Research Article **Open Access**

Cong Feng*, Andrei C. Jalba, and Alexandru C. Telea

Improved Part-Based Segmentation of Voxel Shapes by Skeleton Cut Spaces

Abstract: We present a refined method for part-based segmentation of voxel shapes by constructing partitioning cuts from every voxel of the shape's medial surface. Our cuts have several desirable properties – smoothness, tightness, and orientation with respect to the shape's local symmetry axis, making it a good segmentation tool. We analyze the space of all cuts created for a given shape and detect cuts which are good segment borders. We present a detailed analysis of the parameter space of our method, which yields good preset values for all its parameters. Our method is robust to noise, pose invariant, independent on the shape geometry and genus, and is simple to implement. We demonstrate our method for both automatic and interactive segmentation on a wide selection of 3D shapes.

Keywords: Part-based segmentation, 3D medial surfaces, geodesic cuts, hierarchical clustering

1 Introduction

Shape segmentation aims to decompose a 3D shape into a set of parts that obey certain application-related properties, and is used in many contexts such as image analysis, registration, content-based retrieval, and 3D modeling [43]. *Patch-based* segmentation detects quasi-flat segments whose borders follow local curvature maxima on the shape surface, and is most used for faceted shapes [39]. *Part-based* segmentation follows a semantics-oriented approach, aiming to find shape parts that one would intuitively perceive as being logically distinct, and is used for natural shapes [37].

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

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

***Corresponding Author: Cong Feng:** Institute Johann Bernoulli, University of Groningen, the Netherlands, E-mail: c.feng@rug.nl

Andrei C. Jalba: Dept. of Mathematics and Computer Science, TU Eindhoven, the Netherlands, E-mail: a.c.jalba@tue.nl

Alexandru C. Telea: Institute Johann Bernoulli, University of Groningen, the Netherlands, E-mail: a.c.telea@rug.nl

used to construct PBS segmentations of 3D shapes. This makes this type of skeletal descriptors, which are so far rarely used in shape-processing applications, more interesting for practical purposes.

In this paper, we extend the related segmentation framework proposed in [13] with the following main contributions:

- We present a *clustering-based* segmentation technique using the shape cut-space, which works more robustly, and is easier to use, than the earlier histogram-based technique in [13];
- We present a detailed analysis of the *parameter space* of the entire pipeline, which allows us to find good preset values for the method's free parameters, and also gives detailed insight in the method's behavior;
- We present a new *application* of our cut-space segmentation technique for the interactive, user-driven, segmentation of 3D shapes.

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

The structure of this paper is as follows. Section 2 reviews related work. Section 3 presents the basic method and our proposed enhancements. Section 4 shows how we can use our method for interactive part-based segmentation of 3D shapes. Section 5 presents the parameter space analysis. Section 6 illustrates our method on a wide variety of 3D shapes and also compares it with related methods. Section 7 discusses our method. Section 8 concludes the paper.

2 Related Work

Two main segmentation approaches for 3D shapes exist [1, 5, 44]: *Patch-based* methods segment a shape's surface into quasi-flat patches bounded by sharp surface creases, and are suitable for synthetic shapes such as polyhedral models created by CAD-CAM applications. *Part-based* segmentation (PBS), our focus, cuts a shape's surface into its logical components. Such methods are suitable for shapes formed of articulated parts, *e.g.* human bodies, plants, and other natural structures that exhibit a part-whole hierarchical structure.

Most PBS methods find segments along what a human would see as logical shape parts, in two steps: (a) find *where* to cut a shape to isolate a part; and (b) find *how* to build a cut, once its location is set. These steps are addressed in different ways, as follows. Attene *et al.* segment a shape by fitting primitives from a predefined library to the shape's polygonal surface in a minimal-cost way. This approach works best when reverse-engineering shapes produced by CAD-like modeling, but less well for organic shapes [4]. Lee *et al.* construct partitioning cuts on a surface mesh by analyzing local mesh features such as curvature and centricity, using snakes to optimize for cut smoothness [23, 24]. Liu *et al.* encode the local similarity of faces in a mesh into an affinity matrix which they next decompose by spectral clustering to yield a segmentation [27]. Many similar clustering methods exist, such as based on algebraic multigrid clustering of the surface curvature matrix [9], or the fuzzy clustering approach in [20]. In a related way, mesh models can be segmented by watershed approaches applied to their surface curvature [30, 35]. An important issue of all clustering methods is that it is very hard to explicitly enforce global properties on the resulting cluster borders, which ultimately define the segmentation result.

As the topology of the shape skeleton or medial axis matches the part-whole shape structure [47], many methods use medial axes to place cuts. Au *et al.* use curve skeletons [6], where each skeleton branch maps to a part. Cuts are built by optimizing for cut concavity and length via minimal cuts [19]. Golovinskiy *et al.* create a large randomized cut-set and find part borders as the cuts on which most surface edges lie [15]. Shapira *et al.* note that skeletonization and segmentation are related, and compute a scalar shape-diameter function (SDF) on the shape surface to segments as surface faces with similar SDF values [45]. Their SDF function is related to approaches which compute histograms of shape thickness for shape retrieval tasks [28, 42]. Conversely, Lien *et al.* use shape decomposition to compute progressively refined curve skeletons [26]. Tierny *et al.* segment shapes hierarchically by topological and geometrical

analysis of their Reeb graphs, which are similar to curve skeletons [50]. Chang *et al.* compute shape medial surfaces, separate their manifolds, and back project each manifold on the shape surface to find a segment [8]. A similar segmentation method, using high-resolution point-cloud skeletons computed on the GPU [17] along the method of Reniers *et al.* [40], is proposed in [22]. Similar high-resolution surface and curve skeletons can be computed in voxel space using an advection model [18]. However, such skeletons have not yet been used for segmentation purposes. Dey and Sun extract curve skeletons as the maxima of the medial geodesic function (MGF) which encodes the length of the shortest path between feature points of points in the shape [12], and segment tubular parts as those which minimize the eccentricity of such paths. Reniers *et al.* generalize the MGF metric to extract simplified surface and curve skeletons [40], and next construct a part for each branch of a shape's curve skeleton [37]. Part borders correspond to curve-skeleton junction points, and are created by shortest paths traced on the shape surface around these junctions [12]. However, curve skeletons can contain many spurious junctions which change widely when the shape is slightly perturbed. Reniers *et al.* alleviate this by heuristics that shift cut-points along the curve skeleton to optimize for cut stability and planarity [38]. Yet, this method cannot segment shapes of large geometric, but little topological, variability, like a pawn chess piece: Its curve skeleton has no junction points, so [38] cannot separate the pawn's head, body, and base, although these have different thicknesses.

Summarizing, the two elements of a good PBS (*where* to cut, and *how* to cut) are targeted in complement by different methods: Skeleton-based methods construct good partitioning-cuts efficiently, *e.g.* by shortest-paths [12, 38]. Yet, curve skeletons do not encode enough of the shape geometry. Global search methods that analyze a wide set of shape cuts offer good ways to select where to partition [15, 45]. Yet, they do not offer explicit constraints for the cut shapes, and exhaustive cut-space search is expensive. Our method combines the advantages of the two above classes of methods, while minimizing their limitations.

3 Method

Our method has a simple intuition: Say we want to cut the shape in Fig. 1a close to points $A \dots E$. Which properties should these cuts have to yield a 'natural' PBS? In other words: How would a human draw such cuts? Figure 1 a shows five *undesirable* cuts: A is noisy, although it crosses a perfectly smooth surface zone; B is self-intersecting; C and D are too loose (long); and E is unnaturally slanted – a human asked to cut the shape at that point would arguably do it so across the finger's symmetry axis. Figure 1 b shows five cuts for the same points, computed with the method in this paper. We argue that these cuts are more suitable for PBS than those in Fig. 1 a, as they are (1) tight, (2) locally smooth, (3) self-intersection free, (4) and locally orthogonal to the shape's symmetry axis. An additional property that cuts should satisfy is (5) being closed curves, so that they divide the shape's surface into different parts. Note that these properties follow well-known perceptual principles that model how humans understand shape and its parts, such as the minima and short-cut rules [7, 16, 49].

We construct such cuts as follows: First, we compute a simplified medial surface of the input shape (Sec. 3.1). For each medial point, we next construct a cut having the above properties (Sec. 3.2). This answers the question "how to cut". By analyzing the resulting cut-space, we next select a cut-set that gives us the borders of salient shape-parts (Sec. 3.3). This answers the question "where to cut".

3.1 Skeletonization

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

$$DT_{\partial\Omega}(\mathbf{x} \in \Omega) = \min_{\mathbf{y} \in \partial\Omega} \|\mathbf{x} - \mathbf{y}\|. \tag{1}$$

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

$$S_{\partial\Omega} = \{\mathbf{x} \in \Omega | \exists \{\mathbf{f}_1, \mathbf{f}_2\} \subset \partial\Omega, \mathbf{f}_1 \neq \mathbf{f}_2, \|\mathbf{x} - \mathbf{f}_1\| = \|\mathbf{x} - \mathbf{f}_2\| = DT_{\partial\Omega}(\mathbf{x})\} \tag{2}$$

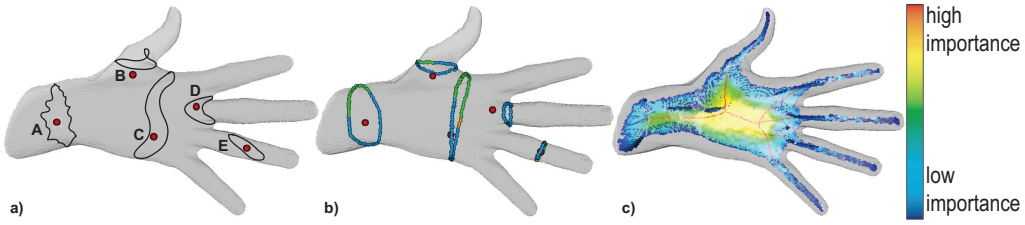

Fig. 1. Possible cuts for part-based segmentation. Suboptimal cuts (a). Cuts created by our method (b). Medial surface colored by its importance metric (c).

where $\mathbf{f}_1$ and $\mathbf{f}_2$ are the contact (or feature) points with $\partial\Omega$ of the maximally inscribed ball in Ω centered at $\mathbf{x}$ [14, 41, 47]. These, in turn, define the so-called feature transform $FT_{\partial\Omega} : \Omega \rightarrow \mathcal{P}(\partial\Omega)$

$$FT_{\partial\Omega}(\mathbf{x} \in \Omega) = \underset{\mathbf{y}\in\partial\Omega}{\operatorname{argmin}} \|\mathbf{x} - \mathbf{y}\|. \tag{3}$$

Medial surfaces are sensitive to small-scale noise on Ω, especially when using voxel-based models to sample the embedding space. To alleviate this, medial surfaces can be *regularized* by a computing a so-called importance metric $\rho : S_{\partial\Omega} \rightarrow \mathbb{R}_+$, such as the medial geodesic function (MGF), which sets $\rho(\mathbf{x})$ to the length of the shortest path on $\partial\Omega$ between the two feature points of $\mathbf{x}$ [12, 40]. As the MGF monotonically increases from the medial-surface boundary to its center, upper thresholding it by a minimal importance ρ_{min} yields connected and noise-free simplified medial surfaces (though tunnel preservation requires additional work) [40]. Figure 1c shows a regularized medial surface obtained by upper-thresholding the MGF metric in [40].

