Vis Comput (2010) 26: 1283–1300
DOI 10.1007/s00371-010-0520-4

ORIGINAL ARTICLE

SkelTre

Robust skeleton extraction from imperfect point clouds

Alexander Bucksch · Roderik Lindenbergh · Massimo Menenti

Published online: 13 August 2010

Abstract Terrestrial laser scanners capture 3D geometry of real world objects as a point cloud. This paper reports on a new algorithm developed for the skeletonization of a laser scanner point cloud. The skeletonization algorithm proposed in this paper consists of three steps: (i) extraction of a graph from an octree organization, (ii) reduction of the graph to a skeleton, and (iii) embedding of the skeleton into the point cloud. For these three steps, only one input parameter is required. The results are validated on laser scanner point clouds representing 2 classes of objects; first on botanic trees as a special application and secondly on popular arbitrary objects. The presented skeleton found its first application in obtaining botanic tree parameters like length and diameter of branches and is presented here in a new, generalized version. Its definition as Reeb Graph, proofs the usefulness of the skeleton for applications like shape analysis. In this paper we show that the resulting skeleton contains the Reeb Graph and investigate the practically relevant parameters: centeredness and topological correctness. The robustness of this skeletonization method against undersampling, varying point density and systematic errors of the point cloud is demonstrated on real data examples.

Keywords Skeletonization · Point cloud · Laser scanning

A. Bucksch (✉) · R. Lindenbergh · M. Menenti
Delft University of Technology, Kluyverweg 1, 2629 HS, Delft, The Netherlands
e-mail: a.k.bucksch@tudelft.nl

R. Lindenbergh
e-mail: r.c.lindenbergh@tudelft.nl

M. Menenti
e-mail: m.menenti@tudelft.nl

1 Introduction

In recent years, instruments capable to measure thousands of distances per second from the instrument to surrounding surfaces became available [22]. One such instrument is a terrestrial laser scanner. These scanners are used to systematically sample the surface of objects by determining the distance to their surroundings. The surrounding is represented as a function of two spherical angles and a distance. The resulting data is called a point cloud. The modeling of botanic tree structures from laser scanned point clouds is a growing topic in both Computer Graphics, e.g. [25, 26] and Remote Sensing, e.g. [13] and [9]. Both fields have in common that a skeleton is used to represent the tree with the goal to extract surface information. Extraction of complex botanic tree structures from a point cloud is difficult for several reasons.

1. Varying point density caused by the spherical scan geometry of the instrument in a single scan.
2. Varying point density caused by the alignment of single scans into a common coordinate system.
3. Undersampling caused by occlusion effects.
4. Noise and systematic errors masking the object structure.

A skeleton is a one-dimensional description of the object structure. Skeletons are represented as curves, collections of ordered points or graphs. Their extraction from a point cloud faces several algorithmical challenges to achieve three main properties of a skeleton. First, topology preservation of the object is essential for navigating to a certain position within the object. Secondly, proper centering of the skeleton within the point cloud enables the extraction of correct surface information. The third property addresses computational efficiency. A point cloud of a small orchard tree, Fig. 1a, already easily consists of 300,000 points. These three properties can be achieved by considering that a point cloud is subject to noise, undersampling, and varying point density.

Table 1 Algorithm classes

Algorithm class	Descriptor dimension	Spatial data structure	Complexity
Morphology	2D	Raster with defined boundary	$O(nw)$,[1,2] [13]
Distance transform	2D	Raster with defined boundary	$O(n)$,[1] [20]
Voronoi diagram	2D	Defined boundary	$O(n^2)$,[1,3] [1]
Clustering	1D	Neighboring structure, e.g. kd-tree or minimum spanning tree	$O(kn^2\Delta^2)$,[1,4,5] [26]
Level set extraction	1D	Raster with defined boundary	$O(n)$,[1,6] [8]
Graph reduction	1D	Octree graph	$O(n)$,[1] [4]

[1] n denotes the number of input cells, point cloud points or vertices

[2] w denotes the size of a structuring element

[3] Worst case scenario, often almost linear in practice, [1]

[4] Quaranteed scenario for k-means clustering [2]

[5] Δ is the spreading of the input points

[6] Referring to the height function, may be higher for other functions

This paper reports on the details of a method for (Skel)etonisation of (Tre)es, here called SkelTre Skeleton, directly from a 3D point cloud by considering all six principal Cartesian directions. A preliminary version has been described elsewhere [5]. The algorithm incorporates three main elements. First, an octree is built from which an octree graph is extracted, representing the connectivity between the octree cells with respect to the point cloud. In a second step, the octree graph is exploited to retract the point cloud to a skeleton. The third element is a strategy to embed the skeleton graph into the point cloud. Topology preservation is enhanced by using a new noise robust criterion to decide on proper connections in the octree graph instead of simple thresholding. It is shown that the skeleton contains a well-known topological structure, the Reeb Graph, which is widely used in computer graphics and shape modeling. The centering is improved by a new embedding strategy taking the approximate shape of the original object into account. This new embedding is less sensitive to varying point density and undersampling. The basis of this strategy is a novel graph reduction method that automatically incorporates the approximate local surface elongation by using suitable vertex labels. These vertex labels form the basis for an increased efficiency, because the new octree graph reduction rules behave linear in time.

2 Related work

In this section, an overview of skeletonization algorithms for point cloud data is given. Notably, their use on point clouds representing botanical trees is highlighted. An overview is of algorithm types is given in Table 1. First algorithms are distinguished based on the dimension of the output descriptor. Two classes are shown in Table 1, first algorithms aiming at the extraction of a 2D descriptor which is reduced further to 1D and second algorithms producing a one-dimensional skeleton directly. Further distinction is achieved by identifying the underlying data structure and computational complexity of the algorithms, which is an important factor when using large data sets.

2.1 2D descriptors

The best known 2D descriptor of an object is the medial axis which is the set of points having more than one closest point on the object boundary [3]. Several frameworks exist to formulate the medial axis extraction. The medial axis may be derived as a subset of the Voronoi diagram of the point cloud, or from a morphological thinning process. One major property of the medial axis of a 3D object is that it in general consists of a set of surfaces. For the reduction of the 2D medial axis to a 1D skeleton, a second processing step is needed. For example, [10] have shown that an approximate medial axis is reducible to a meaningful skeleton. Medial axis approaches commonly need a defined inside and outside of the object. Undersampling and occlusions make this difficult to define on laser scanning data representing a hull.

Morphological thinning methods organize the point cloud in a 3D raster of equally sized cubic cells. From this raster, the outer layer is removed until the skeleton remains. Removing the outer layer makes use of the morphological operations opening and erosion [21]. This class of algorithms requires a defined inner volume of the object to produce a centered skeleton. Depending on the kernel used for the thinning process also the medial axis is derivable. Palagyi et al. [16] introduced a time linear algorithm using 6 sub-iterations. Its application on tree point clouds was proposed by Gorte and Pfeifer [13] and later extended [12]. As stated by the authors, the skeleton does not preserve the topology [13]. Varying point density is only treated by the number of points per cell. Undersampling and occlusions are subject to the cell size. The efficiency of morphological thinning is characterized by a complexity of $O(nw)$, n being the number of raster cells and w being the number of cells used as structuring elements.

In practice, *Distance Transform* based methods often start from a point cloud embedded in a 3D raster of equal sized cubic cells as well [27]. All raster cells are consecutively marked by their distance to the object boundary. The set of cells, where maximal distances occur, form the skeleton. These methods extract the medial axis, and face the same post processing problems as morphological thinning approaches. Furthermore, connectedness of the skeleton is not guaranteed [8]. It's computational complexity is given as $O(n)$, with n being the number of raster cells. An application of the distance transform on botanical tree data was not found.

Voronoi Diagram based approaches also derive an approximation of the medial axis from the point cloud, e.g. [1]. The medial axis is extracted by investigating the poles of the Voronoi diagram of a point cloud, for example [1], distinguished between inner and outer poles of a suitable weighted Voronoi diagram. The set of inner poles containing facets in 3D approximates the medial axes. To our best knowledge, no specific application to botanic trees is known. Only a simple example on a synthetic tree point cloud can be found in [8]. As stated in [1], this method requires a sufficiently dense sampled object as input, as the object has to be assumed watertight. This condition may be difficult to achieve with laser scanned data on trees containing many occlusion effects and undersampling. The construction of the Voronoi diagram determines the efficiency of the algorithm. The complexity is in the worst case $O(n^2)$ and $O(n \log n)$ on average, with n being the number of point cloud points.

2.2 1D descriptors

One-dimensional descriptors have in common that they use neighborhood information to extract the skeleton as a graph. This graph is embedded with an embedding strategy into the point cloud.

Clustering methods produce clusters of point cloud points from a suitable spanning graph, like the minimum spanning tree, to represent a point neighborhood. Some distance metric is used to produce the clusters. Neighboring clusters are connected to a skeleton. In [25], a neighboring graph is used and points with the same quantized distance from the root are considered as belonging to one cluster. The approach shows good results until two-third of the tree height on a test tree carrying leafs. The remaining skeleton is produced by using species dependent allometries. Another promising clustering approach was presented by Yan et al. [26]. They used a kd-tree and k-means clustering to produce the clusters from which the skeleton is derived. It was shown in [26] that embedding of the skeleton is still an issue for these methods. A further drawback is that they need the complete point cloud as an input to perform the skeletonization. The guaranteed termination of k-means clustering algorithms is given by Arthur and Vassilvitskii [2] as $O(kn^2\Delta^2)$, with Δ being the spreading of the data. Nevertheless, it was also shown in [2] that, dependent on the data, better complexities are achievable.

One-dimensional descriptors with proved topological properties, like the *Reeb Graph* [19], were used first in [23]. Reeb Graphs rely on Morse theory describing the extraction of critical points. A discrete formulation of Morse theory came from [11]. Two frameworks exist to extract a skeleton based on topological properties: extracting the level sets based on a Morse function and graph reduction, which is used in this paper.

Level set extraction methods use a function that is defined on the sampled surface. The height function is often used to extract the level sets from a given point cloud, e.g. [24]. Placement of vertices at every centroid of each extracted level set produces a skeleton by connecting the vertices with respect to the chosen function. Every branching point of the resulting skeleton is a saddle point and every one connected vertex is a minimum or maximum. The graph containing only the saddle points and the minima and maxima of curvature is a Reeb Graph [7]. The biggest problems arising with these approaches are the rotational dependency of the height function and the sensitivity of the level set extraction to the sampling density, [8]. The approach of [24] was applied by Côtè et al. [9] on trees. A virtual tree model based on the allometries of the known tree species was used to estimate the finer branches. The virtual tree model is required because of undersampling at the finer branches.

