e} \in \partial\Omega$ to be an accurate approximation of the SG from $\mathbf{s}$ to $\mathbf{e}$. SSGs are computed by tracing multiple straightest geodesics over tangent vectors $\mathbf{v}_i \in T_\mathbf{s}$ and then selecting the one with shortest length $\|\gamma_{S,i}\|$, i.e.,

$$\begin{aligned} \gamma_{S,i}(0) &= \mathbf{s}, \quad \gamma_{S,i}'(0) = \mathbf{v}_i, \\ \gamma_{S,i}(\|\gamma_{S,i}\|) &= \mathbf{e}, \\ \gamma_{se} &= \arg\min_i \left\|\gamma_{S,i}\right\|. \end{aligned} \tag{4.5}$$

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

$$\gamma_{S,i}(0) = \mathbf{f}_1, \qquad \exists t \in \mathbb{R}^+ : \ \gamma_{S,i}(t) = \mathbf{f}_2, \qquad \gamma_{S,i}(\|\gamma_{S,i}\|) = \mathbf{f}_1. \tag{4.6}$$

To compute γ_{se} using Eqs. (4.5) and (4.6), we proceed similarly to [23]: For each skeleton point $\mathbf{x} \in S$, we trace $M = 30$ straightest geodesics $\gamma_{S,i}, 1 \leq i \leq M$, with starting directions $\mathbf{v}_i$ uniformly distributed in $T_\mathbf{s}$ and retain finally the one with minimal length. For each direction $\mathbf{v}_i$, we compute $\gamma_{S,i}$ by intersecting the mesh $\partial\Omega$ with the plane with normal $\mathbf{n}_i = \mathbf{f}_1 \times \mathbf{v}_i$ that passes through $\mathbf{s}$. In contrast to [23], 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

$$\mathcal{CS}_\alpha = \{\gamma_{\mathbf{f}_1\mathbf{f}_2} | (\mathbf{f}_1, \mathbf{f}_2) \in FT_{\partial\Omega|S_\alpha}\}, \tag{4.7}$$

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

4.3.4 CUT-SPACE PARTITIONING

In step 3, we identify how to partition the cut-space $\mathcal{CS}_\alpha$ into cut-sets that correspond to the shape segments. We employ a similar approach to the VCS method, i.e., use

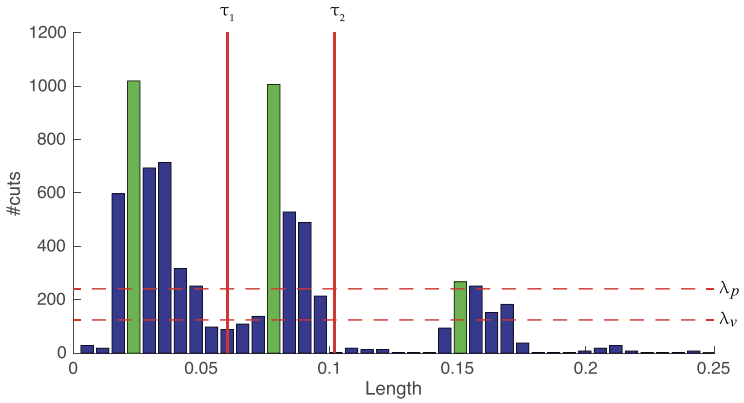

FIGURE 4.3

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

a histogram of cut lengths, in which peaks indicate many cuts with similar lengths, which likely belong to similar shape parts. With this assumption, we can find valleys separating the histogram peaks (Section 4.3.4.1), which in turn provide us with length thresholds to partition $\mathcal{CS}_\alpha$ (Section 4.3.4.2).

4.3.4.1 Histogram Valley Detection

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

To do this, we proceed as follows (see also Fig. 4.3). First, we use the mean shift algorithm [11] on the histogram bin heights to sharpen the differences between peaks and valleys. Next, we start scanning the histogram bins, left to right, for a peak that contains at least $\lambda_p = H_{\text{peak}} \cdot \|\mathcal{CS}_\alpha\|$ cuts. When found, we remember its height h_p and continue scanning and updating h_p as long as higher peaks are found. In the same time, we search also for valleys, i.e., for bins having fewer than $\lambda_v = H_{\text{decrease}} \cdot h_p$ cuts. This way, detection of a valley depends on the height of the peak (h_p) it follows. Just as for peak detection, when a bin having fewer than λ_v cuts is found, we remember its height h_v and continue scanning and updating h_v as long as lower peaks are found. If, during this valley-scan, we find a bin taller than λ_p, we have found a new peak. We have now two peaks and a valley of height h_v

in-between. We store the valley height $\theta_0 = h_v$ and continue scanning the histogram as above. After scanning the entire histogram this way, we obtain a set of valley heights $\Theta = \{\theta_i\}$. These will deliver our thresholds used to partition the cut space, as explained next in Section 4.3.4.2.

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

4.3.4.2 Histogram-Based Cut Space Partitioning

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

As for the VCS method, a cut-set $\mathcal{K}_i$ does not necessarily represent a compact shape segment, but could contain several such segments; for example, the five fingers of a hand have similar thickness, so they end in the same cut-set. To separate segments from cut-sets, we partition each cut-set into connected components, based on the connectivity of the skeleton points that generate the cuts. This was trivial to accomplish for VCS, given the explicit connectivity of skeleton voxels. In our case, the skeleton is an unorganized point-cloud lacking connectivity, as explained in Section 4.3.2. To address this, we define connectivity of skeletal points in terms of the neighborhood relation $\mathcal{N}_S(\mathbf{x})$ of a skeleton point $\mathbf{x} \in S$ to other skeleton points within a distance r from $\mathbf{x}$ as

$$\mathcal{N}_S(\mathbf{x}) = \{\, \mathbf{y} \in S \mid \mathbf{y} \neq \mathbf{x} \wedge \|\mathbf{x} - \mathbf{y}\| < r \,\}, \tag{4.8}$$

where setting r to 1% of the diameter of $\partial\Omega$ gives good results, following similar approaches in, e.g., [23,26].

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

4.3.5 PARTITIONING THE FULL SURFACE SKELETON

The labeling $\mathcal{L}$ computed as outcome of the cut-space partitioning (Section 4.3.4.2) is not our final desired result, i.e., the segmentation $\mathcal{C}^{part}$ of $\partial\Omega$. Indeed, $\mathcal{L}$ only assigns segment IDs to a *subset* S_α of the entire surface skeleton S. To obtain the final part-based segmentation $\mathcal{C}^{part}$, we proceed in two steps. First, we extend the labeling $\mathcal{L}_\alpha$ from the simplified skeleton S_α to a labeling $\mathcal{L}$ of the full surface skeleton S. This is described in this section. Next, we project the full labeling $\mathcal{L}$ from S to the shape boundary $\partial\Omega$. This is described in Section 4.3.6.

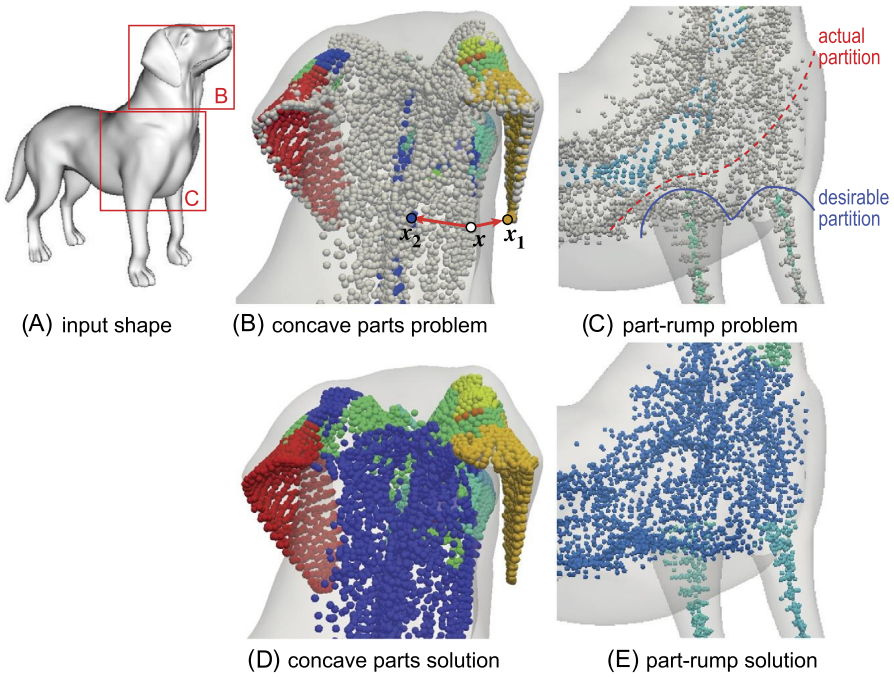

FIGURE 4.4

Label interpolation over the full skeleton. (A) Input shape. (B, C) Problems caused by naive nearest-neighbors interpolation. (D, E) Effective solution using the distance transform of the skeleton points.

