Distance Transformation on Two-Dimensional Irregular Isothetic Grids

Antoine Vacavant[1,⋆], David Coeurjolly[2], and Laure Tougne[1]

[1] LIRIS - UMR 5205, Université Lumière Lyon 2
5, avenue Pierre Mendès-France
69676 Bron cedex, France
[2] LIRIS - UMR 5205, Université Claude Bernard Lyon 1
43, boulevard du 11 novembre 1918
69622 Villeurbanne cedex, France
{antoine.vacavant,david.coeurjolly,laure.tougne}@liris.cnrs.fr

Abstract. In this article, we propose to investigate the extension of the E^2DT (squared Euclidean Distance Transformation) on irregular isothetic grids. We give two algorithms to handle different structurations of grids. We first describe a simple approach based on the complete Voronoi diagram of the background irregular cells. Naturally, this is a fast approach on sparse and chaotic grids. Then, we extend the separable algorithm defined on square regular grids proposed in [22], more convenient for dense grids. Those two methodologies permit to process efficiently E^2DT on every irregular isothetic grids.

1 Introduction

The definition of discrete distances is a very important concept in image analysis and shape description [20,21]. Here, we are interested in the definition of a distance and its application on Irregular Isothetic Grids ($\mathbb{I}$-grids for short) [5], where the cells are rectangles defined by variable positions and sizes, and may be determined by subdivision rules. Those grids are very common, and permit to represent an image in a more compact and adapted manner. Here, we will use two classical $\mathbb{I}$-grids: the Quadtree decomposition [23] and the Run-Length Encoding (or RLE) [11]. In our study, we have chosen to consider that the distance between two cells in a two dimensional (2-D) $\mathbb{I}$-grid is the distance between their centers (which is the same as the regular square grid, where we compute the distance between points of $\mathbb{Z}^2$). The squared Euclidean Distance Transformation (E^2DT) of a binary image is a tool that has been largely investigated for decades, and represents a very common way to analyze the shape of graphical objects, for various applications (see [18] and references of [16] for more details). The purpose of this process is to label each (*foreground*) cell of an object with the distance to the closest cell of its complement (or *background*). Since we consider

⋆ The authors would like to thank Pr. Annick Montanvert for her constructive suggestions during the preparation of this article.

D. Coeurjolly et al. (Eds.): DGCI 2008, LNCS 4992, pp. 238–249, 2008.

the distance between the center of grid cells, this process can be naturally linked with the computation of the Voronoi Diagram (VD) [6,19]. Moreover, we can notice that the VD may also be extracted from the distance map [13,22].

The E^2DT of a binary image generally considers it as a regular square grid. Many studies have aimed to develop a special Distance Transformation (DT) for non-square grids, but these approaches can not be extended to every $\mathbb{I}$-grids. In fact, we want to propose a global model which could be then applied to each sort of $\mathbb{I}$-grid. The DT of Quadtree or Octree based grids [14,23,26] are dependent on the specific structure of the concerned trees. Those approaches compute the DT by propagating distance values from parent nodes to their children nodes. To handle medical images digitized on elongated grids, where the cells are longer along an axis, a lot of methodologies that perform the chamfer distance have been adapted [4,7,9,25]. To use the same technique on an $\mathbb{I}$-grid $\mathbb{I}$, we would have to extend chamfer masks and change them for each cell of $\mathbb{I}$ (non-stationary computation of the DT). In the same way, the algorithms designed on other non-standard grids [10,27], *e.g.* Face-Centered Cubic (FCC) or Body-Centered Cubic (BCC) grids, suppose the regularity of the neighbors of a cell, and may not be adapted to every $\mathbb{I}$-grids. Indeed, we make no hypothesis about the configuration (number, position, size, *etc.*) of the neighbors of a cell in $\mathbb{I}$. In the regular square grid case, most VD-based algorithms [2,16,24] build a partial VD to compute the E^2DT and lead to an optimal linear time complexity for the E^2DT computation, in the number of cells of the grid. Since these methods are separable, *i.e.* they perform operations independently along the two axis, they propose a natural extension to handle d-dimensional images. Here, we propose to extend the separable and linear-time algorithm presented in [22] to compute the squared Euclidean distance transform on $\mathbb{I}$-grids (or $\mathbb{I}$-DT). Thus, we propose an original and efficient approach to perform the $\mathbb{I}$-DT, which can be used for every $\mathbb{I}$-grids we have cited before.

