Inverse Problems and Imaging
Volume X, No. 0X, 200X, X–XX

Web site: http://www.aimSciences.org

EUCLIDEAN SKELETONS USING CLOSEST POINTS

Songting Luo

Department of Mathematics, University of California, Irvine
Irvine, CA 92697-3875, USA

Leonidas J. Guibas

Computer Science Department, Stanford University
Stanford, CA 94305, USA

Hong-Kai Zhao

Department of Mathematics, University of California, Irvine
Irvine, CA 92697-3875, USA

(Communicated by the associate editor name)

Abstract. In this paper, we present an efficient algorithm for computing the Euclidean skeleton of an object directly from a point cloud representation on an underlying grid. The key point of this algorithm is to identify those grid points that are (approximately) on the skeleton using the closest point information of a grid point and its neighbors. The three main ingredients of the algorithm are: (1) computing closest point information efficiently on a grid, (2) identifying possible skeletal points based on the number of closest points of a grid point and its neighbors with smaller distances, (3) applying a distance ordered homotopic thinning process to remove the non-skeletal points while preserving the end points or the edge points of the skeleton. Computational examples in 2D and 3D are presented.

1. Introduction

The skeleton is a compact representation of an object. It has been used in object representation, identification and recognition. The interest of the skeleton as a representation for an object rises from several nice properties: [23, 31] (1) it is homotopic to the original shape, (2) it is invariant under Euclidean transformations, and (3) the object can be reconstructed based on the distance associated with each skeletal point on the skeleton.

Several definitions of the skeleton of an object have been proposed in previous work. For examples,

1. the skeleton consists of the points which have at least two closest points from the boundary of this object [1].
2. the skeleton consists of the centers of all the interior maximal circles (2D) or maximal spheres (3D) [10].

2000 *Mathematics Subject Classification:* Primary: 94A08, 68U10; Secondary: 62H35.

Key words and phrases: Euclidean skeletons, closest points, Eikonal equation, distance map, discrete Voronoi diagrams.

The first author is partially supported by NSF grant DMS-0513073. The third author is partially supported by NSF grant DMS-0513073, ONR grant N00014-02-1-0090 and ARO/MURI grant W911NF-07-1-0185.

3. the skeleton consists of the points where different firefronts (from the boundary) intersect [8].
4. the skeleton is the loci of the singularities of the distance transform [8].

A lot of work has been done to compute the skeleton based on different approaches. Some approaches are based on a discrete representation of the geometry, i.e., point clouds. The Voronoi diagram based methods [26, 28, 29, 30] and the Delaunay triangulation based methods [15, 25] are typically used in this situation. These methods usually preserve the homotopic property pretty well and locate the skeletal points accurately when the data set is dense enough. However, in practice, the results are not invariant under Euclidean transformations, and the computational complexity can be relatively high. Other approaches are based on a continuous representation of the geometry, i.e., curves or surfaces, and the skeleton is defined in terms of the singularities of the distance map one way or the other. For instance, methods based on the Blum's grassfire formulation [3, 9, 19, 24] attempt to simulate the grassfire transform, peeling off layers from an object while keeping the potential skeletal points, but they fail to localize the skeletal points accurately. Methods based on the internal flux or the singularity [4, 16, 21, 20] attempt to detect the local maxima or the corresponding discontinuous derivative of the distance map. The Hamilton-Jacobi skeleton [31, 33] uses the limiting property of the average outward flux of the vector field or the momentum field corresponding to the underlying Hamiltonian system, and combines it with a flux-ordered homotopic thinning process by thresholding the flux. The Euclidean skeleton [23] uses two measures to characterize the singularities and combines them with a topological reconstruction. In these methods the skeleton is defined as the singularities of the distance map to the boundary or similar formulations. Typically, the distance map is computed on a grid and the singularities of the distance map have to be detected through the gradient of the computed distance map. Although the singularities of the distance map are well defined theoretically as the discontinuities of the gradient, it is quite subtle to numerically detect the singularities from the computed distance map and its gradient. In practice, this makes it difficult if impossible to develop numerical procedures for both accurate and robust detection of the singularities. Different formulations are proposed in the above referenced works and the references therein to address this difficulty. Algorithms based on the potential field, the repulsive force field or the gradient vector flow for extracting curve-skeletons were proposed in [11, 12, 17] and references therein.

In this work we develop an algorithm for computing the Euclidean skeleton directly from point clouds on a rectangular grid. Here is a brief summary of our motivations and some key ideas:

- Point cloud representations have become more and more popular in real applications. Our method deals with discrete data sets directly to avoid an ill-posed reconstruction of a continuous representation, which can be expensive, difficult and can introduce artifacts.
- The skeleton is computed on an underlying grid. The data structure and algorithm are very simple.
- The detection of the skeletal points is based on the closest point information of a grid point and its neighbors with smaller distances. No gradient information of the distance map is used. An efficient algorithm based on the fast sweeping method [35, 36] is available to compute the distance and the closest point information on a grid.

- A distance ordered homotopic thinning process [27] is developed. The information of the closest points is used in removing the non-skeletal points while preserving the end or the edge points of the skeleton.

Another direct application of our method is to compute the skeletons for shapes obtained from image segmentation. A binary segmentation, i.e., a classification of all pixels into interior or exterior points, can be used. Our algorithm can take either interior boundary pixels or exterior boundary pixels as the discrete data points and construct the skeleton based on the pixels. Examples are shown in Section 4.

Here is the outline of this paper. In section 2 we present the fast sweeping algorithm for computing the distance and the closest point information on a rectangular grid. In section 3 we present our algorithm with some motivations and explanations. In section 4 we present computational examples.

Before continuing on, we recall some definitions and notations from digital topology [34, 22, 18, 1] on a rectangular grid with grid size h. Given a set O with cardinality denoted as $\#O$,

- Neighborhood and connectivity in 2D: two points p, q are 4-*neighbors* or 4-*adjacent* if $dist(p, q) \leq h$ (8-*neighbors* or 8-*adjacent* if $dist(p, q) \leq \sqrt{2}h$).
- Neighborhood and connectivity in 3D: two points p, q are 6-*neighbors* or 6-*adjacent* if $dist(p, q) \leq h$ (18-*neighbors* or 18-*adjacent* if $dist(p, q) \leq \sqrt{2}h$, 26-*neighbors* or 26-*adjacent* if $dist(p, q) \leq \sqrt{3}h$).
- For $n = 4, 8$ (2D) or $6, 18, 26$ (3D), an n-*path* is a sequence of points $p_1, p_2, \ldots, p_m$ that p_{k-1} and p_k are n-neighbors for $2 \leq k \leq m$. Two points are connected in O if there exists a path in O connecting these two points, which also defines an equivalent relation in O. Each equivalent class is an n-*connected component*.
- In digital topology, a point p of an object is simple if its removal does't change the topology of this object.

2. The closest point solver

In this section, we present an efficient algorithm [13, 35, 36] for computing the closest points and the distance map to a data set on a rectangular grid. Our algorithm for computing the skeleton from point clouds relies on the closest point information and the distance on the grid (see Section 3).