Naive solution: To interpolate $\mathcal{L}_\alpha$ over the entire point-cloud skeleton S, one can use several strategies. A simple (but naive) one is to use a nearest-neighbor scheme where $\mathcal{L}(\mathbf{x} \in S \setminus S_\alpha) = \mathcal{L}_\alpha(\arg\min_{\mathbf{y} \in S_\alpha} \|\mathbf{x} - \mathbf{y}\|)$. This approach has several problems. First, using the $\mathbb{R}^3$ Euclidean distance metric to determine nearest neighbors will not work for nonconvex shapes that have nonconvex surface skeletons. Fig. 4.4B illustrates this for a dog model shown in Fig. 4.4A. Here, colored points are labeled points in S_α, and gray points are points to be labeled in $S \setminus S_\alpha$. The marked (red) point $\mathbf{x}$, which is located on the neck skeleton, is closer to point $\mathbf{x}_1$ (on the dog's ear) than to $\mathbf{x}_2$ (on the neck skeleton), so it will wrongly get the label of $\mathbf{x}_1$, i.e., be assigned as part of the ear. The second problem of this nearest-neighbor interpolation is that labels will always meet halfway between labeled points in S_α. This causes problems at junctions of parts having widely different thicknesses. Fig. 4.4C illustrates this. We see here a detail of the dog model having two label values over S_α, dark blue for rump points and cyan for points in the legs. The area where the two label values would meet, when using the simple nearest-neighbor interpolation, is shown in red, and it is actually the border of a generalized Voronoi diagram having all blue, respective cyan, points as

two sites. This partition is undesirable since it would assign a large part of the rump skeleton to the segments corresponding to the legs. The desirable leg-rump partition is shown by the blue line in Fig. 4.4C.

Effective solution: To correct the first problem, one way would be to consider the geodesic distance along the S manifold rather than Euclidean distance in $\mathbb{R}^3$. Doing so is however not possible for skeletons represented as point clouds, and reconstructing smooth manifolds from such skeletons is a highly complex and expensive process [27]. Also, this would not correct the second problem. We next propose a much simpler and faster solution that addresses both problems. A key to this idea is the observation that the local shape thickness around a skeletal point should determine how far a label should be propagated. This local thickness is precisely represented by the distance transform $DT_{\partial\Omega}$ values on S. Hence, for a labeled point $\mathbf{x} \in S_\alpha$, we search all its neighbors $\mathbf{y}$ in a ball of radius $DT_{\partial\Omega}(\mathbf{x})$ and if $DT_{\partial\Omega}(\mathbf{y}) < DT_{\partial\Omega}(\mathbf{x})$, then assign $\mathcal{L}(\mathbf{y}) \leftarrow \mathcal{L}(\mathbf{x})$. The last distance condition is required to have labels of skeleton points in thick regions dominate those of nearby skeleton points in thin regions and also to ensure that the labeling does not depend on the order of visiting of the skeleton points. Finally, the (few) skeleton points that are still unlabeled after this operation are assigned the label of their nearest neighbor. Figs. 4.4D, E show the results of our proposal. As visible, both concave-part and part-rump problems mentioned earlier are now solved as desired.

4.3.6 PARTITION PROJECTION TO SURFACE

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

In our context, there are several issues with using this method to project the labeling from the skeleton to the surface. First, our point-cloud skeleton does not admit a direct equivalent of the 26-neighbors relationship used for voxel shapes. The nearest-neighbor relation $\mathcal{N}_S$ (Eq. (4.8)) is too coarse to capture such fine details. Much more critically, though, is the fact that our cuts are *planar* SSGs, whereas the ones used by VCS consist of three geodesic curves, not necessarily located in a plane (see Section 4.3.3). As such, VCS cuts have a much larger freedom to model flexible segment boundaries than our cuts. However, despite their flexibility, the VCS cuts are still constrained by their construction process to consist of three geodesic curves (see Section 4.3.1). Although this appears to handle quite well part-based segmentation, it is arguably too rigid for producing good patch-based and mixed segmentations, which are our ultimate goal.

We propose a different way of projecting the segmentation information from the skeleton to the shape surface, which addresses all above issues. For each skeleton

point $\mathbf{x} \in S$ having label $\mathcal{L}(\mathbf{x})$, we assign the label to all closest surface points to $\mathbf{x}$, i.e., set $\mathcal{L}(\mathbf{y} \in FT_{\partial\Omega}(\mathbf{x}) \leftarrow \mathcal{L}(\mathbf{x})$. Note that this may not assign labels to boundary points in convex surface regions since the skeletonization algorithm we use [23] only computes two feature points per skeleton point, rather than the full feature transform (Section 4.3.2). We compensate this by assigning labels to all yet unlabeled surface points by simple nearest-neighbor interpolation. Finally, we derive the part segments $C_i^{part} \subset \partial\Omega$ as the connected-components on $\partial\Omega$ that have the same label values.

4.3.7 PART-BASED PARTITION REFINEMENT

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

Segment validation: First, segments C_i^{part} are checked as to their validity, in terms of what a human observer would qualify as being a "part" or not. Recalling the assumptions of part-based segmentation [20,40,17,16], we see that actual parts in a good part-based segmentation should be covered well by their associated cuts in the cut-space. Intuitively put, if we cut through a part with tight cuts orthogonal to the shape local symmetry axis, then the cuts should stay in the part. Conversely, segments whose cuts cover far more than the segment itself do not coincide with what is typically perceived as a part. To measure this coverage, let $K(C_i^{part}) = \{c \in \mathcal{CS} | c \cap C_i^{part} \neq \varnothing\}$ be the set of cuts in our cut-space that intersects segments C_i^{part}. We then define the coverage of C_i^{part} by cuts as

$$\nu(C_i^{part}) = \frac{\|\{c \in K(C_i^{part}) \mid c \setminus C_i^{part} = \varnothing\}\|}{\|K(C_i^{part})\|}, \tag{4.9}$$

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

The coverage test effectively filters most, but not all, segments that are not part-like. A salient exception are corners of faceted objects such as the box in Fig. 4.5A. These segments are covered by cuts that nearly all stay inside them, as shown by the cut-space of the shape colored by the labels of the respective cuts (Fig. 4.5B). Hence, such segments will not be marked as invalid. A similar problem (without a solution) was highlighted by the curve-skeleton-based part segmentation technique in [40]. To discard such segments, we use a second heuristic: Parts should "stick out" of the shape as much as possible [29], or, in other words, they should have as many points with opposite normals. To test this, we compute angles between all surface

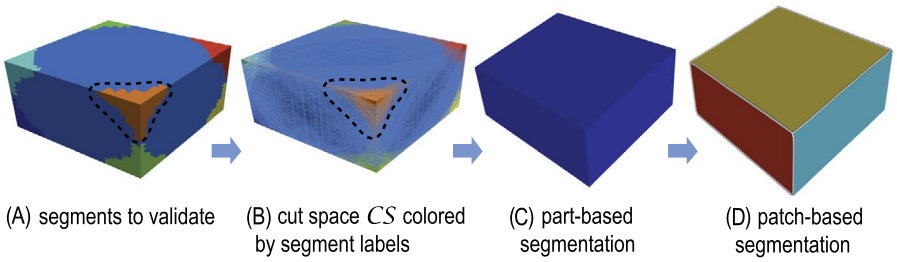

FIGURE 4.5

Part validation for a box shape. The produced segments (A) are validated by computing their cut-coverages (B). This results in a trivial part-based segmentation (C), which finally determines that a patch-based segmentation will handle this shape (D).

normals to points in a segment C_i^{part} and require that the median of these angles be at least 90°. For the box shape, this invalidates the corner segments, yielding a single segment for the entire shape in terms of part-based segmentation (Fig. 4.5C). As such, the patch-based segmentation of the same shape, which we next discuss in Section 4.3.8, gets full freedom to process the shape, leading to the expected result (Fig. 4.5D).