In this article, we first introduce discrete distances on $\mathbb{I}$-grids, and we present the algorithm to compute the $\mathbb{I}$-DT by implementing the complete VD. For some sparse irregular grids, the complete VD based approach seems to be the best way to compute the $\mathbb{I}$-DT. In Section 3, we give details about our extension of [22] on $\mathbb{I}$-grids. Indeed, thanks to a data structure to represent every $\mathbb{I}$-grids, we insure that the $\mathbb{I}$-DT is then error-free, and this allows us to fix an upper bound for the complexity of this method. Then, we propose to compare the speed and the complexity of our method with respect to the direct approach based on the complete VD. We thus propose to measure the performance of the two algorithms, to study what kind of grids they efficiently handle. We finally discuss the applications and possible extensions of our approach.

2 A Simple Approach Based on the Complete Voronoi Diagram Implementation

We first define an $\mathbb{I}$-grid as a tiling of the plane with isothetic rectangles. We shortly recall that each rectangle R (also called *cell*) of $\mathbb{I}$ is defined by its center

$(x_R, y_R) \in \mathbb{R}^2$ and a size $(l_R^x, l_R^y) \in \mathbb{R}^2$. The position and the size of R may be controlled by different level of constraints [5]. Each cell of the grid is also associated with a foreground label (1) or a background label (0). Here, we denote $\mathbb{I}_F$ the set of foreground cells in the grid, and $\mathbb{I}_B$ the set of background ones. The sizes of those sets are denoted n_F and n_B respectively. The distance between two cells R and R' is the distance between their centers. If we denote p (respectively p') the center of a cell R (respectively R'), we have

$$d^2(R, R') = d^2(p, p') = (x_R - x_{R'})^2 + (y_R - y_{R'})^2 \tag{1}$$

for the square of the Euclidean distance. For a cell $R \in \mathbb{I}$, the squared Euclidean distance transformation on $\mathbb{I}$-grids ($\mathbb{I}$-DT) is given by:

$$\mathbb{I}\text{–DT}(R) = \min_{R'} \left\{ d^2(R, R');\ R' \in \mathbb{I}_B \right\} \tag{2}$$

and is exactely the E^{2}DT if we consider a regular square grid $\mathbb{I}$. This irregular discrete distance computed between cell centers can be applied in Geographical Information Systems (GIS), where an irregular spatial structure [1,23] permits to quickly locate points and to compute the distance between them. We could also choose to extend the distance proposed by H. Samet [23] (used in the framework of a chessboard distance transformation, that refers to the underlying regular square grid) on every $\mathbb{I}$-grids . In this case, we would have to compute the distance between a foreground cell center p and the frontier between the object containing it and the background. Let $\mathcal{S}$ be the set of segments so that the intersection between a cell $R \in \mathbb{I}_F$ and a cell $R' \in \mathbb{I}_B$ adjacent with R belongs to $\mathcal{S}$. In this case, the $\mathbb{I}$-DT should be defined as follows:

$$\mathbb{I} - \text{DT}(R) = \min_{\alpha} \left\{ d^2(p, \alpha);\ \alpha \in \mathcal{S} \right\}. \tag{3}$$

Contrary to the $\mathbb{I}$-DT given in Equation 2, this process does not take into account the representation of the background. More precisely, we do not consider the centers of the backgound cells of $\mathbb{I}$.

