Pattern Recognition 43 (2010) 3348–3358

Contents lists available at ScienceDirect

Pattern Recognition

journal homepage: www.elsevier.com/locate/pr

Fast distance transformation on irregular two-dimensional grids

Antoine Vacavant

Université de Lyon, CNRS, Université Lyon 2, LIRIS, UMR5205, F-69676, France

ARTICLE INFO

Article history:
Received 13 November 2009
Received in revised form
23 February 2010
Accepted 22 April 2010

Keywords:
Squared Euclidean distance transformation
Irregular grids
Quadtree
Run length encoding
Voronoi diagram
Medial axis extraction

ABSTRACT

In this article, we propose a new fast algorithm to compute the squared Euclidean distance transform (E^2DT) on every two-dimensional (2-D) irregular isothetic grids (regular square grids, quadtree based grids, etc.). Our new fast algorithm is an extension of the E^2DT method proposed by Breu et al. [3]. It is based on the implicit order of the cells in the grid, and builds a partial Voronoi diagram of the centers of background cells thanks to a data structure of lists. We compare the execution time of our method with the ones of others approaches we developed in previous works. In those experiments, we consider various kinds of classical 2-D grids in imagery to show the interest of our methodology, and to point out its robustness. We also show that our method may be interesting regarding an application where we extract an adaptive medial axis construction based on a quadtree decomposition scheme.

1. Introduction

The definition of discrete distance is a very important concept in shape analysis and description [20,28,29]. The distance transformation (DT) of a binary image consists in labeling each point of a discrete object $\mathcal{E}$ with its distance to the complement of $\mathcal{E}$. The extraction of a skeleton or medial axis (MA), centered within a shape is a classical application of the DT and permits to get a compressed representation of this object while preserving its topology. The DT and MA processes can be used for shape matching [26], shape measures [10], image enhancement [47], etc. These processes are widely studied and developed on classical regular grids (see for example states of the art in [14] for DT techniques and in [19] for MA extraction). There also exist some specific extensions of the DT on non-regular grids, for example rectangular grids [5,34], quadtree based grids [31,44], hexagonal grids [23,36], etc. In a similar way, some works aimed to extend the computation of a MA or a skeleton to other non-standard regular grids (triangular, hexagonal, rectangular grids) [6,13,45], and to quadtree/octree representations [31,46]. However, to our knowledge, no generic framework has been designed for DT and MA extraction processes on every kinds of regular and irregular image representations.

This article deals with generalizing the DT computation on *irregular isothetic grids* (or $\mathbb{I}$−grid for short) in two dimensions (2-D). This kind of grid is defined with rectangular cells whose edges are aligned along the two axis. The size and the position of those nonoverlapping cells are defined without constraint (see next section for a detailed definition). The quadtree decomposition and the RLE (run length encoding) grouping schemes are examples of classical techniques in imagery which induce an $\mathbb{I}$−grid (recursive and adaptive grids as well [4,18]). We also used this model to generalize the polygonal and topological reconstruction of complex irregular objects [9,39,41].

Here, we focus our interest on generalizing techniques that compute an *exact* Euclidean distance map (which excludes approaches using approximate chamfer distances [15] or chamfer sequences [25,35], vector propagation techniques [10,11], fast marching DT methods [33,37], etc.). In particular, the E^2DT (squared Euclidean DT) of a 2-D binary image I can be handled by linear-time algorithms (with respect to the size of I) [3,22,24] and may become very fast techniques to compute the E^2DT on $\mathbb{I}$−grids. Many of those methodologies can be linked to the computation of a discrete and partial Voronoi diagram of the background pixels [8,17]. In our framework, a naive approach based on the complete Voronoi diagram of the centers of the background cells can be naturally developed, as we have shown in [40]. Unfortunately, this method suffers from non-optimal time complexity, and is not convenient for very dense and complex grids. We have also proposed in previous works [40,42] algorithms to compute E^2DT on $\mathbb{I}$−grids in dimension two and in higher dimensions. However, these approaches are not optimal according to time complexity. In this article, we thus propose a new fast algorithm that computes the DT on every 2-D $\mathbb{I}$−grids, which is an extension of the *sweep line* based approach proposed by Breu et al. [3].

E-mail address: antoine.vacavant@liris.cnrs.fr
URL: http://liris.cnrs.fr/antoine.vacavant

0031-3203/$ - see front matter © 2010 Elsevier Ltd. All rights reserved.
doi:10.1016/j.patcog.2010.04.018

The next section of this article presents the definition of an $\mathbb{I}$−grid and the data structure used to develop our contribution. Then, we recall the concept of the E^{2}DT on $\mathbb{I}$−grids and its relations with the Voronoi diagram. In Section 3, we describe our new algorithm inspired from [3] aimed to generalize the computation of the E^{2}DT, with respect to this definition. We then show an experimental analysis to compare our proposal with those we have described in [40,42] (in terms of speed and complexity) and to present MA extraction on $\mathbb{I}$−grids.

2. Preliminaries and previous work

2.1. $\mathbb{I}$−grid *model*

In this section, we first introduce the concept of irregular isothetic grids ($\mathbb{I}$−grids), with the following definition [9] (see Fig. 1 for some examples):

Definition 1 (*2-D* $\mathbb{I}$−grid). Let $\mathtt{I} \subset \mathbb{R}^2$ be a closed rectangular support. A 2-D $\mathbb{I}$−grid $\mathbb{I}$ is a tiling of $\mathtt{I}$ with nonoverlapping rectangular cells which edges are parallel to the X and Y axis. The position (x_R, y_R) and the size (l_R^x, l_R^y) of a cell R in $\mathbb{I}$ are given without constraint:

$$(x_R, y_R) \in \mathbb{R}^2, \quad (l_R^x, l_R^y) \in \mathbb{R}_+^* 2. \tag{1}$$

From this very general definition, we also define an order relation $\preceq_y$ between the cells of an $\mathbb{I}$−grid (we define the relation $\succeq_y$ in a similar way):

Definition 2 (*Order relation on an* $\mathbb{I}$−grid). Let R_1 and R_2 be two cells of an $\mathbb{I}$−grid $\mathbb{I}$. We define the total order relation $\preceq_y$, based on the cell centers:

$$\forall R_1, R_2 \in \mathbb{I}, \quad R_1 \preceq_y R_2 \iff y_{R_1} < y_{R_2} \vee (y_{R_1} = y_{R_2} \wedge x_{R_1} \leq x_{R_2}). \tag{2}$$

This relation is also a lexicographic order over $(\mathbb{R}, \leq) \times (\mathbb{R}, \leq)$. Furthermore, if the tiling of $\mathbb{I}$ is bounded (contained in a compact domain), a first cell (denoted by R_1) exists as well as a last cell (denoted by R_n) in $\mathbb{I}$. If we come back to the regular case, two pixels may be compared with this relation, which implies a lexicographic order over $(\mathbb{Z}, \leq) \times (\mathbb{Z}, \leq)$. To scan efficiently the cells of an $\mathbb{I}$−grid $\mathbb{I}$, we use this order relation to develop our sweep line based approach. We consider the following data structure model:

- The procedure `next`(R), for some R in $\mathbb{I}$, returns the next cell R' in the lexicographic order $\preceq_y$. This operation is handled in $\mathcal{O}(1)$.
- `min`($\mathbb{I}$) and `max`($\mathbb{I}$), respectively, return the lowest and the greatest cell in $\mathbb{I}$ with respect to $\preceq_y$. Those two operations are processed in constant time $\mathcal{O}(1)$.

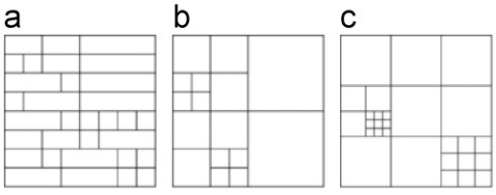


Fig. 1. Examples of $\mathbb{I}$−grids: grid obtained with a RLE scheme (a), quadtree-based grid (b) and an adaptive grid (c).

With the two previous definitions we gave, we are able to scan all the cells of $\mathbb{I}$ according to the $\preceq_y$ order. In particular, we can access to the list of cells whose center is aligned with the line $y = r \in \mathbb{R}$. We denote by $\mathcal{E}_r$ this list, computed in the worst case in $\mathcal{O}(n)$, where n is the total number of cells in $\mathbb{I}$. And if we denote by $R_1 = \mathtt{min}(\mathbb{I})$ and $R_n = \mathtt{max}(\mathbb{I})$, the access to $\mathcal{E}_r$ is direct, if we consider $r = y_{R_1}$ and y_{R_n}.

Now, let $\mathcal{E}_r$ be a list of aligned cells. Thanks to the `next`() procedure, the access to the list $\mathcal{E}_t$, so that $t > r$ and the first cell of $\mathcal{E}_t$ is next to the last cell of $\mathcal{E}_r$, is in $\mathcal{O}(1)$. More precisely, we store the lists $\mathcal{E}_{r_1}, \ldots, \mathcal{E}_{r_{n_l}}$, with $r_1, \ldots, r_{n_l} \in \mathbb{R}$, $r_1 < r_2 < \cdots < r_{n_l - 1} < r_{n_l}$. And getting $\mathcal{E}_{r_{i+1}}$ from $\mathcal{E}_{r_i}$, $i \in \{1, n_l - 1\}$, is handled in $\mathcal{O}(1)$.

