Skeleton-based hierarchical shape segmentation

Citation for published version (APA):

Document status and date:
Published: 01/01/2006

Document Version:
Publisher’s PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:
• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher’s website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the “Taverne” license above, please follow below link for the End User Agreement:
www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:
openaccess@tue.nl
providing details and we will investigate your claim.
Skeleton-based Hierarchical Shape Segmentation

Dennie Reniers, Alexandru Telea
Department of Mathematics and Computer Science
Eindhoven University of Technology
PO Box 513, 5600 MB Eindhoven, The Netherlands
d.reniers@tue.nl, aleet@win.tue.nl

Abstract

We present an effective framework for segmenting 3D shapes into meaningful components using the curve skeleton. Our algorithm identifies a number of critical points on the curve skeleton, either fully automatically as the junctions of the curve skeleton, or based on user input. We use these points to construct a partitioning of the object surface using geodesics. Since it is based on the curve skeleton, our segmentation intrinsically reflects the shape symmetry and topology. By using geodesics we obtain segments that have smooth, minimally twisting borders. Finally, we present a hierarchical segmentation of shapes which reflects the hierarchical structure of the curve skeleton. We demonstrate the framework on various real-world 3D shapes.

1. Introduction

Shape segmentation is the task of decomposing a 3D shape into its meaningful components. In this context, meaningful components are those that a human being intuitively perceives as distinct, logical parts of the shape. Segmentations are useful in shape analysis, shape matching, medical imaging, collision detection, and other geometric processing methods employing divide-and-conquer strategies.

In defining what characterizes meaningful components, several directions have been pursued in the past. One of these is to use curve skeletons. The curve skeleton is a compact shape descriptor, much like a stick-figure representation. The curve skeleton of a 3D shape is a connected 1D structure that is centered within the shape, reflects its circular symmetries, and efficiently captures its topological and geometrical properties [3]. The structure of the curve skeleton, consisting of branches and junctions, reflects the hierarchy of the meaningful components. The branches are associated with meaningful components, whereas the junctions reflect the relationship between components.

In this paper we present a new framework for hierarchical shape segmentation using the curve skeleton. The curve skeleton is formally defined in terms of geodesics on the object’s surface [20]. For each curve-skeleton point these geodesics divide the boundary into a set of connected components, providing a natural skeleton-to-boundary mapping that is intrinsic to the definition of the curve skeleton. Being geodesics, segment borders are smooth and minimally twisting. After computing the meaningful components, we combine them into a hierarchical level-of-detail segmentation. Although we use the curve-skeleton junctions to provide the meaningful components, other points can be used as well, depending on the target application. Besides the segmentation framework, this paper contributes a robust algorithm for computing the curve skeleton of voxelized objects which enhances the method in [20], and a method to robustly detect junctions.

Figure 1 provides an overview of our approach, consisting of four stages. First, the curve skeleton is computed from the voxelized input shape. Second, the critical points are chosen either automatically as the junctions of the curve skeleton, or manually by the user. Third, we compute the component sets of the critical points. Finally, a level-of-detail segmentation is created from the component sets.

The outline of this paper is as follows. In the next section we review related work. In Section 3 we present some preliminary definitions. In Section 4 we provide the details of the curve-skeleton definition and computation. In Section 5 we elaborate on computation of the component sets, and how based on them junctions can be detected robustly. Section 6 deals with creating the hierarchical segmentation from the component sets. In Section 7 we present the results of the algorithm. Section 8 presents a discussion and we conclude in Section 9.
2. Related Work

Shapes can be segmented by considering their boundary or interior. Approaches that are based on the boundary usually define segment borders at object-surface concavities. Katz and Tal [10] use fuzzy clustering on geodesic and angular distances between surface elements to obtain a hierarchical mesh decomposition with non-jaggy borders. They show that their segmentation can be used to compute a (control) curve skeleton. In [9], pose-invariant segmentations are produced by extracting feature points and cores. In [2], a fuzzy clustering on quasi-flat surface features separated by curvature extrema is used to obtain a multiscale segmentation of point sets. However, the different scales do not form a strict hierarchy. In [13], an automatic scissoring scheme is proposed based on 3D snakes.