For completeness, we should note that our part validation heuristic is not the only possible one. For instance, Serino et al. propose a so-called "visibility criterion" [45], which aims to detect whether a peripheral part, far from the object main rump, actively contributes to a meaningful segmentation. This heuristic uses only the curve skeleton in its computation in a voxel-based setting. Since our pipeline only uses the *surface* skeleton and since computing the curve skeleton of mesh-based shapes is more complex when considering the technique in [23], which forms the backbone of our approach, the possibility of integrating this criterion in our framework is a topic of further investigation.

Segment border smoothing: Recall that segments C_i^{part} on $\partial\Omega$ are essentially backprojections of skeleton fragments sharing the same label values via the feature transform (Section 4.3.6). In the ideal continuous case, equivalent to an infinitely dense sampling of a smooth $\partial\Omega$, and a similar sampling of S, the segments would have smooth continuous borders. However, real-world meshes have a limited *and* often highly nonuniform sampling resolution. The used skeletonization method [23] essentially copies this sampling resolution to S. More critically, this method only uses the mesh vertices of $\partial\Omega$ as feature points for estimating the feature transform (Eq. (4.3)). As such, backprojecting labeled skeleton points to $\partial\Omega$ do not result in smooth segment borders. Fig. 4.6A shows this for a densely and uniformly sampled hand model of 197K vertices. As visible, segment borders exhibit problematic small-scale fractal-like noise. We solve this issue by computing smooth segment borders as follows. First, we create borders ∂C_i^{part} of the segments C_i^{part} by connecting the

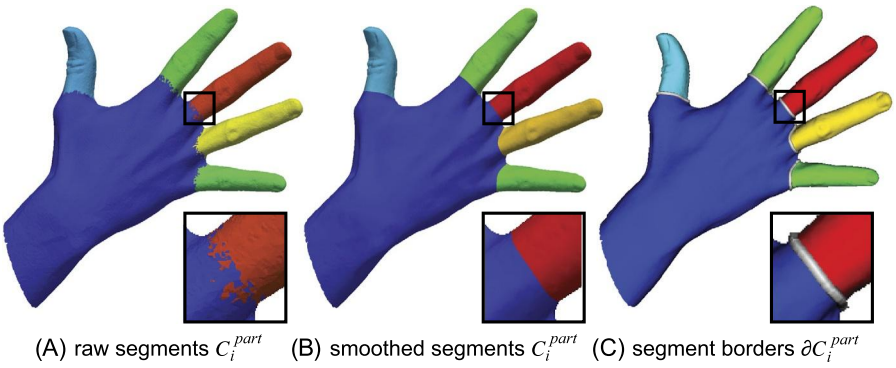

FIGURE 4.6

Computing part-segment borders. (A) Raw borders produced by skeleton-labeling backprojection. (B) Borders after Laplacian smoothing. (C) Visually emphasized borders.

midpoints of surface triangle-cell edges whose vertices have different labels (thus fall into different segments). Next, we apply classical Laplacian smoothing [54] to remove small-scale wiggles along these boundaries. After each smoothing pass (10 in total), we reproject the smoothed boundary ∂C_i^{part} to $\partial\Omega$ since unconstrained 3D smoothing makes the boundary curve leave the shape surface. The overall effect is identical to performing Laplacian smoothing *constrained* on the surface $\partial\Omega$. As the smoothed segment border moves on $\partial\Omega$, we update the labels of the vertices to ensure consistency. Fig. 4.6B shows the result: The smoothed boundaries stay roughly in the same position as the initial ones but are considerably tighter and smoother, thus similar in terms of desirable requirements to the original piecewise-geodesic boundaries of the VCS method. Finally, we draw these smooth boundaries as thick 3D tubes for ease of perception (Fig. 4.6C).

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

4.3.8 UNIFIED (PART AND PATCH) SEGMENTATION

So far, we described how to produce *part-based* segmentations $\mathcal{C}^{part}$ of mesh shapes. However, as outlined already in Section 4.1, our goal is to segment any shape and thus to propose a way to combine part-based and patch-based segmentations in a flexible way. We present here a way to combine the two segmentation types in a new segmentation model, which we refer to as *unified* (part-and-patch) segmentation $\mathcal{C}$. We first introduce the patch-based segmentation method we use (Section 4.3.8.1). Next,

we discuss the desirable properties of a good unification method and possible strategies to implement it (Section 4.3.8.2). Finally, the unification technique we propose is presented in Section 4.3.8.3.

4.3.8.1 Patch-Type Segmentation Using Surface Skeletons

For patch-based segmentation, we use the skeleton boundary backprojection (SBB) method of Kustra et al. [26], which has several desirable properties in our context. First, this method treats high-resolution mesh shapes, which is our application target. Secondly, the method is also based on surface skeletons, which matches our toplevel goal of showing how such skeletons can effectively support shape segmentation. Thirdly, the method uses the same skeletonization technique [23] as our part-based segmentation, which makes easy the technical combination of part and patch segmentation. We next briefly explain this method and also outline its limitations relevant to our unified segmentation goal.

As all other patch segmentation methods, Kustra et al. define patches C_i^{patch} implicitly by requiring that borders separating them should occur in high surface-curvature areas. To do this, they proceed as follows. First, the point-cloud surface skeleton S of the input shape Ω is computed using the technique in [23]. Next, points on the boundary ∂S of the surface skeleton, or so-called A_3 points [19], are detected as those skeletal points whose images on $\partial\Omega$, via the feature transform, contain a single compact cluster. However, as noted earlier in this paper, the underlying skeletonization method [23] does not compute the full feature transform, but only two feature points per skeleton point. Kustra et al. note that computing the exact (full) feature transform is sensitive, so they propose instead the so-called extended feature transform

$$FT^{\tau}_{\partial\Omega}(\mathbf{x} \in S) = \{\mathbf{f} \in \partial\Omega | \|\mathbf{x} - \mathbf{f}\| \leq DT_{\partial\Omega}(\mathbf{x}) + \tau\}, \tag{4.10}$$

which gathers, for each skeletal point $\mathbf{x}$, all boundary points within a range $DT_{\partial\Omega}(\mathbf{x}) + \tau$, where τ is a small positive value. The extended feature transform provides a conservative approximation of the actual feature transform (Eq. (4.3)), and can be readily used to detect ∂S as outlined above. Similar conservative approximations of the feature transform are also proposed by the VCS method for voxel shapes [17,16].

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

$$\Phi = \{\mathbf{f} \in FT^{\tau}_{\partial\Omega}(\mathbf{x}) \mid \mathbf{x} \in \partial S\}, \tag{4.11}$$

which conservatively captures convex edges on $\partial\Omega$. Patches C_i^{patch} are now easily found as the connected components of $\partial\Omega \setminus \Phi$. Finally, points in Φ get assigned to the closest established patch, thereby completing the patch-based segmentation $\mathcal{C}^{patch}$ of $\partial\Omega$.