We depict in Fig. 2 the lexicographic order underlying an $\mathbb{I}$−grid $\mathbb{I}$, and some lists $\{\mathcal{E}_{r_i}\}_{i=1,n_l}$ associated with this order. Considering implementation issues, a simple list structure can be used to implement our model.

2.2. *Toward high-performance algorithms on adaptive grids*

An important fact for the rest of this article is that every lists $\mathcal{E}_{r_i}$ are sorted along the X-axis, thanks to the lexicographic order over $\mathbb{I}$. The construction of the lists $\mathcal{E}_{r_i}$ depends on the structure of the input $\mathbb{I}$−grid.

When we treat regular square, rectangular, or hexagonal grids [21], we can determine the $\mathcal{E}_{r_i}$ lists in $\mathcal{O}(n)$ time: we just have to scan the cells following the natural total order induced by $\mathbb{Z}^2$. For an $\mathbb{I}$−grid computed from a RLE grouping scheme, we also handle the filling of our structure in $\mathcal{O}(n)$ time, since this is a mono-dimensional process. In this case, the cells are organized following a *snake-like* order, which is similar to our total relation order. For numerical analysis or geometrical modeling purpose, hierarchy of grids [4] or recursive grids [18] can be used to adaptively represent the regions of interest of the space. In those adaptive grids, grids with various resolutions are constructed, and can be a very interesting way to speed up classical methodologies defined on regular grids. Let us now consider a simple hierarchy with two grids. The construction of our structure first consists in building the $\mathcal{E}_{r_i}$ lists associated with those grids. Since they are generally regular square grids, this stage is handled in linear time. Then we just have to merge those list structures to construct the global $\mathcal{E}_{r_i}$ lists of the hierarchy. Since each list is sorted along an axis, this merge operation can also be done in linear time. In a more general way, it is clear that the construction of our structure can be

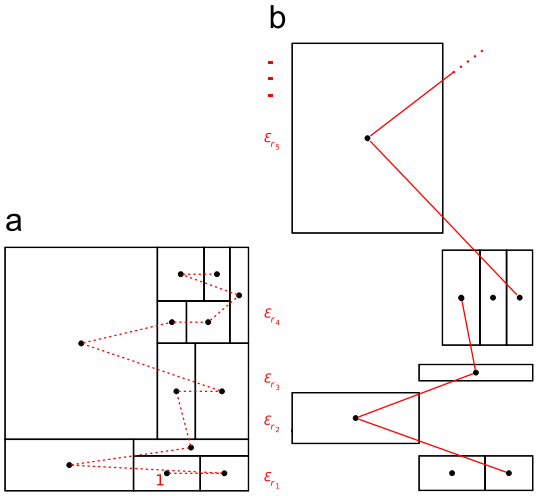


Fig. 2. Structure of lists representing the $\preceq_y$ relation. We depict in (b) the first lists $\mathcal{E}_{r_1}, \ldots, \mathcal{E}_{r_5}$ associated with the cells in (a). The "1" in (a) indicates the first cell in the $\preceq_y$ order.

handled in $\mathcal{O}(n)$ time, since this a sequence of merge operations of ordered sub-structures.

When an image is decomposed thanks to a tree-based process, i.e. a quadtree [32] or a k d-tree [1] for example, the constructed cells are ordered thanks to the underlying tree structure (e.g. Morton or Peano orders). In the worst case scenario, the construction of the $\mathcal{E}_{r_i}$ lists from the induced *tree-based* grid may be then handled in $\mathcal{O}(n\log n)$ time. But we can also choose several optimizations consisting in coding the adjoining relations between cells in the tree. For example, during the decomposition stage into a k d-tree, we can add pointers between two nodes of the tree when the associated cells are adjacent [16]. In this case, we only scan the leaves of the tree in $\mathcal{O}(n)$, with respect to our relation order. Finally, if the input $\mathbb{I}$−grid $\mathbb{I}$ is not supported by any specific structure, we have to first sort the cells of $\mathbb{I}$ before constructing our set of lists. This pre-processing step clearly requires $\mathcal{O}(n\log n)$ execution time but is rarely necessary for the most common $\mathbb{I}$−grids in imagery.

2.3. Distance transformation on $\mathbb{I}$−grids and Voronoi diagram

In our framework, we consider *labeled* $\mathbb{I}$−grids, i.e. each cell of the grid has a *foreground* or *background* label (its value is, respectively, "0" or "1" for example). For an $\mathbb{I}$−grid $\mathbb{I}$, we denote by $\mathbb{I}_F$ and $\mathbb{I}_B$ the sets of foreground and background cells in $\mathbb{I}$. We consider here that the distance between two cells R and R' is the distance between their centers. If we denote by $p=(x_R,y_R)$ and $p'=(x_{R'},y_{R'})$ these points, and $d_e^2(p,p')$ the squared Euclidean distance between them, the $\mathbb{I}$−CDT (center-based DT on $\mathbb{I}$−grids) of a cell R is defined as follows:

$$\mathbb{I}\text{-CDT}(R)=\min_{R'}\{d_e^2(p,p');R'\in\mathbb{I}_B\}, \tag{3}$$

and is exactly the E^{2}DT if we consider $\mathbb{I}$ as a regular classical grid. We can draw a parallel between this definition of the E^{2}DT on an $\mathbb{I}$−grid and the computation of a Voronoi diagram (VD). More precisely, as in the regular case, the $\mathbb{I}$−CDT can be linked with the VD of the centers of the background cells (i.e. *Voronoi sites*) [40]. The VD of a set of points $\mathcal{P}=\{p_i\}$ is a tiling of the plane into *Voronoi cells* that respect the following property [43,12]: each point q of the plane belongs to the closed cell C_{p_i} associated with the site p_i if and only if $d_e(q,p_i)\leq d_e(q,p_j)$ for all the sites p_j so that $j\neq i$. In fact, computing the nearest background cell of a cell R consists in searching for the Voronoi site p_i so that the center of R belongs to C_{p_i}. Then, it is just necessary to compute the squared distance between R and p_i to compute $\mathbb{I}$−CDT(R). In Fig. 3, we present an example of the computation of the $\mathbb{I}$−CDT process, and its corresponding underlying VD.

In this case, the $\mathbb{I}$−CDT computation has a $\mathcal{O}(n\log n_B)$ time complexity, where $n=\#\mathbb{I}$ is the total number of cells, and n_B is the number of background cells. This technique is obviously not computationally efficient for every grids, and not adapted to dense grids [40]. The extension of this transformation to

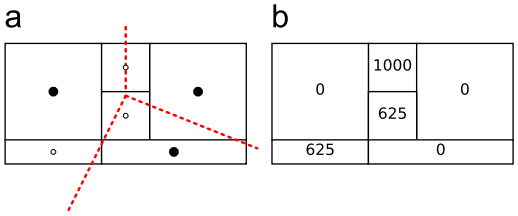


Fig. 3. Computation of the $\mathbb{I}$−CDT (b) of a small labeled $\mathbb{I}$−grid with a size of 100×50. White (respectively black) points represent foreground (background) cells centers. We can draw a parallel between the $\mathbb{I}$−CDT and the construction of a VD of points (a).

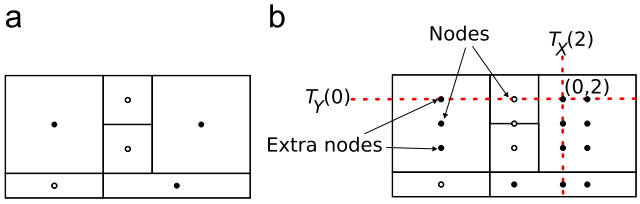


Fig. 4. Construction of the irregular matrix associated to a simple $\mathbb{I}$−grid (a). In (b), the extra node (0,2) of the matrix is depicted at the intersection of the dotted lines.

d-dimensional (d-D) $\mathbb{I}$−grids is a hard work. A VD can be computed in d-D with a $\mathcal{O}(n_B\log n_B+n_B^{\lceil d/2\rceil})$ time complexity (thanks to a gift-wrapping approach [12] for example). However, localizing a point in the VD is an arduous task, and an additional structure like subdivision grids [27] should be constructed to handle this operation.

2.4. Separable algorithms for $\mathbb{I}$−CDT

In [40,42], we presented separable algorithms inspired from [22,30] to compute the $\mathbb{I}$−CDT. To develop a separable process on $\mathbb{I}$−grids (i.e. which can be extended to the d-D case), we use the *irregular matrix* associated to a labeled $\mathbb{I}$−grid $\mathbb{I}$ introduced in [40]. This data structure aims to organize the cells of the grid along the X and Y axis by adding virtual cell centers (see Fig. 4 for an example). The irregular matrix contains as many columns (respectively, rows) as the X-coordinates (Y-coordinates) in the grid. In the following, we denote by n_1 (respectively, n_2) the number of X-coordinates (Y-coordinates) in the grid. At the intersection of two X and Y coordinates, a node in this structure may represent the cell center or not (i.e. this is an *extra node*, see Fig. 4b). The extra nodes are used to propagate the distance values through the irregular matrix and then compute a correct distance value for each cell center.