Segmentation methods using the curve skeleton consider the shape’s interior. Our approach falls into this category. To obtain meaningful components, these methods require a mapping from the curve skeleton to the object boundary. Various mappings have been proposed. In [14] for example, a combination of planar cross-sections and space sweeping is used. The approach in [4] uses force-following of boundary particles.

Our work is similar to that of Li et al. [14], in which the segmentation is also based on curve skeletons. However, there are some important differences. Our curve skeleton algorithm has a more fundamental underpinning (see [20] for details), is more noise resistant, and is connected by default. Li et al. use a planar cross-section sweeping along the curve skeleton between critical points to obtain components. The definition of our curve skeleton is such that it provides a natural skeleton-to-boundary mapping. Instead of taking the planar cross-section as borders between components, we use the actual curve-skeleton definition and take the shortest geodesics between feature points as borders. This makes for more natural borders between segments (compare e.g. the Hand object in Fig. 11 with [14, Fig.13]). Furthermore, our framework provides a hierarchical segmentation.

3. Definitions

The surface skeleton $S$ of a 3D object $\Omega$ with boundary $\partial\Omega$ is defined as those points in $\Omega$ having at least two boundary points at minimum distance:

$$S = \{ p \in \Omega | \exists a, b \in \partial\Omega, a \neq b,\]

$$dist(p, a) = dist(p, b) = \min_{k \in \partial\Omega} dist(p, k)\},$$

where $dist$ is the Euclidean distance in $\mathbb{R}^3$. Points $a, b$ are the points at minimum distance of $p$ and are called the feature points of $p$. Let $F: \Omega \rightarrow \mathcal{P}(\partial\Omega)$ be the feature transform which assigns to each object point its set of feature points, where $\mathcal{P}$ denotes the power set. The surface skeleton consists of 2D manifolds, called sheets, and of 1D curves [7]. The sheets intersect in sheet-intersection curves. Points on these intersection curves have more than three feature points.

The algorithms we present in this paper all work in binary voxel space [12]. When used in a discrete-space context, $\Omega$ denotes the set of foreground, or object voxels.

4. Curve skeleton computation

First, we define the curve skeleton $C$ for the generic case, namely when the curve skeleton is not (partly) incident
with a sheet-intersection curve, so that there are two feature points for a point \( p \in \mathcal{C} \). The curve skeleton is defined as those points having at least two shortest geodesics \( \gamma_i, \gamma_j \) between their two feature points \( F(p) \):

\[
p \in \mathcal{C} \iff \exists \text{ two shortest geodesics } \gamma_i \neq \gamma_j \text{ between } F(p)
\]

(2)

This definition was first presented in [6], and elaborated in [20]. It ensures that the curve skeleton is centered in two ways. First, the curve skeleton is included in the surface skeleton, as the surface skeleton is defined as those points having at least two feature points. Second, it is also centered within the surface skeleton structure, as the two shortest geodesics \( \gamma_i, \gamma_j \) necessarily have the same length. This definition of the curve skeleton yields a connected 1D structure, as shown in [6, 20]. Since we perform our computations in discrete voxel space, we compute the shortest geodesics as shortest paths in the boundary graph in which the surface voxels are the nodes and their neighborhood relations represent the edges.

In the non-generic case, the curve skeleton is partly incident with a sheet-intersection curve, and there can be more than two feature points for a point \( p \in \mathcal{C} \). For example, Figure 2a depicts the surface skeleton of a box with a vertical ridge. Point \( p \) lies on the sheet-intersection curve of the box’s sheet and the ridge’s sheet, and has three feature points. Point \( p \) clearly is a curve-skeleton point (Fig. 2b), but definition (2) does not detect this. Consequently, this definition cannot be used to detect all curve-skeleton points, so we modify it as follows. With each intersecting sheet a feature-point pair \( a, b \) is associated and thus a shortest geodesic \( \gamma_i \) between \( a, b \). For example, for three intersecting sheets there are three shortest geodesics. The combination of these shortest geodesics for a point \( p \in \Omega \) is called the \textit{shortest-geodesic set} \( \Gamma(p) \):

\[
\Gamma(p) = \{ \gamma_i \},
\]

(3)

where \( \gamma_i \) is a shortest geodesic between a feature-point pair \( a, b \in F(p) \) that is associated with a sheet. We observe that \( \Gamma(p \in \mathcal{C}) \) forms a Jordan curve on \( \partial \Omega \) in the generic case, \( |F(p)| = 2 \).