Graph reduction based approaches, [4], extract an initial graph from a spatial subdivision. This initial graph is reduced by a set of rules to a skeleton. These rules consider the connectivity between different parts of the point cloud. Several advantages of such a approach could be demonstrated: a high robustness to noise on imperfect data, a good centeredness and a good connectivity. Centeredness is achieved by embedding the graph into the point cloud. Topological correctness is achieved by choosing a proper decision criterion to place connections between the different point cloud parts and the careful design of the reduction rules. Another benefit is that no preprocessing is necessary to define the inside and outside of the given object. Problems arise with the method of [4], because varying point density and noise are only treated by the cell size. This fact leads to faults in the embedding. In case of large undersampling, erroneous loops may appear at higher resolutions. As a consequence, the topology of the object is locally not represented anymore. The overall complexity was given as $O(n)$.

3 SkelTre skeletonization algorithm

This paper introduces a linear-time algorithm for computing a skeleton from an unorganized 3D point cloud sampling. Unorganized point clouds contain no neighboring information on the points; see Fig. 1a. From here, the point cloud is divided into subsets by an octree subdivision. The centroids of the point cloud points within an octree cell, are the vertices of the octree graph; see Fig. 1b. Two 6-adjacent vertices are connected by an edge if the point cloud parts in the two corresponding octree cells fulfill a suitable connectivity criterion. These edges in the octree graph are labeled by a direction label to indicate the local object elongation. The goal of this section is to show that the retraction of the octree graph using so-called E-Pairs and V-Pairs preserves the topology and embeds it into a well-known topological data structure, the so-called Reeb Graph [19]. Moreover, an explanation of the octree generation and the octree graph extraction and labeling technique is given, followed by the actual formulation of the complete SkelTre Skeleton algorithm.

3.1 Octree generation

An octree is a hierarchical subdivision of a starting cube into 8 equally sized subcubes, so-called octree cells. These subcubes are subdivided further until the subdivision is terminated.

Let the surface of an object be represented by a point cloud Σ. A spatial subdivision of the point cloud Σ into subsets Σ_i is obtained by an octree. The octree subdivides the space occupied by the point cloud into cubic cells containing point cloud points. These cubic cells have equal size.

Definition 1 The octree space is modeled as a cubical region consisting of $2^n \times 2^n \times 2^n$ unit cubes, where n is the subdivision depth. Each unit cube has value 0 or 1, depending on whether it contains data points or not, adapted from [6].

Note that this definition of the octree assumes equal octree space length. In fact, the algorithm is capable to handle adaptive octrees, where the length is locally adapted to the point cloud. Nevertheless, we focus within this paper on the graph reduction principle and do not use adaptive octrees throughout this paper.

Ideally, our intermediate result of the octree generation is a subdivision which separates all parts of the object that are also spatially separated. Clearly, the separating power of the subdivision depends on the minimum resolution of the octree.

3.2 Extraction and labeling of the octree-graph

We are aiming on a graph-reduction method. Because of that an initial graph, so-called octree-graph, is generated. This octree-graph is later reduced to the SkeTre skeleton. An octree-graph is the face dual of the octree, whose vertices correspond to octree cells. The vertices of the octree-graph are simply placed at the center of gravity of all points belonging to a cell and connected by an edge if two octree cells have adjacent faces.

Definition 2 Let $OC_i, i = 1, \dots, n$ be a collection of octree cells. And let CS_{jk} be the shared sides of an octree. The octree-graph $OG(V, E)$ contains the vertices V dual to OC_i connected via the edges E dual to CS_{jk}.

The benefit of using this dual is, that it exhibits local grid graph properties, which is illustrated in Fig. 2 for the 3D case.

Definition 3 A three-dimensional grid graph is an $m \times n \times r$ graph that is the graph Cartesian product of path graphs on m, n and r vertices [18] and denoted as $G(m, n, r)$.

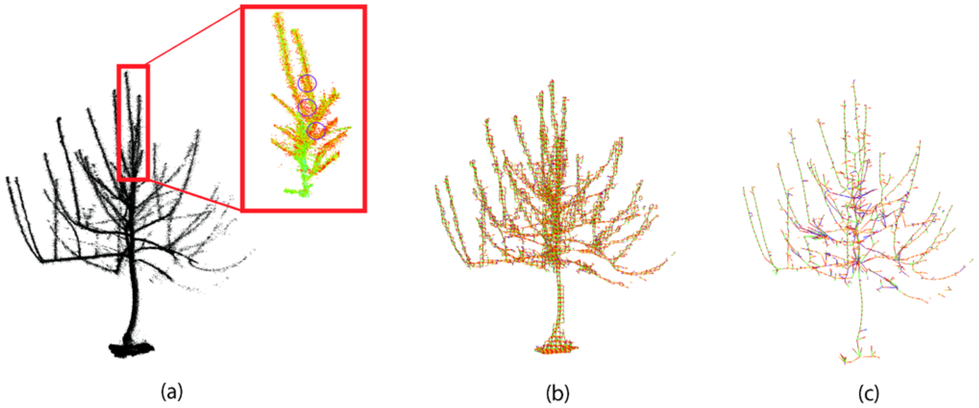


Fig. 1 (**a**) Botanic tree point cloud with a zoom into the inner crown of a leafless orchard tree. The zoom is colored by intensity. *The three marked areas* show examples of noise where the separation of branches is even hard by visual inspection. (**b**) Example of an extracted octree graph, which is reduced to the skeleton in (**c**)

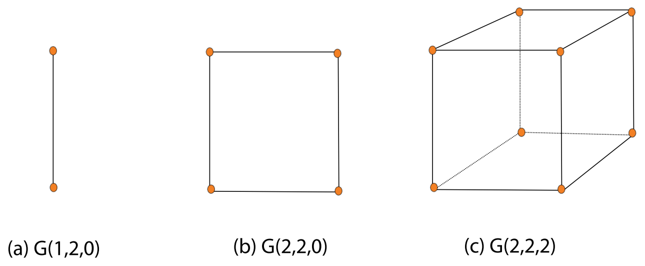


Fig. 2 (**a**) a $G(1,2,0)$ subgraph (**b**) a $G(2,2,0)$ subgraph (**c**) a $G(2,2,2)$ subgraph

Figure 2 shows the grid graph configurations relevant in the context of the paper. Let $m=1, n=2$ and $r=0$, then $G(1,2,0)$ corresponds to two vertices connected by one edge (Fig. 2a), forming a line segment. Let $m=2, n=2$ and $r=0$, then $G(2,2,0)$ corresponds to 4 vertices connected by 4 edges (Fig. 2b), forming a squared structure. Let $m=2, n=2$ and $r=2$, then $G(2,2,2)$ corresponds to 6 vertices connected by 12 edges (Fig. 2c), forming a cubic structure.

Ideally, the octree-graph represents the local directions of the object surface and connects only object parts. As stated above, terrestrial laser scan data is subject to noise, undersampling and varying point density. These data driven problems can result in both overconnecting and underconnecting. Overconnecting occurs when additional erroneous connections are created due to noise and outliers, underconnecting occurs because of undersampling due to occlusions. The criterion to handle undersampling and noise is a decision criterion to place connections between neighboring octree cells. Note that the extraction is based on the intersection direction of the octree cell. Therefore, this extraction overcomes the known rotational dependency problems of octrees [4].

3.2.1 *Robustness criterion*

The formulation of a robustness criterion is motivated by two aspects. Firstly, to address the problem of noise, which is covering and maybe hiding the underlying object topology. And secondly, it provides the possibility to address instrument specific error models. This robust criterion whether to connect two octree-graph vertices, corresponding to two adjacent octree cells, by an edge, is based on the distances of the cell points to three suitable planes (Fig. 3). Let C_1 and C_2 be the centroids of the point cloud points in two adjacent octree cells Ω_1 and Ω_2. Let C_{12} be the midpoint of the line segment $\overline{C_1C_2}$. The three suitable planes P_1, P_2, and P_{12}, are the planes through the points C_1, C_2, and C_{12}, perpendicular to the line through C_1 and C_2. Let d_1, d_2, and d_{12} be the median values of the squared distances of the points in Ω_1, Ω_2 and $\Omega_1 \cup \Omega_2$ to the planes P_1, P_2 and P_{12}. Under ideal conditions, the $\sqrt{d_{1,2}}$ of two connected cells would be at least $\frac{1}{4}$ of the distance between C_1 and C_2 to

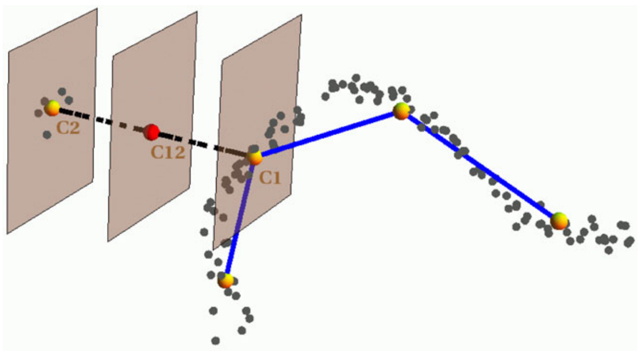


Fig. 3 Robustness criterion to connect octree-graph vertices C_1 and C_2 by an edge

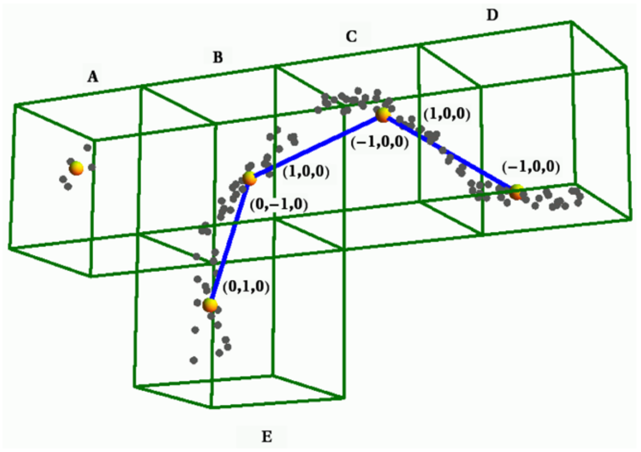


Fig. 4 Connecting and labeling an octree-graph: Five complete octree cells, containing some *black* data points. The vertices of the octree-graph corresponding to the octree cells are shown in *orange*. They are positioned in the center of gravity of the local point cloud points. The connectedness of the vertices is based on a robustness criterion

indicate a connection between the two corresponding point cloud parts. In that sense, we use $\frac{1}{16}d_{12} \leq \min(d_1, d_2)$ as a criterion to place connections in the octree-graph.

Now that the octree-graph is extracted and defined, the graph should be labeled. A label belongs to an edge, but is always associated to a vertex. Every label corresponds to the unique Cartesian direction of the edge from the view point of one of the two incident vertices v_i and v_j. The direction of the graph edges is incorporated by labeling them with a direction label.