Let Γ be a set of discrete points that represents the boundary of an object Ω. For a point $p \in \Omega$, a point $p^\star \in \Gamma$ is closest to p if

$$dist(p, p^\star) = \min_{q \in \Gamma} dist(p, q)$$

$p^\star$ may not be unique if p is a skeletal point. Once such a closest point is found, the distance $u(p)$ can be computed easily as

$$u(p) = dist(p, \Gamma) = dist(p, p^\star)$$

Theoretically, the distance map can also be defined as the viscosity solution of the Eikonal equation:

$$|\nabla u| = 1, \ u(p) = 0, \ p \in \Gamma. \tag{1}$$

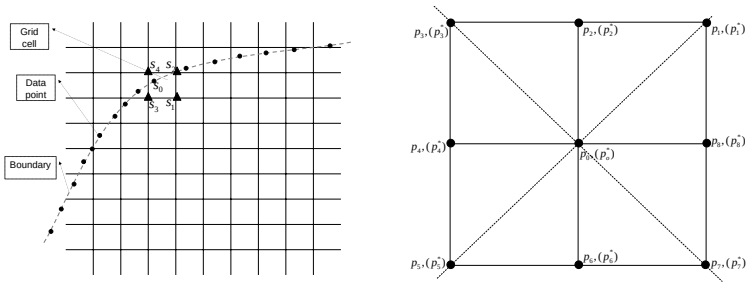
For the above nonlinear hyperbolic partial differential equation (PDE), information (e.g. boundary conditions and closest points) propagates through the characteristics that are defined through the characteristic system for the Eikonal equation (1) [14], and the distance value increases along the characteristics from the boundary.

In a discrete setting, the fast sweeping algorithm utilizes this observation to compute the viscosity solution and the closest points. All propagation directions can be divided into a few groups which can be followed efficiently by a few alternating orderings. Here is the algorithm [13, 35, 36]:

Algorithm (the closest point solver)

1. Initialization:
   Put each data point into a grid cell. Assign one closest point and the corresponding distance to each vertex of all the grid cells containing at least one data point.
2. Sweeping:
   Go over the whole grid and update the closest point and the distance at each point in the following way: the closest point information and the distance at a grid point is updated through the closest point information and the distance of its neighbors.
   Iterate this sweeping using different orderings until no further change is detected.

Here are more detailed explanations, referring to Figure 1. We give explanations in 2D, extension to 3D is straightforward. In the initialization step, for a grid cell



(a) Initialization (b) Sweeping

FIGURE 1. Fast sweeping method: initialization and sweeping

containing some data points, each vertex of this cell is assigned a closest point and the distance to this closest point. See Figure 1(a), we only needs to go through the list of all data points once in the following way: when going through the list, for a data point s_0 in a grid cell with vertices s_1, s_2, s_3 and s_4, each of the four vertices may have already been assigned a closest point, say $s_1^\star, s_2^\star, s_3^\star$ and $s_4^\star$ respectively. We determine for each vertex, whether $s_i^\star$ $(i = 1, 2, 3, 4)$ should be replaced by s_0 simply by calculating the distance to s_0 and choosing the minimum one compared to old distance that has already been assigned at this vertex. Hence the complexity of this step is proportional to the number of the data points.

In the sweeping step, see Figure 1(b), for a center grid point p_0 and all its neighbors $p_1, p_2, ...$ and p_n, each of them may have already been assigned a closest point $p_0^\star, p_1^\star, p_2^\star, ...$ and $p_n^\star$ respectively. In order to update the information at p_0, we calculate the values $u_i = dist(p_0, p_i^\star)$ for $i = 0, 1, 2, ..., n$, if $u_j = \min_{i=0,1,2,...,n} u_i$ for some $0 \le j \le n$, we update $p_0^\star = p_j^\star$ and $u(p_0) = u_j$.

For a rectangular grid, we use 8-neighbors in 2D and 26-neighbors in 3D to improve the accuracy. In 2D the sweeping follows the four natural orderings alternatingly: (1) $i = 1 : I, j = 1 : J$ (2) $i = I : 1, j = 1 : J$ (3) $i = I : 1, j = J : 1$ (4) $i = 1 : I, j = J : 1$. For a rectangular grid in 3D, there are eight different orderings. The sweeping will converge in a finite number of iterations [36]. Typically, 4 iterations in 2D and 8 iterations in 3D are enough. Hence the complexity of this step is proportional to the total number of the grid points.

A key idea of the above algorithm is to propagate the closest point relation, which is a global information with respect to the data set, through neighbors efficiently from those grid points near the data set to the whole grid by alternating sweepings. However, the above closest point solver may not be exact for two reasons:

- The data set is not well resolved by the underlying grid. For example, if there are more data points than the vertices in a grid cell, some data points will be missed during the initialization step.
- A finite number of neighboring grid points aligned with a finite number of the directions cannot cover all closest point configurations [13].

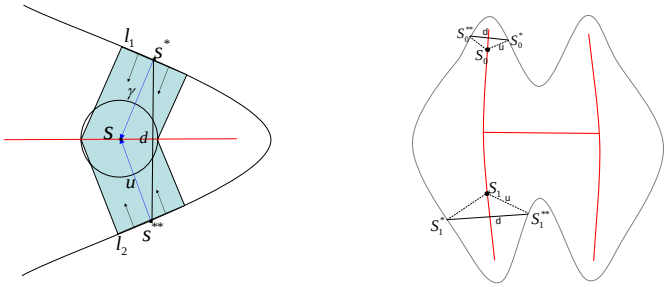
The error analysis of the computed closest point and distance value has been given in [35, 36].

With the information of the closest point and the distance value available for each grid point we will design criteria to select skeletal points based on this information in the following section.

3. Motivation and Algorithm

The starting point of our algorithm is: for a given point, by counting the number of distinct closest points of its neighbors with smaller distances and itself, one can detect the convergence of different characteristics and hence the possible skeletal points. This observation is directly related to the definition of the skeleton. The reason why we only choose neighbors with smaller distances is because of the causality of the distance map observed through the characteristics, i.e., information propagates from points with smaller distances to those with larger distances along the characteristics.

However, since the shape is described by discrete points, we need to distinguish the spurious skeletal points, which are of equal distance to neighboring data points, from the real skeletal points. We use an example in 2D for the illustration purpose. By the definition, each skeletal point has at least two closest points from the boundary. For example, in Figure 2(a), s is a skeletal point. Two of its closest points are $s^\star$ and $s^{\star\star}$. The characteristics emanating from $s^\star$ and $s^{\star\star}$ meet at s. In other words, any point on these two characteristics also has $s^\star$ or $s^{\star\star}$ as one of its closest points respectively. This information can be passed along the characteristics from the boundary to the interior. One natural criterion for a true skeletal point is that the separation distance between $s^\star$ and $s^{\star\star}$ should not be too small, e.g., larger than the distance between two neighboring data points. This partially describes a skeletal point. But the separation distance is not a dimensionless quantity, i.e., it is not scale invariant. For example, Figure 2(b) shows that it may not work well. The separation distances are very different at points s_0 and s_1. In practice, only the separation distance is not enough to distinguish the skeletal and non-skeletal points. We also apply the ratio $\frac{d/2}{u}$ to classify skeletal points, where d is the largest distance between any two closest points and u is the distance at s. This ratio is an



(a) (b)

FIGURE 2. Skeletal point

approximation to the so-called bisector function in [23, 6, 5]. As in Figure 2(b), the ratio at s_0 is comparable to that at s_1. The above argument also implies that the algorithm can detect a possible skeletal point that has more than two closest points as long as we can detect at least two of them that are well separated compared to the grid size. Hence our closest point solver, which may not be exact, should be a good approximation for this purpose.