The first algorithm [40] inspired from the work of Saito et al. [30]has a time complexity in 2-D in $\mathcal{O}(n_1n_2\log n_2)$. This algorithm aims to minimize a quadratic form by computing the lower envelope of a set of parabolas. If we consider a d-D labeled $\mathbb{I}$−grid, the cost of the consecutive steps is in $\mathcal{O}(n_1\times\cdots\times n_d\times\log n_2\times\cdots\times\log n_d)$, where the dimension of the irregular matrix is $n_1\times\cdots\times n_d$. The second method [42] is an adaptation of the algorithm of Maurer et al. [22]. It is a linear $\mathbb{I}$−CDT approach with respect to the associated irregular matrix size, i.e. in $\mathcal{O}(n_1n_2)$ time complexity in 2-D. Dimension by dimension, the Voronoi diagram is pruned in order to keep only the Voronoi sites impacting the treated dimension. If we consider a labeled d-D $\mathbb{I}$−grid, which associated irregular matrix size is $n_1\times\cdots\times n_d$, the time complexity of our second algorithm is thus in $\mathcal{O}(n_1\times\cdots\times n_d)$.

In the following, we present a faster algorithm to compute the $\mathbb{I}$−CDT on 2-D $\mathbb{I}$−grids. To do so, we use the lists structure presented in Section 2.1 instead of the irregular matrix, and we adapt a sweep-line based E^{2}DT algorithm [3].

3. A fast and sweep-line based approach in 2-D

3.1. Description of the method

We now present our simpler and faster algorithm, based on the order relation, the lists structure we have described in Section 2.1 and the elements introduced by Breu et al. [3]. A simple approach to compute the E^{2}DT would be to pre-compute the complete VD of the grid background, and to locate foreground points in the VD. This technique is obviously not computationally efficient, and not adapted to digital images. The basic idea of Breu

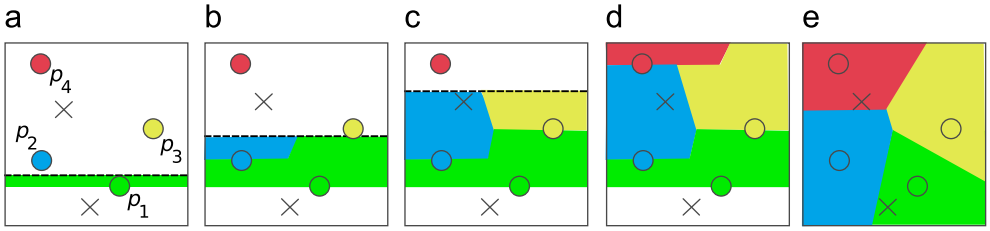


Fig. 5. Several stages of our algorithm in an example with four background cell centers $p_1,\ldots,p_4$ (circles) and two foreground cell centers (crosses). During the first sweeping stage (a–d), we build the discrete irregular VD, intersection between the $\mathbb{I}-$grid points and the complete VD of the background cell centers. In (c), the point p_1 is not considered anymore, because it is hidden by p_2 and p_3. Figure (e) is the result of the second stage, where the foreground points are affected by the correct $\mathbb{I}-$CDT.

et al. is thus to treat the points in $\mathbb{Z}^2$ by building directly the *discrete* VD (intersection of the complete VD and $\mathbb{Z}^2$), to compute the final E^{2}DT. In this article, we directly present the generalization of this approach to obtain the $\mathbb{I}-$CDT, that treat cells centers of an $\mathbb{I}-$grid, i.e. points of $\mathbb{R}^2$ (see Fig. 5 for an illustration of the algorithm).

We consider the points whose Y-coordinate is lower or equal to $r \in \mathbb{R}$. With the notion of $\preceq_y$ order we have presented before, we can parse the $\mathbb{I}-$grid $\mathbb{I}$ by successively considering the cells with an Y-coordinate aligned with $y = r_i \in \mathbb{R}$, where r_i increases. We thus know the different Y-coordinates of the cells in $\mathbb{I}$. We recall that we denote by $\{r_i\}_{i=1,n_l}, r_i \in \mathbb{R}$ this set of n_l values, $\mathcal{E}_{r_i}$ the set of points aligned with r_i. We also denote by $\mathcal{V}_{\mathcal{E}_{r_i}}$ the corresponding discrete VD. While sweeping the $\mathbb{I}-$grid, we build a set of lists $\{\mathcal{L}_{r_i}\}_{i=1,n_l}$. Each list corresponds to background cell centers (i.e. Voronoi sites) whose Voronoi cell in $\mathcal{V}_{\mathcal{E}_{r_i}}$ intersects the line given by $y = r_i$. The goal of our algorithm, adapted from [3], is to deduce $\mathcal{L}_{r_i}$ from $\mathcal{L}_{r_{i-1}}$. We first make two observations:

- Let $u \in \mathcal{L}_{r_i}$ and $v \in \mathcal{L}_{r_{i-1}}$ be two Voronoi sites so that their respective X-coordinates coincide, i.e. $u_x = v_x$, and we suppose that their respective Y-coordinates respect $u_y > v_y$. Obviously, the Voronoi cell attached to v does not intersect the line $y = r_i$. This implies that there is only one site in $\mathcal{L}_{r_i}$ per X-coordinate.
- If the sites of $\mathcal{E}_{r_i}$ are sorted along X axis, then the Voronoi cells intersecting the line $y = r_i$ are sorted in the same way.

Then, we define the predicate `hidden_by()` (see Fig. 6) that permits to prune sites in $\mathcal{L}_{r_{i-1}}$ to build the list $\mathcal{L}_{r_i}$. This predicate takes three points $u,v,w \in \mathbb{R}^2$ as parameters, and indicates that v is *hidden by* u and w along the line $y = r_i$ if the Voronoi cell associated to v does not intersect this line, whereas u and v do.

With those elements, our algorithm mainly consists in deleting points from a list of *candidates* $\mathcal{C}_{r_i}$, where the maximal Y-coordinate is $y \leq r_i$, and sorted along X-axis, with the following property:

Property 1. *Let v be a site in the set $\mathcal{C}_{r_i}$ and a line l given by $y = r_i$, v does not belong to the list $\mathcal{L}_{r_i}$ if and only if there exists at least a couple of sites (u,w) from $\mathcal{C}_{r_i} \backslash \{v\}$ so that u and w hide v along l.*

To determine the $\mathbb{I}-$CDT of each foreground point p, we just have to find the nearest site, only by considering the X-coordinate of p (i.e. find the Voronoi cell of the VD where p is located).

In Fig. 6, we give our algorithm, adapted from [3] to deal with irregular structures. We also present in this figure the function `delete_sites()` and the predicate `hidden_by()`. The algorithm $\mathbb{I}-$`CDT()` presented in Fig. 6 is based on a double sweeping process over the cells of an input $\mathbb{I}-$grid $\mathbb{I}$. We recall that the cells $\mathcal{E}_{r_i}$ aligned with the line $y = r_i$ are sorted along the X-axis, and that we know the number of different possible Y-coordinates r_i in $\mathbb{I}$, n_l. If a cell $R \in \mathcal{E}_{r_i}$ belongs to the background, then we update the associated list of candidates, and so ($\cup_x$ is the union between ordered sets, along the X axis):

$$\begin{aligned}\mathcal{C}_{r_i} &= \mathcal{L}_{r_{i-1}} \underset{x}{\cup} \{p \text{ center of } R \in \mathbb{I}_B; y_R = r_i\},\ i \geq 2 \\ &= \mathcal{L}_{r_{i-1}} \underset{x}{\cup} \{R \in \mathcal{E}_{r_i} \wedge R \in \mathbb{I}_B; y_R = r_i\},\ i \geq 2, \end{aligned} \tag{4}$$

and if $i = 1$, then the left member of the union does not exist. We insist on the fact that the invariant of our algorithm is that $\mathcal{L}_{r_{i-1}}$ is ordered along the X-axis, and since $\mathcal{E}_{r_i}$ is also sorted (by the lexicographic order induced by $\preceq_y$), the computation of $\mathcal{L}_{r_i}$ is linear in time and this list is naturally ordered along X. We use the function `delete_sites()` to eliminate the candidate sites, hidden by two other sites (Property 1). We recall that this function is processed in linear time $\mathcal{O}(\#\mathcal{C}_{r_i})$. When the list $\mathcal{L}_{r_i}$ is determined, we scan the set of foreground points ($\mathcal{F}$) and we affect them the squared distance to the nearest background point in $\mathcal{L}_{r_i}$. We can notice that this loop implies two linear scans of the centers of the cells with the same Y-coordinate r_i, because they are sorted along X-axis. Finally, a second sweep line stage is processed backward, which is very similar to the first one. In this case, we have to consider an other order to parse the cells of $\mathbb{I}$. Briefly, we choose here the total order relation $\succeq_y$ instead of $\preceq_y$.