Definition 4 A label associated with an edge of the octree graph indicates the direction of the edge with a direction vector. The labels for all 3D-directions are:

Left/Right: $(\pm 1, 0, 0)$,
Up/Down: $(0, \pm 1, 0)$,
Front/Back: $(0, 0, \pm 1)$.

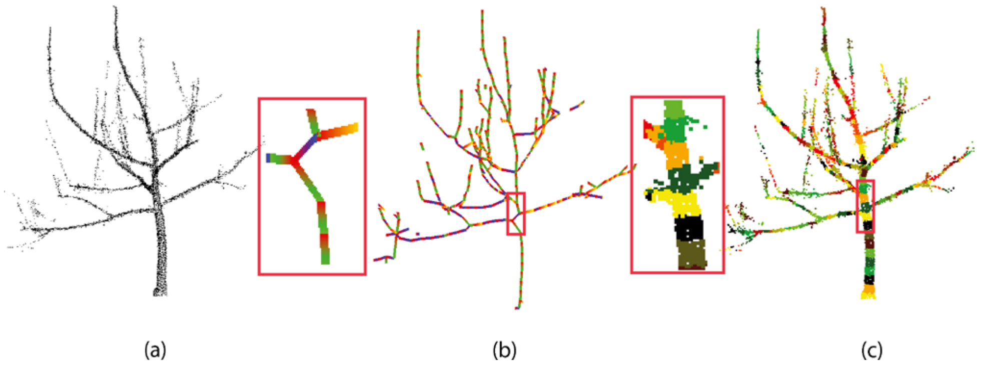


Fig. 5 (**a**) Point cloud of a tree (**b**) derived skeleton with *color* labeled direction labels and a zoom into the skeleton (**c**) *each color* corresponds to a subset of the point cloud associated to one contour and belonging to one vertex in the skeleton graph and a zoom into the level sets

The resulting octree-graph should be interpreted as a bidirectional graph, as every edge gets two labels (Fig. 4). Suppose, that two vertices $v_i, i = 1, 2$, with Cartesian coordinates (x_i, y_i, z_i) are connected. And let $x_1 < x_2$ such that $y_1 = y_2$ and $z_1 = z_2$. Then the edge e_{12} gets the label $(1, 0, 0)$ and the edge e_{21} the label $(-1, 0, 0)$. Note that the sum of the labels belonging to one edge is the zero-vector $(0, 0, 0)$ in 3D. Two graphs with different directional flows can be obtained by taking the $(0, 0, 0)$ labels and either labels containing a positive element in the 3D vector of the edge label or labels containing a negative element into account. This strategy is depict in Fig. 10.

3.2.2 Localized directions

The direction labels will be used to identify the local object elongation which in turn will enable a local reduction of the octree-graph. This localized approach has as its major benefit that it overcomes the rotational dependency, [8], of considering the levels sets s_z of a global height function $f : \mathbb{R}^2 \to \mathbb{R}$. Therefore, we consider level sets locally with respect to one of the three Cartesian directions. That is, dependent on the local object elongation, the height change in either the x-, the y- or the z-direction is considered. This choice is founded in the fact that parts of the surface parallel to the x- or y-axis will collapse to one remaining vertex in the skeleton graph, when taking only a global height function into account. Thus, a local choice of f based on the surface elongation will preserve the representation of surface parts parallel to one of the axis.

The resulting skeleton graph is augmented by vertices between the saddle points, minima and maxima. The minima and maxima are represented as vertices that have one incident edge in the skeleton graph and the saddle points are vertices with 3 or more incident edges. Note here that every vertex in the skeleton graph corresponds to a contour (compare Fig. 5(b) and (c)) derived with respect to the local surface elongation.

3.3 Computational framework

The goal of this section is to show, that the extracted skeleton contains a known topological structure. This structure is the Reeb-graph, which was first introduced by [19] and is strongly linked to Morse theory [14].

The input to the graph reduction is a labeled octree graph. Based on the introduced concept of octree-graphs, Sect. 3.1, and the underlying concept of the grid graph (Definition 3), a graph reduction for the skeletonization algorithm is described. This graph reduction will be defined by merging suitable pairs of neighboring vertices to be specified below. The merging operation of two vertices will be denoted as $\oplus$.

The operations on the octree-graph are intuitively explained as the merge of two vertices incident to the same edges. To quantify the reduction, we introduce the vertex dimension, denoted as *vdim*.

Definition 5 The number of distinct associated edge labels of a vertex v_i is called the dimension $vdim(v_i)$ of a vertex.

In Fig. 4, the vertex in cell A has dimension 0, while the vertices in cell D and E have dimension 1. The vertices in cell B and C both have dimension 2. The vertex in cell C has two associated labels corresponding to two direction labels of opposite direction while the vertex in cell B has labels indicating different principal directions.

In 3D, the dimension of a vertex is at most 6. The convergence toward the skeleton and the preservation of the shape elongation is established by a notion of a local direction. This direction is defined per vertex as vertex direction *vdir*.

Definition 6 The sum $vdir(v_i)$ over the distinct associated edge labels of a vertex v_i is called the vertex direction.

Each label in 3D is a 3D vector, which allows adding up the labels. In Fig. 4, the vertex v_1 in cell B, with the associated labels $(0, -1, 0)$ and $(1, 0, 0)$ has vertex direction

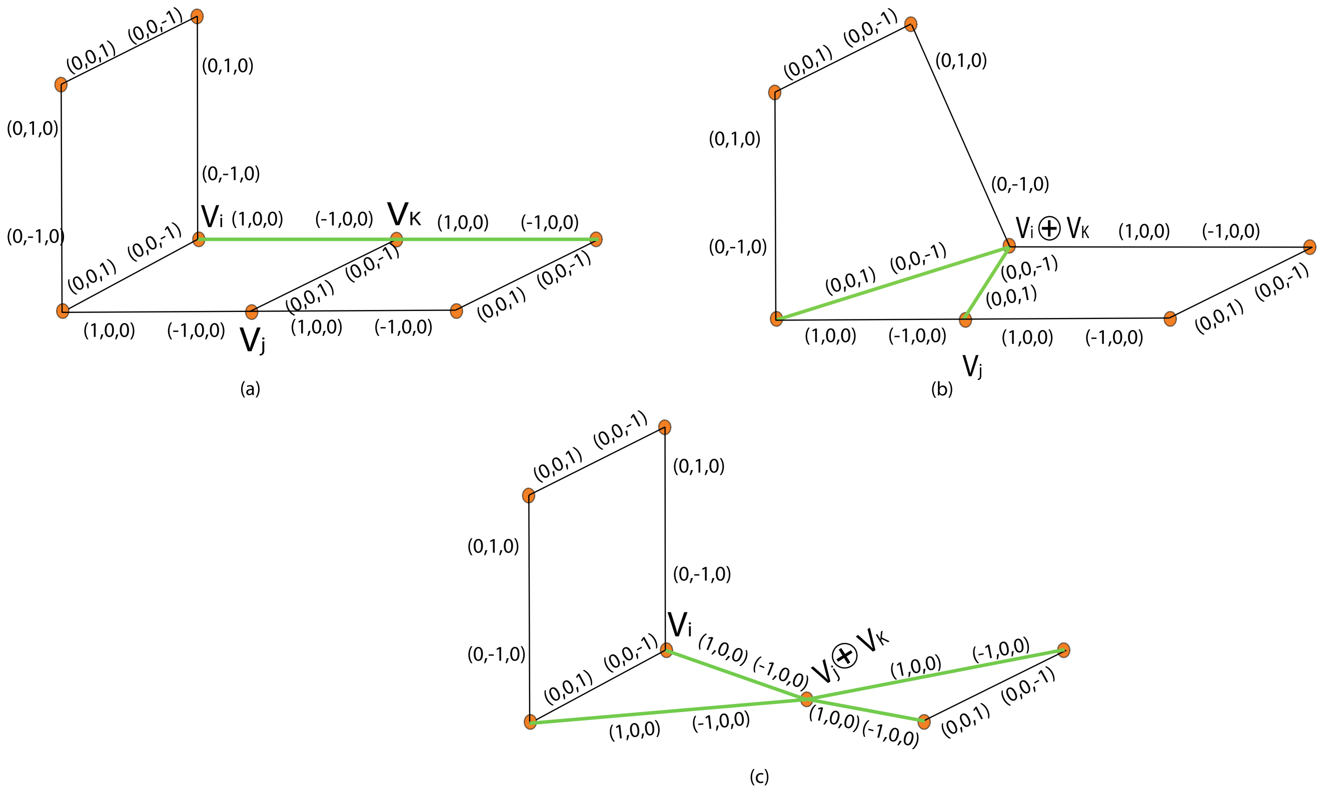


Fig. 6 The dominant direction in v_k. (**a**) The minimal configuration. (**b**) A merge of v_i and v_k without taking the norm value into account resulting in a changed dominant direction (**c**) the merge of v_j and v_k preserving the dominant direction

$vdir(v_1) = (1, -1, 0)$. The vertex v_2 in cell C has $vdir(v_2) = (0, 0, 0)$.

The vertex direction encodes local surface elongation properties. Such a direction is called dominant direction. On the other hand a direction corresponding to a nonzero value for x_i is called a nondominant direction.

Definition 7 Let x_1, x_2, x_3 be the three components of $vdir(v) = (x_1, x_2, x_3)$. The direction of v is trivial if the Cartesian entries are zero for any associated edge label. A direction is dominant at v, if $x_i = 0$ for some $i = \{1, 2, 3\}$, but not trivial.

For example, in Fig. 6a, the dominant direction of vertex v_k is marked green.

Until here, a set of definitions is given for the purpose of valid vertex merging. Still, constraints apply to dimension 3 vertices. Consider the minimal configuration of $G(2, 2, 0)$ subgraphs as given in Fig. 6a with three vertices v_i, v_j, v_k, all three having vertex dimension 3. In the framework given so far, it might happen that v_i is merged to v_k, before v_j is merged to v_k, resulting in the leakage of a dominant direction. Figure 6b demonstrates such a leakage, where the dominant direction in v_k is not represented anymore in the resulting merged vertex $v_i \oplus v_k$.

A solution to this problem is to take the norm of a vertex into account.

Definition 8 Let $vdir(v_i) = (x_1, x_2, x_3)$. The norm of v_i is

$$norm\big(vdir(v_i)\big) = norm(x_1, x_2, x_3) = |x_1| + |x_2| + |x_3|. \quad (1)$$

In case of vertex dimension 3, vertices with smaller norm value are reduced first. Taking the norm into account results in a unchanged dominant direction as can be seen in Fig. 6c. Because $norm(v_k) = 1$ and $norm(v_j) = 1$ (Fig. 6a) and $norm(v_i) = 3$, v_k and v_j are merged before v_i.