The discrete space equivalent of \( \Gamma \) is called the \textit{shortest-path set} and covers a set of boundary voxels. Testing whether this set represents a Jordan curve on \( \partial \Omega \) in the continuous case can be done by testing whether the genus of \( \Gamma \) is 1 with respect to \( \partial \Omega \) in discrete space. For objects of genus 0, this is equivalent to checking whether \( \Gamma \) divides \( \partial \Omega \) into two components. The new definition of the curve skeleton replaces (2) and becomes:

\[
p \in \mathcal{C} \iff \text{genus}(\Gamma(p)) \geq 1
\]

(4)

Furthermore, we observe that the genus of \( \Gamma \) can be used to differentiate between junction and non-junction points, called \textit{regular} points. A junction is a point on the curve skeleton where at least three branches come together. If the genus of \( \Gamma(p) \) is larger than 1, then \( p \) is a \( C \)-junction point:

\[
p \in \text{junctions}(\mathcal{C}) \iff \text{genus}(\Gamma(p)) \geq 2
\]

(5)

4.1. Algorithm

Based on the above definitions, we now present our algorithm for computing the curve skeleton of a voxelized object \( \Omega \). The algorithm consists of several steps that can be executed in parallel for each object voxel. The different stages are depicted in Figure 3. In the first step, we compute the feature transform \( F \) and extended feature transform \( \overline{F} \) of \( \Omega \). The \textit{extended feature transform} merges the feature set of each voxel \( p \) with that of its 26-neighbors in the first octant: \( \overline{F} = \bigcup_{x,y,z \in \{0,1\}} F(p + x, p_y + y, p_z + z) \). The result is that a regular curve-skeleton point has two (compact) groups of feature voxels (Fig. 3a), one on each side of the surface skeleton. The reason for computing \( \overline{F} \) is to solve the well-known problem that in discrete space the feature set of a skeleton voxel might contain only one feature voxel, since there might be no voxel exactly in the center of the voxelized boundary [5, 16]. After computing \( \overline{F} \), we compute the shortest paths \( \Gamma(p) \) as 3D chain codes [11] in the boundary graph between all feature voxels \( \overline{F}(p) \). For this we use the A* algorithm [8], a modification of Dijkstra’s algorithm, with Euclidean distance as the search heuristic. Now the second reason to use \( \overline{F} \) becomes apparent. Because \( \overline{F} \) contains groups of voxels, we will find numerous shortest paths for a point with only two feature points. Together they form a discrete (noisy) Jordan curve around the object, even though we use a simple shortest-path algorithm that finds only one shortest path between two given voxels.

**Figure 4. The boundaries of two dilated paths.**

After computing the path set \( \Gamma(p) \), we determine its genus on \( \partial \Omega \), as follows. First we dilate \( \Gamma(p) \) so that we obtain a surface band \( \Gamma'(p) \) centered at the 1D structure \( \Gamma(p) \) (Fig. 3c). The dilation also eliminates small holes from the path set that are caused by the discrete nature of the shortest paths (Fig. 3b). The dilation is performed by propagating
the voxels in $\Gamma(p)$ a short distance outward using a distance-ordered flood fill. Next, we determine the number of compact boundary pieces that the area $\Gamma'(p)$ has. If two or more boundaries are found, $p$ is concluded to be a curve-skeleton voxel (Fig. 3d). When only one boundary is found, the voxel is concluded to be a non-curve skeleton voxel (Figure 4a). Empirical studies on an extensive family of real-world 3D shapes show that a (geodesic) dilation distance of 5.0 is enough to obtain two disjoint boundaries if $\Gamma(p)$ is the discrete representation of a Jordan curve. In principle, when three or more boundaries for $\Gamma'(p)$ are found, $p$ is a junction. In Figure 4b for example, three boundaries are found for the junction voxel. However, due to the discrete nature of the shortest paths, curve-skeleton voxels might be incorrectly classified as junctions, i.e., junction detection by analyzing the genus of $\Gamma$ is conservative. This is problematic if we want to use the junctions in segmentation (Sec. 6). We address this problem in Section 5.4.