After we apply the closest point solver, each point will be assigned a closest point and a distance value. Then we define the Neighbor-Closest Point (NCP).

Definition 3.1 (Neighbor-Closest Point (NCP))**.** For each grid point s, the set of the closest points of its n-neighbors with smaller distances and itself is defined as the neighbor-closest points (NCP) of s, denoted as $NCP(s,n)$, where $n = 4, 8$ $(2D)$ or $6, 18, 26$ $(3D)$.

Then for each grid point s, we define d by

$$d = \max_{s^\star, s^{\star\star} \in NCP(s,n)} dist(s^\star, s^{\star\star})$$

If $\#NCP = 1$, $d = 0$. In this work, we use $n = 4$ in 2D and $n = 6$ in 3D. At each point, the computational cost for NCP and d is at most $O(\frac{k(k-1)}{2})$ ($k \leq 5$ in 2D, $k \leq 7$ in 3D).

As discussed above, at a skeletal point s, $\#NCP(s) \geq 2$. Therefore the number, $\#NCP$, can be used to detect possible skeletal points. All the points with $\#NCP \geq 2$ will be located on or near the Voronoi boundary of Γ. Therefore, by identifying all points with $\#NCP \geq 2$, we locate the real skeletal points as well as Voronoi boundaries of Γ in a discrete sense. In 2D, the set of all Voronoi vertices is a good approximation of the skeleton when the data set is dense [1, 2]. Referring to Figure 3, if we remove all secondary branches (shown in black), the rest of the Voronoi boundary is a well-defined skeleton (principal branches shown in red). In 3D, not all Voronoi vertices contribute to the approximation of the skeleton, instead, only a subset of the Voronoi vertices gives an approximation of the skeleton [1, 2].

In a discrete setting, simply locating all points with $\#NCP \geq 2$ results in a thick set containing spurious branches, holes and cavities. So in order to avoid getting holes or cavities by locating points with $\#NCP \geq 2$, we apply a distance ordered homotopic thinning process [27] and design rules to remove non-skeletal points and preserve skeletal points.

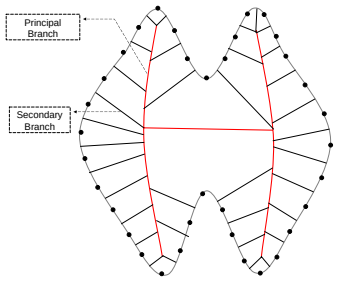


FIGURE 3. The "Voronoi diagram" of a data set

3.1. 2D SKELETON. In 2D, [34, 22, 31] have a clear description of a simple point and an end point in digital topology.

Proposition 1. *A point p is simple if its removal doesn't disconnect the object or create a hole.*

Definition 3.2 (2D endpoints)**.** A point p on the skeleton is an end point if and only if it has only one 8-neighbor on the skeleton or it has two 8-neighbors on the skeleton, both of which are 4-neighbors to each other.

Figure 4 shows a few important observations after we apply the closest point solver for a data set on a rectangular grid: (a) shows the "Voronoi diagram" after locating all points with $\#NCP \geq 2$, which has holes (enlarged in (b)), and the $\#NCP$ at each grid point. We see that points on secondary branches as well as points on the skeleton may have $\#NCP \geq 2$ due to discrete nature of the data set. So $\#NCP \geq 2$ is not a sufficient criterion for skeletal points. (c) shows the separation distance d at each point. The separation distance d of a point on a secondary branch is much smaller than that of a point on a principal branch, which depends on the sampling of the data points. And (d) shows the ratio $\frac{d/2}{u}$ at each grid point. In particular we have (1) each point in the "interior" of a Voronoi cell has $\#NCP = 1$. (2) each Voronoi vertex or end point of the skeleton has $\#NCP \geq 3$ and relatively high ratio $\frac{d/2}{u}$, (3) each point on a principal branch has $\#NCP \geq 2$ and the ratio $\frac{d/2}{u}$ is relatively high and pretty uniform, (4) most points on a secondary branch have $\#NCP \leq 2$ and the separation distance d ($\sim$ the separation distance of neighboring data points) or the ratio $\frac{d/2}{u}$ is small. More subtle points are those close to a Voronoi vertex (Figure 4(a)). Due to converging characteristics, a fixed resolution of the underlying grid and the property of the closest point solver on a discrete setting, some of these points may have $\#NCP \geq 3$ and relatively large separation distance d as well as large ratio $\frac{d/2}{u}$. Our strategy is to efficiently locate the Voronoi boundary, and remove secondary branches while retaining principal branches by preserving Voronoi vertices when they become end points. For this purpose we construct a distance ordered homotopic thinning process [27] which removes the non-skeletal points while preserving the end points of the skeleton and hence the skeleton. The thinning process consists of two steps. The first step is to remove simple points that are in the "interior" of Voronoi cells ($\#NCP = 1$), and simple points on or near secondary branches with either $\#NCP \leq 2$, small d or small $\frac{d/2}{u}$. The remaining set consists of points on or near the skeleton, i.e. a little

bit thicker set containing the skeleton. So the next step is a post-process to further remove non-skeletal points. Here is our 2D algorithm with explanations.

2D Algorithm

step 1 All the points are ordered according to the distance. This set is denoted as S.

step 2 According to the ascendent ordering, go over all the points in S. For each point p in S, if p is simple:

- if p is an end point:
  - if $\#NCP \leq 2$, then $S = S\backslash\{p\}$.
  - or if $\frac{d/2}{u} < threshold1$, then $S = S\backslash\{p\}$.
  - or if $d < threshold2$, then $S = S\backslash\{p\}$.
- else,
  - if $\#NCP \leq 2$, then $S = S\backslash\{p\}$.
  - or if $\frac{d/2}{u} < threshold1$, then $S = S\backslash\{p\}$.
  - or if $d < threshold2$, then $S = S\backslash\{p\}$.
- repeat the process until no further points are removed. Then move to next step.

step 3 According to the ascendent ordering, go over all the points in S. For each point p in S, if p is simple:

- if p is an end point:
  - if $\#NCP \leq 2$, then $S = S\backslash\{p\}$.
  - or if $\frac{d/2}{u} < threshold1$, then $S = S\backslash\{p\}$.
  - or if $d < threshold2$, then $S = S\backslash\{p\}$.
- else, $S = S\backslash\{p\}$.
- repeat the process until no further points are removed.