Fig. 4.7 (top row) shows the results of the patch-based segmentation method described above on two shapes (*fandisk* and *horse*). For *fandisk*, which has clear

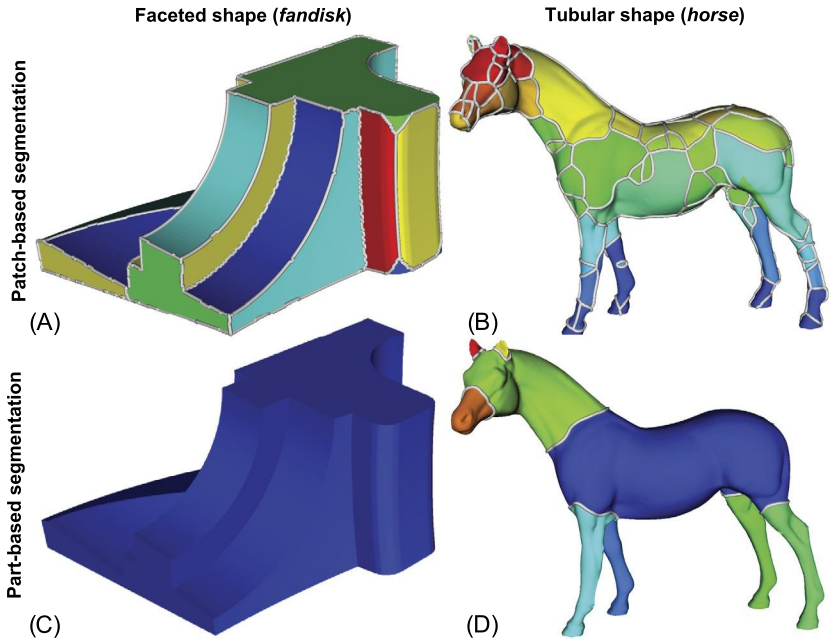

FIGURE 4.7

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

and salient edges, a very good patch segmentation is produced (Fig. 4.7A). The *horse* shape has much softer and fuzzier edges and as such yields a poor patch-type segmentation (Fig. 4.7B). This can be explained as follows: Finding a reliable set Φ that captures edges on the surface $\partial\Omega$ requires computing stable A_3 points to detect the skeleton boundary ∂S (Eq. (4.11)). However, the heuristic described earlier of finding A_3 points via the extended feature transform fails for cylinder-like parts of a shape, as discussed in [26]. Although fine-tuning various parameters of the method of Kustra et al. sometimes improve results, patch-type segmentation does not work well for shape regions having too soft edges. In contrast, using our part-based segmentation method described in Sections 4.3.2–4.3.7 yields very good results for tubular soft-edged shapes like *horse* (Fig. 4.7D). Conversely, part-based segmentation produces a poor result for the faceted shapes like *fandisk* (Fig. 4.7C). In this case, no part is identified, which is correct, given the part-validation criteria outlined in Section 4.3.7. Hence, to obtain the best segmentation results for all shape types, including those that have a mix of faceted and tubular parts, a unification of part- and patch-based segmentation is needed. Such a method is proposed next.

4.3.8.2 Unification Desirable Properties

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

1. *hybrid:* the strategy should handle shapes admitting a full part-based segmentation, shapes admitting a full patch-based segmentation, and also shapes admitting a mix of the two segmentation types in various areas;
2. *intuitive:* the unified segmentation should make sense according to human perception, that is, the produced parts, respectively patches, should visually make sense with respect to the part and patch properties established in Sections 4.2.2.1 and 4.2.2.2;
3. *balanced:* unified segmentations should not be oversegmented due to incorporating both segment types.

The simplest unification approach would be to compute separate part-based and patch-based segmentations and then choose the result that maximizes the respective quality properties of the two segmentations. However, this solution cannot handle shapes that require a hybrid segmentation. An alternative is to compute the two segmentation types separately and then *mix* their results in terms of selecting the optimal segments with respect to both local quality criteria (that decide which type of segmentation best fits a region of the shape) and global criteria (the user's preference for part- or patch-based segmentations). This approach can satisfy all above-mentioned requirements. As such, we next choose this way as follows. For part-based segmentation $\mathcal{C}^{part}$, we use our pipeline presented in Sections 4.3.2–4.3.7. For patch-based segmentation $\mathcal{C}^{patch}$, we use the method discussed in Section 4.3.8.1. Our unification method is discussed next.

4.3.8.3 Unification Method

As outlined in Section 4.3.8.2, our approach to a unified segmentation is to compute both part- and patch-based segmentations of the shape and next decide which of the produced segments are *valid* in terms of specific part and patch requirements. Valid segments are kept and finally merged to yield the unified segmentation. The process is detailed next.

Part validation: For part-based segmentation, we use the segment validation procedure described in Section 4.3.7, which consists of the cut-coverage and angle-coverage criteria. As such, part-based segmentation will only produce valid part-segments C_i^{part} in areas where such segments can be computed. When and where valid segments cannot be computed, the shape will be left unsegmented.

Patch validation: For patch-based segmentation, the method of Kustra et al. does not provide patch validation. Hence, this method may oversegment the shape or produce otherwise suboptimal patches, such as shown by the example in Fig. 4.7B. Merging such suboptimal patches with otherwise good parts will yield an overall poor unified

segmentation. We address this by proposing a patch validation scheme, similar in spirit to the part validation discussed earlier.

As already outlined, a good (valid) patch C_i^{patch} should have its borders in high-curvature regions of the input surface $\partial\Omega$. Several methods exist for computing curvature on mesh surfaces, such as using the curvature tensor [36,53], using moment analysis [10], or estimating the shape thickness over the surface-skeleton boundary [26]. The first two methods are quite sensitive in terms of setting the scale at which edges are detected, as also noted in [26]. The last method, which is also used to detect patch borders in the segmentation technique described in Section 4.3.8.1, does not have this problem, but captures also soft edges, leading to issues such as the oversegmentation in Fig. 4.7B.

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

$$\mathcal{N}_V(\mathbf{x}) = \{\, \mathbf{y} \in V \mid \|\mathbf{x} - \mathbf{y}\| < r \,\} \tag{4.12}$$

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

$$\partial\mathcal{N}_V(\mathbf{x}) = \{\, \mathbf{y} \in V \setminus \mathcal{N}_V(\mathbf{x}) \mid (\mathbf{x}, \mathbf{y}) \in E \,\} \tag{4.13}$$

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

$$\kappa(\mathbf{x}) = \text{mean}\,\{\, \angle(\mathbf{n}(\mathbf{y}), \mathbf{n}(\mathbf{z})) \mid (\mathbf{y}, \mathbf{z} \neq \mathbf{y}) \subset \partial\mathcal{N}_V(\mathbf{x}) \times \partial\mathcal{N}_V(\mathbf{x}) \,\}\,, \tag{4.14}$$

where $\times$ denotes Cartesian product.

Fig. 4.8 shows our curvature estimator κ for two shapes. As visible, κ captures all zones where salient edges exist, both for the *fandisk* model, which exhibits clear edges, and for the *frontal bone* model, which has much noisier edges. Here, the neighborhood size r (Eq. (4.12)) is set to roughly 2 to 5% of the diameter of $\partial\Omega$. This setting has given good results for all other tested shapes.

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

$$\begin{aligned}\mathcal{B}_{ij} = &\left\{ \mathbf{x} \in C_i^{patch} \,\middle|\, \exists \mathbf{y} \in C_j^{patch}, (\mathbf{x}, \mathbf{y}) \in E \right\} \\ &\cup \left\{ \mathbf{x} \in C_j^{patch} \,\middle|\, \exists \mathbf{x} \in C_i^{patch}, (\mathbf{x}, \mathbf{y}) \in E \right\}.\end{aligned} \tag{4.15}$$

For each such border fragment, we compute the median curvature $\hat{\kappa}(\mathcal{B}_{ij})$ over all its vertices $\mathbf{x} \in \mathcal{B}_{ij}$. If $\hat{\kappa}(\mathcal{B}_{ij})$ is larger than a threshold $\beta = \pi/4$, determined empirically for our studied models, then $\mathcal{B}_{ij}$ is a valid (good) boundary of our patch-based

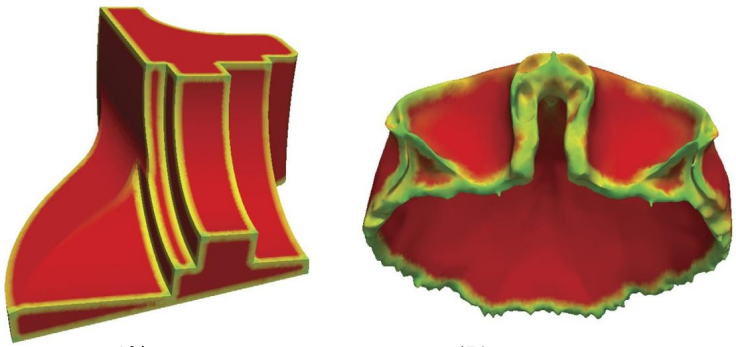

FIGURE 4.8

Proposed curvature detector κ for two shapes.

segmentation, so we keep it. If not, then we erase $\mathcal{B}_{ij}$ from the segmentation, i.e., merge patches C_i^{patch} and C_j^{patch}. To ensure a deterministic patch-validation result, we process patch border segments $\mathcal{B}_{ij}$ in increasing order of $\hat{\kappa}(\mathcal{B}_{ij})$.