5. Computing Component Sets

In the previous section we described the detection of a curve skeleton point $p$ by analyzing the genus of its associated geodesic set $\Gamma(p)$. In this definition of the curve skeleton lies a natural skeleton-to-boundary mapping. We observe that $\Gamma(p)$ for a point $p \in C$ divides the object surface $\partial \Omega$ into a set of connected components. The component set of a point $p$ is denoted $C(p)$. Figure 6 shows the component sets for two selected curve-skeleton points $p, q$, where $p$ and $q$ are a junction and regular point respectively. The component sets $|C|$ of the junctions will be used in the final segmentation (Sec. 6).

In our voxel space representation, the component set $C(p)$ for a voxel $p$ is computed by labeling the connected components in the boundary graph from which the voxels from the dilated path-set $\Gamma'(p)$ are removed. The flood fills involved are sped up using a simple spatial-subdivision scheme. The voxels in the dilated path-set have not been assigned a component in this manner. Therefore, we erode $\Gamma'(p)$ and assign its pixels to the nearest component.

5.1. Topological Properties

It is important to note that the number of components $|C(p)|$ for a point $p \in C$ is related to whether the shape $\Omega$
has tunnels, as this will affect the final segmentation. For shapes without tunnels or holes (genus 0), the curve skeleton has a tree structure. For objects with tunnels (genus \( \geq 1 \)), the curve skeleton is a graph that contains a loop around each tunnel. Since there is a skeleton-to-boundary mapping through the use of \( \gamma \), whether \( \Gamma(p) \) divides the boundary into multiple components is equivalent to whether \( p \) divides the graph into multiple components. In graph theory, a point \( p \) is called a cut vertex dividing the graph into multiple connected components is equivalent to whether \( \Gamma(p) \) divides the boundary into multiple components.

\[ |C(p)| = \max \{ \text{genus}(\Gamma(p)) + 1 - L(p), 1 \}. \quad (6) \]

Figure 7 shows an example shape with two tunnels and four selected points. Points \( p, q \) are regular (non-junction) points (genus \( \Gamma(p, q) = 1 \)), whereas \( r, s \) are junctions of three branches (genus \( \Gamma(r, s) = 2 \)). Point \( p \) is not on a loop, points \( q, r \) are both on one loop, and \( s \) is on two loops. Indeed, we verify that the cardinality of the component sets \( |C| \) generated by \( p, q, r, s \) is \( 2, 1, 2, 1 \) respectively. Points \( p, r \) are cut vertices in the graph \( C \), whereas \( q, s \) are not.

\[ \rho(p) = \frac{1}{\max \rho} \]

We normalize \( \rho \) by its maximum value over the entire curve skeleton. The maximum value is reached for the point where \( |C_k| \) is smallest. This point is called the root of \( \rho \). In case the root \( r \) is regular point, \( \rho(r) \) is half the surface area: \( \rho(r) = \frac{1}{2} |\partial \Omega| \). Note that this importance measure can only be computed for non-loop points. A regular loop-point has only one component and no foreground components (e.g. Fig. 7, point \( q \)), so that \( \rho \) is undefined. Simplified curve-skeletons can be computed from a noisy curve-skeleton \( C \) by discarding all points with an importance below a certain threshold \( \tau \):

\[ C_{\tau} = \{ p \in C \mid \rho(p) \geq \tau \} \quad (9) \]
The threshold \( \tau \) is a user-parameter which controls what is considered noise. The meaning of this parameter is intuitive: all \( C \)-points representing a smaller surface area than \( \tau \) are discarded. Furthermore, \( C_\tau \) can be considered a multiscale representation of the curve skeleton. The simplified skeletons are connected by default at all scales, because \( \rho \) is monotonic, which follows from (7). The simplified curve-skeletons \( C_\tau \) of several shapes are shown in Figure 11, with \( \tau = 10^{-3} \). Thus, curve-skeleton points representing an area less than 0.1% of the object surface are discarded.

This definition of \( \rho \) has been presented in [20]. There, the curve skeleton detection was integrated with the computation of \( \rho \). In contrast, the skeletonization method presented here computes a curve skeleton by analyzing the genus of \( \Gamma \), making it much faster to compute, and making it possible to handle objects with tunnels. Furthermore, in this paper we also present a method to robustly detect junctions, as detailed in the next section.