3.2 Cut model

The first step of our PBS is to compute a rich set of cuts, or cut-space $\mathcal{S}$, which all satisfy properties (1-5) listed in Sec. 3. To build a cut $c \in \mathcal{S}$, consider a point $\mathbf{x} \in S_{\partial\Omega}$. Here and in the following, we use for $S_{\partial\Omega}$ the simplified surface skeleton of Ω, obtained by upper-thresholding the MGF importance metric by a given value ρ_{min}. By definition, $\mathbf{x}$ has at least two feature points $\mathbf{f}_1$ and $\mathbf{f}_2$ on $\partial\Omega$ (Eqn. 2). Consider, for now, that there are precisely two such points. We first trace the shortest path $\gamma_1 \subset \partial\Omega$ between $\mathbf{f}_1$ and $\mathbf{f}_2$ (Fig. 2a), whose length is the MGF value for $\mathbf{x}$ (Sec. 3.1). Next, we find the midpoint $\mathbf{m}$ of γ_1, *i.e.* the voxel of γ_1 furthest in arc-length distance from both $\mathbf{f}_1$ and $\mathbf{f}_2$. We then trace a ray through $\mathbf{x}$ and oriented in the direction $\mathbf{x} - \mathbf{m}$, and find the point $\mathbf{o}$ where this ray 'exits' Ω (Fig. 2b). Intuitively, $\mathbf{o}$ is on the 'other side' of $S_{\partial\Omega}$ as opposed to $\mathbf{m}$. Finally, we trace the two shortest paths on $\partial\Omega$ connecting $(\mathbf{f}_1, \mathbf{o})$ and $(\mathbf{f}_2, \mathbf{o})$ respectively (Fig. 2 c,d). Our final cut c for point $\mathbf{x}$ is given by $\gamma_1 \cup \gamma_2 \cup \gamma_3$.

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

3.2.1 Cut properties:

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

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

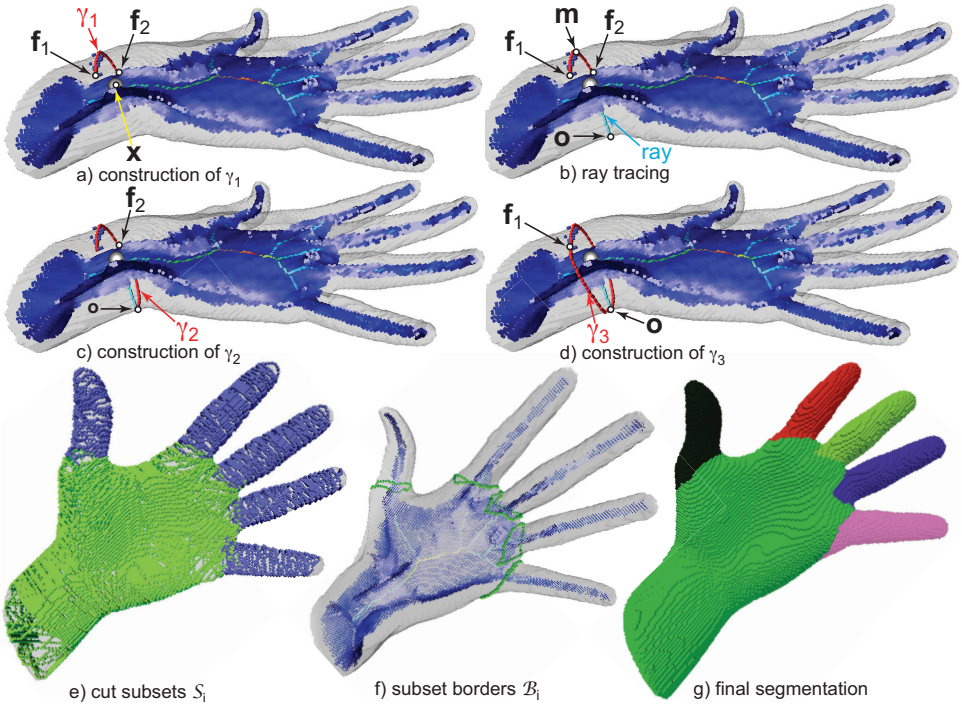

Fig. 2. Cut construction (a-d) and cut-space analysis (e-g) for part-based segmentation.

2. *Smooth:* Smoothness is guaranteed by the same properties as for tightness, *i.e.*, piecewise geodesicness and constrained Laplacian smoothing;
3. *Self-intersection free:* c is a geodesic triangle (three geodesics linking three *different* points on $\partial\Omega$) whose edges do not intersect except at endpoints, by definition;
4. *Locally orthogonal* to the symmetry axis: The cut $c(\mathbf{x})$ surrounds the medial surface $S_{\partial\Omega}$ around point $\mathbf{x}$, by construction. Hence, it also surrounds the so-called *curve skeleton* of $\partial\Omega$, which is a 1D structure locally centered within $S_{\partial\Omega}$ with respect to its boundary $\partial S_{\partial\Omega}$. While we do not have a formal proof of local orthogonality, we observed in practice that our construction always creates cuts that are visually orthogonal to the curve skeleton;
5. *Closed:* The cut c is a closed (Jordan) curve by construction.

3.2.2 Implementation

Our method requires the efficient and robust computation of regularized medial surfaces for 3D voxel shapes. For this, we tested the methods in [40] and [18]. As also described in [18], both methods produce very similar medial surfaces, and also deliver a skeleton importance metric, required to simplify skeletons to *e.g.* eliminate noise or small details. As the method in [18] is on average 10 times faster, we use this technique to compute our surface skeletons.

To build γ_1, we need two feature points $\mathbf{f}_1$ and $\mathbf{f}_2$ for each medial surface point $\mathbf{x}$. Two issues exist here: (1) Computing the feature transform $FT(\mathbf{x})$ on digital shapes cannot be done via Eqn. 3, given the finite voxel grid resolution [36, 40]. To fix this, we compute the so-called extended feature transform $EFT(\mathbf{x})$ which finds all closest-points on $\partial\Omega$ to all 26 neighbors of $\mathbf{x}$, and which is a superset of $FT(\mathbf{x})$ [40]. From this superset, we select exactly two feature points that best represent the symmetric embedding of $S_{\partial\Omega}$ in Ω. For this, we select the two feature points $\{\mathbf{f}_1, \mathbf{f}_2\} \subset EFT(\mathbf{x})$ that maximize the angle $\widehat{\mathbf{f}_1\mathbf{x}\mathbf{f}_2}$. We trace the ray used to find $\mathbf{o}$ by Bresenham's 3D line-tracing algorithm on the voxel shape. We compute geodesics by Dijkstra's shortest-path algorithm on the connectivity graph of voxels of $\partial\Omega$, using A^* heuristics to speed the search, and using edge weights that approximate neighbor-voxel distances by Eppstein's scheme [21] for better path-length accuracy. Finally, we reproject Laplacian-smoothed points on the shape surface by using the fast ANN library for finding nearest-neighbors [33].

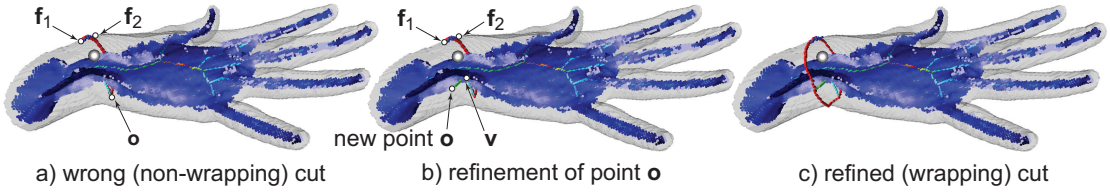

Fig. 3. Refinement of cut construction.

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

3.3 Cut space partitioning

For any voxel $\mathbf{x}$ of a shape's medial surface $S_{\partial\Omega}$, we can create a cut $c(\mathbf{x})$ which has good properties for PBS. Intuitively, $c(\mathbf{x})$ is a good way to cut the shape at point $\mathbf{x}$, *if* we want a cut there. We now must decide *where* we want to cut to get a PBS with desired global properties. Let $\mathcal{S} = \{c(\mathbf{x}) | \mathbf{x} \in S_{\partial\Omega}\}$ be the space of all cuts created from $S_{\partial\Omega}$. Given our cut properties, cuts on the same shape-part share similar properties *e.g.* position, orientation, and length. Cuts for different parts have different properties. Consider our hand model: Finger cuts are short; wrist cuts have average length; and palm cuts are longest. For a shape having a rump and protruding parts, cuts for parts are shorter than cuts for the rump. We use these insights to partition $\mathcal{S}$ in subsets $\mathcal{S}_i$ so that $\cup_i \mathcal{S}_i = \mathcal{S}$ and $\mathcal{S}_i \cup \mathcal{S}_{j \neq i} = \varnothing$. We discuss next two ways to achieve this partitioning.