3.2. Complexity analysis

For each iteration ($y = r_i$, $i = 1,\ldots,n_l$), we consider the set $\mathcal{E}_{r_i}$ in $\mathcal{O}(\#\mathcal{E}_{r_i})$. Then, as it is ordered in the X direction, the stages of building $\mathcal{C}_{r_i}$ and $\mathcal{L}_{r_i}$ are handled, respectively, in $\mathcal{O}(\#\mathcal{C}_{r_i})$ and $\mathcal{O}(\#\mathcal{L}_{r_i})$. Thanks to this property of order along X-axis, the value assignment of $\mathbb{I}-$CDT(R) for each foreground cell $R \in \mathcal{E}_{r_i}$ can be performed in linear time in the number of elements in $\mathcal{E}_{r_i}$. Hence, in a general way, our algorithm has a time complexity in $\mathcal{O}(\sum_{i=1,n_l} \#\mathcal{E}_{r_i} + \mathcal{C}_{r_i} + \mathcal{L}_{r_i})$. We now give further details about this complexity regarding two cases.

In the best case, the lists $\mathcal{L}_{r_i}$ evolve and are updated during the sweep-line process, i.e. Voronoi sites are added then deleted because other sites hide them. This means that for each ($y = r_i$, $i = 1,\ldots,n_l$), our algorithm needs $\mathcal{O}(\#\mathcal{E}_{r_i})$ time. In the experiments we have driven, this is the more common behaviour we have observed for our algorithm. For each scan of the algorithm, the time complexity in the best case is thus in $\mathcal{O}(\sum_{i=1,n_l} \#\mathcal{E}_{r_i}) = \mathcal{O}(n)$.

In the worst case, we keep $\mathcal{L}_{r_1}$ unchanged during the whole algorithm, and we have $\max_i(\#\mathcal{E}_{r_i}) = \mathcal{L}_{r_1}$. This may append if $\mathcal{E}_{r_1}$ is a set of background cells that are never deleted with the `hidden_by()` predicate. Hence, our new algorithm $\mathbb{I}-$`CDT()` is processed in $\mathcal{O}(n_l \max_i(\#\mathcal{E}_{r_i}))$ in the worst case, where we may have $n_l \max_i(\#\mathcal{E}_{r_i}) \gg n$ (see Fig. 7 for an example). According to the previous algorithms that we developed [40,42] (see Section 2.4), we can bound the complexity of our new method thanks to the irregular matrix size. Indeed, we can notice that we have $n_l = n_2$ and $\max_i(\#\mathcal{E}_{r_i}) \leq n_1$, where n_1,n_2 represent the size of the irregular

a

Predicate `hidden_by()`

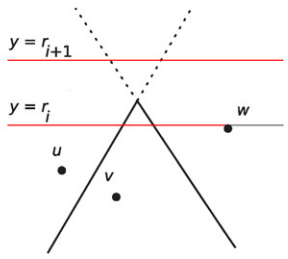
input: points u, v and w of $\mathbb{R}^2$,
the line given by $y = r \in \mathbb{R}$.
output: is v hidden by u and w on the line $y = r$?

return $(w_x - v_x) \times (v_x^2 - u_x^2 - 2r(v_y - u_y) + v_y^2 - u_y^2) \geq (v_x - u_x) \times (w_x^2 - v_x^2 - 2r(w_y - v_y) + w_y^2 - v_y^2)$;

Function `delete_sites()`

input: a list of candidate sites $\mathcal{C}_{r_i}$.
output: a list of sites $\mathcal{L}_{r_i}$ respecting property ??.

$\mathcal{L}_{r_i}[1] \leftarrow \mathcal{C}_{r_i}[1]$;
$\mathcal{L}_{r_i}[2] \leftarrow \mathcal{C}_{r_i}[2]$;
$k \leftarrow 2,\ l \leftarrow 3,\ c \leftarrow \#\mathcal{C}_{r_i}$;
while $l \leq c$ **do**
  $w \leftarrow \mathcal{C}_{r_i}[l]$;
  while $k \geq 2 \wedge$ `hidden_by`$(\mathcal{L}_{r_i}[k-1], \mathcal{L}_{r_i}[k], w, r)$ **do**
    $k \leftarrow k - 1$;
  end while
  $k \leftarrow k + 1$;
  $l \leftarrow l + 1$;
  $\mathcal{L}_{r_i}[k] \leftarrow w$;
end while
return $\mathcal{L}_{r_i}$;



b

Algorithm $\mathbb{I}$-`CDT()`

input: a labeled $\mathbb{I}$-grid $\mathbb{I}$.
output: the $\mathbb{I}$-CDT of $\mathbb{I}$.

▷ *Stage 1, from bottom to top, in respect to* $\preceq_y$
for $i \leftarrow 1, n_l$ **do**
  $\mathcal{F} \leftarrow \emptyset$; ▷ *List of foreground cells*
  if $i \geq 2$ **then** $\mathcal{C}_{r_i} \leftarrow \mathcal{L}_{r_{i-1}}$;
  else $\mathcal{C}_{r_i} \leftarrow \emptyset$;
  end if
  ▷ $\mathcal{E}_{r_i}$ *is the list of cells aligned with* $y = r_i$
  for all center p of a cell $R \in \mathcal{E}_{r_i}$ **do**
    if $R \in \mathbb{I}_B$ **then**
      $\mathcal{C}_{r_i} \leftarrow \mathcal{C}_{r_i} \cup_x \{p\}$;
      $\mathbb{I}$-CDT$(R) \leftarrow 0$;
    else
      $\mathcal{F} \leftarrow \mathcal{F} \cup_x \{p\}$;
      $\mathbb{I}$-CDT$(R) \leftarrow \infty$;
    end if
  end for
  $\mathcal{L}_{r_i} \leftarrow$ `delete_sites`$(\mathcal{C}_{r_i})$;
  $k \leftarrow 1,\ l \leftarrow \#\mathcal{L}_{r_i}$;
  for all point $p \in \mathcal{F}$ **do**
    $d_1 \leftarrow d_e(p, \mathcal{L}_{r_i}[k])$;
    $d_2 \leftarrow d_e(p, \mathcal{L}_{r_i}[k+1])$;
    while $d_1 \geq d_2 \ \wedge \ k < l$ **do** $k \leftarrow k + 1$;
    end while
    if $d_1 <$ $\mathbb{I}$-CDT(R) **then**
      $\mathbb{I}$-CDT$(R) \leftarrow d_1^2$;
    end if
  end for
end for
▷ *Stage 2, from top to bottom, in respect to* $\succeq_y$
for $i \leftarrow n_l, 1$ **do**
  ⋮
end for

Fig. 6. In (a), we present the predicate `hidden_by()` and the function `delete_sites()` that builds $\mathcal{L}_{r_i}$ from $\mathcal{C}_{r_i}$. We also present an example of deleting sites with the `hidden_by()` predicate. In this example, we have $\mathcal{L}_{r_i} = \{u,v,w\}$ and $\mathcal{L}_{r_{i+1}} = \{u,w\}$ because v is hidden by u and w. We use this function in our algorithm to compute the $\mathbb{I}$−CDT of an $\mathbb{I}$−grid $\mathbb{I}$ (b).

matrix associated with the treated $\mathbb{I}$−grid. A coarse time complexity of $\mathbb{I}$−`CDT()` is thus $\mathcal{O}(n_1 n_2)$, in the worst case.

In the next section, we compare our new approach with the ones we have proposed in previous works [40,42]. We consider the execution time in different configurations of grids, built from natural images or by a random generating process.

4. Experiments and analysis of our contribution

4.1. Experiments on regular and irregular 2-D grids

Here, we propose to study the computation of the $\mathbb{I}$−CDT on several classical 2-D grids in imagery. We consider in our experiments three regular grids (square, rectangular and hexagonal), then two image-dependent grids, based on a quadtree decomposition and a RLE grouping scheme. Since the definition of the $\mathbb{I}$−CDT is only based on the centers of the cells, we can treat all those grids that belong to the $\mathbb{I}$−grid model, and other non-isothetic grids, like hexagonal or triangular grids. Firstly, we present the computation results of the $\mathbb{I}$−CDT for those grids, based on the small image *ghost* (16×16 pixels) in Fig. 8. In this figure, the rectangular grid cells have a height two times greater than the width.

We now compare the speed of our new $\mathbb{I}$−CDT algorithm with the methods we have developed in previous works [40,42]. Therefore, we use the notations given in Table 1. In this table, we depict the time and space complexities of the four tested algorithms (see Sections 2 and 3 for a description of those methodologies).

We now consider the three images *noise* (100×100 pixels), *lena* (208×222 pixels) and *canon* (512×512 pixels) depicted in Fig. 9. The image *lena* was obtained by thresholding the original grey-level picture at level 113. In Fig. 10, we present the computation results of the $\mathbb{I}$−CDT for the regular and quadtree-based grids constructed from the *lena* image. For each distance

map, the $\mathbb{I}$−CDT of a cell d is represented with a color c so that $c=d \bmod 255$. In this figure, we also depict an elevation map associated with each case. The X and Y axis of this 3-D plot correspond to the axis of the 2-D image, and the Z axis stands for the squared Euclidean distance computed for each cell. Thanks to the elevation maps, we can notice that the distance information is roughly preserved between regular and irregular representations. For example, the global maximum distance value stands in the shoulder of *lena* in both maps. In Table 2, we now show execution times of the four algorithms for the regular square case, hexagonal grids, quadtree and RLE based $\mathbb{I}$−grids. We recall that we do not need to sort the cells to construct the list structures in the regular and RLE cases. For the quadtree based $\mathbb{I}$−grid, this $\preceq_y$ relation has been computed in $\mathcal{O}(n\log n)$ time, but is negligible in respect of our method execution time. This implies that our contribution is always faster than Algorithms 1 and 2. The localization stage in the complete VD can be very complex if we consider regular grids (35 s for the image *canon*), and Algorithm 2 is not computationally efficient, according to the irregular matrix size. Algorithm 4 is slightly slower than Algorithm 3 in the regular square case. The approach described in [22] stands as a reference for computing the E^2DT, as stated in [14]. But even for a very large image, our new contribution computes the $\mathbb{I}$−CDT in less than 0.5 s (0.3 s for *canon* image). Thanks to those experiments, we show that our contribution is very efficient for every classical $\mathbb{I}$−grids in imagery, and also a fast algorithm.