5.4. Robust Junction Detection

As indicated in Section 4.1, the genus of \( \Gamma'(p) \) is a conservative criterion for detecting junctions, which is problematic if we want to use them for segmentation. Inspired by the importance measure from the previous section, we discard junctions which have small components among their foreground components. These junctions are likely the result of noise on the object boundary, or due to discretization artifacts. A point \( p \) is a junction, \( p \in J_\tau \), at a certain scale \( \tau \) if and only if it has at least two foreground components \( F_\tau \) that are larger than \( \tau \):

\[
J_\tau = \{ p \in J_0 \mid |F_\tau(p)| \geq 2 \} \tag{10}
\]

\[
F_\tau(p) = \{ C_i(p) \mid 1 \leq i < k \land |C_i(p)| \geq \tau \} , \tag{11}
\]

where \( J_0 \) is the set of conservatively detected junctions from Section 4.1. Although related to the computation of \( \rho \), we do not have to compute \( \rho \) to compute \( J_\tau \): We only have to compute the component sets for the conservative junctions \( J_0 \). Equivalent to the computation of \( C_\tau \), the scale \( \tau \) is used to distinguish between small-scale noise and signal. For all objects we tested, a value of \( \tau = 10^{-3} \) gives good results. Like the importance measure, a disadvantage of this method is that junctions on \( C \)-loops are discarded, as these junctions have only one foreground component (Sec. 5.1).

In Figure 8a, the conservative junctions \( J_0 \) on \( C \) are shown for an object with noise on its boundary, computed by analyzing the genus of \( \Gamma \). Figure 8b shows the robust subset selected by applying (10).

6. Segmentation

We now present two methods for computing a shape segmentation from the critical points. By segmentation we mean the final segmentation \( S \) of the object into disjoint and connected segments \( S_i \). In order to obtain a segmentation into meaningful components, the junctions of the curve skeleton are used as critical points. Alternatively, we can pick critical points by some other (semi) automatic process.

The first segmentation method (Sec. 6.1) we present is the most straightforward. It is based on the geodesic sets \( \Gamma \) of \( P \), and produces a flat (non-hierarchical) segmentation at the finest scale. The second method (Sec. 6.2) is based on the component sets of \( P \), and produces a hierarchical, level-of-detail segmentation.

6.1. Flat Segmentation

We produce a simple, flat segmentation by considering the shortest-path sets \( \Gamma \) of all critical points \( P \) at once. The connected components in the boundary graph due to all these shortest paths are labeled by using simple flood fills. A segmentation that is obtained in this way is shown in Figure 10b, using the shortest-path sets of the critical points shown in Fig. 10a. One property of this segmentation is that it splits tunnel-parts of the object into multiple segments.

6.2. Hierarchical Segmentation

In order to produce a hierarchical, level-of-detail segmentation, we consider the component sets \( C \) of the critical points \( P \). In Section 5.3 we distinguished between foreground and background components. In a component set \( C \) consisting of \( k \) components \( C_{1..k} \), the largest component \( C_k \) is called the background component, the remaining ones are called foreground components. The foreground components are those that one would consider meaningful and intuitively associate with the critical point. The background component is merely the remaining surface area. In Figure 6 for example, the background components are light...
1: $F \leftarrow \{C_i(p) | p \in P, 1 \leq i \leq k \} \cup \{C^*(p) | p \in P\}$
2: $F \leftarrow F \cup \{\emptyset\}$
3: procedure computeS($\tau$)
4: $S \leftarrow \emptyset$
5: for each $f \in F$ in ascending order of $|f|$ do
6: if $|f| \geq \tau$ then
7: $s \leftarrow f \setminus S$
8: if $f \cap S \neq \emptyset \Rightarrow |s| > 0.1 \times |f \cap S|$ then
9: $S \leftarrow S \cup \{s\}$
10: end if
11: end if
12: end for
13: end procedure
14: procedure computeLevelsOfDetail($\tau$)
15: for each $f_i, f_{i+1}$ in $F$ in ascending order of $|f|$ do
16: if $|f_{i+1}| - |f_i| > 0.1 \times |f_i|$ then
17: computeS($f_i$)
18: end if
19: end for
20: end procedure

Figure 9. Algorithm for computing a level-of-detail segmentation.