3.3.1 Histogram-based partitioning

A first way to do this is to use the histogram of cut lengths over $\mathcal{S}$, as described in [13]. This works as follows. Histogram peaks show large similar-length cuts, so partitioning it by thresholds in the valleys between peaks gives our desired subsets $\mathcal{S}_i$. Figure 4a shows the cut-length histogram for the hand model. Its three main peaks describe cuts on the fingers, wrist, and palm; the two valleys give the two thresholds needed to separate fingers from the palm and the palm from the wrist. Figure 4b shows the final segmentation computed by partitioning the histogram into the three aforementioned parts.

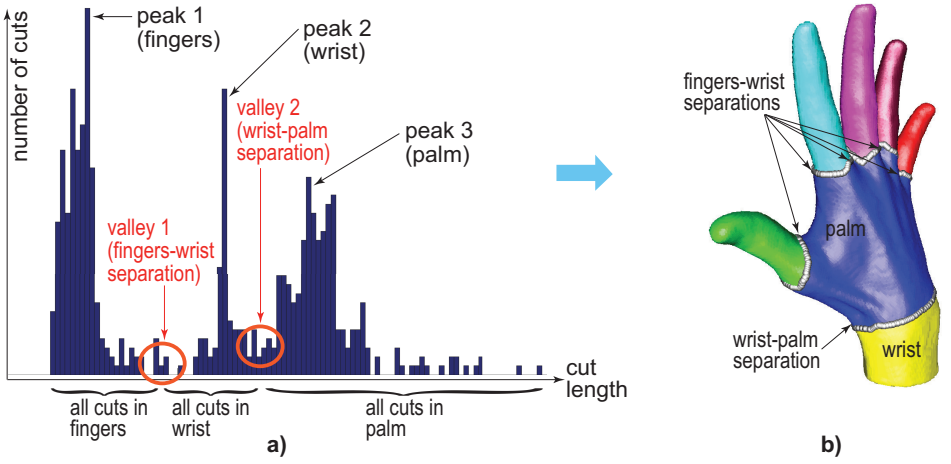

Fig. 4. (a) Cut-length histogram for hand model. (b) Segmented hand model based on left histogram (Sec. 3.3.1).

An important problem of the histogram-based partitioning is how to find its valleys *robustly* and *automatically*. As visible in Fig. 4a, the cut-length histogram is quite noisy, mainly due to the discrete nature of our cuts which are constructed in voxel space. Hence, robustly finding these valleys is a delicate process. To decrease the noise influence, we filter the histogram by mean shift [10]. This has the effect of 'sharpening' the cut-distribution and separate peaks from valleys more clearly. Following [13], we define a peak as a histogram value exceeding λ times the cut count $\|\mathcal{S}\|$, and a valley as a value less than a fraction μ of λ. Setting $\lambda \simeq 0.01$ and $\mu \simeq \lambda/3$ gives good results for a large range of shapes. However, problems appear for shapes having small-scale surface details. Such small details, on the one hand, cause noise-level variations in the cut lengths; on the other hand, their cut counts $\|\mathcal{S}\|$ are low, and thus separated from each other by very shallow, thus hard to detect, valleys in the histogram. Figure 5 shows such an example. Here, ideally, we would like to segment the limbs, head, and details (fingers, hears, muzzle) of the armadillo model. The four instances in the figure show different results obtained for quite similar values of λ and μ. All these results show various degrees of over- or undersegmentation. A second issue of the histogram-based partitioning is that it does not offer an intuitive control of the parameters λ and μ: We cannot easily determine optimal values for them based on the number of segments we would finally like to obtain.

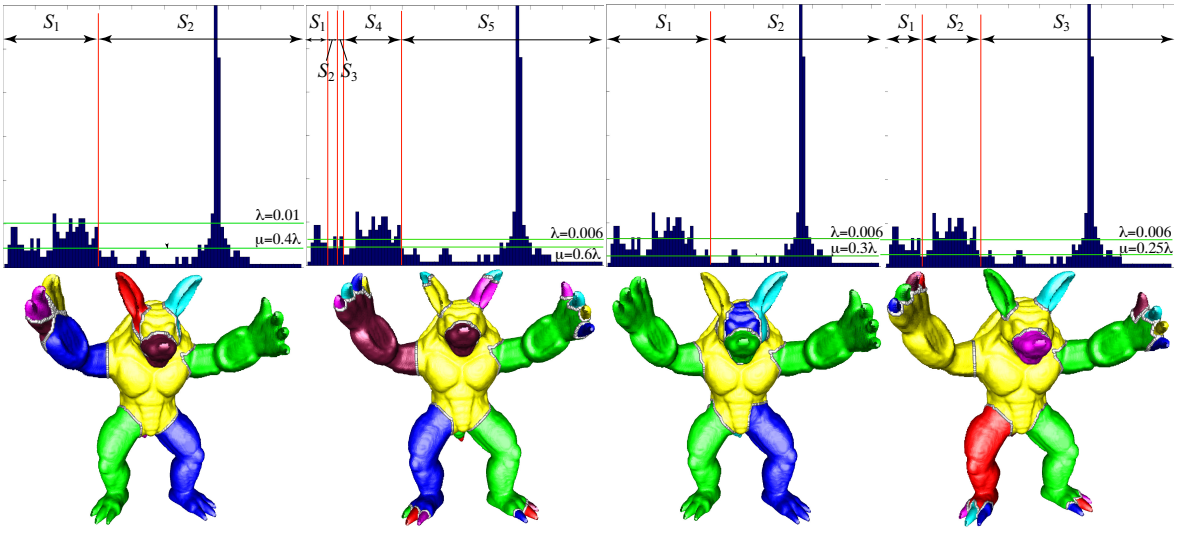

Fig. 5. Subsets and corresponding segmentations obtained for different values of λ and μ (Sec. 3.3.1).

3.3.2 Clustering-based partitioning

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

$$\delta(c_1 \in \mathcal{S}, c_2 \in \mathcal{S}) = \alpha\|l(c_1) - l(c_2)\| + \beta\|\mathbf{x}(c_1) - \mathbf{x}(c_2)\|, \tag{4}$$

where, for a cut c, $l(c)$ denotes the cut length and $\mathbf{x}(c)$ denotes the location of the skeleton-point from which c was generated, respectively (see Fig. 2); and $\alpha \in [0,1]$ and $\beta \in [0,1]$ are weight factors for the length, respectively distance, components of δ. To allow for a meaningful comparison between cut-lengths and cut-positions, we first normalize all cut lengths to the range $[0,1]$ and the voxel shape to the range $[0,1]^3$, respectively. The function δ will thus take low values for cuts which are similar in length and close to each other, and high values for cuts of different lengths and/or located far away over $\partial\Omega$. Next, we use hierarchical bottom-up agglomerative clustering to iteratively group all cuts in $\mathcal{S}$, represented by the distance matrix given by δ. During this process, the most similar two cut-clusters, as

determined by a so-called *linkage* function, are iteratively merged in a new cluster, until a single cluster containing all cuts is obtained. This creates a binary tree, or dendrogram, D. Full algorithm details, including a public implementation, are available at [11]. Cutting D at a desired level from its root next gives us a set of nodes, which are precisely our partitions $\mathcal{S}_i$.

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

3.3.3 Segment border construction

Subsets $\mathcal{S}_i$ do not (yet) coincide with our desired segments. Indeed, an $\mathcal{S}_i$ can contain logically disjoint cuts of similar lengths – *e.g.* all cuts on the fingers (blue in Fig. 2 e) are in the same subset. Also, $\mathcal{S}$ does not fully cover $\partial\Omega$, since we compute it from the *simplified* medial surface (Sec. 3.2).This is shown by the gaps between cuts in Fig. 2 d. To fix this, the method in [13] proposes to define a cut $c(\mathbf{x})$ as being a border $\mathcal{B}_i$ of subset $\mathcal{S}_i$ if $c(\mathbf{x})$ belongs to a different subset than any of the cuts $c(\mathbf{y})$, where $\mathbf{y}$ are the 26-neighbors of $\mathbf{x}$ on $S_{\partial\Omega}$. Using this definition, we can find the set of cuts $\{\mathcal{B}_i\}$ that represent the borders of our final segments (Fig. 2 f). Note that, if a cut is marked as border, at least one of its neighbor cuts will be in a different cut subset, by definition. Hence, that neighbor cut will also be a border, so more than one border will be produced from a $3 \times 3 \times 3$ voxel neighborhood. To remove such duplicates, we keep, for each such neighborhood, the shortest border.

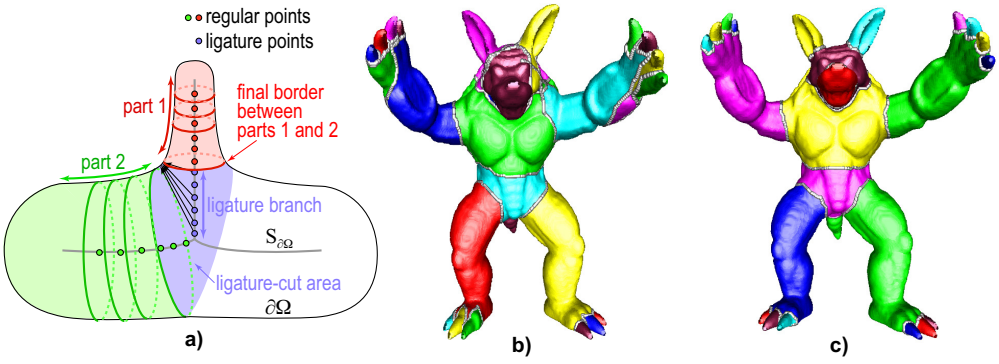

Fig. 6. (a) Border construction problems withture regions. Part borders computed by (b) the original method in [13] and (c) our new ligature-sensitive method (Sec. 3.3.3).