The local reduction of the graph will be driven and rules exploiting the underlying grid graph. Two types of special configurations are considered: so-called E-Pairs and V-Pairs. The two configurations are shown in Fig. 8 and Fig. 7. The definition of E-pairs and V-Pairs find their identification in the grid graph properties of the octree-graph. Because of the underlying octree organization the octree-graph is a collection of various connected grid graphs, as introduced in Definition 3. The primary goal is to remove the overrepresented graph parts from the graph by merging vertices. These subgraphs, Fig. 2, are mostly $G(2, 2, 0)$ and $G(2, 2, 2)$ grid graphs, forming loops.

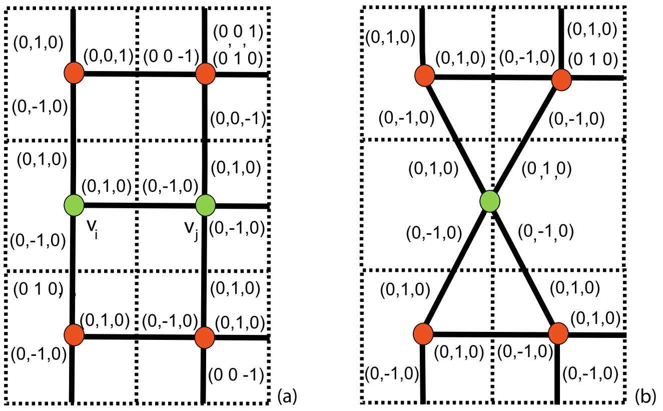


Fig. 7 *Circles* indicate the vertices. The E-Pair and its merging are indicated in *green. The dotted lines* denote the cell sides and the labels are shown along *the black edges*. Note here that the position of $vdim(v_i \oplus v_j)$ is chosen arbitrarily. (**a**) An E-Pair configuration and (**d**) a merging result of the E-Pair in (**b**)

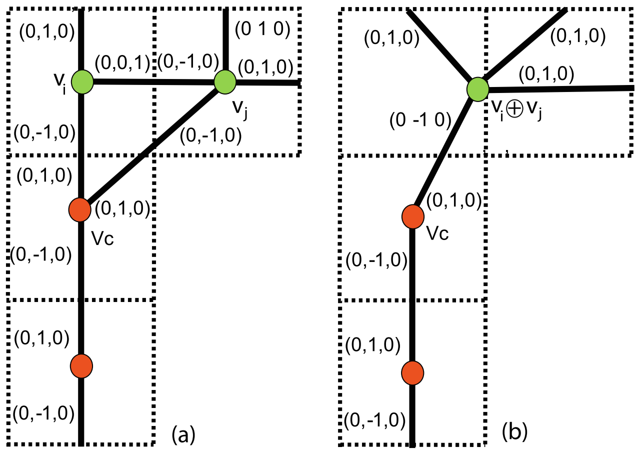


Fig. 8 The vertices are circular. The V-Pair and its merging are indicated in *green. The dotted lines* denote the cell sides and the labels are shown along *the black edges*. Not here that the position of $vdim(v_i \oplus v_j)$ is chosen arbitrarily. (**a**) Example of a V-Pair configuration and (**b**) a merging result of the V-Pair in (**a**)

Definition 9 Let v_i and v_j be two adjacent vertices with $vdim(v_i) \leq vdim(v_j)$. Then v_i forms an E-Pair with v_j if:

1. $vdim(v_i \oplus v_j) \leq \max(vdim(v_i), vdim(v_j))$;
2. $vdir(v_i) \neq (0,0,0)$ and v_i and v_j are connected in the same direction of one of the nonzero entries of $vdir(v_i)$;
3. v_i and v_j do not form a $G(1,2,0)$ subgraph.

Note here that a $G(1,2,0)$ subgraph is a part of the resulting skeleton, which should not be reduced further. In Fig. 7(a), vertices v_i and v_j form an E-pair. In Fig. 7(b) vertices v_i and v_j are merged to $v_i \oplus v_j$.

Definition 10 Two vertices v_i and v_j both incident to a vertex v_c are called a V-Pair if:

1. the labels of edges $v_i v_c$ and $v_j v_c$ are identical;
2. $vdim(v_i \oplus v_j) \leq \max(vdim(v_i), vdim(v_j))$.

The two vertices v_i and v_j shown in Fig. 8(a) form a V-pair, because the edge labels $v_i v_c = (0,-1,0)$ and $v_j v_c = (0,-1,0)$ are identical and $vdim(v_i \oplus v_j) = 3$ is smaller than $\max(vdim(v_i) = 3, vdim(v_j) = 4) = 4$. Figure 8(b) shows the resulting labels of $v_i \oplus v_j$. Remember here that V-Pairs are generated by E-Pairs and because of that the example in Fig. 8 is a intermediate configuration in the reduction process.

3.4 Topological and geometrical correctness

In order to simplify the following explanations, we assume ideal conditions for the principal proof. It is assumed that first the point cloud is without noise, and second that it is sufficiently dense in the sense that it covers every directional change of the surface. The octree-graph extracted from the octree cells encodes both topology and surface information under these idealized conditions. We conclude from that the octree-graph is an alternative for representing the directional behavior of the sampled surface, embedded into 3D Euclidean space under the assumption made above.

The octree-graph, consisting of its vertices and edges labeled by direction, is ideally to be retracted to a graph embedded into 3D-Euclidean space containing only loops representing the topological genus of the sampled object. At this stage, we are ready to connect the resulting skeleton graph to the properties of a known skeleton graph, the so-called Reeb-graph.

Given a piecewise linear function f, the level set of a value $s \in \mathbb{R}$ is defined as the set of points with a function value equal to $s \in \mathbb{R}$. Recall that the octree-graph is embedded into 3D Euclidean space, and because of that every vertex has 3D-coordinates. The construction of a Reeb-graph is based on the analysis of the evolution of the connected components of the level sets generated by f. We call such a level set to be merged a contour. In the following, we show that the given retraction rules are an analysis of the evolution of the connected components of the level sets generated by f. Practically, a Reeb-graph connects the contours with respect to f. For a function value s_i of f where the number of contours increases compared to s_{i-1}, the Reeb-graph will split and indicate a saddle point of f. For values s_i having no successor s_{i+1}, the Reeb-graph represents a maximum. A minimum is present if s_i has no predecessor s_{i-1}.

In this paragraph, we use a local elongation function $e(vdir(v))$ on the octree-graph as the piecewise linear function f. Here, $e(vdir(v))$ gives the local elongation of the object surface at a vertex v_i by its vertex direction $vdir(v_i)$.

Definition 11 Let $v = (x, y, z)$ be a vertex in $\mathbb{R}^3$, with $x, y, z \in \mathbb{R}$. Let $vdir(v) = (d_1, d_2, d_3)$ with $d_i \in \{-1, 0, 1\}$,

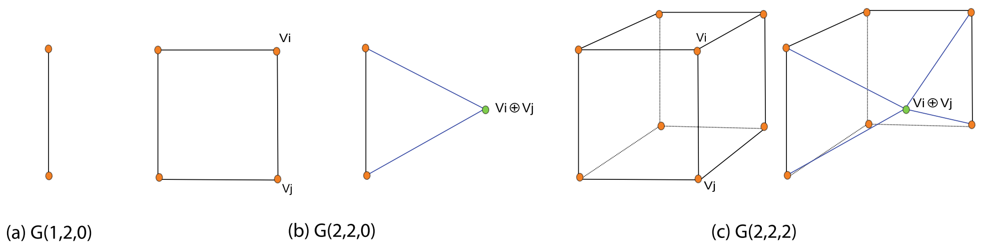


Fig. 9 (**a**) A $G(1,2,0)$ subgraph, belonging to the skeleton. (**b**) A $G(2,2,0)$ subgraph and a possible V-Pair derived from it. (**c**) A $G(2,2,2)$ subgraph with an example of a possible derived V-Pair configuration. $\oplus$ denotes the merge of two neighboring vertices

where $d_i = 0$ indicates a dominant direction. The function to extract the contours is then defined as

$$e(vdir(v)) = \begin{cases} x & \text{if } vdir(v) = (0,*,*); \\ y & \text{if } vdir(v) = (*,0,*); \\ z & \text{if } vdir(v) = (*,*,0). \end{cases}$$

$$\text{with } * \in \{-1,0,1\} \qquad (2)$$

Note that this definition is not unique in case there is more than one dominant direction. This, however, is not a problem. A different choice of dominant direction only corresponds to a different order of reduction of the octree-graph. Careful investigation of the elongation function $e(vdir(v_i))$ reveals that the height function is included $e(vdir(v_i)) = z$, if the object is only elongated in the Cartesian z-direction.

Proposition 1 *Let OG be an octree graph derived from a sampled object. The merging operations on OG defined by E-Pairs and V-Pairs result in a skeleton containing the Reeb-graph.*

We prove Proposition 1 by proving three lemmas. First, we prove that the local elongation is correctly represented and the dominant direction stays unchanged during merging operations in Lemma 1. Then we prove in Lemma 2 that E-Pairs always result in V-Pairs, which assures the convergence of the algorithm towards a skeletal line. At last, we prove in Lemma 3 that merging V-Pairs does not change the dominant direction. Lemma 3 relies on the correct representation of the elongation in Lemma 1 and the guaranteed convergence in Lemma 2. Proposition 1 is proven, by showing the three lemmas in the following setting:

Let OG be an octree-graph derived from an octree subdivision, as described in Sect. 3.2, formed of $G(1,2,0)$, $G(2,2,0)$ and $G(2,2,2)$ subgraphs. These three underlying subgraphs are the minimal cases considered to locally describe a merge of two connected vertices. A $G(1,2,0)$ is the trivial case belonging to the skeleton demanding no further processing (compare Fig. 9(a)). Every $G(2,2,2)$ subgraph, Fig. 9(c), is the union of vertices and edges of six $G(2,2,0)$ subgraphs, Fig. 9(b), inducing four valid E-Pair configurations. This coherence between E-Pairs and the $G(2,2,0)$ grid graph allows to prove Lemma 1 and Lemma 2 on an valid E-Pair configurations in a $G(2,2,0)$ setting. Recall, that every edge of OG is labeled with two labels, containing a positive and a negative component. In the following lemma, we make use of the fact that two graphs with opposite directional flows can be obtained. The first directed graph is derived by removing all edge labels from the octree-graph containing a negative entry, Fig. 10(c). The second directed graph is derived by removing all edge labels from the octree-graph containing a positive entry, Fig. 10(b).

Lemma 1 *Let E be an E-Pair consisting of two vertices v_i and v_j. The merging operation $v_i \oplus v_j$ preserves the dominant directions of v_i.*

Proof Consider two connected $G(2,2,0)$ subgraphs A and B. Two cases have to be considered. First, the case where A and B share exactly one vertex resulting in maximal 2 valid E-Pair configurations, corresponding to exactly one dominant direction (Fig. 11 subgraph A). The second case consists of two subgraphs sharing exactly one edge and 2 vertices. The second case results in exactly one possible E-Pair configuration (Fig. 11 subgraph B). The E-Pair configuration of the subgraph contains only one non-dominant direction. According to Definition 9, merges only occur in nondominant directions, preserving the dominant one. □

Note that an example of the elongation description by an E-Pair was already given in Fig. 6a.