Merging segmentations: After removing invalid patches, we have both a validated part-based ($\mathcal{C}^{part}$) and patch-based ($\mathcal{C}^{patch}$) segmentation. We now construct the unified segmentation $\mathcal{C}$ by merging $\mathcal{C}^{part}$ and $\mathcal{C}^{patch}$ as follows. We first initialize $\mathcal{C}$ with $\mathcal{C}^{part}$, i.e., keep all part segments that have been already validated (see Section 4.3.8.3). Next, we merge in $\mathcal{C}$ the patches C_i^{patch}. This essentially refines, or subdivides, those parts that have been found to consist of several patches in $\mathcal{C}^{patch}$.

Let us explain how the unified segmentation yields the desirable results for the shapes in Fig. 4.7. For the *fandisk* shape, $\mathcal{C}^{patch}$ contains patches having very high curvature along their borders (Fig. 4.7A). Hence, all borders shown in the figure will be validated and kept as shown. In contrast, its $\mathcal{C}^{part}$ contains a single part (Fig. 4.7C). The merging process outlined above will split this single part into precisely the patches of $\mathcal{C}^{patch}$. Hence, the unified result will be identical to $\mathcal{C}^{patch}$, as desired. For the *horse* shape, $\mathcal{C}^{patch}$ contains patches having (very) low curvature values along their borders (Fig. 4.7B). As such, all these patches will be invalidated, yielding $\mathcal{C}^{patch} = \varnothing$. Hence, the merging process will keep $\mathcal{C}^{part}$ unchanged. This is the desired result since $\mathcal{C}^{part}$ for this shape is of good quality (Fig. 4.7D).

4.4 RESULTS

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

We start by comparing part-based segmentations. Fig. 4.9 compares our method with the original voxel-based cut-space segmentation (VCS) in [16]. As visible, our

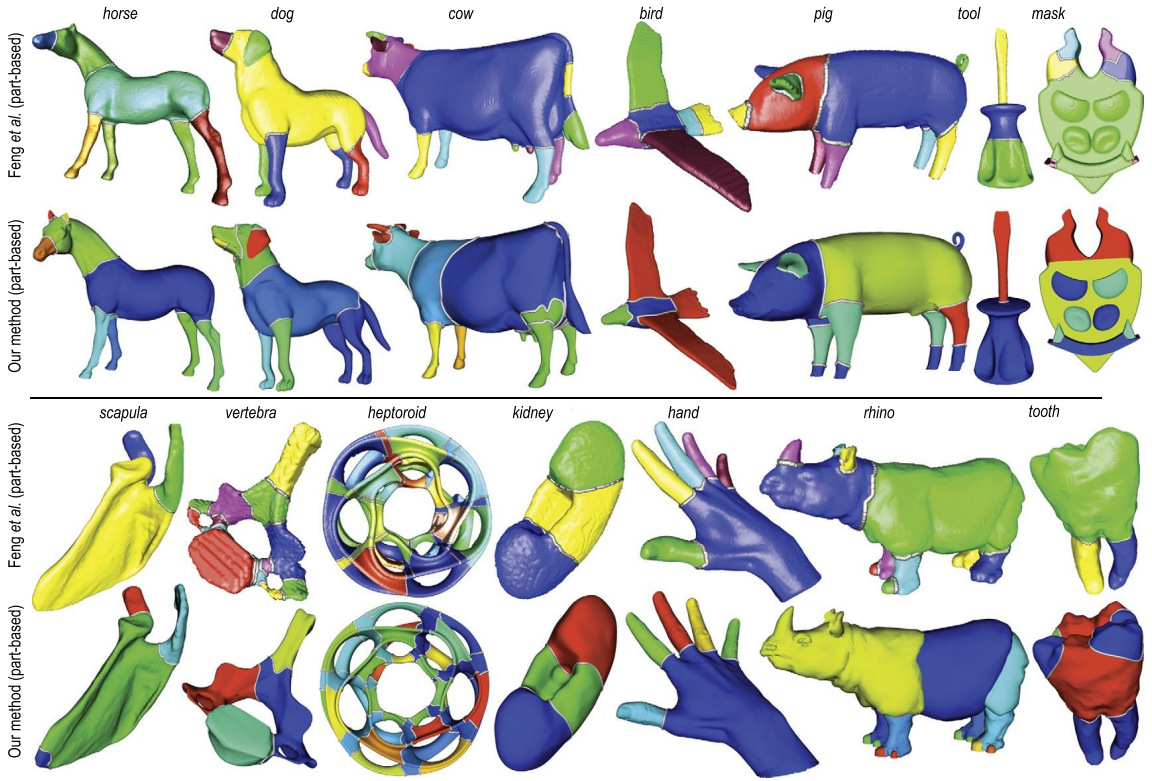

FIGURE 4.9

Comparison of segmentations between VCS (Feng et al. [16]) and our part-based method.

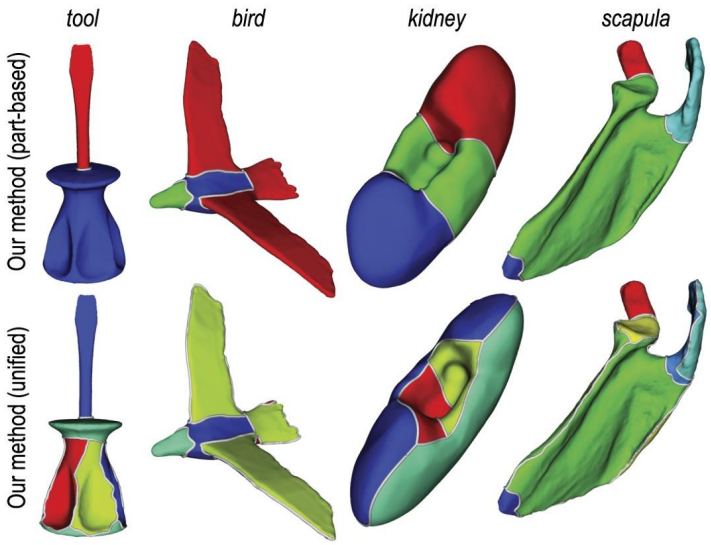

FIGURE 4.10

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

results are very similar in terms of position, smoothness, and tightness of the delivered segments. We also see that our method succeeds in cases where VCS visibly undersegments the shape (*tool*, *mask*). Small-scale details may differ between the two segmentations, such as for the *rhino* model where VCS undersegments the toes, whereas our method undersegments the horn and ears. In this particular case (*rhino*), the VCS segmentation is likely better since separating the larger horns and ear details should be more important than separating the smaller toes. These variations are explainable by the two different parameter settings of the two methods.

Fig. 4.10 shows the added-value of the unified segmentation for a set of shapes having relatively soft (shallow) edges. We see how the unified segmentation (bottom row) refines the already-discussed part-based segmentation (top row) by splitting segments along lines of high curvature, e.g., the edge of the bird wing or of the scapular bone. This allows getting a mix of parts that capture the shape topology, and faces, or patches, that capture the shape geometry. The unified segmentation can thus be seen as a refinement of the part-based segmentation.