While this method finds borders located close to areas where different partitions $\mathcal{S}_i$ meet, it has problems for parts which meet along so-called *ligature* skeleton branches [47]. To explain this, consider the situation sketched in Fig. 6a. The skeleton $S_{\partial\Omega}$ consists here of three branches that correspond to the three shape parts that meet at the central junction point. Consider now the vertical branch that describes the thinner (red) part. The first part of this branch corresponds to so-called regular skeleton points, which have a one-to-one mapping with the shape surface $\partial\Omega$ via the feature transform $FT_{\partial\Omega}$. The second part of this branch contains ligature points (blue), which have a many-to-one mapping to $\partial\Omega$, as also indicated by the black feature vectors in the drawing. Ligature points do also generate cuts in our cut-space, like regular points. However, as compared to regular cuts, such as the red and green ones drawn in the figure, ligature cuts are far less stable – they can fall anywhere in the blue surface area indicated in the figure. Separately, note that our desired red and green segments will meet precisely in this ligature area, so their separating border, when computed by the method in [13], can fall anywhere in this area.

We propose next a way to fix this problem. Consider two parts $\mathcal{S}_1$ and $\mathcal{S}_2$ which are adjacent, *i.e.*, have at least two neighbor cuts in the sense described earlier in this section, *e.g.* the green and red parts in Fig. 6a. Let l_1 and l_2 be the average cut-lengths over $\mathcal{S}_1$ and $\mathcal{S}_2$ respectively. We next decide that the border $\mathcal{B}$ separating $\mathcal{S}_1$ from $\mathcal{S}_2$ should come from the part $\mathcal{S}_i$ that has the smaller average-length l_i. This heuristic models the idea that we want to cut the smaller part $\mathcal{S}_1$ as precisely as possible from the larger adjoining part $\mathcal{S}_2$. To find the exact location of this border, we proceed as follows. Let $\mathcal{S}_1$ be the part having the smaller average-length, *i.e.*, $l_1 < l_2$. We next collect all cuts $P = \{c_i\} \subset \mathcal{S}_1$ whose lengths $l(c_i)$ are smaller than $l_1 + a \cdot \sigma_1$, where σ_1 is the cut-length standard deviation over $\mathcal{S}_1$, and a is a constant set to 1.2 for all tested shapes. The set P skips the potential ligature-cuts in $\mathcal{S}_1$, which are longer than regular cuts. From this candidate border-set P, we next select the border $\mathcal{B}$ as being the cut which is geometrically closest to $\mathcal{S}_2$, *i.e.* cuts $\mathcal{S}_2$ as closely as possible with respect to $\mathcal{S}_2$. This also favors creating *short* borders, in line with requirement 1 (Sec. 3.2.1). For the shape in Fig. 6a, this yields the border $\mathcal{B}$ indicated in the drawing, which is outside the ligature area *and* is short. Figure 6 compares our new variance-based borders with the ones produced by the original method in [13] for a shape having many ligature regions (*e.g.* palm-hand, arm-torso, and ears-head junctions). As visible, the new borders separate the perceived shape segments better than the original ones, and are also shorter, while producing the same overall segmentation (number and location of parts).

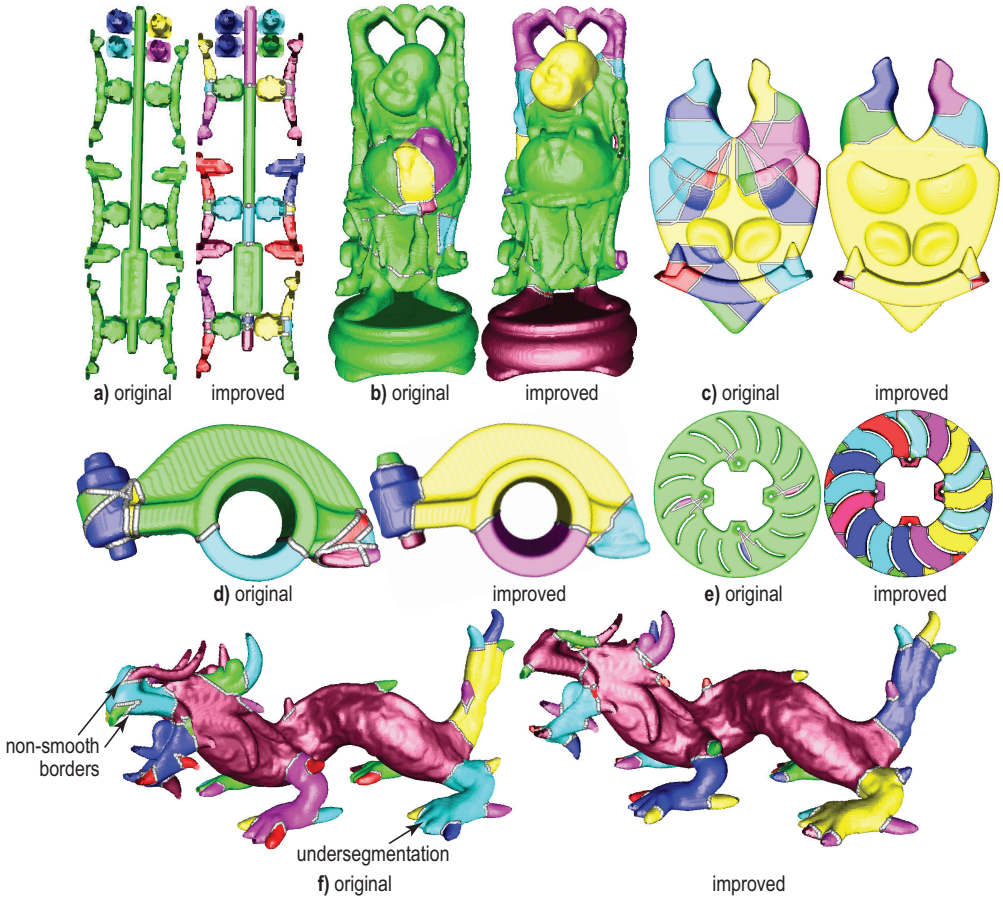

Fig. 7. Comparison of original border construction [13] and improved ligature-sensitive method (Sec. 3.3.3).

Once the part borders $\mathcal{B}_i$ are determined, we compute the final segments by finding the connected components of $\partial\Omega$ separated by these borders, via a simple flood-fill algorithm on $\partial\Omega$, and visualize these segments, for illustration, by coloring them so that adjacent segments get different colors (see Fig. 4b and following figures in the paper).

3.3.4 Final results

Figure 7 shows several examples where we compare our improved segmentation pipeline (using clustering-based partitioning of the cut space and ligature-sensitive border creation) with the original segmentation results in [13] (which use histogram-based partitioning and the ligature-agnostic border creation). For shapes (a,b,e) the new method removes the severe undersegmentation effects of the old method which are due to the difficulty of finding appropriate histogram thresholds λ and μ. For shapes (c,d), the new method removes the border instability due to skeleton ligature points, and creates tighter and better oriented segment borders. Image (f) shows that the new method detects more small-scale details and also creates smoother segment borders (see markers in figure). Additionally, images (b,d,e) show that our method can handle shapes of genus larger than zero, *i.e.*, having tunnels.

4 Interactive segmentation for shape editing

While useful, fully automated segmentation is not a solution in many contexts. Consider, for instance, the task of editing a 3D shape to *e.g.* enhance, deform, or remove certain features. The main time-consuming part here is accurately *selecting* the shape parts to process. This is typically done by interactive selection tools such as 2D or 3D bounding boxes or lasso tools [32, 52]. While such tools are quite efficient in a 2D setting, selecting details from complex 3D shapes still requires considerable user effort [9, 51, 52].

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

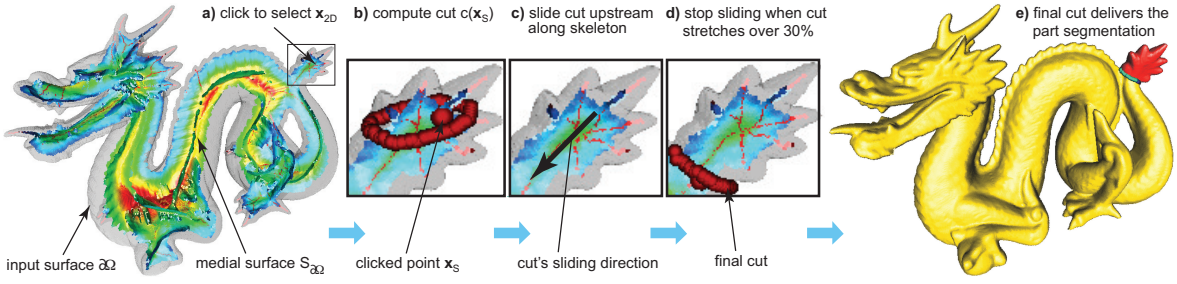

Fig. 8. Interactive segmentation pipeline, starting from clicking a surface point (a) until obtaining the surrounding part (e).