The octree-graph is retracted by merging vertices forming a V-Pair. If no V-Pair is present in the graph to be retracted, a V-pair is created by an E-Pair representing local elongation of the sampled surface. Now that it is shown that the merge of an E-Pair preserves the elongation, it has to be shown that every E-Pair is resulting in a V-Pair, for continuation of the retraction process.

Lemma 2 *The merging of an E-pair results in at least one V-Pair.*

Proof A $G(2,2,0)$ contains 4 valid E-Pair configurations, as illustrated in Fig. 12. Consider one arbitrary E-Pair configuration of a $G(2,2,0)$. Merging the two vertices involved

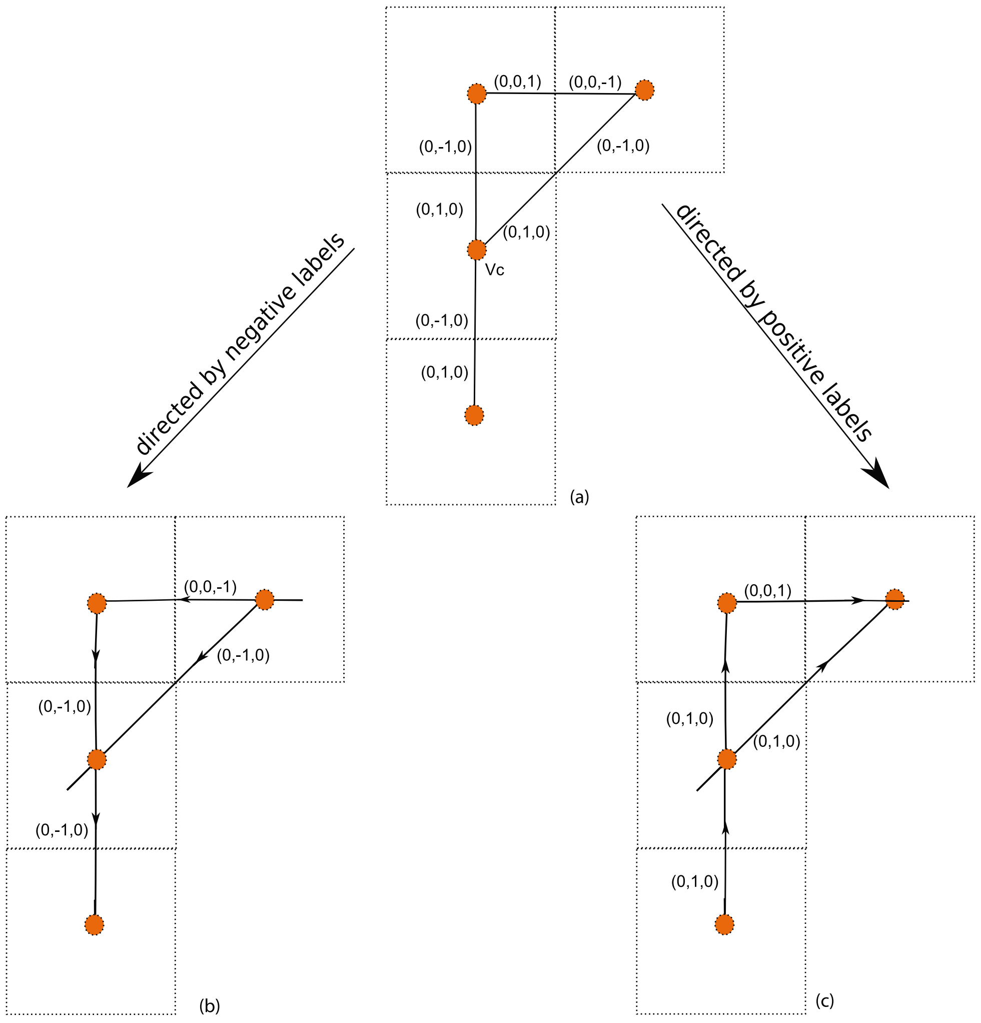


Fig. 10 (**a**) A labeled octree graph is shown. (**b**) and (**c**) the two derived graphs are shown. These derived graphs are directed. (**b**) Contains only label elements ≤ 0 and (**c**) contains only label elements ≥ 0

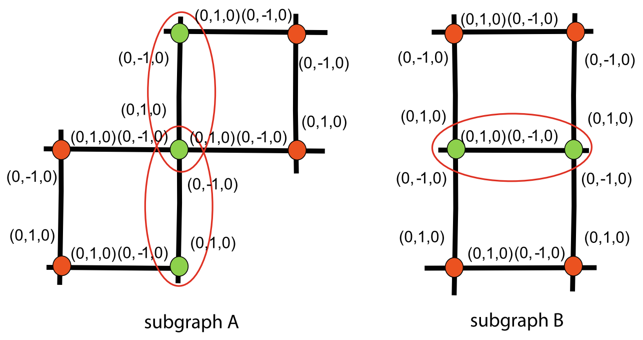


Fig. 11 The two considered configurations of Lemma 1, with possible E-Pairs marked by an ellipse

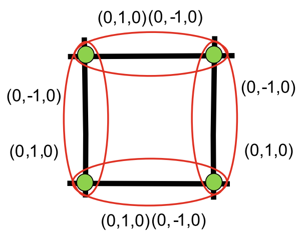


Fig. 12 The considered configuration of Lemma 22, with possible E-Pairs marked by an ellipse

in the E-Pair, reduces the squared structure to a triangle satisfying the definition of a V-Pair according to Definition 10. □

Concluding that every E-Pair results in a V-Pair, as shown in Fig. 6c on an example, it is finally necessary to show that the merge of a V-Pair does not change the dominant direction in the graph.

Lemma 3 *Merging of V-Pairs does not change the local dominant direction.*

Proof Consider the triangle structure of a V-Pair, Fig. 8. The triangle structure of a V-Pair contains at least one direction label exactly two times, and the label to be removed by the merging operation exactly one time. The merge removes exactly one dominant label, which preserves the difference between the amount of dominant and non-dominant labels. □

3.5 Graph embedding

The desired centeredness of the skeleton is achieved via a graph embedding. Thus, the octree graph is embedded into the point cloud by averaging the points Σ_i belonging to an octree-cell. The embedding explained in Pascucci et al. [17] was adapted to points clouds, because their embedding can be applied during the computation of the SkelTre Skeleton. Every vertex in the octree graph has an initial weight w which is equal to the number of points belonging to the corresponding octree cell. During the merging process the weighted average of the 3D positions of two merged vertices v_1 and v_2 is taken for the position of the newly created vertex v_{new}. The weight w_{new} of v_{new} is then the sum of the previous weights, compare Fig. 13, that is, $w_{\text{new}} = w_1 + w_2$. These weights are used to compute the coordinates of v_{new} into the point cloud. The position of v_{new} is calculated as: $v_{\text{new}} = \frac{w_1 \cdot v_1 + w_2 \cdot v_2}{w_1 + w_2}$ In case of different levels of subdivision, the subdivision level is multiplied to the weight to obtain a centered skeleton as a result. Embedding can be problematic, if the hull of Σ_i is concave, because then the centroid is not necessarily equal to the weighted average described above. In case of trees this occurs on vertices, where the skeleton branches. For such cases, a post-processing step is required. The post-processing step treats the 3 or more connected vertices of the skeleton, by investigating the distance of point cloud points Σ_i belonging to one vertex to their bounding box. Note here that Σ_i is a subset of the point cloud Σ. Let the 6 sides of a bounding box be the 6 Cartesian directions of some Σ_i. And let every point p_i be closest to one of these sides, then the $\{x, y, z\}$-coordinate component closest to a side of the bounding box is selected. The corrected vertex coordinate is computed by averaging selected $\{x, y, z\}$ values for every coordinate component.

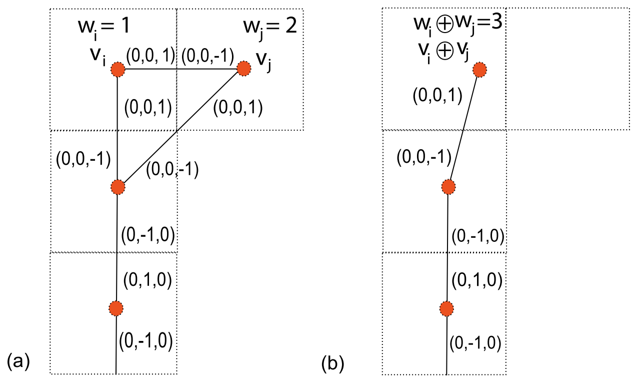


Fig. 13 Example of an embedding. (**a**) Shows a graph before merging is applied to v_1 and v_2 and (**b**) the graph with the new vertex v_{new}

4 Algorithm analysis

This section demonstrates the implementation of the algorithm as pseudo code. Furthermore, the computational complexity of the algorithm is discussed together with its behavior per vertex dimension in practice.

4.1 Implementation of the framework

This section discusses an implementation of the given computational framework. For the vertex dimensions counting from $n = 5$ to $n = 2$, successively the following steps have to be implemented. The merging is visualized on a simple 2D example in Fig. 14.

1. Initialize a vertex-list *VPairList* containing all vertices of dimension n. For each vertex v_i in *VPairList* test if new V-Pairs can be formed with its direct neighbors v_j, until

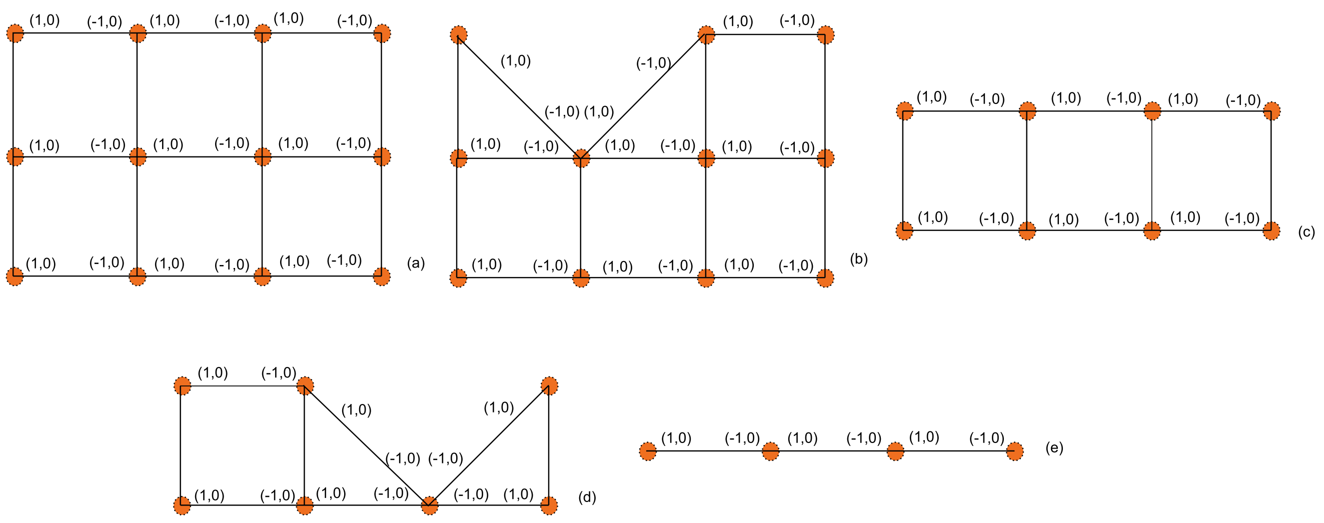


Fig. 14 Example of the collapsing procedure

either a V-Pair is found, or no direct neighbors to test are left. All V-Pairs found are stored in a list.