Fig. 4.11 compares our unified segmentation with the SBB method of Kustra et al. [26], which is also using surface skeletons to produce patch-based segmentations of 3D shapes. The figure contains the anatomic shapes used as benchmark in [26]. These are quite challenging to segment since they have complex geometries, a mix of sharp and soft tortuous edges, and consist of both parts and patches. As visible, our method is able to find all patch-like segments that the SBB method can. However,

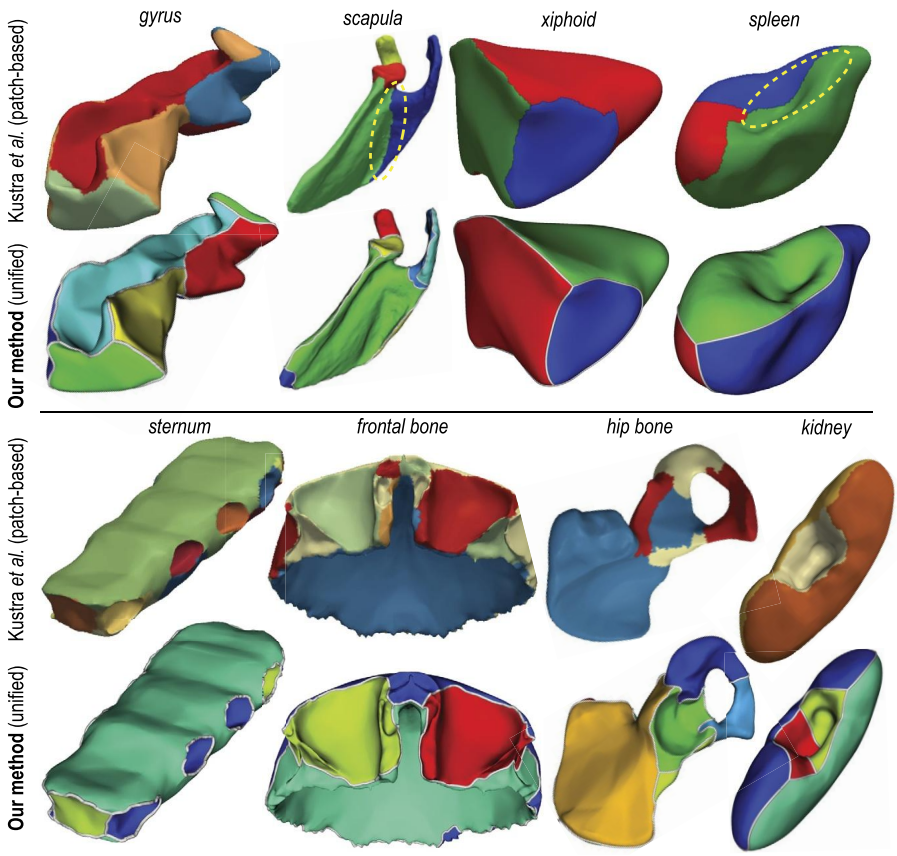

FIGURE 4.11

Anatomic shapes segmented by the SBB method of Kustra et al. [26] and our unified method.

our method generates visibly smoother segment borders, which also better follow the shape edges, even in the case of very complex geometries such as *frontal bone* and *gyrus*. We also see that SBB places two segment borders in areas where no apparent patch or part transition occurs (*scapula* and *spleen*, marked details). In contrast, our unified method does not allow such borders to exist due to its part and patch validation steps (Sections 4.3.7 and 4.3.8.3).

4.5 DISCUSSION

We next discuss several aspects of our unified segmentation method.

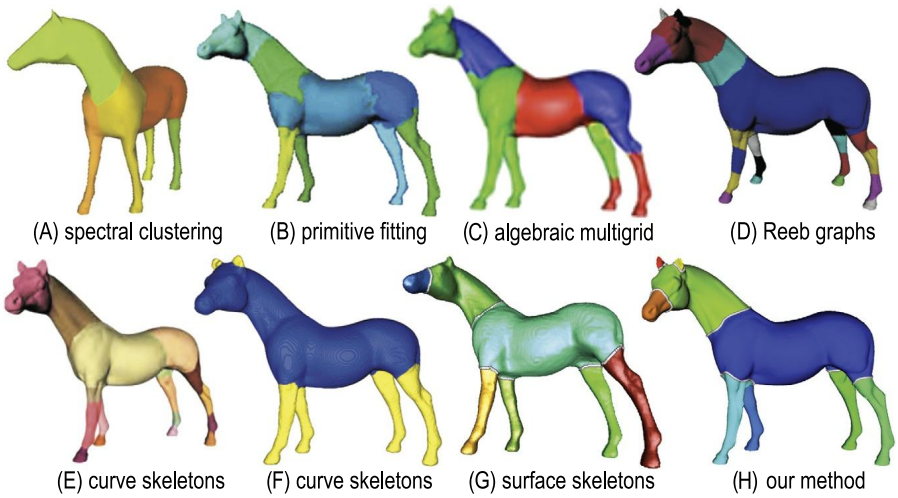

FIGURE 4.12

Comparison of eight part-based segmentation methods. (A) Liu and Zhang [33]; (B) Attene et al. [2]; (C) Clarenz et al. [10]; (D) Tierny et al. [55]; (E) Lien et al. [32]; (F) Reniers et al. [40]; (G) Feng et al. [16]; (H) our method.

Comparison: Section 4.4 has compared our method with the two main related methods, which use surface skeletons to segment shapes (VCS [16] and SBB [26]). It is also interesting to compare our method with the larger class of shape segmentation techniques out there. Although it is impossible to do so in the same detail as provided in Section 4.4, Fig. 4.12 shows a qualitative comparison of our method with seven well-known part-based segmentation methods. Images (A–D) correspond to methods that do not use skeletons. Images (E–F) correspond to methods that use either curve or surface skeletons. For an overview of these methods, we refer to Section 4.2.2.1.

The most salient observation on Fig. 4.12 is that skeleton-based methods tend, in general, to provide more "natural" part-based segmentations than the other studied methods in terms of positioning and smoothness of the segment borders. This is due to the inherent ability of skeletons to model a shape local axis of symmetry, across which segment borders can be fit. In contrast, non-skeleton-based methods cannot ensure this proper border orientation. Secondly, we see that the cut-space based methods (Figs. 4.12G, H) do not *oversegment*. This observation is also confirmed by all other earlier examples showing our method (Figs. 4.9, 4.10, 4.11). This is due to two design elements in our method: (1) the cut-space partitioning ensures that similar-length cuts will never yield different parts (Section 4.3.4), and (2) the part and patch validations ensure that superfluous parts and patches are automatically removed (Sections 4.3.7 and 4.3.8.3). This is in contrast to several of the other methods depicted here (Figs. 4.12B–E). All in all, the above observations strongly plead for the added-value of skeletons for shape segmentation.

Table 4.1 Performance of the proposed unified segmentation method

Shape	$\|\partial\Omega\|$	$\|\mathcal{CS}_\alpha\|$	t_{skel}	t_{cuts}	t_{patch}	t_{unif}	$T_{unified}$
Horse	49,749	7453	5.6	63.2	5.2	6.4	5866
Hound	16,158	364	0.7	5.1	2.0	2.7	1153
Cow	137,862	16,447	23.0	242.3	10.9	63.0	9357
Bird	47,184	26,485	6.3	262.0	5.3	14.1	4769
Pig	4800	756	0.2	2.0	0.6	0.6	1814
Vertebra	22,789	7779	0.2	57.4	2.4	2.2	3088
Scapula	117,432	83,340	17.3	2325.7	11.0	26.6	4208
Heptoroid	79,056	63,876	5.0	539.1	6.3	5.8	9367
Kidney	30,389	9986	6.5	177.7	4.0	27.0	1707
Hand	49,546	2815	7.3	47.9	4.2	11.1	2911

The fact that there is quite some variation in the segmentations produced by different methods and, implicitly, in their perceived quality, should however be interpreted with care. The comparison presented here, as is far from exhaustive, cannot thus be used to derive generalizing value judgments of one method vs another. For instance, the examples in Fig. 4.12E, F show that curve-skeleton-based methods can sometimes oversegment (see horse legs, Fig. 4.12E) and sometimes undersegment (see horse neck and rump, Fig. 4.12F). The fact that surface-skeleton-based methods do not show such artifacts (Fig. 4.12G, H) should not be interpreted as pointing to a general superiority of such methods as opposed to curve-skeleton-based methods. The main conclusion from this comparison is limited to showing that segmentation methods using *surface* skeletons are an interesting and viable alternative for *part*-based shape segmentation.