In this article, we propose a VD-based algorithm to compute the $\mathbb{I}$-DT defined in Equation 2. The first step of this algorithm is to compute the VD where the sites are the centers of the background cells in $\mathbb{I}_B$. Then, we perform the $\mathbb{I}$-DT of each foreground cell R by locating its center p in the VD. We search for one of the nearest Voronoi site s to finally compute the $\mathbb{I}$-DT of R (see Algorithm 1 for more details). We show in Figure 1 examples of results obtained with a small binary image digitized with a regular square grid, a Quadtree grid, and a RLE along X grid. We have chosen those two $\mathbb{I}$-grids because they are common in image analysis applications, but we could also consider Kd-tree based grids [1] for example. The Voronoi Diagram $\mathcal{V}$ is computed thanks to the `CGAL` library [3,15], and has an optimal time complexity $\mathcal{O}(n_B \log n_B)$ and fill $\mathcal{O}(n_B)$ space [8]. Then, the main loop of Algorithm 1 consists in locating each foreground cell P in $\mathcal{V}$ (with its center p), and in searching for its nearest Voronoi site s in $\mathcal{V}$. The location query is proved to have a $\mathcal{O}(\log n_B)$ complexity [8]. Thus, we perform this loop in $\mathcal{O}(n_F \log n_B)$ time. Indeed, searching for the nearest site

Algorithm 1. $\mathbb{I}$-DT based on the complete Voronoi diagram

input : A labelled $\mathbb{I}$-grid $\mathbb{I}$.
output: The $\mathbb{I}$-DT of $\mathbb{I}$.

```
1  Compute the Voronoi Diagram V of the points {p; R ∈ I_B};
2  foreach cell R ∈ I_F do
3      Locate p in V;
4      if p belongs to a Voronoi vertex v of V then
5          s := Voronoi site of an adjacent cell of v;
6      else if p belongs to a Voronoi edge e in V then
7          s := Voronoi site of an adjacent cell of e;
8      else
           // p belongs to a Voronoi cell c of V
9          s := Voronoi site of c;
10     I-DT(R) := d^2(s, p);
11 foreach cell R ∈ I_B do
12     I-DT(R) := 0;
```

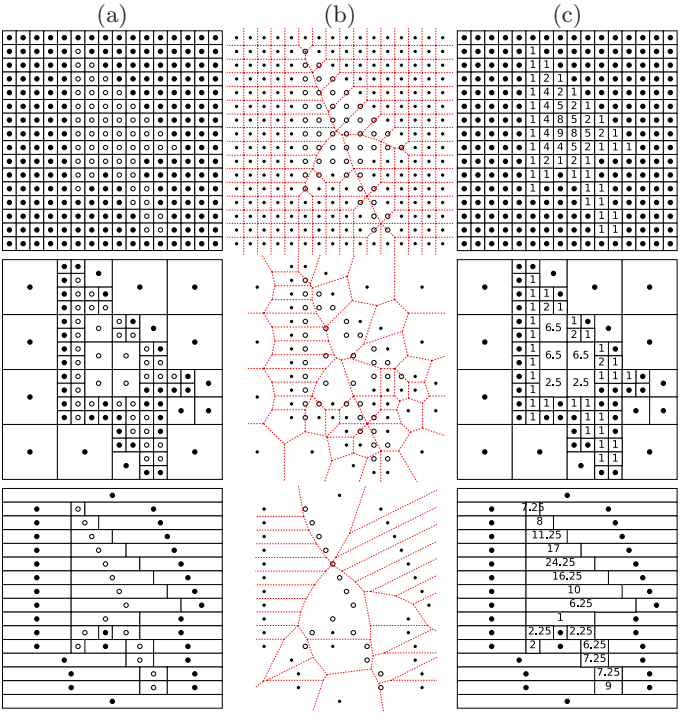

Fig. 1. Examples of results of Algorithm 1 with a small binary image of size 16 x 16. The image *cursor* is digitized with a regular square grid, a Quadtree decomposition, and a RLE along the X axis *(a)*. Background points are illustrated in black, and foreground ones in white. We present the Voronoi diagram for each case in dotted lines *(b)*, and the final distance map for the d^2 distance *(c)*.

s is processed in constant time, in the three cases: p is inside a Voronoi cell, p stands on a Voronoi edge, and p is on a Voronoi vertex. When p stands on a vertex or on an edge of $\mathcal{V}$, the choice of the Voronoi cell containing s is arbitrary. Thus, Algorithm 1 is performed in $\mathcal{O}(N \log n_B)$ time and $\mathcal{O}(N)$ space, where $N = n_B + n_F$ is the total number of cells in $\mathbb{I}$.