The complexity of the ordering in step 1 is $O(n \log n)$, with n the number of grid points. The complexity of step 2 and 3 is $O(\frac{k(k-1)}{2}n')$ with $n' = \#S, k \leq 5$. $\frac{k(k-1)}{2}$ comes from the computation of $\#NCP$ and the maximum separation distance d among NCP. In step 2 and 3, the removal of an end point depends on its $\#NCP$, $\frac{d/2}{u}$ and d. The set of Voronoi vertices is a good approximation of the skeleton if the data set is dense [1, 2]. Theoretically, an end point has at least three closest points, and a Voronoi vertex has at least three closest points too. Therefore, when an end point has $\#NCP \geq 3$, relatively large $\frac{d/2}{u}$ and d, it is a possible skeletal end point. The difference between step 2 and 3 is how to deal with simple but not end points. In step 3, if a point is a simple but not an end point, it is removed immediately. But in step 2, the removal of such a point is more cautious. It also depends on its $\#NCP$, $\frac{d/2}{u}$ and d. When a point has $\#NCP \geq 3$, relatively large $\frac{d/2}{u}$ and d, it is very likely to be a skeletal point. In particular, if it is an end point, removal of it may cause damage of the whole skeleton in successive thinning process. For example, see Figure 5, p and q are two possible skeletal points, $u(p) \leq u(q)$, but $\#NCP(p) \geq 3$, $\#NCP(q) \leq 2$. The thinning process reaches p first since it is distance-ordered. p is a simple point but not an end point. It should be removed immediately if without checking its $\#NCP$, $\frac{d/2}{u}$ and d. After p is removed, q is removed because $\#NCP(q) \leq 2$, then the whole branch may be removed by $\#NCP \leq 2$. The purpose of step 2 is to remove q but retain p, thus the whole branch is preserved. The resulting set of step 2 may still have simple but not end points left, so step 3 serves as a post-process of step 2 to remove such points (e.g. Example 1, Figure 7 in section 4). Step 2 and 3 may be repeated.

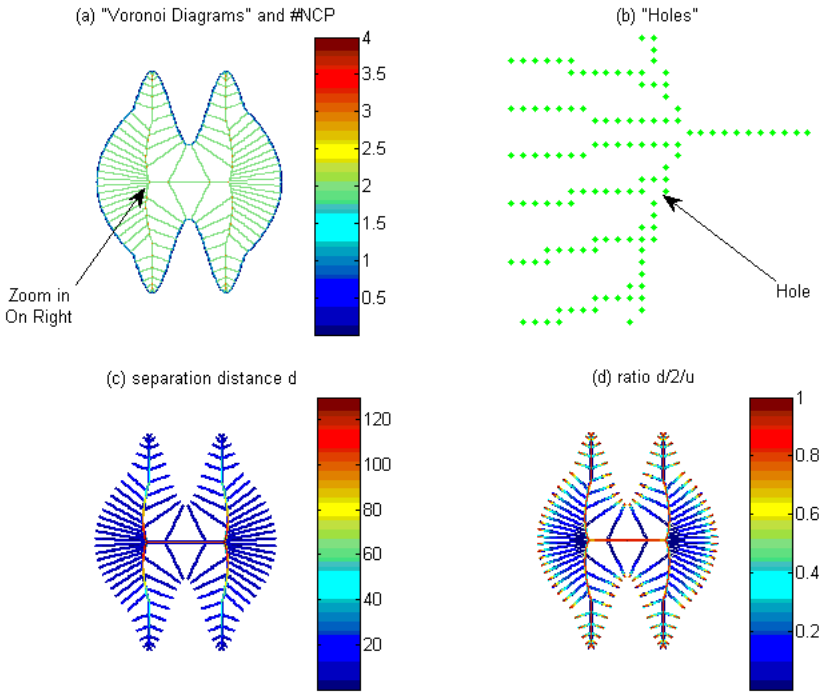


FIGURE 4. The "Voronoi diagram" of D

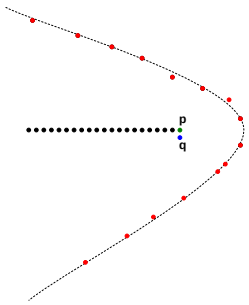


FIGURE 5. An example to illustrate the purpose of step 2.

Remark 1. One important issue of our grid based skeleton construction for point clouds is the grid size, which affects the resolution, accuracy and complexity of the algorithm. To construct the skeleton for a uniformly distributed data set with best possible resolution and accuracy, the grid size should be totally determined by the sampling density of the data set, i.e, the grid size should be comparable to separation distance between neighboring points. However, an advantage of our approach is that coarser grid may be used for faster construction of a low resolution skeleton approximation. If the data is non-uniform the grid size should be adaptive to the local sampling density ideally, which is not done in this paper. For applications to image segmentation, a natural choice is the underlying image grid/pixel.

Remark 2. Our measures $\frac{d/2}{u}$ and d are similar to those used in [23, 6, 5]. In general the thresholds for $\frac{d/2}{u}$ and d depend on sampling density, feature size and noise level. For example, d should be comparable to the separation distance between neighboring data points or feature/perturbation size we would like to filter out from the data set. $\frac{d/2}{u}$ is an approximation of the bisector angle. An ideal choice should be adaptive to local sampling density, feature size and noise level. If the data contains noise, these thresholds can be used also as low pass filter and should depend on noise level. For example, small features as well as noise will be smeared if we choose large threshold for d, i.e., small shocks due to small perturbations will be absorbed into large shocks a little bit further away. Computational examples are presented in section 4 to show the results with different thresholds.

Remark 3. In 2D, $\#NCP$ is very useful for the removal of most points on or near secondary branches of the Voronoi diagrams. In particular, when data points are non-uniform, the separation distance criteria d varies a lot and $\frac{d/2}{u}$ is always large for points on secondary branches that are close to the boundary.

3.1.1. *3D Skeleton.* In 3D, a simple point p of an object O can be classified by two numbers in digital topology [22]: (1) $C^\star$: the number of 26-connected components 26-adjacent to p in $O \cap N_{26}^\star(p)$ and (2) $\bar{C}$: the number of 6-connected components 6-adjacent to p in $O^c \cap N_{18}(p)$. Where $N_n(p)$ denotes the n-neighbors of p, and $N_n^\star(p) = N_n(p) \backslash \{p\}$.

Proposition 2. *A point p is simple if $C^\star = 1$ and $\bar{C} = 1$.*

The characterization of the end and the edge points in 3D is more complicated than that in 2D. In [27], criteria are designed to detect the end points of a curve and the edge points of a surface: a point is an end point of a curve if it has less than two 26-neighbors, and a point is an edge point of a surface if it is an end point of a curve on any of the 9 digital planes shown in Figure 6. For details please refer to [27] and references therein.