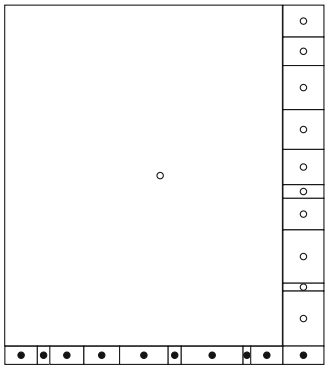


Fig. 7. Example of an input labeled $\mathbb{I}$−grid, leading to the worst case complexity for our algorithm. In this situation, the first list $\mathcal{L}_{r_1}$ do not change during the sweep-line process (no other couple of Voronoi sites hide the points in $\mathcal{L}_{r_1}$). In this example, we have $n_l \times \max_i(\#\mathcal{E}_{r_i}) = 11 \times 10 \gg n$, since $n=21$.

4.2. Comparison with a random generating process

In this section, we are interested in treating $\mathbb{I}$−grids which we do not suppose any underlying order. We thus have to first sort the cells to apply our new algorithm. We introduce now a randomized process to generate labeled $\mathbb{I}$−grids, with an increasing number of cells. This recursive procedure is based on a k d-tree [1], and consists in repeating the subdivision of cells into two sub-cells. This subdivision is performed by cutting a cell with a line aligned alternatively with the X and Y axis. In a general manner, we take into account three main parameters to define our system:

- l_{max} is the maximal number of recursion, i.e. the height of the k d-tree.
- p_{leaf} is the probability (with respect to an uniform law) that a cell is a leaf in the k d-tree, i.e. is not subdivided.
- b_{mid} is a Boolean which indicates if we always choose to subdivide a cell into two cells with the same size or not. If $b_{mid}=0$, then we randomly determine the position of the line that cuts a cell into two.

Table 1
The four compared algorithms, and their associated time and space complexities.

Id.	Algorithm	Time	Space
1	Complete VD [40]	$\mathcal{O}(n\log n_B)$	$\mathcal{O}(n)$
2	From Saito et al. [30,40]	$\mathcal{O}(n_1 n_2 \log n_2)$	$\mathcal{O}(n_1 n_2)$
3	From Maurer et al. [22,42]	$\mathcal{O}(n_1 n_2)$	$\mathcal{O}(n_1 n_2)$
4	New contribution	Worst: $\mathcal{O}(n_1 n_2)$, best: $\mathcal{O}(n)$	$\mathcal{O}(n)$

Fig. 9. The three binary images we consider in our tests: *noise* (a), *lena* (b) and *canon* (c).

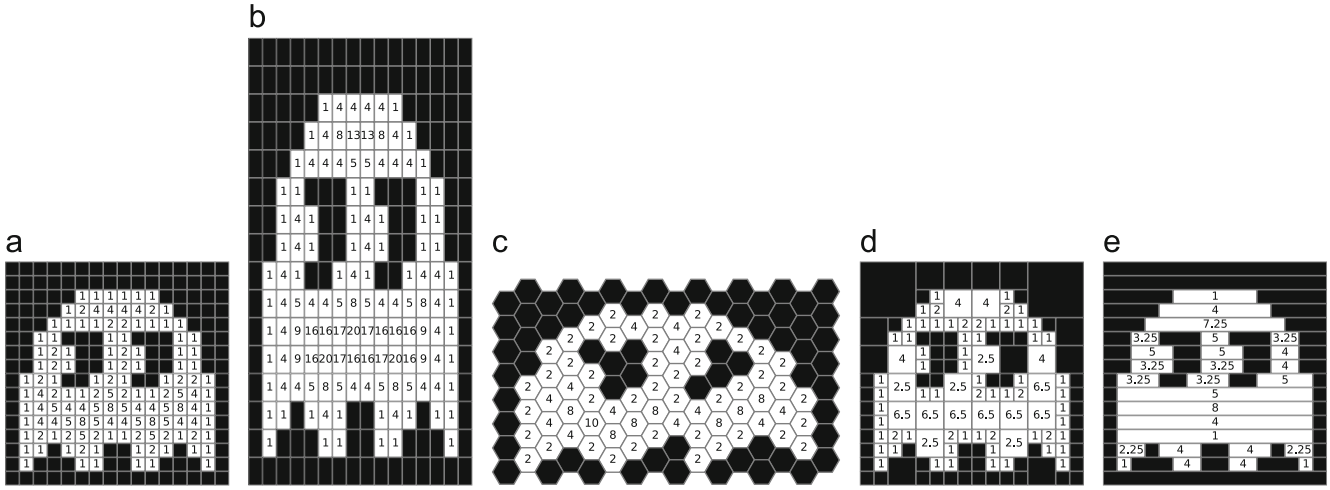


Fig. 8. Results of the $\mathbb{I}$−CDT for classical 2-D grids constructed from the image *ghost*: (a) regular square grid, (b) rectangular grid, (c) hexagonal grid, (d) quadtree-based grid, and (e) a grid built with a RLE along X axis.

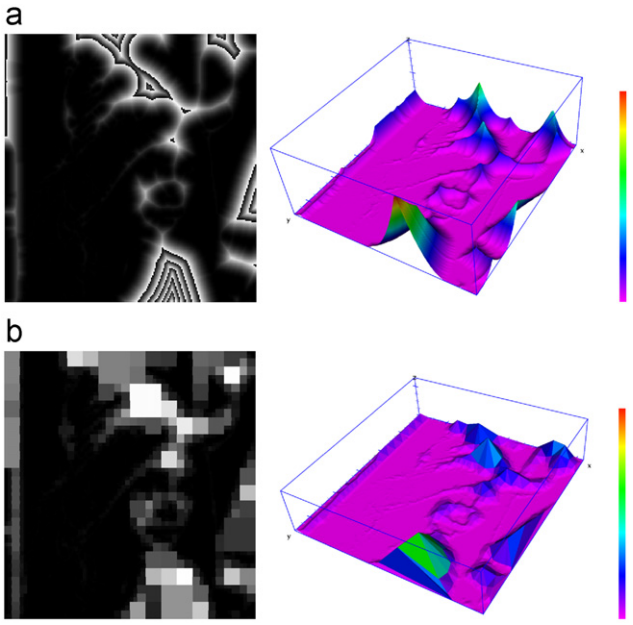


Fig. 10. Distance maps and elevation maps computed by our $\mathbb{I}$–CDT algorithm for classical 2-D $\mathbb{I}$–grids built from *lena* image (a) regular square grid, (b) quadtree-based grid.

Table 2
Execution times (in seconds) of the four tested algorithms for the $\mathbb{I}$–grids built from binary images.

	Alg. 1	Alg. 2	Alg. 3	Alg. 4
(a) Regular grids				
noise	0.228	0.019	0.005	0.011
lena	1.278	0.167	0.029	0.032
canon	35.341	2.025	0.213	0.337
(b) Hexagonal grids				
noise	0.076	0.018	0.007	0.006
lena	0.394	0.186	0.031	0.019
canon	9.189	2.210	0.228	0.217
(c) Quadtree grids				
noise	0.056	0.032	0.014	0.011
lena	0.071	0.323	0.053	0.016
canon	0.035	1.207	0.242	0.022
(d) RLE grids				
noise	0.035	0.027	0.010	0.005
lena	0.029	0.200	0.037	0.006
canon	0.014	1.291	0.178	0.009

We can notice that we do not control the cells value in our process. In fact, we have noticed that the number of background and foreground cells in the grid does not imply important variations in the execution times of the tested methods. In Table 3, we sum up the three configurations we have chosen for our experiments. For each of these configurations, we have generated 1000 grids with an increasing level $l_{max}=5,6,\ldots,14,15$, and we have kept the execution time of the three $\mathbb{I}$–CDT algorithms 1, 3 and 4 (see Table 1) for each grid. We do not consider Algorithm 2 anymore, because its execution times would be obviously greater than the ones of Algorithm 3, as we have shown experimentally in the previous section. For each level l_{max}, and for the method $i=\{1,3,4\}$, the mean time $\mu_i(l_{max})$ and the standard deviation $\sigma_i(l_{max})$, over the 1000 tests, are computed. In Fig. 11, we depict examples of input grids of our system. For each configuration, and for the three methods, we also plot the curves $y_i(l_{max})=\mu_i(l_{max})$ (plain lines) and $y_i(l_{max})=\mu_i(l_{max})\pm\sqrt{\sigma_i(l_{max})/1000}$ (dashed lines for Algorithm 1, dotted lines for Algorithm 3 and dash–dot–dash lines for Algorithm 4).