3 A *d*-Dimensional Algorithm for $\mathbb{I}$-DT

We first propose a data structure to represent any $\mathbb{I}$-grids, and to simplify the cell scanning. An *irregular matrix* $\mathbf{A}$ associated to the labelled $\mathbb{I}$-grid $\mathbb{I}$ is built by organizing aligned cells along X and Y axis. The value of a node in $\mathbf{A}$ is fixed according to two cases: (1) $\mathbf{A}(i,j)$ is the center of a cell in $\mathbb{I}$, this node is a foreground node ($\mathbf{A}(i,j) = 1$) or a background node ($\mathbf{A}(i,j) = 0$); (2) it does not correspond to a cell center in $\mathbb{I}$, and we set it as a foreground *extra node* ($\mathbf{A}(i,j) = 1$). Those extra nodes permit to compute the $\mathbb{I}$-DT between the centers of the cells (see Figure 2 for more details). More precisely, we create as many columns as the different X-coordinates of the cell centers of $\mathbb{I}$. We make the same with the columns of $\mathbf{A}$, when we consider the Y-coordinate of the cell centers. We store the X-coordinates and Y-coordinates in two tables T_X and T_Y, and we denote $n_X = |T_X|$ and $n_Y = |T_Y|$ the number of columns and rows of $\mathbf{A}$. So, the X-coordinate of $\mathbf{A}(i,j)$ is $T_X(i)$. We propose to extend the separable

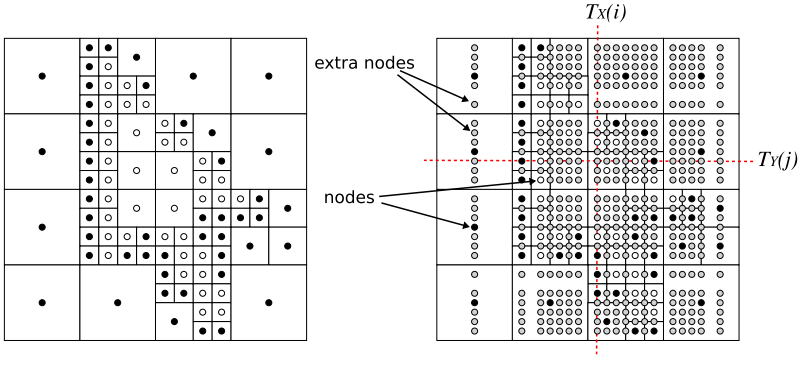

Fig. 2. The irregular matrix $\mathbf{A}$ associated with the image *cursor*, digitized on a Quadtree grid. A node $\mathbf{A}(i,j)$ is depicted as the intersection of the dotted lines, and extra nodes are filled in light grey.

algorithm described in [22] and we adapt it on the irregular matrix. The $\mathbb{I}$-DT of an $\mathbb{I}$-grid $\mathbb{I}$ given in Equation 2 is represented by the irregular matrix:

$$\mathbf{C}(i,j) = \min_{x,y} \big\{ (T_X(i) - T_X(x))^2 + (T_Y(j) - T_Y(y))^2; \\ x \in \{0,...,n_X - 1\},\ y \in \{0,...,n_Y - 1\},\ \mathbf{C}(x,y) = 0 \big\}. \quad (4)$$

This min operation is processed over all the elements of $\mathbf{C}$ (nodes and extra nodes). To compute this $\mathbb{I}$-DT, our algorithm may be decomposed into two steps:

1. Let $\mathbf{A}$ be the irregular matrix built from the $\mathbb{I}$-grid $\mathbb{I}$. We perform here a one-dimensional $\mathbb{I}$-DT along X axis, stored in the irregular matrix $\mathbf{B}$ such that:

$$\mathbf{B}(i,j) = \min_{x} \big\{ |T_X(i) - T_X(x)|;\ x \in \{0,...,n_X - 1\},\ \mathbf{A}(x,j) = 0 \big\}. \quad (5)$$

2. We perform then a Y axis process to build the final irregular matrix $\mathbf{C}$:

$$\mathbf{C}(i,j) = \min_{y} \big\{ \mathbf{B}(i,y)^2 + (T_Y(j) - T_Y(y))^2;\ y \in \{0,...,n_Y - 1\} \big\}. \quad (6)$$

We also present in Algorithm 2 the two steps of our approach. In the first step, we can notice that the only difference with the regular square case [6,22] is the computation of the distance, lines 2 and 2. Indeed, we have to consider in those operations the distance between the point $\mathbf{B}(i,j)$ and its neighbor (*e.g.* $|T_X(i) - T_X(i-1)|$ in line 2). The second step of our algorithm is in fact the computation of the lower envelope of a set of parabolas [6]. After Step 1, we can consider the set of parabolas $\mathcal{F}^i_y(j) = \mathbf{B}(i,y)^2 + (T_Y(j) - T_Y(y))^2$ on the column $\{\mathbf{B}(i,y)\}_{0 \le y \le n_Y}$. With Step 2, the column $\{\mathbf{C}(i,y)\}_{0 \le y \le n_Y}$ is the lower envelope of the set $\{\mathcal{F}^i_y\}_{0 \le y \le n_Y}$. The function $Sep^i(u,v)$ is the exact coordinate of the intersection point between two parabolas [6,17] (we will simply denote $\mathcal{F}_y(j) = \mathcal{F}^i_y(j)$ and $Sep(u,v) = Sep^i(u,v)$ when the parameter i is fixed):

$$Sep^i(u,v) = \big(T_Y(v)^2 - T_Y(u)^2 + \mathbf{B}(i,v)^2 - \mathbf{B}(i,u)^2\big) / \big(2(T_Y(v) - T_Y(u))\big). \quad (7)$$

Algorithm 2. $\mathbb{I}$-DT by a separable approach

```
   input : A labelled I-grid I.
   output: The I-DT of I, stored in an irregular matrix C.
2  Build the irregular matrix A associated to I;
   // Step 1 along X-axis
3  for j = 0 to n_Y − 1 do
4      if A(0, j) = 0 then
5          B(0, j) := 0;
6      else
7          B(0, j) := ∞;
8      for i = 1 to n_X − 1 do
9          if A(i, j) = 0 then
10             B(i, j) := 0;
11         else
12             B(i, j) := |T_X(i)−T_X(i−1)|+
                          B(i − 1, j);
13     for i = n_X − 2 to 0 do
14         if B(i + 1, j) < B(i, j) then
15             B(i, j) := |T_X(i)−T_X(i+1)|+
                          B(i + 1, j);
17
   // Step 2 along Y-axis
19 for i = 0 to n_X − 1 do
20     q := 0; s[0] := 0; t[0] := 0;
21     for j = 1 to n_Y − 1 do
22         while q ≥ 0 ∧ F_{s[q]}(t[q]) > F_j(t[q])
           do
23             q := q − 1;
24         if q < 0 then
25             q := 0; s[q] := j;
26         else
27             w := Sep(s[q], j);
28             if w ≤ T_Y(n_Y − 1) then
29                 Find the node B(i, k), k ∈
                   {s[q], ..., n_Y − 1} such
                   that T_Y(k) > w;
30             q := q+1; s[q] := k; t[q] := w;
31 for j := n_Y − 1 to 0 do
32     C(i, j) := F_{s[q]}(j);
33     if T_Y(j) = t[q] then
34         q := q − 1;
```