Our proposal works as follows (see Fig. 8). Given a 3D rendering of the input shape Ω, the user clicks on a surface point $\mathbf{x}_{2D}$ thereof, in a classical 2D rendering of Ω (Fig. 8a). We next determine the corresponding 3D point $\mathbf{x}_{3D} \in \partial\Omega$. Thirdly, we find the closest surface-skeleton point $\mathbf{x}_S \in S_{\partial\Omega}$ to $\mathbf{x}_{3D}$, using the inverse of the feature transform FT of $\partial\Omega$, and construct the corresponding cut $c(\mathbf{x}_{3D})$, following the method outlined in Sec. 3.2 (Fig. 8b). This cut represents a way to 'slice' the input shape based on the clicked location $\mathbf{x}_{2D}$. Assuming the user clicked *anywhere* on a shape detail, this does not yet give us the *entire* shape detail containing the point $\mathbf{x}_{3D}$. To segment this detail from the rest of the shape, we proceed as follows. Let $\rho(\mathbf{x}_S)$ be the MGF importance of the skeleton point $\mathbf{x}_S \in S_{\partial\Omega}$ (Sec. 3.1). We then move $\mathbf{x}_S$ along the medial surface $S_{\partial\Omega}$ upstream, with a distance of one voxel, in strictly increasing order of the medial-surface importance $\rho(\mathbf{x}_S)$ (Fig. 8c). Since the importance ρ increases *monotonically* from the medial-surface boundary to its center [18, 40], the point $\mathbf{x}_S$ moves strictly 'upstream' along the medial surface $S_{\partial\Omega}$, towards the center of the skeleton $S_{\partial\Omega}$, which is the point of maximal importance [18, 40]. We stop this motion when the cut-length $\|c(\mathbf{x}_S)\|$ for the current skeleton point $\mathbf{x}_S$ increases over 30% as compared to the previous skeleton point in this upstream motion process. Practically, this stops the upstream motion of $\mathbf{x}_S$ once the sliding cut reaches the location where

a part joins the main shape rump (Fig. 8d). Note that this location precisely corresponds to a large negative-curvature loop on the shape surface, which in turn is exactly the definition of the minima rules proposed by many researchers to segment protruding parts from a shape [7, 16, 19, 49]. When this event is detected, we use the current cut $c(\mathbf{x}_S)$ to separate the clicked shape part from the rump (Fig. 8e). Next, any shape-processing operations can be applied on this separated part, as desired by the user.

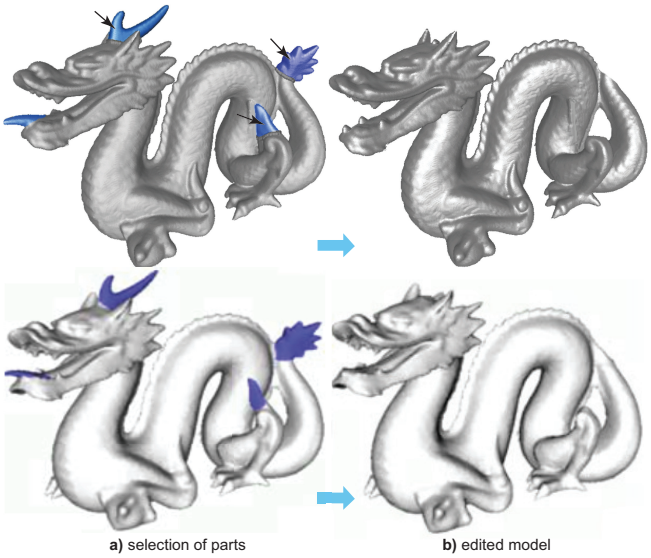

Fig. 9. (a) Model with selected parts in blue. (b) Model with deleted parts. Bottom row: Method of Clarenz *et al.* [9]. Top row: our method.

Figure 9 shows a simple example of such editing. Here, the user clicked three times, once inside each detail marked in blue in Fig. 9a, top-row. Using our part-selection procedure described earlier, we automatically select the three clicked parts, *i.e.*, the dragon's horn, tail, and hind leg spike. Next, we apply a simple erasing operation (for illustration purposes) to remove the selected details. Other shape-editing operations can be applied with the same ease, as desired. The final result is shown in Fig. 9b, top row. For comparison, Fig. 9 bottom-row shows the selection and editing operations performed by the related method of Clarenz *et al.* [9]. Our method achieves the same results, while being significantly simpler. Indeed, Clarenz *et al.* need to compute a differential surface classifier, encode it into a matrix, feed the matrix to an algebraic multigrid method that decomposes the matrix into a multiscale representation, select a suitable multiscale level, and threshold the basis functions representing the classifier on that level to find the clicked segment (for full details, we refer to [9]). In contrast, we only need to compute the shape's medial surface, select a point on it, and slide the cut generated by this point upstream the medial surface until its length increases by a desired threshold. Our interactive part selection method works in real-time as, upon a user click, we only need to compute a few tens of cuts from consecutive medial-surface points.

5 Parameter analysis

Our proposed segmentation pipeline involves several parameter values. For the method to be practically usable, end users need to understand (a) how these parameters affect the segmentation results, and (b) what are good preset values for them. In this section, we explore our method's parameter space and thereby address the above understanding goals. For this, we vary every parameter over its allowable range while keeping all other parameters at their preset values, and analyze the resulting segmentation results. The complete set of parameters of our method is listed in Tab. 1 and discussed next.

Description	Introduced in	Allowed values	Good preset
Skeleton simplification	**Sec. 3.2**	$\rho_{min} \in (0, 1)$	$\rho_{min} = 0.01$
Cut dissimilarity δ	**Eqn. 4**	$\{length\text{–}only\ (\beta = 0), length\text{–}and\text{–}position\ (\beta > 0)\}$	$length\text{–}only$
Linkage choice	**Sec. 3.3.2**	$\{single, centroid, full, average\}$	$average$
Input resolution	**Sec. 1**	$\|\Omega\| > 0$	$\|\Omega\| > 200^3$
Number of desired parts	**Sec. 3.3.2**	$N \in \mathbb{N}_{>0}$	**task-dependent**

Table 1. Complete set of method parameters with optimal preset values.

Simplification level: We use a simplified surface skeleton $S_{\partial\Omega}$ so as to avoid creating cuts from irrelevant spurious skeleton branches. Besides this, simplification allows removing skeleton details corresponding to small shape parts, to produce coarser segmentations. Thirdly, using simplified skeletons reduces computation time, as our method needs to create one cut per skeleton voxel. We empirically found that a skeleton simplification level of $\rho_{min} = 0.01\|\partial\Omega\|$ gives optimal results in terms of removing noise but keeping small shape details, and use this value as default for ρ_{min}. This result is in line with the independent observation that the same simplification level yields noise-free skeletons that capture all significant details of a 3D shape [40]. Additionally, simplified skeletons have voxels with large importance values, which in turn implies far-apart feature points $\mathbf{f}_1$ and $\mathbf{f}_2$ (see definition of the MGF importance metric in [12, 40]). This ensures that the ray casting used to compute cuts robustly finds cuts that wrap around the medial surface (Sec. 3.2). Figure 10 shows the effects of varying the simplification level ρ_{min} for the armadillo shape: Low ρ_{min} values capture finer-scale shape parts, while higher values produce coarser segmentations.

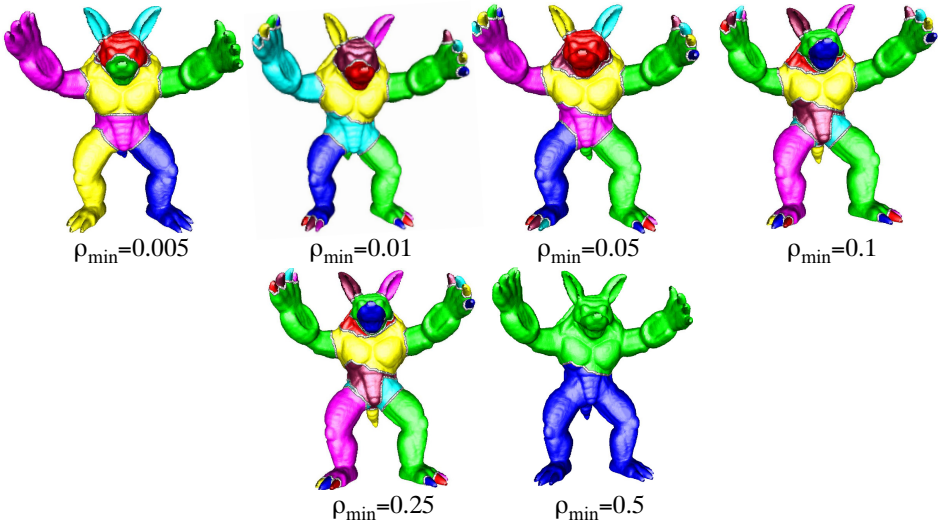

Fig. 10. Segmentation results as function of skeleton simplification level ρ_{min}.

Linkage choice: Hierarchical bottom-up clustering works by iteratively merging the two most similar cut-clusters. To compute the similarity of two clusters $\mathcal{S}_1$ and $\mathcal{S}_2$, a so-called linkage function is used [11]. Well-known variants hereof are *single* linkage (the minimum of all pairwise distances between cuts in $\mathcal{S}_1$ and $\mathcal{S}_2$); *full* linkage (the maximum of all pairwise distances between cuts in $\mathcal{S}_1$ and $\mathcal{S}_2$); *average* linkage (the average of all pairwise distances between cuts in $\mathcal{S}_1$ and $\mathcal{S}_2$); and *centroid* linkage (distance between the averages of cuts in $\mathcal{S}_1$ and $\mathcal{S}_2$). We tested all four linkage strategies for the shapes presented in this paper. An example is shown in Fig. 11. Single linkage yields no segmentation, since border-cuts are shared by adjoining segments, so the single linkage of such segments is zero. Centroid linkage typically produces a visible degree of undersegmentation, as the cut averaging acts like a low-pass filter eliminating the effect of small shape details. Full linkage, in contrast, yields a small amount of oversegmentation,

due to the maximum function involved in its computation. Finally, average linkage yields, in all tested cases, a balanced segmentation. As such, we set average linkage as the default value for our pipeline.

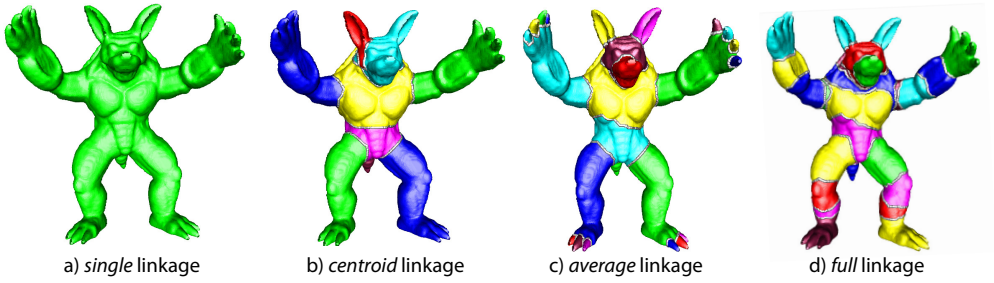

Fig. 11. Segmentation results as function of the linkage method used in hierarchical clustering.