In the case of the first configuration (chaotic k d-trees), we can notice that Algorithms 1 and 4 have similar performances, while Algorithm 3 is the slowest one. Algorithm 4 is slightly slower than Algorithm 1 (for the highest level $l_{max}=15$, with a factor about 1.5 times). We can also confirm this remark for any $p_{leaf}>0$ and $b_{mid}=0$. For the two other configurations, our new algorithm is clearly faster than the complete VD based method. In particular, for the second configuration (classical k d-trees), and $l_{max}=15$ the mean time is 0.4 and 0.07 s for Algorithms 1 and 4, respectively. Finally, for the third configuration, we have alternatively square and rectangular grids (when l_{max} is even or odd, respectively). In this case, we have measured for example that the mean time of the highest level of generation is 0.7 s for the first algorithm, and 0.07 for our new contribution. We could make similar remarks for any value of $p_{leaf}>0$, when $b_{mid}=1$. Algorithm 3 has the same behaviour as Algorithm 4 in the third configuration, which implies that those methods run in linear-time (with respect to the number of cells n). With these experiments, we show that even if we have to sort the cells of the treated $\mathbb{I}$–grid, our algorithm is generally more convenient to compute the $\mathbb{I}$–CDT than other approaches we developed in previous work. This is a fast algorithm that handles various configurations of complex $\mathbb{I}$–grids.

4.3. Application to medial axis extraction on $\mathbb{I}$–grids

In [38], we presented a separable algorithm to extract a reduced medial axis from a shape digitized on a 2-D labeled $\mathbb{I}$–grid. From the distance map obtained from an $\mathbb{I}$–CDT process, as the one we present in this article, we then compute such a medial axis by computing the upper envelope of a set of parabolas, as in the regular square case [8,30].

We have chosen three different binary images that we have digitized on a regular square grid, an elongated grid where cells are 1.3 longer along the Y axis, and a hexagonal grid. As in the previous experiments, even if we have supposed that we treat $\mathbb{I}$–grids, we are able to handle the latter case, where we consider the hexagons centers instead of rectangular cell centers. In Fig. 12, we present the grid considered for each case, and for each image (black and white cells). We also illustrate the computed MA with white points, which is a set of nodes of the irregular matrix associated with the input grid. We can notice that it may contain many parasite points (consider for example the elongated grid for *cartoon eye* image). It is clear that a further simplification process should be adapted to $\mathbb{I}$–grids to delete these elements. The construction of a *minimal* MA is an NP-complete process on regular grids [7]. We could adapt classical minimal MA algorithms (as [2]) to $\mathbb{I}$–grids to remove these parasite points.

We now present an application of our algorithm for computing an adaptive MA. We consider a quadtree decomposition of the binary images of Fig. 12 (with a finer resolution), with an increasing level of subdivision. In Fig. 13, for each level of decomposition, we present the reduced MA in the same way as in Fig. 12. We are able to compute a progressive MA, becoming more and more precise (i.e. which contains more and more points converging to the center of the shape) as the level of the quadtree increases.

5. Conclusion and future work

In this article, we have proposed a fast algorithm to compute the $\mathbb{I}$–CDT of a labeled $\mathbb{I}$–grid. Thanks to a structure of

Table 3
The three configurations of our generation of labeled ℓ−grids parameters.

Id.	p_{leaf}	b_{mid}	Comments
1	0.30	0	Chaotic k d-tree, where the centers of—very variable—cells are not aligned
2	0.05	1	Classical k d-tree, centers of cells can be aligned and easily ordered
3	0.00	1	Regular square and rectangular grids, where cells are naturally ordered

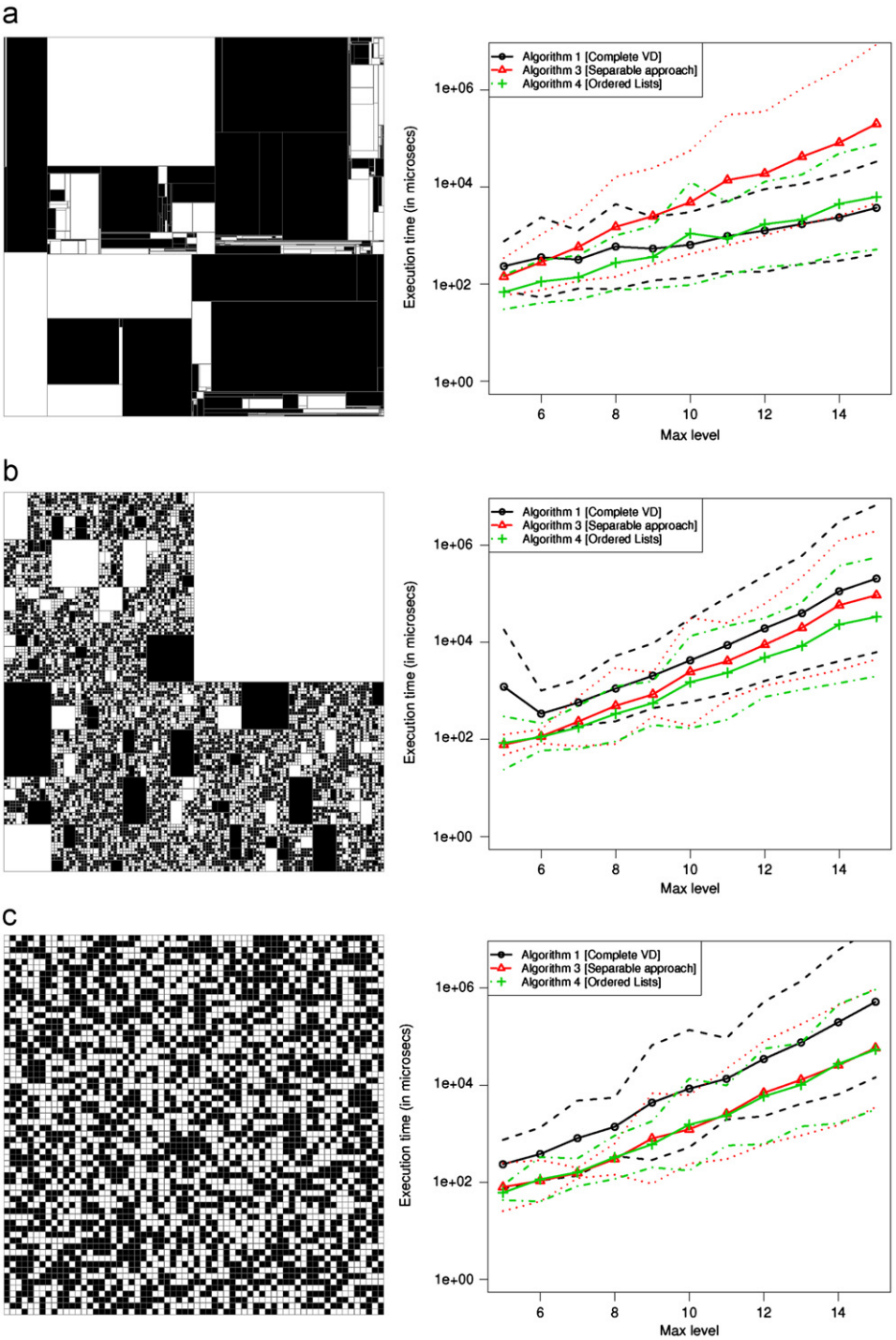


Fig. 11. We depict an example of a high level ℓ−grid generated with the three configurations we have enumerated in Table 3. We also plot the mean time and the associated standard deviation all over the tests, for Algorithm 1 (circles and dashed lines), Algorithm 3 (triangles and dotted lines), and Algorithm 4 (crosses and dash-dot-dash lines): (a) Configuration 1, (b) Configuration 2, and (c) Configuration 3.