Unification: If we study the state-of-the-art in shape segmentation methods [46,3], then we see that most such methods are, implicitly or explicitly, focused on handling either part-based or patch-based segmentations, but rarely both. This is easy to explain since we have seen that the criteria defining (good) parts and patches are very different, as the two have different natures. Indeed, parts are inherently volumetrically described; whereas patches are best described on the shape surface [40,41]. As such, one typically needs to know in practice what is the nature of shapes one wants to segment in order to choose the best segmentation method for that task; or else, one needs to manually run several such methods and hopes that one of them will be optimal for the shapes at hand. Our unified segmentation proposed here shows that one can use a single descriptor—surface skeletons—to automatically compute *and* combine both segmentation types. This allows users to simply "drop" their shapes in the tool and let it choose the optimal mix of parts and patches to segment with.

Performance: Table 4.1 shows the performance of our unified method, implemented in single-threaded C++, on an Intel Core i7 3.8 GHz PC with 32 GB RAM. The tested

Table 4.2 Performance of the VCS part-based segmentation method of Feng et al. [16]

Shape	$\|\partial\Omega\|$	$\|\mathcal{CS}\|$	t_{skel}	t_{cuts}	T_{VCS}
Horse	109,555	884	1.24	9.58	10,109
Hound	245,759	1530	1.51	23.24	16,179
Cow	143,938	1009	0.96	8.15	17,820
Bird	45,638	476	0.18	2.28	9527
Pig	145,215	959	1.51	10.97	12,694
Vertebra	68,632	683	0.37	8.56	5472
Scapula	285,854	4329	30.0	301.3	4106
Heptoroid	651,478	4873	3.36	400.5	7926
Kidney	31,874	403	0.16	3.91	3278
Hand	58,071	584	0.22	2.15	15,773

shapes vary considerably in terms of mesh resolution ($\|\partial\Omega\|$) and mesh-sampling uniformity, and also in terms of the number of cuts we compute ($\|\mathcal{CS}_\alpha\|$). We next see that the total cost is dominated by the cut-space computation. At first sight, the overall cost appears to be quite high. To better assess this cost, we compute the throughput of our method

$$T_{unified} = \frac{1}{1000}\frac{|\partial\Omega\| \cdot \|\mathcal{CS}_\alpha\|}{t_{cuts}}, \tag{4.16}$$

i.e., the number of cuts, for a given resolution of the mesh (in thousands of vertices) that the method can deliver per second. Using the mesh resolution in Eq. (4.16) accounts for the fact that the cut computation cost is proportional with the mesh resolution. Table 4.2 shows the throughput T_{VCS} for the original voxel-based VCS method. Here, $\|\partial\Omega\|$ denotes the number of voxels on the shape surface. The ratio of the average T_{VCS} to the average $T_{unified}$ is 2.32, i.e., the VCS method is 2.32 times faster than ours. However, the original VCS method does use CPU parallelism (8 threads) to compute cuts, whereas our method is purely sequential. Parallelizing our cut computation is very easy since cuts are traced completely independently. This makes our method potentially over three times faster than the VCS method. Further, using GPU parallelism to compute the cuts and following the technique originally proposed by [23] for computing surface-skeleton importance would make our method significantly faster than the VCS method, which lends itself far less to GPU parallelization. Interestingly, we see that the relative throughput of our method as compared to VCS increases significantly for shapes where many cuts are used, e.g., *scapula* and *heptoroid*—for the second shape, our method has an even higher throughput than VCS. Memory-wise, our method needs to store only the input mesh, point-cloud skeleton, and two feature points per skeleton point, its memory cost being $O(\|\partial\Omega\|)$. In contrast, VCS needs to store four full densely-sampled volumes, yielding a memory cost of $O(\|\partial\Omega\|^{3/2})$ (for details, see the underlying skeletoniza-

tion method [24]). All in all, we see that our method scales far better than VCS with respect to shape size.

Implementation: Although at first sight complex, our unified method requires only a few ingredients to be implemented: the point-cloud skeletonization method for mesh models in [23], which delivers surface skeletons, distance transforms, and feature transforms; and a way to find the nearest neighbors in a point cloud (Eqs. (4.8) and (4.12)). The former is provided by the respective algorithm, and the latter is readily available via the ANN package [37].

Limitations: Although delivering good-quality segmentations of complex shapes, our unified method still has several limitations. First and foremost, its quality essentially depends on the underlying qualities of the parts and patches delivered by its two branches (Section 4.3). Although the part and patch validation steps we introduce considerably help delivering good quality parts and patches, the overall result still depends on the possibility of *independently* finding good parts and patches on a shape. There are, obviously, cases where this assumption does not hold. In such cases, designing a new "joint segment" model able to capture the full continuum between parts and patches is desirable, and is a challenge for future work.

Secondly, our unified method is technically more complex than other part-based and patch-based segmentation methods taken separately. Indeed, it comprises a full part-based segmentation pipeline (7 steps, Sections 4.3.2–4.3.7), a full patch-based segmentation pipeline (Section 4.3.8.1), and a unification pipeline (two steps, Section 4.3.8.3). However, this is justified by the fact that our method is, to our knowledge, the only existing one that can handle mixed part-patch-based segmentations with good results.

The comparison of our method with related segmentation methods [17,16,40,26,33,2,10,55,32] is, of course, not covering the entire spectrum of segmentation methods out there. This would be highly challenging, if not impossible, to do given the available space and the availability of implementations of such methods. More effort (from the entire shape segmentation community) is required here. Yet, we argue that our main point, showing that our method can leverage surface skeletons to generate part-patch-based segmentations than other tools in the same class, has been well defended by the presented results.

4.6 CONCLUSION

In this chapter, we have presented a novel method to create segmentations of 3D mesh shapes. In contrast to most existing segmentation methods out there, we show that it is possible to design a method that effectively handles tubular (articulated) shapes, faceted shapes, and shapes containing a mix of the two. On a more fundamental level, we show that surface skeletons are an effective tool to support complex segmentation tasks. Thereby, we strengthen the existing evidence that this type of

medial descriptors, so far sparsely used in actual applications, are effective tools with practical added value. We support our claims by comparing our method with the most relevant part-based and patch-based segmentation methods using 3D skeletons on a collection of shapes ranging from purely faceted to purely tubular.

Future work can handle several directions. First and foremost, the current results show that it is possible to create hybrid part-patch segmentations using a single descriptor type (surface skeletons). As such, it is interesting to explore refinements of the presented part and patch detection and merging heuristics to design methods where users can control the resulting segmentation more easily and intuitively. Secondly, low-hanging fruits include the GPU acceleration of all steps of the proposed pipeline to yield a single method able to compete speed-wise with current state-of-the-art segmentation methods.

REFERENCES

[1] C. Arcelli, G. Sanniti di Baja, L. Serino, Distance-driven skeletonization in voxel images, IEEE Trans. Pattern Anal. Mach. Intell. 33 (4) (2011) 709–720.
[2] M. Attene, B. Falcidieno, M. Spagnuolo, Hierarchical mesh segmentation based on fitting primitives, Vis. Comput. 22 (3) (2006).
[3] M. Attene, S. Katz, M. Mortara, G. Patané, M. Spagnuolo, A. Tal, Mesh segmentation – a comparative study, in: Proc. IEEE SMI, 2006, pp. 134–141.
[4] O.K.C. Au, C. Tai, H. Chu, D. Cohen-Or, T. Lee, Skeleton extraction by mesh contraction, in: Proc. ACM SIGGRAPH, 2008, pp. 441–449.
[5] J. Bloomenthal, C. Bajaj, J. Blinn, M.-P. Cani, A. Rockwood, B. Wyvill, G. Wyvill, Introduction to Implicit Surfaces, Morgan Kaufmann, 1997.
[6] H. Blum, A transformation for extracting new descriptors of shape, in: Models for the Perception of Speech and Visual Form, MIT Press, 1967, pp. 362–380.
[7] M. Botsch, L. Kobbelt, M. Pauly, P. Alliez, B. Lévy, Polygon Mesh Processing, A K Peters, 2010.
[8] M. Braunstein, D. Hoffman, A. Saidpour, Parts of visual objects: and experimental test of the minima rule, Perception 18 (1989) 817–826.
[9] M. Chang, F. Leymarie, B. Kimia, Surface reconstruction from point clouds by transforming the medial scaffold, Comput. Vis. Image Underst. 113 (11) (2009) 1130–1146.
[10] U. Clarenz, M. Griebel, M. Schewitzer, A. Telea, Feature sensitive multiscale editing on surfaces, Vis. Comput. 20 (5) (2004) 329–343.
[11] D. Comaniciu, P. Meer, Mean shift: a robust approach toward feature space analysis, IEEE Trans. Pattern Anal. Mach. Intell. 24 (5) (2002) 603–619.
[12] N. Cornea, D. Silver, P. Min, Curve-skeleton properties, applications, and algorithms, IEEE Trans. Vis. Comput. Graph. 13 (3) (2007) 87–95.
[13] N. Cornea, D. Silver, X. Yuan, R. Balasubramanian, Computing hierarchical curve-skeletons of 3D objects, Vis. Comput. 21 (11) (2005) 945–955.
[14] J. Damon, Global medial structure of regions in $\mathbb{R}^3$, Geom. Topol. 10 (2006) 2385–2429.
[15] T. Dey, J. Sun, Defining and computing curve skeletons with medial geodesic functions, in: Proc. IEEE SGP, 2006, pp. 143–152.
[16] C. Feng, A.C. Jalba, A.C. Telea, Improved part-based segmentation of voxel shapes by skeleton cut spaces, Math. Morphol. Theory Appl. 1 (1) (2015) 1–20.