Dissimilarity function: As explained in Sec. 3.3.2, we construct partitions by clustering by comparing cuts based on their length only or length-and-position, as determined by the ratio of the parameters α and β in Eqn. 4. To test the effect of these parameters, we fix $\alpha = 1$ (since we always want to compare cut lengths), vary β between 0 and 1, and analyze the produced segmentations. Figure 12 shows several results. For testing, we use here a shape exhibiting both thick and very thin parts and also having several elongated parts, so that both components of the dissimilarity function δ become important (Eqn. 4). We see that, when we use a non-zero importance β for the cut position (Fig. 12b), we obtain an oversegmentation of the length-only result: Long tubular-like parts, such as the trident shaft, torso, or limbs, are split into shorter segments. Also, we see a slight undersegmentation of details which only slightly differ in terms of local thickness, such as the bulge at the basis of the trident fork (Fig. 12a). Increasing β further yields an undersegmentation of the length-only result (Fig. 12c), as close cuts will be grouped in the same segment, regardless of their length – see *e.g.* the grouping of the trident spikes or fingers in the same segment. If oversegmentation of long tubular parts is not desired, then setting δ to length-only ($\beta = 0$) is a good default value. This is the value used for all examples in this paper except Fig. 12.

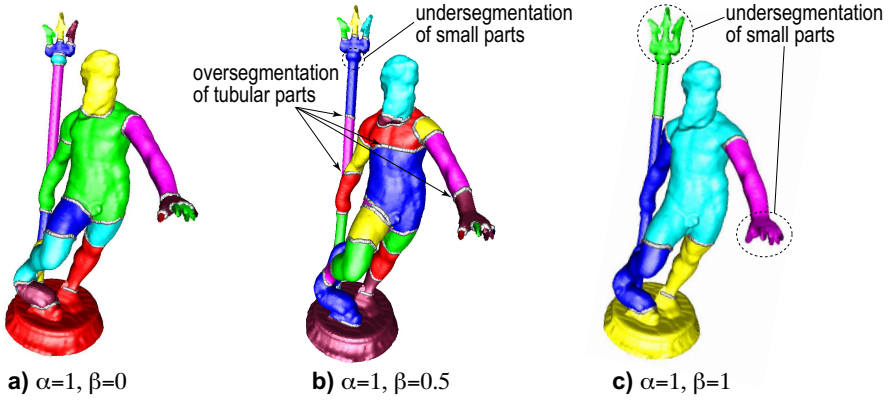

Fig. 12. Segmentation results as function of the dissimilarity function δ. (a) Length-only. (b,c) Length-and-position.

Resolution: As our entire pipeline works in voxel space, the sampling resolution, or number of voxels used to represent our input shape, its skeleton, and the cut-space, is an important parameter to examine. Figure 13 shows the segmentation results for four different resolutions. Overall, we see that the same segments are detected in all four cases, which tells that our method is robust with respect to sampling resolution. This is due to the fact that, once the used resolution is fine enough to capture skeletal details corresponding to small shape parts, then segments for those parts will be detected. Separately, we notice however an effect of the resolution in terms of *smoothness* of the produced cuts (see marked cut on the armadillo torso in Fig. 13). Low resolutions produce less smooth cuts, since the extended

feature transform EFT of the input shape becomes inaccurate (see Sec. 3.2.2), and thus the feature-points used in our cut construction get noisy. As the resolution increases, so does the accuracy of our EFT in approximating the true FT, and thus the cuts become smoother *and* more orthogonal to the local symmetry axis of the shape. Combining the previous observations, we noticed, in practice, that a resolution of 400^3 voxels is sufficient to capture all salient shape segments and also produce smooth and well-oriented cuts.

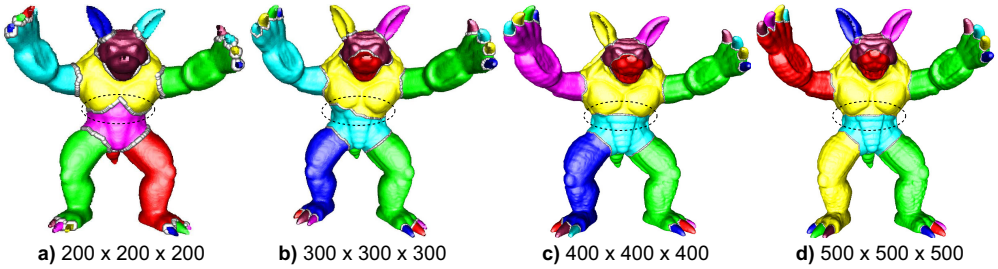

Fig. 13. Segmentation results as function of the voxel resolution of the input shape.

Number of desired parts: The last parameter of our pipeline determines the number of desired parts to be produced by segmentation (value N, Sec. 3.3.2). This is the *single* free parameter of our method. Its setting depends largely on the specific application context, *e.g.*, what is the scale of details that we consider relevant and thus want to segment separately; and what is the amount of noise that is present on the input shape, which we do not want to yield separate segments. As such, we leave the setting of N to the end user. Segmentations for different N values can be created *interactively*, since the most expensive part of our pipeline, skeleton computation and cut creation, needs to be done only once for a given shape (see Sec. 6, Tab. 2 next). Figure 14 shows three settings for N for two different models which have a clear part-whole structure. As visible, increasing N produces more detailed segmentations, in a multiscale fashion.

Fig. 14. Segmentation results for different numbers of desired parts (increasing from left to right).

6 Results and Comparison

We have tested our method on over 70 shapes provided as 3D polygon meshes, from the well-known shape repositories [2, 31], which we voxelized by *binvox* [34] at resolutions between 100^3 and 500^3 voxels. Results and comparison with related methods are discussed next.

Medial PBS methods: We first compare our results with [38], the best voxel-based PBS method that we are aware of which also uses medial descriptors for segmentation. We get very similar results, but find more fine-grained segments than [38] – see finger and ear details of the animal models, pig tail, dragon spikes, and microscope lens. Segment borders are smooth and locally orthogonal to the shape's symmetry axis, *i.e.*, similar to how a human would cut the shape at the respective places (Sec. 3). Our method finds segments of various sizes, ranging from details (dragon's tail, hound's ears), to large parts (limbs of various models).

General PBS methods: We next compare our method with a larger class of general-purpose PBS methods (Fig. 16 a-k). The considered methods are [4, 24–27, 37, 38, 50]. Here, Reniers *et al.* (1) denotes [37], and Reniers *et al.* (2) denotes [38]. These methods span from voxel-based to mesh-based, and use various segmentation heuristics (skeleton, curvature, salience, and topology-based). We argue that our method creates equally or, in some cases, more plausible PBSs. Since both our method and [38] use medial descriptors, computed by the same underlying method [40], a relevant question is how the two methods differ. We use (a) medial *surfaces*, while [38] uses *curve* skeletons; and (b) we find segment borders by analyzing *all* possible cuts, while [38] places such borders around the curve-skeleton branch junctions. Fig. 16 l-p shows five examples where the public implementation of [38] fails to segment at all. We find two causes for this: The shape parts in Fig. 16 l cannot be well described by curve-skeleton branches, as they are nearly rotationally symmetric. As few (if any) such junctions exist, [38] fails. The shape in Fig. 16 n is described by a mix of medial surfaces (base plate) and curve skeletons (tubular parts). As [38] only uses curve skeletons, data on the base plate is incomplete or missing. For the shapes in Fig. 16 m-p, the many heuristics in [38] to select cuts centered on the curve-skeleton fail, as they imply that such cuts should be nearly planar. This does not happen for the above shapes.

Multiscale: As described in Sec. 5, we can produce a multiscale segmentation by simply changing the number N of desired parts. This is a much simpler way to specify the desired level-of-detail than the earlier proposal in [13], where one had to simultaneously control two parameters λ and μ to yield the same result (see Sec. 3.3.1). Figure 16 r shows three such scales for the armadillo shape.

Invariance: Our method is pose invariant, as shown in Fig. 16 s. Indeed, our cut-space essentially captures local shape thickness, which does not depend on pose. Additionally, as the cut clustering essentially depends on the *relative* difference in cut lengths and positions, and not on their absolute values, our method is also scale, translation, and rotation invariant.

Performance: Table 2 shows the time for creating cuts (t_{cuts}), medial surfaces (t_{skel}), cut-space analysis (t_{space}), the total time of the original method in [13] (t_{total}), and total time for [38] ($t_{Reniers}$), for our method coded in C++ on an 8-core 3.5 GHz PC. As cuts are computed independently, we parallelized our method by *pthreads*, getting a speed boost factor of 7, close to the optimal value of 8 for our hardware. As visible, the original method [13] is slightly faster than [38]. We observe that most of the time is spent in the cut computation (t_{cuts} *vs* t_{total}). As such, we optimized the A^* method used to trace geodesics for cut construction (Sec. 3.2.2), by using a fast priority queue implementation. The performance yielded by this optimization (Tab. 2, t_{optim}) is now significantly higher than the original method (t_{total}) and also much higher then [38]. Finally, we note that our method could successfully segment all tested shapes, while [38] failed on several shapes (Tab. 2, empty cells in column $t_{Reniers}$).