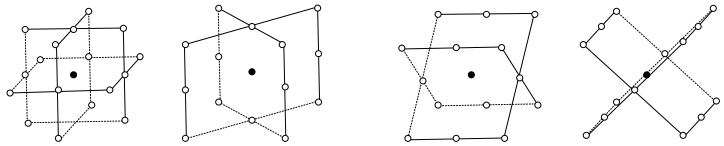


FIGURE 6. 9 planes

The following is our 3D algorithm. The idea and strategy are similar to those in the 2D algorithm: remove the non-skeletal points but preserve the skeletal points by keeping the skeletal end and edge points during the thinning process.

3D Algorithm

step 1 All the points are ordered according to the distance. This set is denoted as S.

step 2 According to the ascendent ordering, go over all the points in S. For each point p in S, if p is simple:

- if p is an end point or has 2 26-neighbors in S:

  - if $\#NCP \leq 4$, then $S = S\backslash\{p\}$.
  - or if $\frac{d/2}{u} < threshold1$, then $S = S\backslash\{p\}$.
  - or if $d < threshold2$, then $S = S\backslash\{p\}$.
- else if p is an edge point,
  - if $\frac{d/2}{u} < threshold3$, then $S = S\backslash\{p\}$.
  - or if $d < threshold4$, then $S = S\backslash\{p\}$.
- else,
  - if $\frac{d/2}{u} < threshold3$, then $S = S\backslash\{p\}$.
  - or if $d < threshold4$, then $S = S\backslash\{p\}$.
- repeat the process until no further points are removed. Then move to next step.

step 3 According to the ascendent ordering, go over all the points in S. For each point p in S, if p is simple:

- if p is an end point or has 2 26-neighbors in S:
  - if $\#NCP \leq 4$, then $S = S\backslash\{p\}$.
  - or if $\frac{d/2}{u} < threshold1$, then $S = S\backslash\{p\}$.
  - or if $d < shreshold2$, then $S = S\backslash\{p\}$.
- else if p is an edge point,
  - if $\frac{d/2}{u} < threshold3$, then $S = S\backslash\{p\}$.
  - or if $d < shreshold4$, then $S = S\backslash\{p\}$.
- else, $S = S\backslash\{p\}$.
- repeat the process until no further points are removed.

The 3D algorithm is an extension of the 2D algorithm. In 3D, the classification of simple points including end and edge points is more complicated. Moreover, only a subset of the Voronoi vertices contributes to the skeleton [1, 2]. Theoretically, an edge point of the skeleton has at least three closest points and an end point of the skeleton has at least five closest points. However, some Voronoi vertices that are not on the skeleton may have the same properties. So, the way to distinguish such points depends on $\frac{d/2}{u}$ and d. In step 2 or 3, the removal of an end point depends on its $\#NCP$, $\frac{d/2}{u}$ and d. The removal of an edge points depends on $\frac{d/2}{u}$ and d. The only difference between step 2 and 3 is how to deal with simple points that are not end or edge points. In step 3, a simple point is removed immediately if it is neither an end nor an edge point. In step 2, a more cautious procedure is taken. The removal of such a simple point depends also on $\frac{d/2}{u}$ and d for reasons similar to 2D case. Once the main structure of the skeleton has been preserved, step 3 serves as a post-process of step 2. Step 2 and 3 have complexity $O(\frac{k(k-1)}{2}n')$ with $n' = \#S, k \leq 7$. More detailed thresholds are put on the end and the edge points according to $\#NCP$. Both step 2 and 3 may be repeated. Computational examples are present in section 4 to show the results for each step, with CPU time for different thresholds.

Remark 4. On a surface, a point with three 26-neighbors lies on the corner of this surface. And a point with more than three 26-neighbors lies on the rim of the surface.

Remark 5. Since only a subset of the Voronoi vertices contributes to the approximation of the skeleton in 3D, $\#NCP$ doesn't serve as a strong measure as in 2D to distinguish skeletal and non-skeletal points anymore, especially for edge points.

4. Numerical Examples

In this section, we use a few examples to demonstrate our algorithm[1]. Both clean data and noisy data are used. For clean data we give points $\{p_i\}$ on the real boundary of an object. For noisy data we use perturbed points, $\{q_i\} = p_i + e_i$ where e_i is some random perturbation. We show both computed skeleton and reconstruction from the skeleton, i.e., the union of disks (2D) or balls (3D). Reconstruction from noisy data shows some denoising effect which is due to thresholding, e.g., the threshold for the separation distance. Programs are coded with VC++ 6.0, MS Windows XP, Intel(R)Core(TM)2CPU T5500 @ 1.66GHz 980MHz, 1.00GB of RAM.

4.1. 2D examples.

A few 2D examples are presented, including two shapes from image segmentation.

Example 1: Figure 7 shows a synthetic example with clean data points. The object is represented by 258 points. The computation is performed on a 100×100 grid. From the figure, we see that step 3 serves as a post-process of step 2.

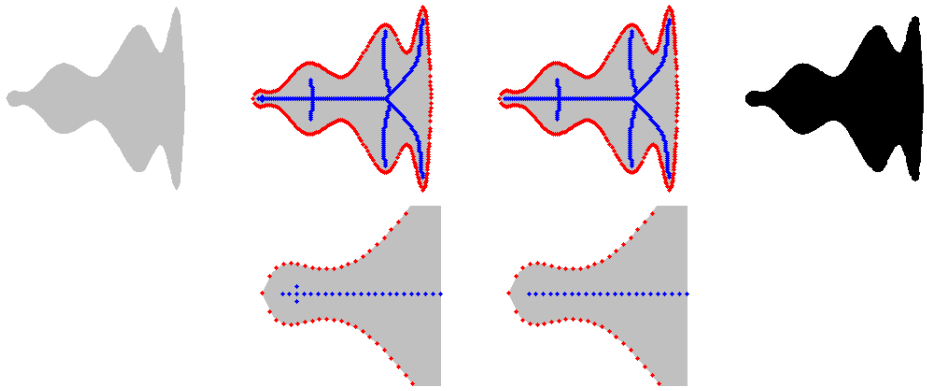


Figure 7. Example 1. Clean data. Top from left to right: the object, result after step 2, result after step 3 and reconstruction. Bottom: zoom in. $(threshold1, threshold2) = (0.65, 4.4h)$.

Figure 8 shows the same example with perturbations on the clean data, that is, the noisy data points $(x, y) = (x_0, y_0(1 + 0.05 * rand\{-1, 1\}))$ with (x_0, y_0) the clean data points, where $rand\{-1, 1\}$ represents random uniform distribution on $[-1\ 1]$. Different thresholds are chosen to show the comparisons. The closest point solver and the 2D skeleton algorithm take CPU time much less than one second.

Example 2: Another example with both clean and noisy data points is presented. The object is represented by 200 points. The computation is performed on a 200×200 grid. Figure 9 shows the results. First we use clean data (left), then we add noise to this data set with above method. Two different sets of noisy data points are used (middle and right). We show the computed skeletons (top row) and the corresponding reconstructions (bottom row).

[1]The sorting algorithm used in our examples is quicksort algorithm.