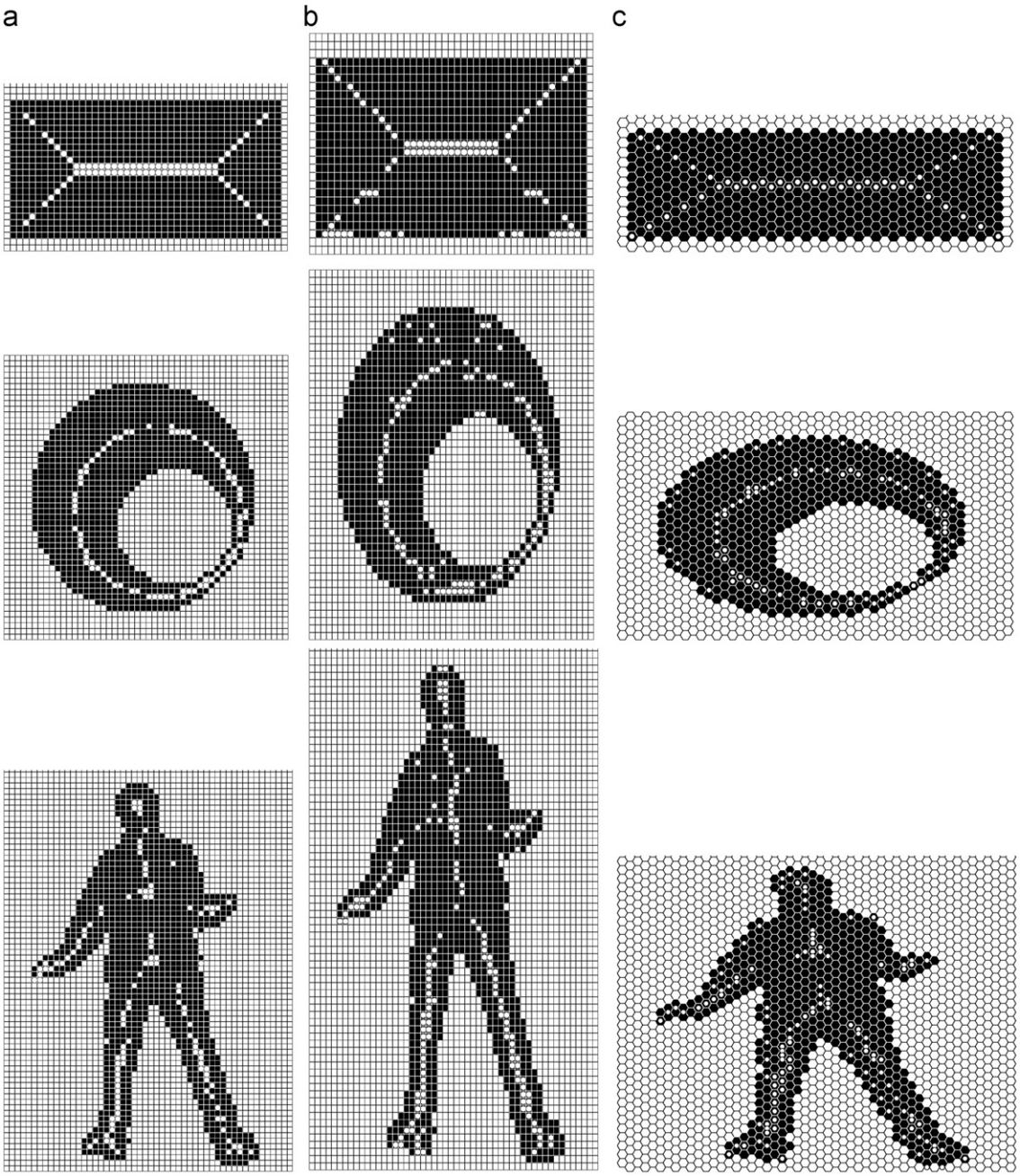


Fig. 12. The reduced discrete medial axis obtained for regular square grids (a), rectangular grids (b) and hexagonal grids (c) built from three different images: *black board* (top), *cartoon eye* (center) and *tracked human* (bottom).

ordered lists of cell centers, its time complexity may be optimal and linear in the number of cells of the grid, in the best case. We have presented various and relevant experiments to show that our contribution is more efficient than algorithms we developed in previous works. We can also notice that the order of the cells in an $\mathbb{I}$−grid $\mathbb{I}$ can be either naturally deduced from the structure of $\mathbb{I}$ (this is the case of the regular grids), or easily determined when there is an underlying data structure (e.g. quadtree, *k* d-tree based grids or hierarchy of grids).

We would like to keep on studying the computation of a medial axis of an object digitized on a labeled $\mathbb{I}$−grid. We clearly search for the fastest algorithms to handle this transformation, as we have presented here to compute the $\mathbb{I}$−CDT. The extension of the algorithm of Breu [3] to three-dimensional (3-D) binary images was efficiently handled by Maurer [22]. We would like to make an equivalent adaptation of our approach in higher dimensions, to treat 3-D labeled $\mathbb{I}$−grids, and propose a 3-D medial axis representation of irregular objects.

The $\mathbb{I}$−CDT definition (Eq. (3)) implies that we only consider the cells centers to compute the distance between two cells in the grid. In [42], we have noticed that this model strongly depends on the background representation, and is not clearly convenient for shape analysis on very irregular structures. Hence, we introduced a new extension of the E^2DT on $\mathbb{I}$−grids that take into account the borders of the cells, and we proposed a DT algorithm, which is not dependent on the background encoding. This is an extension of the DT proposed by Samet [31,32] (used in the framework of a chessboard DT that refers to the underlying regular square grid) on every $\mathbb{I}$−grids. In this case, we have to compute the distance between a foreground cell center p and the frontier between the

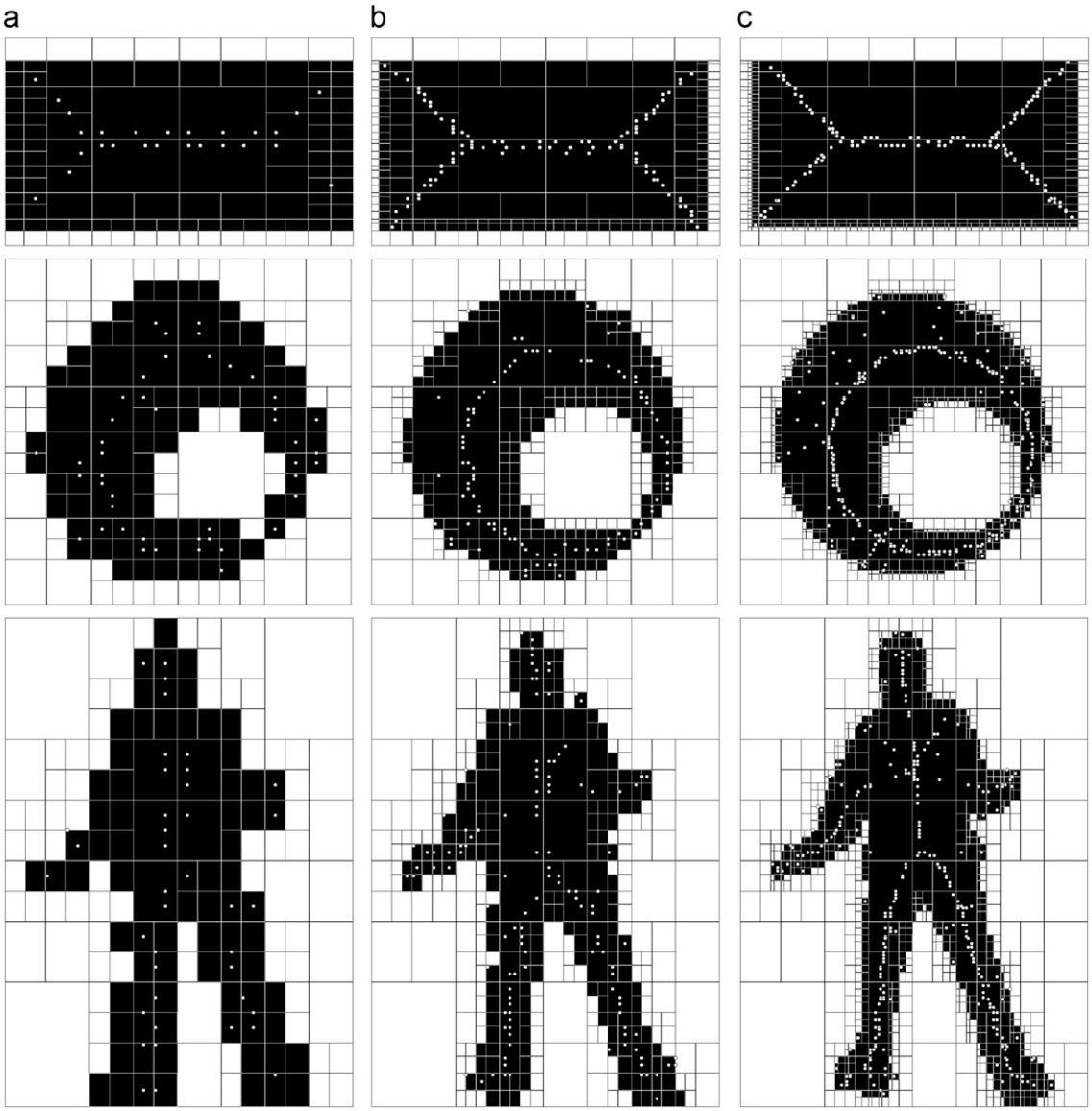


Fig. 13. The reduced discrete medial axis obtained for our three sample images digitized with a quadtree decomposition scheme of level 4 (a), 5 (b) and 6 (c).

object containing it and the background. We would like to develop fast algorithms for DT and MA extraction according to this new definition.

Finally, we have shown in this article that our algorithm may be applied for adaptive grids that represent an interesting tool to develop high-performance algorithms. Since they are group of grids with various resolutions, the DT algorithms can be easily designed on them and may be significantly improved.

Acknowledgment

The author would like to thank David Coeurjolly and Laure Tougne for their constructive help during the preparation of this manuscript.

References

[1] J.L. Bentley, Multidimensional binary search trees used for associative searching, Communication on the ACM 18 (9) (1975) 509–517.

[2] G. Borgefors, Distance transformations in digital images, Computer Vision, Graphics, and Image Processing 34 (3) (1986) 344–371.