2. If during the loop through the vertex list *VPairList* no V-Pair was found, go through the vertex list *VPairList* again. Whenever possible, an E-Pair is merged to one or more V-Pairs with the direct neighbors v_j of v_i, until either an E-pair is found, or no direct neighbors to test are left. Each E-pair is merged to one or more V-Pairs which are added to the list of V-Pairs.
3. All vertex pairs in the *VPairList* are merged. Directly after merging it is tested whether the merging resulted in the creation of new V-Pairs. If so, these are added at the end of the V-Pairs list.

Remember that the merging of two vertices v_i and v_j along a common edge is denoted by $v_i \oplus v_j$. The resulting merged vertex v_c inherits all incident edges from its ancestors v_i and v_j. If v_i and v_j were both incident to a common vertex v_c, then the two edges $v_i v_c$ and $v_j v_c$ are collapsed to a common edge $(v_i \oplus v_j)v_c$. Under ideal conditions, these edges represent the connection between two connected subsets of the sampled surface Σ. For this reason, the operation $v_i \oplus v_j$ is only performed on vertices representing two neighboring subsets of Σ with the same direction characteristic, as indicated by the identical edge labels of $v_i v_c$ and $v_j v_c$.

4.2 Computational complexity

In practice, the computation of the skeletons operates on a far smaller set of vertices than the number of points in the point cloud. Our focus lies on the explanation, that the graph-reduction of the SkelTre algorithm is linear in time. A pseudo code to implement the algorithm is shown in Procedure 1 and Procedure 2.

Let v_i be a vertex of the set of vertices V of the processed graph. The dimension of v_i is denoted as $vdim(v_i)$ and the number of incident edges as $k(v_i)$. Let v_j denote a direct neighboring vertex of v_i and $c_1 \leq 6$ a constant. The procedure *computeSkeleton* contains an outer for-loop, which is bounded by $O(1)$. As can be noticed in Procedure 1, *dimList* is always initialized with $O(V)$. Note that V is decreasing after every dimension. The inner while-loop is operating on a subset of V with at most $\frac{V}{2}$ operations, which results in $O(\frac{V}{2})$ as an upper bound.

The procedure *computeVPair()*, selects the first unprocessed entry v_i in *dimList* that fulfills Definition 9, and loops through all elements of *dimList*. We show the influence of this condition on the inner while-loop for two extreme cases:

Case 1 The input graph is already a skeleton, e.g., a combination of $G(2,1,0)$ subgraphs, all connected on only 2 vertices. This combination will lead to no merge at all; because of that the whole inner while-loop of *computeSkeleton()* stays $O(V)$ by checking one time *Definition* 9 $=$ *true*.

Case 2 All $vdim(v_i)$ are equal within V, e.g. a graph formed by $G(2,1,0)$ subgraphs, which all are connected on three vertices. This will lead to exactly one check of condition *Definition* 9 $=$ *true* in the given example, and c_2 calls. c_2 is bounded by the minimal number of aligned $G(2,2,0)$ subgraphs in one of the principal Cartesian directions.

Procedure 1 `computeSkeleton`

```
Input: An Octree graph
Output: SkelTre Skeleton

contains the vertices of the of the processed dimension;
dimList[];
contains found VPairs;
VPairList[];
for $dim = 5$ to $2$ do
    $dimList := \{v_i | vdim(v_i) \geq dim, i = 0 \dots n\}$;
    forall $v_i \in dimList$ do
        if $VPair = true$ for some $v_k$ of $v_i$ then
            add found pair to the end of VPairList;
        end
    end
    while $VPairList \neq \emptyset$ and $createVPair()$ return true do
        $\{v_i, v_j\} =$ first unprocessed entry in VPairList;
        $v_{\text{new}} = v_i \otimes v_j$;
        if ($VPair = true$ for some $v_k$ of $v_{\text{new}}$) then
            add found pair to the end of VPairList;
        end
        if $vdim(v_{\text{new}}) \geq \max(vdim(v_i, v_j))$ then
            add $v_{\text{new}}$ to the end of dimList;
        end
        remove $\{v_i, v_j\}$ from VPairList;
    end
end
```

Procedure 2 `createVPair`

```
Input: a vertex $v_i$
Output: true or false

if $vdir(v_i) \neq \emptyset$ then
    foreach $v_j$ adjacent to $v_i$ do
        if $EPair = true$ for $\{v_i, v_j\}$ then
            add found pair to the end of VPairList;
            return true;
        end
    end
end
return false
```

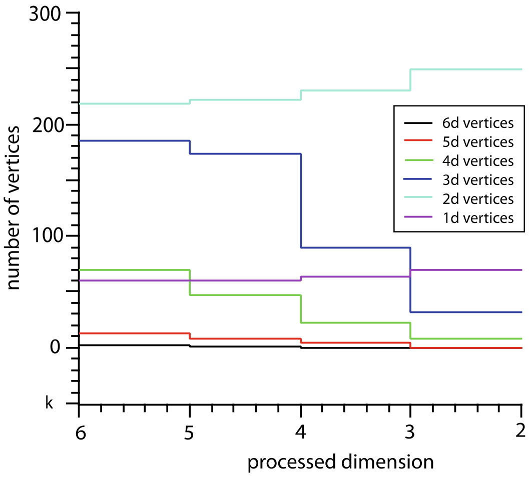
Now that it is shown that the influence of *computeVPair()* on the inner while loop is $O(c_2)$ or $O(V)$, the algorithms overall complexity can be calculated as follows from the upper bound. $O(1) \cdot (O(V) + O(\frac{V}{2}) + O(V)) = O(V)$. Note here, that the special case of dimension 3 vertices are simply three lists, each for every norm value. Because of that, they have no influence on the complexity analysis given here.

4.3 Algorithm behavior

Vertex dimension 6 results in vertex direction 0, 0, 0 in all cases and because of that it can be omitted in the reduction. The reduction of the graph is therefore limited to vertex dimension 5 to 2. This successive reduction from vertex dimension 5 to 2 guarantees that first the $G(2, 2, 2)$ graph parts are reduced, before $G(2, 2, 0)$ subgraph regions are processed. Another example to depict the algorithm is to characterize vertices of a certain dimension. For example, $vdim(v) = 3$ can be either a corner of a $G(2, 2, 2)$ with $vdir(v) = (\pm1, \pm1, \pm1)$ or a vertex on the boundary of $G(2, 2, 0)$ subgraph with one entry $\neq 0$ in $vdir(v)$. Boundary vertices of a $G(2, 2, 2)$, which have dimension 4, are processed before the "corner"-vertices of a $G(2, 2, 2)$ area, and the boundary of a $G(2, 2, 0)$ region is processed before its "corners" of $vdim(v) = 2$.

In Fig. 15, the convergence of the algorithm is illustrated. The values correspond to the skeleton in Fig. 5(b). Figure 15(top) depicts the algorithmic behavior of SkelTre per vertex dimension. It is observable that the amount of $vdim = 1$ and $vdim = 2$ vertices is increasing, as they form mainly the targeted skeleton. Meanwhile, $vdim = 5$ and $vdim = 6$ vertices are vanishing as expected, also a rapid decrease of $vdim = 3$ and $vdim = 4$ vertices is recognized. This behavior of the algorithm is coherent to the behavior described in theory.

The red curve in Fig. 15 (bottom) shows that the number of graph vertices is decreasing after every processed vertex dimension. The black curve shows the overall number of merging operations after every processed vertex dimension, which is almost constant between dimension 3 and 2. This approximate constant behavior is explainable by the influence of the procedure *createVPair()* (compare Sect. 4.2, Case 1), because this intermediate case is already close to the desired skeleton. This observed performance is also in compliance with the expectations described theoretically.



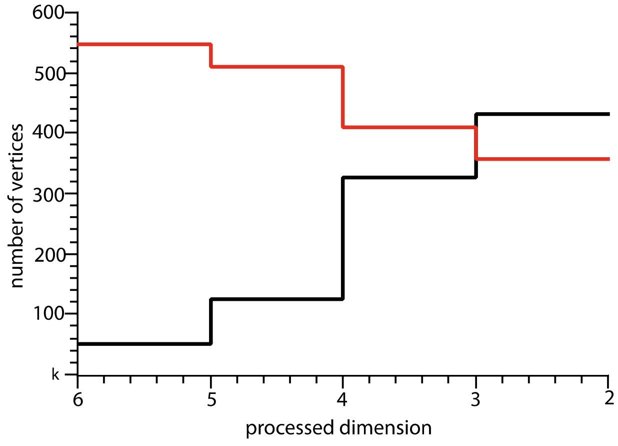


Fig. 15 (*Left*) amount of vertices per dimension after every processed vertex dimension. (*Right*) *The red* graph shows the vertices belonging to the intermediate skeleton and *the black* graph the number of merged vertices after every processed vertex dimension

5 Results and practical validation

The evaluation of the extracted skeletons considers several examples and validation parameters. This section is divided into three parts, trees and nontree objects. The trees and objects use a variety of scanners and produce different density, sampling, and noise characteristics. Limitations are pointed out to explain the algorithm behavior. In all cases, the skeleton edges are colored by their resulting direction labels. Yellow denotes a $(1, 0, 0)$ label, blue a $(0, 1, 0)$ label, and green a $(0, 0, 1)$ label. In all labeling cases, the corresponding negative label is indicated in red.

No post processing is applied to the skeleton. Centeredness is analyzed by considering the (average) Euclidean distance of every point cloud point to the skeleton. First results on botanical trees are presented and secondly results on nontree-like objects are shown.

In Sect. 5.3, running time aspects of our implementation in a proprietary software are shown. This evaluation gives the parameters used for the examples and the running times.