The idea is that we consider all components $f \in F$ in ascending order of their area (line 5), but only those components that are larger than the specified scale $\tau$ (line 6). The potential segment $s$ is computed as the set difference between $f$ and the existing segments in $S$: $s = f \setminus S$ (line 7). Before adding the potential segment $s$ to $S$, we check whether $f$ overlaps any existing segments from $S$. If not, $s$ is added without restriction. If $f$ overlaps, we only add $s$ if it contributes enough to the segmentation, that is, if it adds at least 10% of the area that it overlaps (line 8). This is to prevent tiny segments due to different junctions having similar components among their component sets. This occurs for example due to the fact that junctions computed by the algorithm in Section 4 may consist of multiple voxels, having almost the same component sets. After processing all components in $F$, because the background components have been left out in the segmentation, the object surface might not be fully covered. Therefore, we add to $F$ the whole boundary as the largest component (line 2), which ends up as a segment filling up the remaining part.

In order to compute multiple segmentations from fine to coarse (line 14), we can simply consider all components from $F$ in ascending order of area, and produce a segmentation for each area. To limit the amount of generated hierarchy levels, in our implementation we only compute a hierarchy if two consecutive areas differ by at least 10% from the smaller area $f_i$ (line 16). The different segmentations produced at the various scales actually form a hierarchy, because every segment is included in a segment from a coarser scale (7). Besides being hierarchical, another difference between this segmentation method and the flat one is that tunnel-parts are not segmented and are left intact (see Fig. 10c).

Figure 10. Critical points $P$ and their path sets (a). Flat segmentation (b). Finest level of the hierarchical segmentation (c).

7. Results

We have implemented our framework in C++ and have run it on a Pentium 4, 3GHz with 1GB of RAM. In computing the shortest paths we used a cache to prevent computing the same shortest path twice. As input we used several polygonal meshes from [1, 17], voxelized using binvox [15] for various resolutions. We used both organic and geometric objects. Figure 11 shows for four objects the simplified curve-skeleton $C_\tau$ and robust junctions $J_\tau$, both with $\tau = 10^{-3}$, and three selected levels from the hierarchical segmentation. Figure 12 shows for several objects the segmentation at the finest scale. We observe that our method is able to extract fine details, such as the toes and fingers of the Armadillo and Dinopet objects, and it does not suffer from over-segmentation. The use of geodesics as borders of the surface segments has the very beneficial effect of producing sharp, non-jagged separations, since geodesics are minimally twisting curves on the surface.

Table 1 shows performance measurements on our framework for several objects that are shown in this paper. Columns “object”, “dim”, “$|\Omega|$”, “$|\partial \Omega|$” denote object name, voxel-grid dimensions, number of object voxels, and number of boundary voxels respectively. Column “$C$ time” denotes the time in seconds to compute the non-simplified curve-skeleton $C$, including preprocessing (Sec. 4.1). Column “seg time” denotes the time for computing all levels in the hierarchical segmentation. This time strongly depends on the amount of levels generated. Column “$\rho$ time” denotes the time required to compute $\rho$ on $C$ (Sec. 5.3), that is, to obtain simplified curve-skeletons. Note that computing $\rho$
Figure 11. Simplified curve-skeletons $C_{\tau}$, and three levels of the hierarchical segmentation.

Figure 12. Segmentations at the finest scale.
is not needed for obtaining a segmentation. Column “mem”
gives an indication of the peak memory usage. Comput-
ing the curve skeleton takes the most time, which can be
attributed to the computation of the shortest paths and di-
lations. It takes relatively long for the Plane, which can be
explained by the fact that the Plane has a high surface area
to volume ratio, meaning that the average shortest-path be-
tween feature voxels is relatively long. The time needed to
compute the robust junctions $J_{\tau}$ is not shown as it is negli-
gible: up to 5 seconds for the considered objects.

8. Discussion

Following from the curve-skeleton definition, the bor-
ders of the segments are piecewise geodesic. They do not
necessarily follow local surface-concavities, but instead find
the minimal-length path between feature points. Hence, the
borders exhibit minimal twist on the surface, look natural,
and are robust to noise, as shown by the Noisydino object
(Fig. 12). For segmentation methods based on surface cur-
vature, this noise could be problematic [2].