[3] H. Breu, J. Gil, D. Kirkpatrick, M. Werman, Linear time Euclidean distance algorithms, IEEE Transactions on Pattern Analysis and Machine Intelligence 17 (5) (1995) 529–533.

[4] F. Cazals, G. Drettakis, C. Puech, Filtering, clustering and hierarchy construction: a new solution for ray-tracing complex scenes, Computer Graphics Forum 14 (3) (1995) 371–382.

[5] Y. Chehadeh, D. Coquin, P. Bolon, A skeletonization algorithm using chamfer distance transformation adapted to rectangular grids, in: 13th International Conference on Pattern Recognition (ICPR'96), vol. 2, 1996, pp. 131–135.

[6] M. Ciuc, D. Coquin, P. Bolon, Quantitative assessment of two skeletonization algorithms adapted to rectangular grids, in: International Conference on Image Analysis and Processing (CIAP'97), vol. 1, 1997, pp. 588–595.

[7] D. Coeurjolly, J. Hulin, S. Sivignon, Finding a minimum medial axis of a discrete shape is np-hard, Theoretical Computer Science 406 (1–2) (2008) 72–79.

[8] D. Coeurjolly, A. Montanvert, Optimal separable algorithms to compute the reverse Euclidean distance transformation and discrete medial axis in arbitrary dimension, IEEE Transactions on Pattern Analysis and Machine Intelligence 29 (3) (2007) 437–448.

[9] D. Coeurjolly, L. Zerarga, Supercover model, digital straight line recognition and curve reconstruction on the irregular isothetic grids, Computers & Graphics 30 (1) (2006) 46–53.

[10] O. Cuisenaire, B. Macq, Fast Euclidean distance transformation by propagation using multiple neighborhoods, Computer Vision and Image Understanding 76 (2) (1999) 163–172.

[11] P.E. Danielsson, Euclidean distance mapping, Computer Graphics and Image Processing 14 (1980) 227–248.
[12] M. de Berg, M. van Kreveld, M. Overmars, O. Schwarzkopf, Computational Geometry: Algorithms and Applications, Springer-Verlag, Berlin, Germany, January 2000.
[13] E.S. Deutsch, Thinning algorithms on rectangular, hexagonal, and triangular arrays, Communications of the ACM 15 (9) (1972) 827–837.
[14] R. Fabbri, L.D.F. Costa, J.C. Torelli, O.M. Bruno, 2D Euclidean distance transform algorithms: a comparative survey, ACM Computing Surveys 40 (1) (2008) 1–44.
[15] C. Fouard, G. Malandain, 3-D chamfer distances and norms in anisotropic grids, Image Vision Computing 23 (2) (2005) 143–158.
[16] S.F. Frisken, R.N. Perry, Simple and efficient traversal methods for quadtrees and octrees, Journal of Graphic Tools 7 (3) (2002) 1–11.
[17] W.H. Hesselink, M. Visser, J. Roerdink, Euclidean skeletons of 3D data sets in linear time by the integer medial axis transform, in: C. Ronse, L. Najman, E. Decencire (Eds.), Computational Imaging and Vision, vol. 30, Springer-Verlag, Netherlands, 2005, pp. 259–268.
[18] D. Jevans, B. Wyvill, Adaptive voxel subdivision for ray tracing, in: Graphics Interface'89, 1989, pp. 164–172.
[19] G. Klette, Skeletons in digital image processing, Technical Report CITR-TR-112, Centre for Image Technology and Robotics of The University of Auckland, 2002.
[20] R. Klette, A. Rosenfeld, Digital Geometry: Geometric Methods for Digital Picture Analysis, Morgan Kaufmann Publishers Inc, San Francisco, USA, 2004.
[21] E. Luczak, A. Rosenfeld, Distance on a hexagonal grid, IEEE Transactions on Computer 25 (1976) 532–533.
[22] C.R. Maurer, R. Qi, V. Raghavan, A linear time algorithm for computing exact Euclidean distance transforms of binary images in arbitrary dimensions, IEEE Transactions on Pattern Analysis and Machine Intelligence 25 (2) (2003) 265–270.
[23] A.J.H. Mehnert, P.T. Jackway, On computing the exact Euclidean distance transform on rectangular and hexagonal grids, Journal of Mathematical Imaging Vision 11 (3) (1999) 223–230.
[24] A. Meijster, J. Roerdink, W.H. Hesselink, A general algorithm for computing distance transforms in linear time, in: Mathematical Morphology and its Applications to Image and Signal Processing, 2000, pp. 331–340.
[25] B. Nagy, A comparison among distances based on neighborhood sequences in regular grids, in: 14th Scandinavian Conference on Image Analysis (SCIA05), 2005, pp. 1027–1036.
[26] D.W. Paglieroni, Distance transforms: properties and machine vision applications, Graphical Models for Image Processing 54 (1) (1992) 56–74.
[27] S. Park, S.S. Lee, J. Kim, The Delaunay triangulation by grid subdivision, in: Computational Science and its Applications, 2005, pp. 1033–1042.
[28] A. Rosenfeld, J.L. Pfaltz, Sequential operations in digital picture processing, Journal of the ACM 13 (4) (1966) 471–494.
[29] A. Rosenfeld, J.L. Pfaltz, Distance functions on digital pictures, Pattern Recognition 1 (1) (1968) 33–61.
[30] T. Saito, J.I. Toriwaki, New algorithms for Euclidean distance transformation of an n-dimensional digitized picture with applications, Pattern Recognition 27 (11) (1994) 1551–1565.
[31] H. Samet, A quadtree medial axis transform, Communications of the ACM 26 (9) (1983) 680–693.
[32] H. Samet, Applications of Spatial Data Structures: Computer Graphics, Image Processing, and GIS, Addison-Wesley Longman Publishing Co., Inc, Massachusetts, USA, 1990.
[33] J.A. Sethian, Level Set Methods and Fast Marching Methods, Cambridge University Press, England, 1999.
[34] I.-M. Sintorn, G. Borgefors, Weighted distance transforms for volume images digitized in elongated voxel grids, Pattern Recognition Letters 25 (5) (2004) 571–580.
[35] R. Strand, Weighted distances based on neighbourhood sequences, Pattern Recognition Letters 28 (15) (2007) 2029–2036.
[36] R. Strand, G. Borgefors, Distance transforms for three-dimensional grids with non-cubic voxels, Computer Vision and Image Understanding 100 (3) (2005) 294–311.
[37] R. Strand, K. Norell, The polar distance transform by fast-marching, in: 19th International Conference on Pattern Recognition (ICPR 2008), 2008, pp. 1–4.
[38] A. Vacavant, D. Coeurjolly, Medial axis extraction on irregular isothetic grids, in: 13th International Workshop on Combinatorial Image Analysis (IWCIA 2009), Progress in Combinatorial Image Analysis, Research Publishing Service, Singapore, 2009, pp. 207–220.
[39] A. Vacavant, D. Coeurjolly, L. Tougne, Topological and geometrical reconstruction of complex objects on irregular isothetic grids, in: 13th International Conference on Discrete Geometry for Computer Imagery (DGCI 2006), Lecture Notes in Computer Science, vol. 4245, Springer, Berlin, Germany, 2006, pp. 470–481.
[40] A. Vacavant, D. Coeurjolly, L. Tougne, Distance transformation on two-dimensional irregular isothetic grids, in: 14th International Conference on Discrete Geometry for Computer Imagery (DGCI 2008), Lecture Notes in Computer Science, vol. 4292, Springer, Berlin, Germany, 2008, pp. 238–249.
[41] A. Vacavant, D. Coeurjolly, L. Tougne, A framework for dynamic implicit curve approximation by an irregular discrete approach, Graphical Models 71 (3) (2009) 113–124.
[42] A. Vacavant, D. Coeurjolly, L. Tougne, A novel algorithm for distance transformation on irregular isothetic grids, in: 15th International Conference on Discrete Geometry for Computer Imagery (DGCI 2009), Lecture Notes in Computer Science, vol. 5810, Springer, 2009, pp. 469–480.
[43] G. Voronoi, Nouvelles applications des paramètres continus la théorie des formes quadratiques deuxième mémoire: Recherches sur les parallélloèdres primitifs, Journal für die reine und angewandte Mathematik 134 (1908) 198–287.
[44] J. Vörös, Low-cost implementation of distance maps for path planning using matrix quadtrees and octrees, Robotics and Computer-Integrated Manufacturing 17 (13) (2001) 447–459.
[45] P. Wiederhold, S. Morales, Thinning on quadratic, triangular, and hexagonal cell complexes, in: 12th International Workshop on Combinatorial Image Analysis (IWCIA'2008), 2008, pp. 13–25.
[46] W. Wong, F. Shih, T. Su, Thinning algorithms based on quadtree and octree representations, Information Sciences 176 (10) (2006) 1379–1394.
[47] P. Zeng, T. Hirata, Distance map based enhancement for interpolated images, in: Theoretical Foundations of Computer Vision, 2002, pp. 86–100.

About the Author—ANTOINE VACAVANT obtained the master's degree from the Université Lyon 1, France, in 2005, and the PhD degree in computer science from the Université Lumière Lyon 2 in 2008. He is a teaching assistant at Université Lumière Lyon 2. His main research topics are discrete geometry, image analysis, and computer graphics.