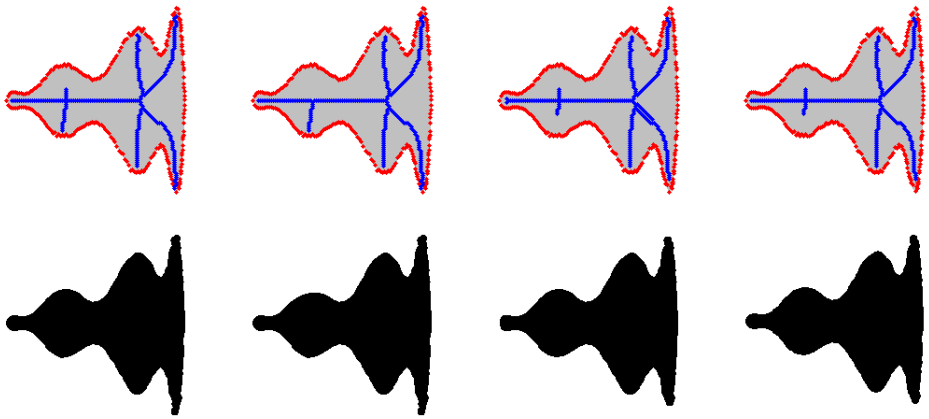


FIGURE 8. Example 1. Noisy data. Top: data points and the computed skeleton. Bottom: reconstruction. From left to right: $(threshold1, threshold2) = (0.75, 3.5h), (0.85, 3.5h), (0.75, 4.1h), (0.85, 4.1h)$.

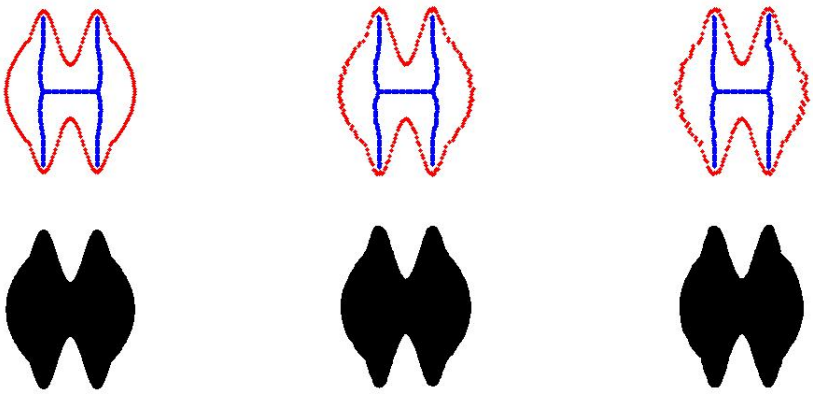


FIGURE 9. Example 2. From left to right: clean data, noisy data and noisy data; $(threshold1, threshold2) = (0.8, 8h), (0.8, 8.8h)$ and $(0.8, 8.8h)$; Top row: computed skeleton, bottom row: reconstruction

Example 3: In this example, we use two shapes from image segmentation to illustrate our 2D algorithm. Figure 10 and 11 show the results. The image segmentations[2] are obtained by Chan-Vese active contour model. For our method we don't need an explicit representation of the contour. Instead we only use a binary classification of pixels based on any segmentation method and use those exterior (or interior) boundary pixels as our discrete data points. The computation is performed on sub-images that consist of 500×500 pixels and 300×300 pixels respectively. For

[2]By courtesy of Dr. Shingyu Leung, University of California, Irvine.

the two images, both the closest point solver and the 2D skeleton algorithm take CPU time much less one second.

Note that the skeleton for the horse in Figure 10 has feet connected due to shadows in the original image and the particular image segmentation. On the other hand, some thinner shadows are removed by the thresholds.

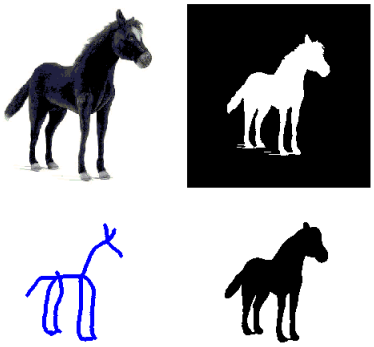


FIGURE 10. Example 3. (horse) From left to right, top to bottom: original image, image segmentation, skeleton and reconstruction. $(threshold1, threshold2) = (0.7, 5.9h)$.

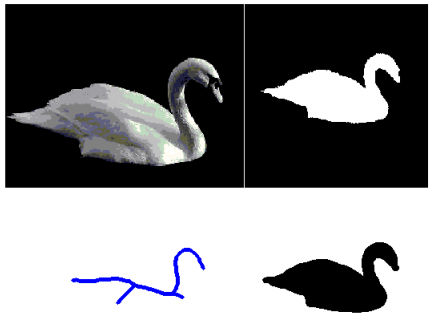


FIGURE 11. Example 3. (swan) From left to right, top to bottom: original image, image segmentation, skeleton and reconstruction. $(threshold1, threshold2) = (0.7, 4h)$.

4.2. 3D EXAMPLES. Here we present a few 3D examples.

Example 1: Figure 12 shows an example with clean data (7510 points). Figure 15 shows the same example with noisy data (7510 points). Different thresholds are chosen for comparisons. CPU times for both the closest point solver and the 3D skeleton algorithm are presented. The number of skeletal points left after each step is recorded. Table 1 shows the corresponding choices of different thresholds, number of points after step 2 and 3, CPU times (seconds) for closest point solver and 3D skeleton solver. The computation is performed on a $100 \times 100 \times 100$ grid.

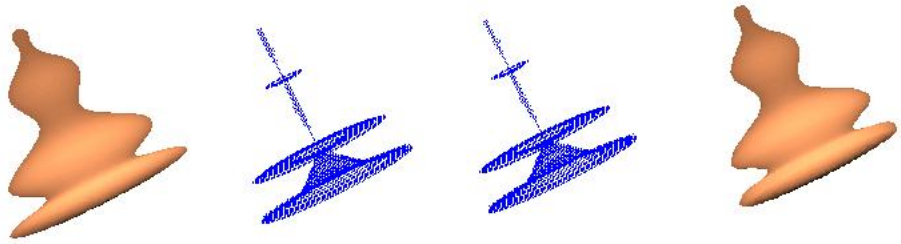


FIGURE 12. Form left to right: the object, skeletal points after step 2 and step 3, and reconstruction. $(threshold1, threshold2, threshold3, threshold4) = (0.9, 5h, 0.65, 4.6h)$. points of step 2 and 3= $(11027, 10673)$. CPU Time for the closest point solver and skeleton algorithm= $(30, 10)$ seconds.

Example 2: In this example, we show another object represented by 6450 scanned data points. Different thresholds are chosen to compute the skeletons. Figure 13 shows the computed skeletons and corresponding reconstructions. The computation is performed on a $23 \times 83 \times 257$ grid.