[17] C. Feng, A.C. Jalba, A.C. Telea, Part-based segmentation by skeleton cut space analysis, in: Proc. ISMM, Springer, Jan. 2015, pp. 1–12.
[18] M. Foskey, M. Lin, D. Manocha, Efficient computation of a simplified medial axis, in: Proc. SMA, 2003, pp. 135–142.
[19] P. Giblin, B.B. Kimia, A formal classification of 3D medial axis points and their local geometry, IEEE Trans. Pattern Anal. Mach. Intell. 26 (2) (2004) 238–251.
[20] A. Golovinskiy, T. Funkhouser, Randomized cuts for 3D mesh analysis, ACM Trans. Graph. 27 (2008) 454–463.
[21] W. Hesselink, J. Roerdink, Euclidean skeletons of digital image and volume data in linear time by the integer medial axis transform, IEEE Trans. Pattern Anal. Mach. Intell. 30 (12) (2008) 2204–2217.
[22] D. Hoffman, W. Richards, Parts of recognition, Cognition 18 (1984) 65–96.
[23] A. Jalba, J. Kustra, A. Telea, Surface and curve skeletonization of large 3D models on the GPU, IEEE Trans. Pattern Anal. Mach. Intell. 35 (6) (2013) 1495–1508.
[24] A.C. Jalba, A. Sobiecki, A.C. Telea, An unified multiscale framework for planar, surface, and curve skeletonization, IEEE Trans. Vis. Comput. Graph. 38 (1) (2016) 30–45.
[25] S. Katz, A. Tal, Hierarchical mesh decomposition using fuzzy clustering and cuts, ACM Trans. Graph. 22 (2003) 954–961.
[26] J. Kustra, A. Jalba, A. Telea, Computing refined skeletal features from medial point clouds, Pattern Recognit. Lett. 76 (2014) 13–21.
[27] J. Kustra, A. Jalba, A. Telea, Robust segmentation of multiple intersecting manifolds from unoriented noisy point clouds, Comput. Graph. Forum 33 (1) (2014) 73–87.
[28] Y. Lee, S. Lee, A. Shamir, D. Cohen-Or, Intelligent mesh scissoring using 3D snakes, in: Proc. IEEE Pacific Graphics, 2004, pp. 279–287.
[29] Y. Lee, S. Lee, A. Shamir, D. Cohen-Or, H.P. Seidel, Mesh scissoring with minima rule and part salience, Comput. Aided Geom. Des. 22 (2005) 444–465.
[30] F. Leymarie, B. Kimia, The medial scaffold of 3D unorganized point clouds, IEEE Trans. Vis. Comput. Graph. 29 (2) (2007) 313–330.
[31] X. Li, T.W. Woon, T.S. Tan, Z. Huang, Decomposing polygon meshes for interactive applications, in: Proc. ACM I3D, 2001, pp. 35–42.
[32] J. Lien, J. Keyser, N. Amato, Simultaneous shape decomposition and skeletonization, in: Proc. ACM SPM, 2005, pp. 219–228.
[33] R. Liu, H. Zhang, Segmentation of 3D meshes through spectral clustering, in: Proc. Pacific Graphics, 2004, pp. 298–305.
[34] J. Ma, S. Bae, S. Choi, 3D medial axis point approximation using nearest neighbors and the normal field, Vis. Comput. 28 (1) (2012) 7–19.
[35] A. Mangan, R. Whitaker, Partitioning 3D surface meshes using watershed segmentation, IEEE Trans. Vis. Comput. Graph. 5 (4) (1999) 308–321.
[36] H. Moreton, C. Séquin, Functional optimization for fair surface design, in: Proc. ACM SIGGRAPH, 1992, pp. 167–176.
[37] D. Mount, S. Arya, Approximate nearest neighbor search software, www.cs.umd.edu/~mount/ANN, 2016.
[38] D. Page, A. Koschan, M. Abidi, Perception-based 3D triangle mesh segmentation using fast marching watersheds, in: Proc. IEEE CVPR, 2003, pp. 27–32.
[39] D. Reniers, A. Jalba, A. Telea, Robust classification and analysis of anatomical surfaces using 3D skeletons, in: Proc. VCBM, Eurographics, 2008, pp. 61–68.
[40] D. Reniers, A. Telea, Part-type segmentation of articulated voxel-shapes using the junction rule, Comput. Graph. Forum 27 (7) (2008) 1837–1844.

[41] D. Reniers, A. Telea, Patch-type segmentation of voxel shapes using simplified surface skeletons, Comput. Graph. Forum 27 (7) (2008) 1837–1844.
[42] D. Reniers, A.C. Telea, Skeleton-based hierarchical shape segmentation, in: Proc. IEEE SMA, 2007, pp. 179–188.
[43] D. Reniers, J.J. van Wijk, A. Telea, Computing multiscale skeletons of genus 0 objects using a global importance measure, IEEE Trans. Vis. Comput. Graph. 14 (2) (2008) 355–368.
[44] L. Serino, C. Arcelli, G.S. di Baja, From skeleton branches to object parts, Comput. Vis. Image Underst. 129C (2014) 42–51.
[45] L. Serino, G.S. di Baja, C. Arcelli, Using the skeleton for 3D object decomposition, in: Proc. SCIA, in: Lect. Notes Comput. Sci., Springer, 2011, pp. 447–456.
[46] A. Shamir, A survey on mesh segmentation techniques, Comput. Graph. Forum 27 (6) (2008) 1539–1556.
[47] L. Shapira, A. Shamir, D. Cohen-Or, Consistent mesh partitioning and skeletonisation using the shape diameter function, Vis. Comput. 24 (2008) 249–259.
[48] K. Siddiqi, S. Bouix, A. Tannenbaum, S. Zucker, Hamilton–Jacobi skeletons, Int. J. Comput. Vis. 48 (3) (2002) 215–231.
[49] K. Siddiqi, S. Pizer, Medial Representations: Mathematics, Algorithms and Applications, Springer, 2009.
[50] M. Singh, G. Seyranian, D. Hoffman, Parsing silhouettes: the short-cut rule, Percept. Psychophys. 61 (4) (1999) 636–660.
[51] A. Tagliasacchi, I. Alhashim, M. Olson, H. Zhang, Skeletonization by mean curvature flow, in: Proc. Symp. Geom. Proc., 2012, pp. 342–350.
[52] A. Tagliasacchi, T. Delame, M. Spagnuolo, N. Amenta, A. Telea, 3D skeletons: a state-of-the-art report, Comput. Graph. Forum (2016), http://dx.doi.org/10.1111/cgf.12865.
[53] G. Taubin, Estimating the tensor of curvature of a surface from a polyhedral approximation, in: Proc. ICCV, 1995, pp. 902–907.
[54] G. Taubin, Geometric signal processing on polygonal meshes, in: Proc. Eurographics – STARs, Eurographics Association, 2000.
[55] J. Tierny, J. Vandeborre, M. Daoudi, Topology driven 3D mesh hierarchical segmentation, in: Proc. SMI, 2007, pp. 215–220.