5.1 Trees

In Fig. 16, results of the SkelTre algorithm on point clouds representing three different trees scanned with three different terrestrial laser scanners: a Simple Tree, an Apple tree,

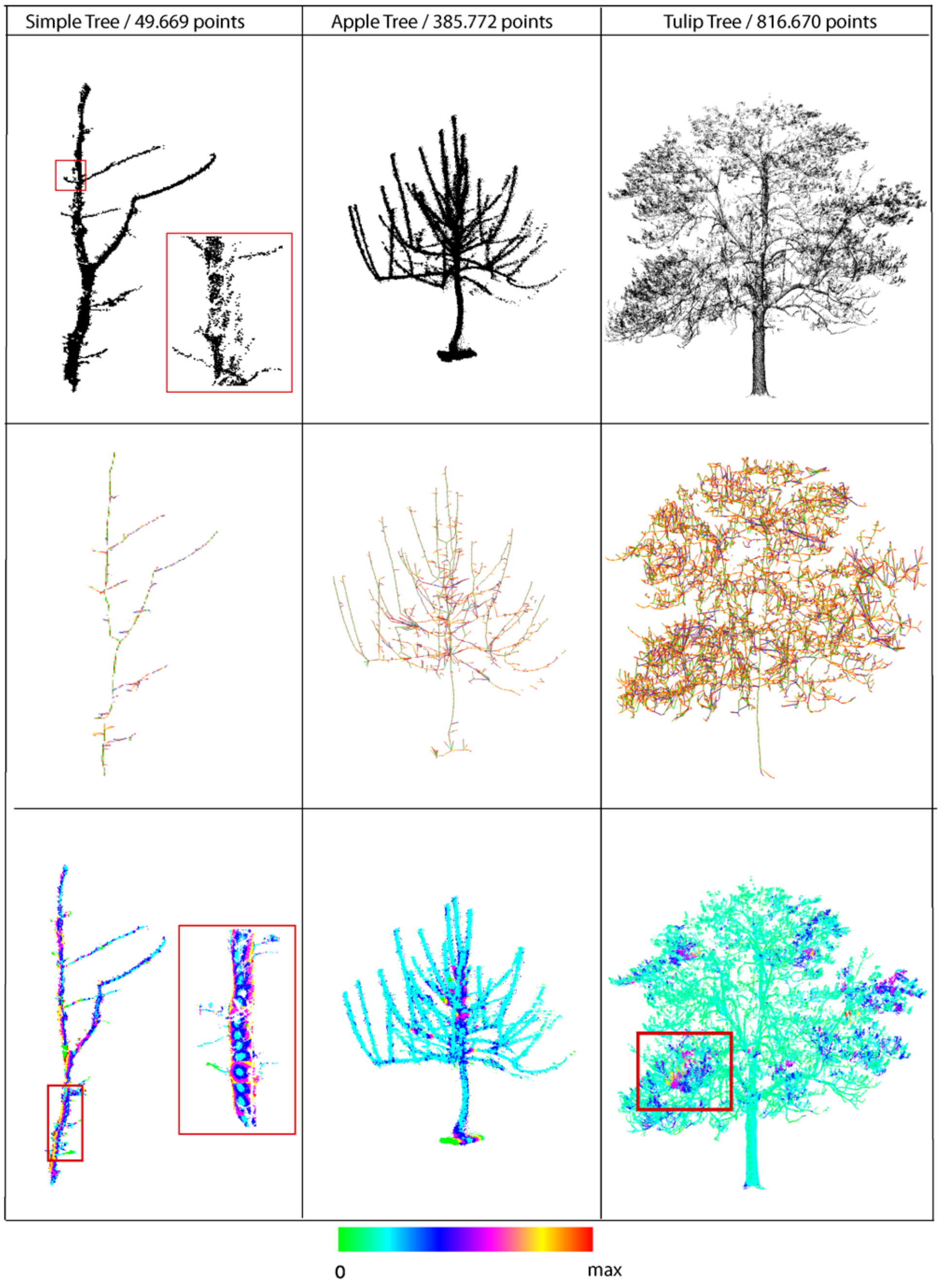

Simple Tree / 49.669 points
Apple Tree / 385.772 points
Tulip Tree / 816.670 points
0
max

Fig. 16 *First row* shows the raw point cloud. *The second row* shows the skeleton colored by direction labels. All negative ends are labeled *red*. The directions Up/Left/Front are colored with *yellow*/*blue*/*green*. *The third row* shows the distances to the skeleton according to the color scheme given *on the bottom of the figure* (*red* indicates more than 10 cm for the Simple Tree and the Apple Tree and more than 100 cm for the Tulip Tree). *The black point* cloud part belonging to the Simple Tree shows strong undersampling due to occlusions. The tree *in the second column* is an orchard cherry tree with small blossoms, which can be recognized as noise. The third tree is a huge Tulip tree of 11,5 meters height with leafs

and a Tulip tree. For the Simple tree and the Apple tree, the maximum of the color scale indicates distances to the skeleton larger than 10 cm. For the Tulip tree, the maximum of the color scale indicates distances to the skeleton larger than 100 cm.

The Simple tree was scanned with a Leica Scan Station laser scanner and was previously used in the publication of [13]. This tree of 4.07 m height is sampled by 49,669 points The simple tree shows good connectedness in the 18 detected branches, even under bad sampling conditions (upper red box in Fig. 16). The distances to the skeleton show a symmetric pattern. The Simple Tree is represented by a single scan from one viewpoint. On this single scan the skeleton is attracted to one side (light blue color), because the back of the tree was not scanned. The sampling density shows a very noisy pattern already on a single scan. Spurious branches on the boundary indicate the noise on the object boundary.

The Apple Tree was scanned with the Zoller and Froehlich scanner Imager 5006 in high resolution. Therefore, the

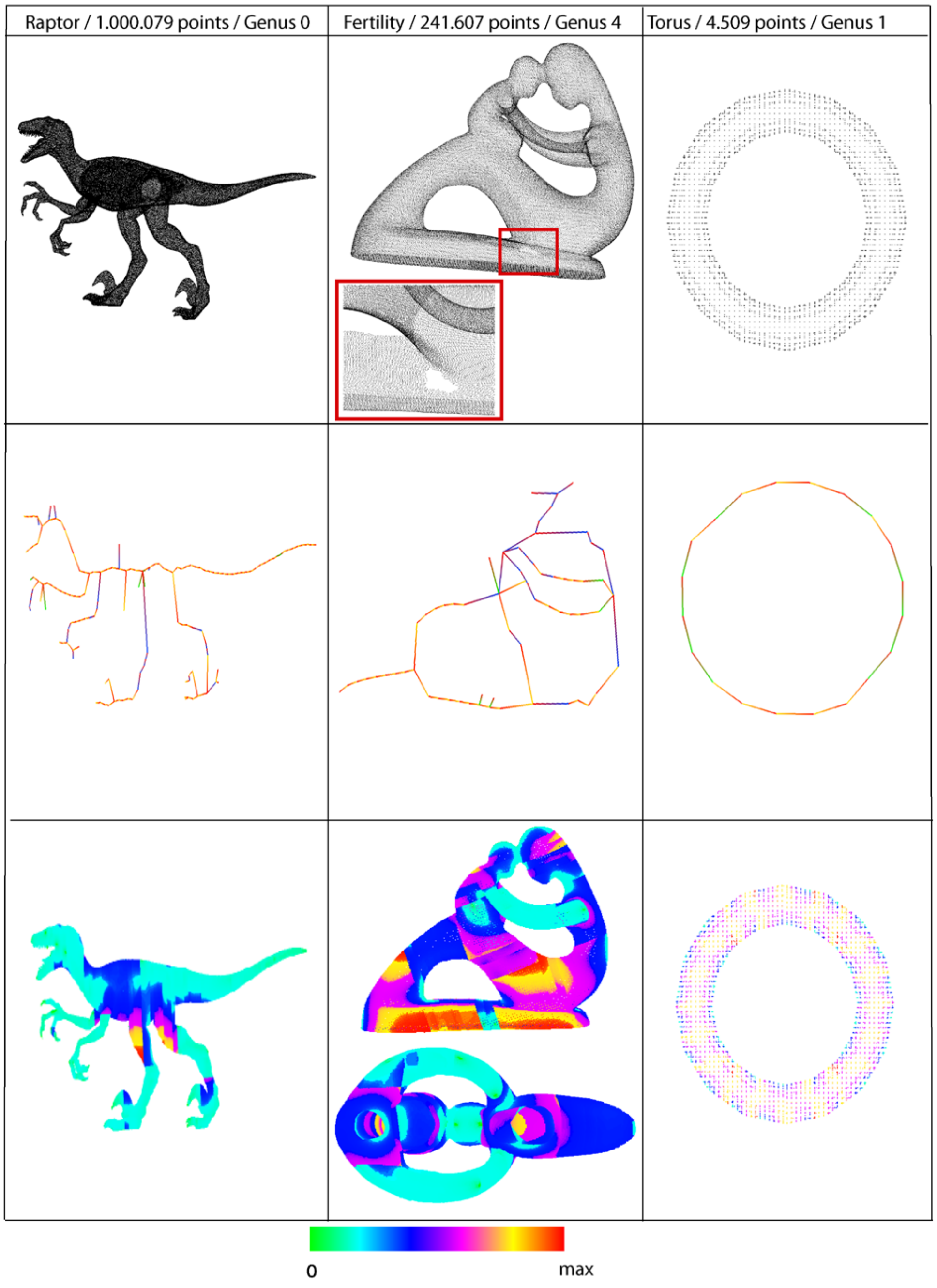

Raptor / 1.000.079 points / Genus 0
Fertility / 241.607 points / Genus 4
Torus / 4.509 points / Genus 1
0
max

Fig. 17 3 test objects. *First row* shows the raw point cloud. *The second row* shows the skeleton colored by with direction labels. All negative ends are labeled *red*. The directions Up/Left/Front are colored with *yellow*/*blue*/*green*. *The third row* shows the distances to the skeleton according to the color scheme given *on the bottom of the figure*

Apple tree was sampled by 385,772 points by Stefan Fleck from the University of Göttingen. The Apple tree is 1.99 m high and its largest extension in the crown is 1.62 m. The derived skeleton of the Apple tree contains only one erroneous loop due to unresolvable noise problems in the dense crown containing 136 detected branches. The distances to the skeleton get larger in the inner crown, because the crown contains a huge amount of noise, as discussed before in this paper.

The Tulip tree was scanned with a Calidus scanner. It is 11.75 m high and 14.47 m wide at the largest extension of the crown. This massive tree was scanned by Forstliche Versuchs- und Forschungsanstalt Freiburg near Karlsruhe, Germany. It is sampled by 816,670 points. The tree was scanned during summer time and contains leaf. The marked region show the difficulty on this particular tree. Noise, e.g., because of moving leaves during the scan procedure, makes a meaningful extraction of the skeleton impossible. The locally bad extracted skeleton is visible as distances to the skeleton in the order of 5–8 cm on fine branches. Still, the major branches and the trunk are extracted correctly.