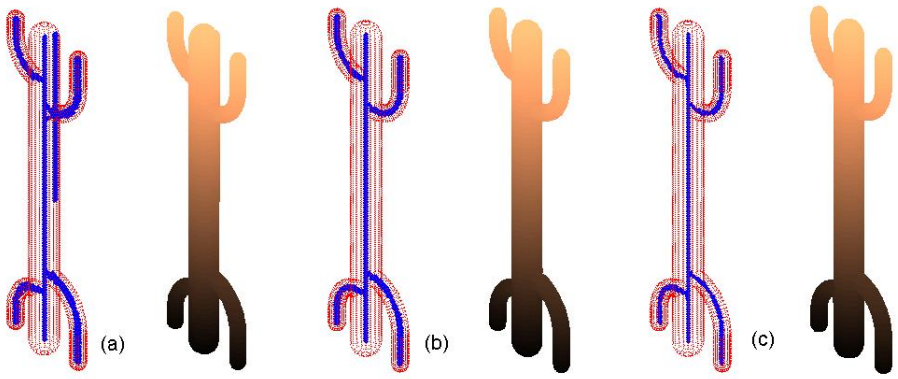


FIGURE 13. Computed skeletons and reconstructions. From left to right: $(threshold1, threshold2, threshold3, threshold4)$ are (a) $(0.7, 4h, 0.85, 5h)$, (b) $(0.7, 4h, 0.85, 6h)$ and (c) $(0.7, 4h, 0.95, 6h)$.

Example 3: Figure 14 shows a computational example with both clean and noisy data sets (9070 points). The computation is performed on a $100 \times 100 \times 100$ grid.

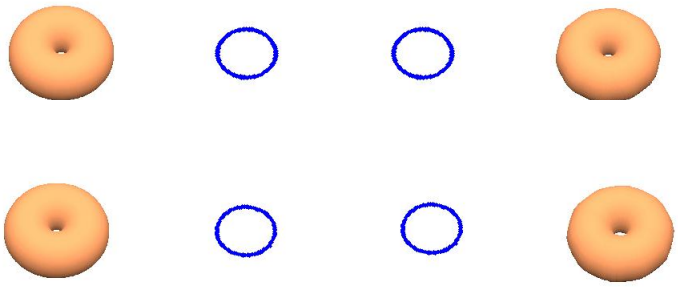


FIGURE 14. From left to right: the object, step 2, step 3 and reconstruction. Top: clean data; Bottom: noisy data.

Data of Figure 15							
*threshold*1	*threshold*2	*threshold*3	*threshold*4	Step 2	Step 3	CPU time-closest point	CPU time-skeleton
0.8	$5h$	0.65	$4.5h$	11290	10880	29	12
0.8	$5.5h$	0.65	$4.5h$	11286	10876	29	12
0.8	$5.5h$	0.8	$4.5h$	10451	10320	29	11
0.8	$5.5h$	0.65	$5h$	10055	9654	29	10
0.8	$5.5h$	0.7	$5h$	9914	9624	29	10
0.8	$5.5h$	0.8	$5h$	9405	9271	29	10

TABLE 1. Data of Figure 15: choices of different thresholds, points after step 2 and 3, CPU times (seconds) for closest point solver and 3D skeleton algorithm.

Remark 6. From Figure 8 and 15 with noisy data points, smaller thresholds capture fine feature and give more detailed skeletons and reconstructions, while larger thresholds give more simplified skeletons and reconstructions and more robust to noise. The choice of thresholds depends on the problems such as the sampling density and noise level.

5. CONCLUSION

An efficient algorithm is proposed for computing the Euclidean skeleton of an object represented by point clouds on an underlying rectangular grid. The algorithm is based on a distance ordered homotopic thinning process. Information of the closest points is used to determine possible skeletal points and design rules to remove non-skeletal points but preserve skeletal points, especially the end or the edge points. The closest point and the distance are computed with the fast sweeping method. Numerical examples show that number of closest points can be used as a criterion to remove non-skeletal points, especially in 2D. The overall complexity is $O(m + n \log n)$, with m the number of the data points and n the number of the grid points.

ACKNOWLEDGEMENTS

We would like to thank the referees very much for their valuable comments and suggestions.

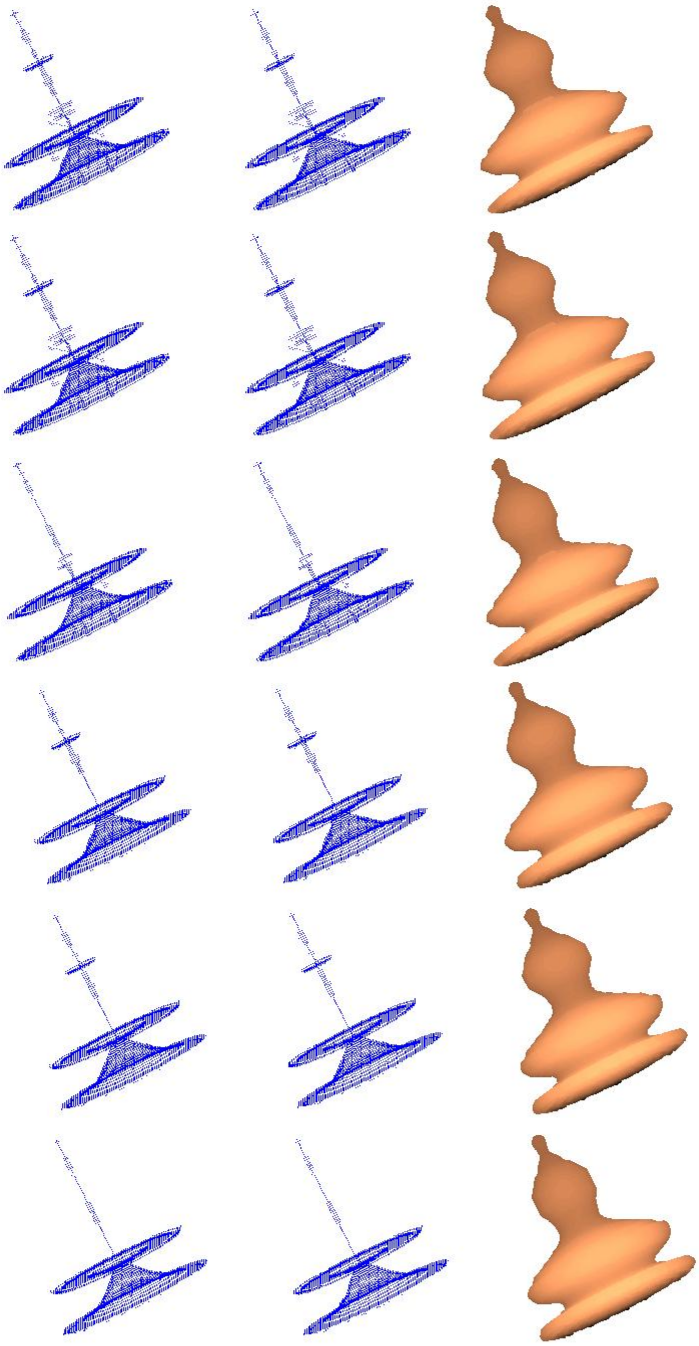


FIGURE 15. Form left to right: skeletal points after step 2 and step 3, and reconstruction. From top to bottom, different thresholds as shown in Table 1.