7 Discussion

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

Global search: We create a PBS by finding all part-inducing cuts from the medial surface, and selecting a cut-subset by globally optimizing for part-similarity as captured by cut lengths and/or positions. In contrast to purely topological PBS methods [37, 38], we search a much wider space of possible partitionings; yet, our search space is much smaller than that of other methods which look for cuts of *any* possible orientation [15], thereby achieving a good flexibility-performance balance. This is also visible if we compare our running times (Tab. 2, t_{optim}) with those reported in [15]: We process

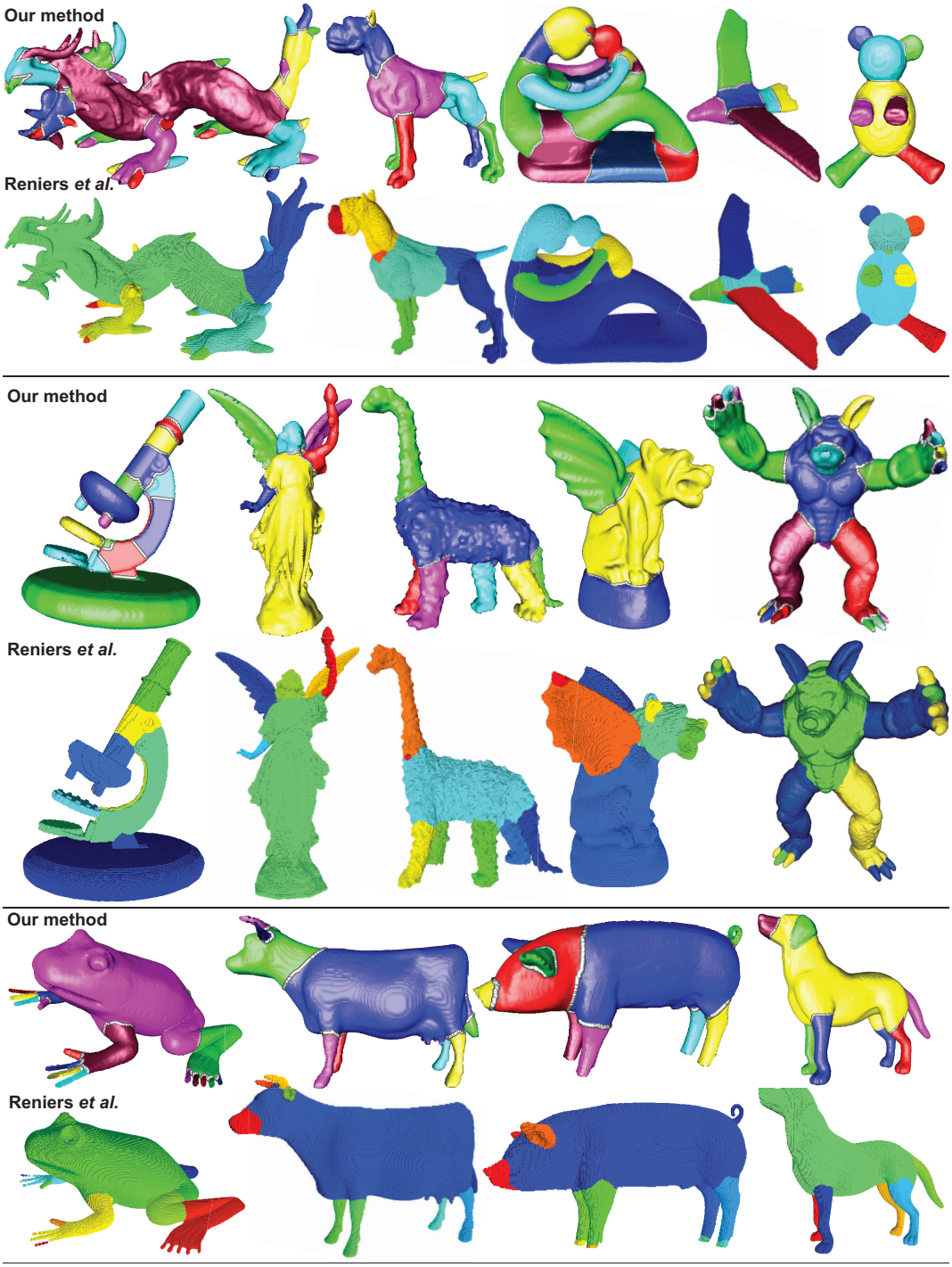

Fig. 15. Part-based segmentations of our method *vs* Reniers *et al.* [38] (Sec. 6).

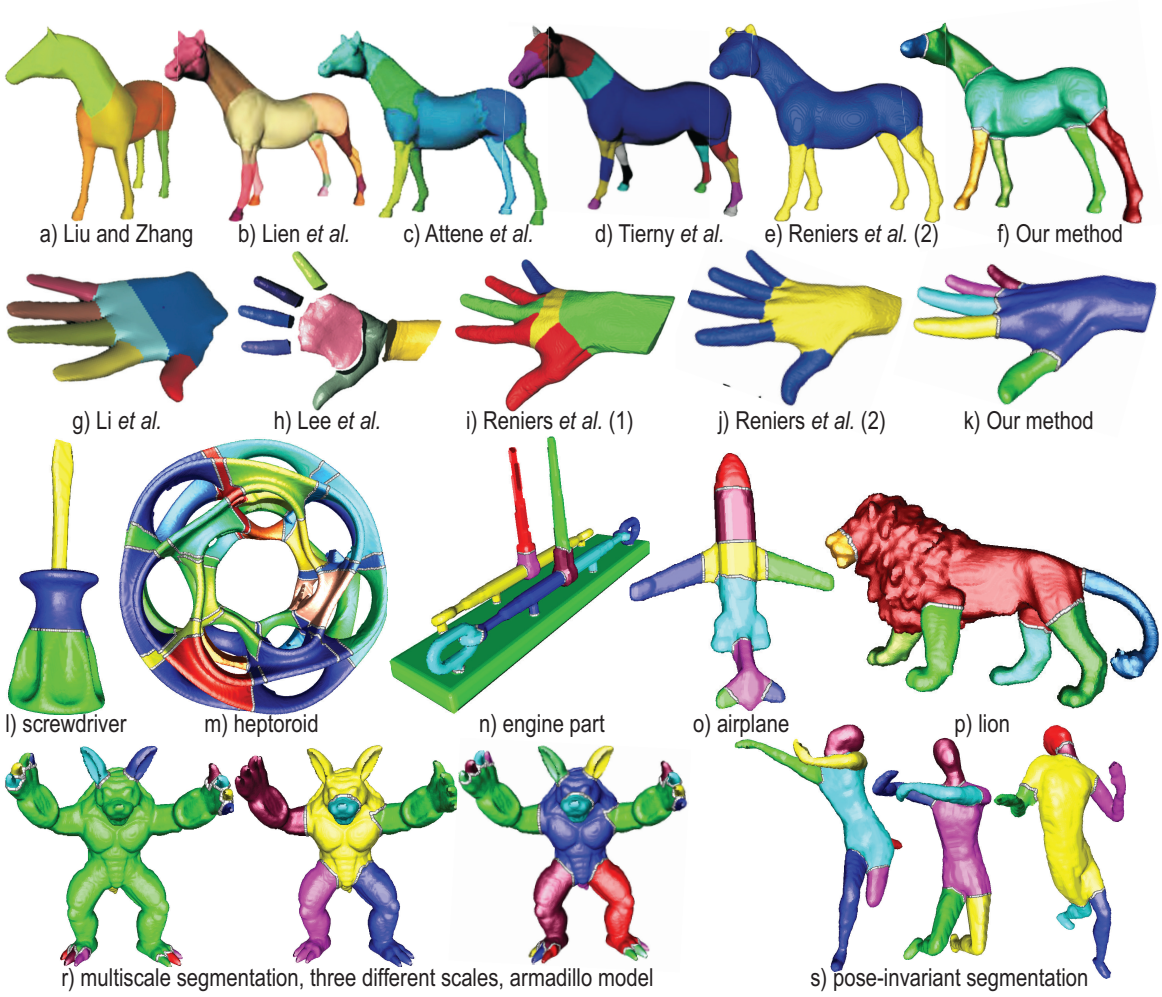

Fig. 16. Comparison of our method with eight PBS methods (a-k). Our results for shapes where Reniers *et al.* fails (l-p). Multiscale (r) and pose-invariant (s) segmentations.

voxel shapes having tens up to hundreds of thousands of surface voxels ($\|\partial\Omega\|$) in under 10 seconds; on similar hardware, [15] processes meshes having *only* 4000 triangles in 4 minutes on average.

Simplicity: In our approach, we can use *any* medial surface skeletonization method, *e.g.* [3, 18, 40, 41, 46], as long as it outputs regularized skeletons. This makes our method directly applicable to mesh-based shapes, which allow fast medial-surface extraction [17], without the additional cost of voxelization.

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

Invariance: Our method is scale, translation, rotation, and pose invariant [38, 48], as shown by the model in Fig. 16 s (which is also used in [48] to show pose invariance). Note that pose-invariance is not guaranteed by default by other cut-space segmentation methods, *e.g.* [15].

Robustness: We robustly segment noisy or detail-rich surfaces, *e.g. dragon* and *dino* (Fig. 15) or *lion* (Fig. 16). Segment borders are smooth by construction (Sec. 3.2). Since our segmentation uses a subset of these cuts, and only considers *integral* cut properties (length, position) rather than differential ones (*e.g.* curvature), noise and/or small-scale details are robustly handled. Moreover, we avoid constructing segment borders from unstable cuts created from ligature skeletal points (Sec. 3.3.3).

Limitations: Our method's cost is $O(\|S_{\partial\Omega}\|\|\partial\Omega\|\log\|\partial\Omega\|)$. As our method parallelizes easily (Sec. 6), its practical cost is much lower than other skeleton-based PBS methods [37, 38] or cut-based methods [15]. For space constraints, we compare with only eight related PBS methods. More PBS methods exist, and quantitative metrics can be further used to measure segmentation quality [29]. Yet, even without such

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

Table 2. Shape sizes and segmentation times (in seconds) for [13], our optimized method, and Reniers *et al.* [38].

extra insights, we argue that our goal of showing that *surface* skeletons have added both theoretical and practical value for PBS, as opposed to the well-known use of *curve* skeletons for PBS, is well defended.