Table 2 Running time for the example objects

Modell	Number of points	Octree construction time	Graph reduction time	Octree graph vertices	Octree cells size	Data range
Simple tree	49,669	0.45 s	1.1 min	667	0.1	x: [225.0–229.1] y: [393.2–397.3] z: [102.2–106.3]
Apple Tree	385,772	2.7 s	192.0 min	5064	0.05	x: [225.0–229.1] y: [393.2–397.3] z: [102.2–106.3]
Tulip Tree	816,670	1.8 s	275.6 min	5084	0.5	x: [225.0–229.1] y: [393.2–397.3] z: [102.2–106.3]
Raptor	1,000,079	2.3 s	10.5 min	2404	0.04	x: [225.0–229.1] y: [393.2–397.3] z: [102.2–106.3]
Fertility	241,607	0.5 s	9.9 min	2132	10.0	x: [−75.7–123.9] y: [−76.9–122.8] z: [−36.9–162.8]
Torus	4,509	0.1 s	0.01 min	92	0.5	x: [−1.2–1.2] y: [−0.2–2.2] z: [−1.2–1.2]

5.2 Nontree objects

In this paragraph, examples of nontree objects are given. Note here that distances are not given with a unit, because the unit of the data sets is not known.

The Raptor model with lots of ripple on its skin is taken from the aim@shape repository. It consists of 1,000,079 points and no information about the construction process is known. The skeleton shows good connectedness and clearly represents the saddles and maxima of the object. The skeleton of the left foot shows a straight element, which results in unexpected high distances to the skeleton. In this case, not all directional changes are covered by the chosen minimum cell size. Instead of a curved skeleton of the toe, a straight line is the result. If not all directional changes of the surface are modeled, the skeleton is not embedded correctly, due to the fixed octree cell size used in this paper. Nevertheless, the local surface maximum is modeled preserved.

The Fertility model is a genus 4 stone sculpture and taken from the aim@shape repository as well. It consists of 241,607 points. It was obtained with a Roland LPX-w50 laser range scanner. The marked region shows a huge hole in the point cloud of the statue, which prevented the use of fine resolutions. This results in insufficient skeleton resolution on the left and in the head region of the statue, because not all directional changes of the statue could be modeled by the octree graph. The genus is still represented correctly in the skeleton, and the maxima are modeled correctly as far the octree-graph covered them (e.g., on the baby's stomach).

The Torus model is sampled by 4,509 points from a polygonal surface. The sampling is rough to show the influence of strong undersampling. From theory, one would expect two maxima on a ring; compare Definition 11. The strong undersampling is the reason for the vanishing maxima. Furthermore, a pattern is observable in the distances to the skeleton, because of the discreteness of the skeleton-graph.

5.3 Running time

The calculation time for six examples given in Fig. 16 and Fig. 17 is given in Table 2. The table shows the size of the point cloud as number of point cloud points and its data range, the time needed to construct the octree under a fixed octree cell size. Furthermore, the number of octree graph vertices is given and the time needed to reduce the graph with the SkelTre algorithm. The calculation times given refer to a Intel Dual-Core processor 6700 running at 2.66 GHz having 3.5 GB memory. The operating system used was Windows XP with Service Pack 3 installed.

Table 2 shows the time needed to perform the octree construction and the graph reduction. The running times are given for the algorithm as given in Sect. 4.1. The implementation uses the vector container of the Standard Template Library, [15] to store the adjacency list of the graph and all other intermediate lists, causing performance loss above 1,000 graph vertices.

6 Conclusion

In this paper, the SkelTre skeletonization method has been presented. The method reduces an initial graph, corresponding to the subdivision of a point cloud by a suitable octree, to the SkelTre skeleton using only one input parameter. We have shown that the produced skeleton incorporates a well-known topological structure, the Reeb-graph. It could be shown that the graph reduction is linear in time, which underlines the efficiency of the method that is designed to skeletonize real, large point clouds of botanic trees. Both the correctness and robustness could be demonstrated on several different point clouds. The capability of the method to skeletonize point clouds of a much larger class of objects was shown on examples. Further research will first focus on applying the retrieved skeletons on the automatic extraction of structural parameters and sizes of objects.

References

1. Amenta, N., Choi, S., Kolluri, R.: The power crust, unions of balls and the medial axis transform. Comput. Geom., Theory Appl. **19**(2–3), 127–153 (2001)
2. Arthur, D., Vassilvitskii, S.: On the worst case complexity of the k-means method. Technical Report, Stanford 698 (2005)
3. Blum, H.: A transformation for extracting new descriptors of shape. In: Proceedings Models for Perception of Speech and Visual Form, pp. 362–380 (1967)
4. Bucksch, A., Lindenbergh, R.: Campino—a skeletonization method for point cloud processing. ISPRS J. Photogramm. Remote Sens. **63**, 115–127 (2008)
5. Bucksch, A., Lindenbergh, R., Menenti, M.: Skeltre—Fast skeletonization of imperfect point clouds of botanic trees. In: Proceedings 3D Object Retrieval Workshop 2009 (3DOR), pp. 13–20 (2009)
6. Chen, H.H., Huang, T.S.: A survey of construction and manipulation of octrees. Comput. Vis. Graph. Image Process. Arch. **43**(3), 409–431 (1988)
7. Cole-McLaughlin, K., Edelsbrunner, H., Harer, J., Natarajan, V., Pascucci, V.: Loops in Reeb graphs of 2-manifolds. Discrete Comput. Geom. **32**(2), 231–244 (2004)
8. Cornea, N.D., Min, P.: Curve-skeleton properties, applications, and algorithms. IEEE Trans. Vis. Comput. Graph. **13**(3), 530–548 (2007)
9. Côtè, J.F., Widlowski, J.L., Fournier, R.A., Verstraete, M.M.: The structural and radiative consistency of three-dimensional tree reconstructions from terrestrial lidar. Remote Sens. Environ. **113**(5), 1067–1081 (2009)
10. Dey, T.K., Sun, J.: Defining and computing curve-skeletons with medial geodesic function. In: Proceedings 4th Eurographics Symposium on Geometry Processing, pp. 143–152 (2006)
11. Foreman, R.: Morse theory for cell complexes. Adv. Math. **134**, 90–145 (1998)
12. Gorte, B.: Skeletonization of laser-scanned trees in the 3d raster domain. In: Proceedings 3DGeoInfo 2006 (2006)
13. Gorte, B., Pfeifer, N.: Structuring laser-scanned trees using 3D mathematical morphology. Int. Arch. Photogramm. Remote Sens. **XXXV**(B5), 929–933 (2004)
14. Milnor, J.: Morse Theory. Princeton University Press, Princeton (1963)
15. Musser, D., Saini, A.: Stl Tutorial and Reference Guide: C++ Programming with the Standard Template Library. Addison-Wesley Professional Computing Series. Addison-Wesley, Reading (1996)
16. Palagyi, K., Sorantin, E., Balogh, E., Kuba, A., Halmai, C., Erdohelyi, B., Hausegger, K.: A sequential 3D thinning algorithm and its medical applications. In: Proceedings 17th International Conference Information Processing in Medical Imaging, pp. 409–415 (2001)
17. Pascucci, V., Scorzelli, G., Bremer, P.T., Mascarenhas, A.: Robust on-line computation of Reeb graphs: simplicity and speed. ACM Trans. Graph. **26**(3), 58 (2007)
18. Pemmaraju, S., Skiena, S.: Computational Discrete Mathematics: Combinatorics and Graph Theory with Mathematica. Cambridge University Press, Cambridge (2003)
19. Reeb, G.: Sur les points singuliers d'une forme de Pfaff complèment intégrable ou d'une fonction numérique. C. R. Acad. Sci. **222**, 847–849 (1946)
20. Saito, T., Toriwaki, J.: New algorithms for Euclidean distance transformation of an n-dimensional digitized picture with applications. Pattern Recogn. **27**, 1551–1565 (1994)
21. Serra, J.: Image Analysis and Mathematical Morphology. Academic Press, London (1982)
22. Shan, J., Todd, C. (eds.): Topographic Laser Ranging and Scanning. CRC Press, Boca Raton (2008)
23. Shinagawa, Y., Kunii, T.L.: Surface coding based on Morse theory. IEEE Comput. Graph. Appl. **11**, 66–78 (1991)
24. Verroust, A., Lazarus, F.: Extracting skeletal curves from 3d scattered data. Vis. Comput. **16**, 15–25 (2000)
25. Xu, H., Gossett, N., Chen, B.: Knowledge and heuristic-based modeling of laser-scanned trees. ACM Trans. Graph. **26**(4), 19 (2007)
26. Yan, D.M., Wintz, J., Mourrain, B., Wang, W., Boudon, F., Godin, C.: Efficient and robust branch model reconstruction from laser scanned points. In: Proceedings 11th IEEE International Conference on Computer-Aided Design and Computer Graphics (2009)
27. Zhou, Y., Kaufman, A., Toga, A.W.: Three-dimensional skeleton and centerline generation based on an approximate minimum distance field. Vis. Comput. **14**(7), 303–314 (1998)

Alexander Bucksch obtained his bachelor's and master's degree in Media and Information Technology at the Brandenburg University of Technology, Germany. Currently, he is a Ph.D. at the Remote Sensing Department at Delft University of Technology. He started working on point clouds obtained by terrestrial laser scanning already during his bachelor thesis. His main research interest is aiming on the development of new point cloud processing methodology. Notably, his current work founds its application in tree analysis for forestry and ecology.

Roderik Lindenbergh studied Mathematics at the University of Amsterdam, the Netherlands. He obtained a Ph.D. in Mathematics from the University of Utrecht, the Netherlands, for his work on limits of Voronoi diagrams. After his Ph.D., he joined the Remote Sensing Department of Delft University of Technology where he is currently employed as an assistant professor. His research interests are focused on those applications of notably laser range data where the signal to noise ratio is critical.

Massimo Menenti holds the Chair of Optical and Laser Remote Sensing. He has expertise in hydrology, water management and land surface processes, and remote sensing. He is coordinating the EU FP7 CEOP–AEGIS project aiming at establishing a prototype observation system of water resources on the Qinghai–Tibetan Plateau and major river basins in Southeast Asia. He has been the lead scientist of the ASI–CSA Joint Hyperspectral Mission and of the ESA SPECTRA and LSPIM missions. He has coordinated an EU research project—Hydrological Determinants of Agricultural Production in South America and an EU network dealing with Climate Impact on Water and Dry lands Agriculture. Past investigations include studies of groundwater hydrology in the deserts of Libya and Egypt, the use of advanced Earth Observation sensors systems to improve the performance of atmospheric models and irrigation water management in several countries. His experience with time series analyses of both ground and satellite measurements dates back to 1973 with studies on precipitation, evaporation, and vegetation.