Figure 13. Pose-invariance

Segment borders are not based on boundary features but
on the curve skeleton, capturing global geometrical (e.g.
symmetrical) and topological properties. This has a number
of advantages. The segmentations respect the object’s cir-
cular symmetry and are invariant for different poses of the
same object. This is because the curve-skeleton structure
in general remains stable under deformations of the object.
Figure 13 shows the segmentations for two poses of the Cat
(poses from [18]).

Borders are not necessarily found at curvature extrema.
In the H-shape object for example, borders are found on
flat parts of the surface, segmenting the tips of the H-shape.
These segments are ascribed to the fact that the tips have
convex corners, which generate branches and junctions in
the curve skeleton. A consequence of choosing only the
junctions as critical points is that we do not find segment
borders for curvature extrema in tubular-like parts of the
object. In Figure 12 for example, the Dinopet object has
no segment borders on its knees. In order to consider more
the object’s geometry in addition to its topology, we could
choose extra critical points on the curve skeleton at curva-
ture extrema, computed either on the curve skeleton or orig-
inal surface.

Figure 14. Manual segmentation of the Octo-
pus.

As mentioned earlier, our framework supports manual
selection of critical points. In Figure 14a, we manually se-
lected three such points on each tentacle of the Octopus.
The resulting segmentation containing three segments per
tentacle is show in Fig. 14b.

9. Conclusion

We have presented a novel framework for hierarchical
segmentation of 3D shapes. Being based on the formally
defined curve skeleton, our framework has a solid underpin-
ing. The produced segmentations inherit several desirable
properties, such as pose-invariance, from the curve skele-
ton and reflect the symmetry of the object. The definition
of the curve skeleton in terms of shortest geodesics gives
rise to a natural skeleton-to-boundary mapping. The mean-
ningful components are found using the curve-skeleton junc-
tions and are combined into a hierarchical, level-of-detail
segmentation. Being piecewise geodesic, the segment bor-
ders are smooth and non-twisting. Interestingly, because
our method is based on the curve skeleton representing the
object’s interior, our method produces both a surface seg-
mentation and a corresponding volumetric segmentation.
The framework supports segmentations based on the critical
points defined as the curve-skeleton junctions, but also de-
finied in other ways. A voxel-based implementation is pro-
vided. Besides the segmentations, it computes multiscale
curve-skeletons that are robust to boundary noise, and per-
forms robust junction detection. We showed that our frame-
work delivers good results on both organic and geometric
objects.

Several extensions can be made to this work. First, seg-
mentations would benefit from automatic detection of cur-
vature extrema of the curve skeleton in addition to the junc-
tions. Second, it would be interesting to see whether alter-
native curve-skeletons definitions, for example using differ-
ent distance metrics give useful segmentations.
Table 1. Table with measurements. See the text for details.

| object | dim | $|\Omega|$ | $\partial \Omega$ | $C$ time (s) | seg time (s) | $\rho$ time (s) | mem (MB) |
|--------|-----|--------|----------|-------------|-------------|-------------|--------|
| Armadillo | 188x245x207 | 905k | 80k | 42.9 | 41.0 | 7.50 | 447 |
| Bird | 197x99x366 | 289k | 54k | 60.3 | 4.98 | 6.86 | 334 |
| Dinoget | 334x366x180 | 1,810k | 136k | 98.3 | 54.2 | 40.0 | 707 |
| Hand | 360x154x257 | 1,300k | 94k | 91.8 | 20.5 | 5.57 | 540 |
| Horse | 360x316x171 | 1,860k | 154k | 105 | 33.7 | 10.2 | 660 |
| Noisydino | 125x346x365 | 1,421k | 114k | 55.2 | 19.4 | 14.0 | 552 |
| Plane | 217x304x98 | 545k | 110k | 228 | 20.5 | 38.5 | 492 |
| Octopus | 366x259x335 | 1,860k | 154k | 28.5 | 9.47 | 10.5 | 600 |

Acknowledgement

This work was supported by the Netherlands Organisation for Scientific Research (NWO) under grant number 612.065.414.

References

[1] AIM@SHAPE repository http://shapes.aim-at-shape.net.