8 Conclusions

We have presented a new method for part-based segmentation of 3D voxel shapes by analyzing the entire space of potential partitioning cuts constructed by using the shape's medial surface. To our knowledge, our approach is the first which uses medial *surfaces* for part-based segmentation, and thereby shows the added-value of medial surfaces for segmentation, as opposed to the well-known use of curve skeletons for the same task. We demonstrate our method on a wide variety of 3D shapes, and compare it with eight related segmentation methods. Our method can produce similar segmentations with less computational effort, and has a single intuitive end-user parameter to set – the number of desired segments.

Different ways to partition the cut space can be easily tried, *e.g.* cut similarities based on *e.g.* curvature, eccentricity, and orientation. This would lead to an entire family of PBS methods in a single simple implementation. Separately, cut-length-and-position histograms computed by our method could be an effective shape descriptor for retrieval and matching [42]. Finally, implementing our method for mesh-based shapes on the GPU should lead to massive scalability increases.

References

[1] Agates, A., Pratikakis, I., Perantonis, S., Sapidis, N., and Azariadis, P. (2007) 3D mesh segmentation methodologies for CAD applications. *Computer-Aided Design & Applications*, **4**, 827–841.

[2] Aim@Shape Consortium (2015), Aim@Shape repository. http://visionair.ge.imati.cnr.it/ontologies/shapes.

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

[4] Attene, M., Falcidieno, B., and Spagnuolo, M. (2006) Hierarchical mesh segmentation based on fitting primitives. *Visual Comput*, **22**, 181–193.

[5] Attene, M., Katz, S., Mortara, M., Patane, G., Spagnuolo, M., and Tal, A. (2006) Mesh segmentation - a comparative study. *Proc. SMI*, pp. 134–141.
[6] Au, O., Tai, C., Chu, H., Cohen-Or, D., and Lee, T. (2008) Skeleton extraction by mesh contraction. *ACM TOG (Proc. ACM SIGGRAPH)*, **27**, 441–449.
[7] Braunstein, M., Hoffman, D., and Saidpour, A. (1989) Parts of visual objects: and experimental test of the minima rule. *Perception*, **18**, 817–826.
[8] Chang, M., Leymarie, F., and Kimia, B. (2009) Surface reconstruction from point clouds by transforming the medial scaffold. *CVIU*, **113**, 1130–1146.
[9] Clarenz, U., Griebel, M., Rumpf, M., Schweitzer, M. A., and Telea, A. (2004) Feature sensitive multiscale editing on surfaces. *Visual Computer*, **20**, 329–343.
[10] Comaniciu, D. and Meer, P. (2002) Mean shift: A robust approach toward feature space analysis. *IEEE TPAMI*, **24**, 603–619.
[11] de Hoon, M. J. J., Imoto, S., Nolan, J., and Miyano, S. (2004) Open source clustering software. *Bioinformatics*, **19**, 1453–1454, software available at http://bonsai.hgc.jp/~mdehoon/software/cluster.
[12] Dey, T. and Sun, J. (2006) Defining and computing curve-skeletons with the medial geodesic function. *Proc. SGP*, pp. 143–152.
[13] Feng, C., Jalba, A., and Telea, A. (2015) Part-based segmentation by skeleton cut space analysis. *Proc. ISMM*, pp. 607–618.
[14] Giblin, P. and Kimia, B. (2004) A formal classification of 3D medial axis points and their local geometry. *IEEE TPAMI*, **26**, 238–251.
[15] Golovinskiy, A. and Funkhouser, T. (2008) Randomized cuts for 3D mesh analysis. *ACM TOG*, **27**, 454–463.
[16] Hoffman, D. and Richards, W. (1984) Parts of recognition. *Cognition*, **18**, 65–96.
[17] Jalba, A., Kustra, J., and Telea, A. (2013) Surface and curve skeletonization of large 3D models on the GPU. *IEEE TPAMI*, **35**, 1495–1508.
[18] Jalba, A., Sobiecki, A., and Telea, A. (2015) An unified multiscale framework for planar, surface, and curve skeletonization. *IEEE TPAMI*, dOI:10.1109/TPAMI.2015.2414420.
[19] Katz, S., Leifman, G., and Tal, A. (2005) Mesh segmentation using feature point and core extraction. *Visual Comput*, **21**, 649–658.
[20] Katz, S. and Tal, A. (2003) Hierarchical mesh decomposition using fuzzy clustering and cuts. *ACM TOG*, **22**, 954–961.
[21] Kiryati, N. and Szekely, G. (1993) Estimating shortest paths and minimal distances on digitized three-dimensional surfaces. *Pattern Recognition*, **26**, 1623–1637.
[22] Kustra, J., Jalba, A., and Telea, A. (2015) Computing refined skeletal features from medial point clouds. *Patt Recog Lett*, dOI:10.1016/j.patrec.2015.05.007.
[23] Lee, Y., Lee, S., Shamir, A., and Cohen-Or, D. (2004) Intelligent mesh scissoring using 3D snakes. *Proc. IEEE Pacific Graphics*, pp. 279–287.
[24] Lee, Y., Lee, S., Shamir, A., Cohen-Or, D., and Seidel, H. P. (2005) Mesh scissoring with minima rule and part salience. *CAGD*, **22**, 444–465.
[25] Li, X., Woon, T., Tan, T., and Huang, Z. (2001) Decomposing polygon meshes for interactive applications. *Proc. I3D*, pp. 35–42.
[26] Lien, J., Keyser, J., and Amato, N. (2006) Simultaneous shape decomposition and skeletonization. *Proc. ACM SPM*, pp. 219–228.
[27] Liu, R. and Zhang, H. (2004) Segmentation of 3D meshes through spectral clustering. *Proc. Pacific Graphics*, pp. 298–305.
[28] Liu, Y., Pu, J., Zha, H., Liu, W., and Uehara, Y. (2004) Thickness histogram and statistical harmonic representation for 3D model retrieval. *Proc. 3D Data Processing, Visualization and Transmission (3DPVT)*, pp. 896–903.
[29] Liu, Z., Tang, S., Bu, S., and Zhang, H. (2013) New evaluation metrics for mesh segmentation. *Computers & Graphics*, **37**, 553–564.
[30] Mangan, A. and Whitaker, R. (1999) Partitioning 3D surface meshes using watershed segmentation. *IEEE TVCG*, **5**, 308–321.
[31] McGill University (2015), McGill 3D Shape Benchmark. http://www.cim.mcgill.ca/~shape/benchMark.
[32] MeshLab Consortium (2015), MeshLab geometry processing tool. meshlab.sourceforge.net.
[33] Mount, D. and Arya, S. (2015), Approximate nearest-neighbor search. www.cs.umd.edu/~mount/ANN.
[34] Nooruddin, F. and Turk, G. (2003) Simplification and repair of polygonal models using volumetric techniques. *IEEE TVCG*, **9**.
[35] Page, D., Koschan, A., and Abidi, M. (2003) Perception-based 3D triangle mesh segmentation using fast marching watersheds. *Proc. IEEE CVPR*, pp. 27–32.
[36] Reniers, D. and Telea, A. (2007) Tolerance-based feature transforms. *Advances in Computer Graphics and Computer Vision*, pp. 187–200, Springer.

[37] Reniers, D. and Telea, A. (2008) Hierarchical part-type segmentation using voxel-based curve skeletons. *Visual Comput*, **24**, 383–395.
[38] Reniers, D. and Telea, A. (2008) Part-type segmentation of articulated voxel-shapes using the junction rule. *CGF*, **27**, 1845–1852.
[39] Reniers, D. and Telea, A. (2008) Patch-type segmentation of voxel shapes using simplified surface skeletons. *CGF*, **27**, 1837–1844.
[40] Reniers, D., van Wijk, J. J., and Telea, A. (2008) Computing multiscale skeletons of genus 0 objects using a global importance measure. *IEEE TVCG*, **14**, 355–368.
[41] Roerdink, J. and Hesselink, W. (2008) Euclidean skeletons of digital image and volume data in linear time by the integer medial axis transform. *IEEE TPAMI*, **30**, 2204–2217.
[42] Schmitt, W., Sotomayor, J., Telea, A., Silva, C., and Comba, J. (2015) A 3D shape descriptor based on depth complexity and thickness histograms. *Proc. IEEE SIBGRAPI*, pp. 261–267.
[43] Shamir, A. (2004) A formulation of boundary mesh segmentation. *Proc. 3DPVT*.
[44] Shamir, A. (2008) A survey on mesh segmentation techniques. *CGF*, **27**, 1539–1556.
[45] Shapira, L., Shamir, A., and Cohen-Or, D. (2008) Consistent mesh partitioning and skeletonisation using the shape diameter function. *Visual Comput*, **24**, 249–259.
[46] Siddiqi, K., Bouix, S., Tannenbaum, A., and Zucker, S. (2002) Hamilton-Jacobi skeletons. *IJCV*, **48**, 215–231.
[47] Siddiqi, K. and Pizer, S. (2008) *Medial representations: mathematics, algorithms and applications*. Springer.
[48] Siddiqi, K., Zhang, J., Macrini, D., Shoukofandeh, A., and Dickinson, S. (2008) Retrieving articulated 3D models using medial surfaces. *Mach. Vis. Appl.*, **19**, 261–275.
[49] Singh, M., Seyranian, G., and Hoffman, D. (1999) Parsing silhouettes: The short-cut rule. *Perception & Psychophysics*, pp. 636–660.
[50] Tierny, J., Vandeborre, J., and Daoudi, M. (2007) Topology driven 3D mesh hierarchical segmentation. *Proc. SMI*, pp. 215–220.
[51] Yu, L., Efstathiou, K., Isenberg, P., and Isenberg, T. (2012) Efficient structure-aware selection techniques for 3D point cloud visualizations with 2DOF input. *IEEE TVCG*, **18**, 2245–2254.
[52] Zwicker, M., Pauly, M., and Gross, O. K. M. (2002) Pointshop 3D: an interactive system for point-based surface editing. *Proc. ACM SIGGRAPH*, pp. 322–329.