References

[1] N. Amenta, M. Bern and D. Eppstein, *T*he crust and the β-skeleton: combinatorial curve reconstruction, Graphic Models and Image Processing, (1998), 125-135.

[2] N. Amenta, M. Bern and M. Kamvysselis, *A* new Voronoi-based surface reconstruction algorithm, Computer Graphic (Proceedings of SIGGRAPH 98), (1998), 415-421.
[3] C. Arcelli and G. S. di Baja, *A* width-independent fast thinning algorithm, IEEE Transactions on Pattern Alalysis and Machine Intelligence, **7(4)** (1995), 464-474.
[4] C. Arcelli and G. S. di Baja, *R*idge Points In Euclidean Distance Maps, Pattern Recognition Letters, **13(4)** (1982), 237-243.
[5] D. Attali and J. O. Lachaud, *D*elaunay conforming iso-surface, skeleton extraction and noise removal, Computational Geometry: Theory and Applications, **19** (2001), 175-189.
[6] D. Attali and A. Montanvert, *M*odeling noise for a better simplification of skeletons, in "Proceeding of International Conference on Image Processing", **3** (1996), 13-16.
[7] G. Bertrand, *A* parallel thinning algorithm for medial surfaces, Pattern Recognition Letters, **16** (1995), 979-986.
[8] H. Blum, *B*iological shape and visual science, Journal of Theoretical Biology, **38** (1973), 205-287.
[9] G. Borgefors, I. Nystrom and G. S. D. Baja, *C*omputing skeletons in three dimensions, Pattern Recognition, **32** (1999), 1225-1236.
[10] L. Calabi, "A study of the skeleton of plane figures", Technical Report 60429, sr-2, Parke Mathematical Laboratories, December 1965.
[11] J.-H. Chuang, C.-H. Tsai and M.-C. Ko, *S*keletonization of Three-Dimensional Object Using Generalized Potential Field, IEEE Trans. Patten Anal. Mach. Intell., **22(11)** (2000), 1241-1251.
[12] N. Cornea, D. Silver, X. Yuan and R. Balasubramanian, *C*omputing hierarchical curve-skeletons of 3d objects, The Visual Computer, **21** (2005), 945-955.
[13] P.-E. Danielsson, *E*uclidean Distance Mapping, Computer Graphics and Image Processing, **14** (1980), 227-248.
[14] L. C. Evans, "Partial Differential Equations", Graduate Studies in Mathematics, **19**, American Society of Mathematics, 1998.
[15] J. A. Goldak, X. Yu, A. Knight and L. Dong, *C*onstructing discrete medial axis of 3-D objects, International Journal of Computational Geometry and Applications, **1(3)** (1991), 327-329.
[16] J. Gomez and O. Faugeras, *L*evel sets and distance functions, In "European Conference on Computer Vision", Dublin, Ireland, **1** (2000), 588-602.
[17] M. S. Hassouna and A. A. Farag, *O*n the Extraction of Curve Skeletons using Gradient Vector Flow, In "11th International Conference on Computer Vision (ICCV)", (2007), 1-8.
[18] D. G. Kirkpatrick and J. D. Radke, *A* framework for computational morphology, in "Computational Geometry" (ed. G. Toussaint), North-Holland, (1998), 217-248.
[19] T.-C. Lee and R. L. Kashyap, *B*uilding skeleton models via 3-D medial surface/axis thinning algorithm, Graphical Models and Image Processing, **56(6)** (1994), 462-478.
[20] F. F. Leymarie, "Three-dimensional shape representation via shock flow", Ph.D dissertation, Brown University, 2003.
[21] F. Leymarie and M. D. Levine, *S*imulating the grassfire transform using an active contour model, IEEE Transactions on Pattern Analysis and Machine Intelligence, **14(1)** (1992), 56-75.
[22] G. Malandain, G. Bertrand and N. Ayache, *T*opological segmentation of discrete surfaces, International Journal of Computer Vision, **10(2)** (1993), 183-197.
[23] G. Malandain and S. Fernandez-Vidal, *E*uclidean skeletons, Image and Vison Computing, **16** (1998), 317-327.
[24] A. Manzanera, T. M. Bernard, F. Pretrux and B. Longuet, *M*edial faces from a concise 3D thinnig algorithm, In "International Conference on Computer Vision", Kerkyra, Greece, (1999), 337-343.
[25] M. Näf, O. Kübler, R. Kikinis, M. E. Shenton and G. Székely, *C*haracterization and recognition of 3D organ shape in medical image analysis using skeletonization, In "IEEE Workshop on Mathematical Methods in Biomedical Image Analysis", 1996.
[26] R. L. Ogniewicz, "Discrete Voronoi Skeletons", Hartung-Gorre, 1993.
[27] C. Pudney, *D*istance-ordered homotopic thinning: A skeletonizaiton algorithm for 3D digital images, Computer Vision and Image Understanding, **72(3)** (1998), 404-413.
[28] M. Schmitt, *S*ome examples of algorithms analysis in computational geometry by means of mathematical morphology techniques, In "Lecture Notes in Computer Science, Geometry and Robotics" (Eds. J. Boissonnat and J. Laumond), **391**, Springer-Verlag: Berlin, (1989), 225-246.

[29] D. J. Sheehy, C. G. Armstrong and D. J. Robinson, *S*hape description by medial surface construction, IEEE Transactions on Visualization and Computer Graphics, **2(1)** (1996), 62-72.
[30] E. C. Sherbrooke, N. Patrikalakis and E. Brisson, *A*n algorithm for the medial axis transform of 3D polyhedrals, IEEE Transactions on Visualization and Computer Graphics, **2(1)** (1996), 44-61.
[31] K. Siddiqi, S. Bouix, A. Tannenbaum and S. Zucher, *H*amilton-Jacobi Skeletons, International Journal of Computer Vision, **48(3)** (2002), 215-221.
[32] H. Tek and B. B. Kimia, *S*ymmetry maps of free-form curve segments via wave propagation, In "International Conference on Computer Vision", Kerkyra, Greece, (1999), 362-369.
[33] A. Torsello and E. R. Hancock, *C*orrecting Curvature-Density Effects in the Hamilton-Jacobi Skeleton, IEEE Transaction on Image Processing, **15(4)** (2006), 877-891.
[34] T. Y. Kong and A. Rosenfeld, *D*igital topology: introduction and survey, Computer Vision, Graphics, and Image Processing, **48** (1989), 357-393.
[35] Y. R. Tsai, *R*apid and Accurate Computation of the Distance Function Using Grids, Journal of Computational Physics, **178** (2002), 175-195.
[36] H.-K. Zhao, *A* fast sweeping method for Eikonal equation, Mathematics of Computation, **74** (2005), 603-627.
[37] Y. Zhou, A. Kaufman and A. W. Toga, *3*D skeleton and centerline generation based on an approximate minimum distance field, International Journal of the Visual Computer, **14(7)** (1998), 303-314.

Received September 2009; revised January 2010.

E-mail address: luos@math.uci.edu
E-mail address: guibas@cs.stanford.edu
E-mail address: zhao@math.uci.edu