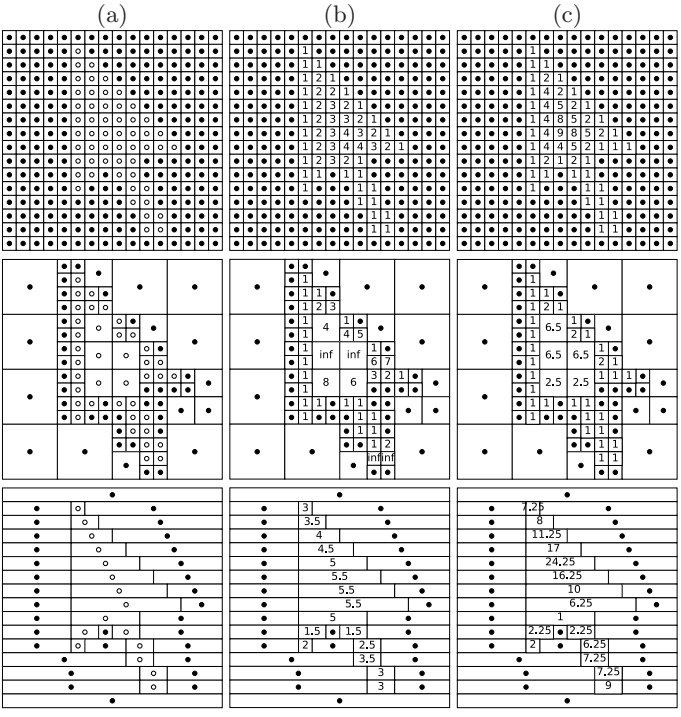

Fig. 3. We present the temporary distance map obtained with Step 1 *(b)* on the image *cursor* digitized with different $\mathbb{I}$-grids *(a)*. We can notice that the *inf* node means that no background node exists on the row containing this point. So, the last *for* loop in Algorithm 2 line 2 does not change $\mathbf{B}(i,j) = \infty$. The final distance transformation is depicted as Figure 1 *(c)*.

In comparison with the regular grid case, we can see that the operator *div* has been replaced by the floating-point operator / in this equation to compute the exact intersection point. For $\mathbb{I}$-grids computed from a cell subdivision or a cell grouping process, we still have an exact arithmetic division. In those cases, coordinates may be half-integers, and we just have to multiply grid cells coordinates by four to compute the integer intersection point with $Sep()$. The computation of w (line 2) only depends on the function $Sep(u,v)$ and then permits to find the intersection point in $\mathbf{B}$ (line 2). Here, this find command is performed with a dichotomous search through the ordered set of nodes $\{\mathbf{B}(i,k)\}_{s[q] \leq k \leq n_Y - 1}$, and has a $\mathcal{O}(\log n_Y)$ time complexity in the worst case. But in our experiments, we have observed that this is a fast operation, since we begin the search from the last intersection point (with index $s[q]$). In Figure 3, we present some results of Algorithm 2 on the small binary image *cursor* used in Figure 1. To apply the $\mathbb{I}$-DT given in Equation 3 (between a foreground cell and the foreground/background frontier), we would have to link each node and each extra node in the irregular matrix to the cell in $\mathbb{I}$ that contains it. This permits to change equations 4, 5,

and 6 to take into account the size of the cells. For example, if we denote $R_{i,j}$ the cell in $\mathbb{I}$ associated with the node $\mathbf{A}(i,j)$, Equation 5 becomes:

$$\mathbf{B}(i,j) = \min_x \left\{ |T_X(i) - T_X(x)| - \frac{l^x_{R_{x,j}}}{2};\ x \in \{0,...,n_X - 1\},\ \mathbf{A}(x,j) = 0 \right\}. \tag{8}$$

We have presented here a separable algorithm on $\mathbb{I}$-grids. The first operation (build the irregular matrix) is performed in $\mathcal{O}(n_X n_Y)$ time. More precisely, we first scan all the cells of $\mathbb{I}$ to get the n_Y rows and n_X columns of $\mathbf{A}$. Then, we consider each node of $\mathbf{A}$ and assign its value by checking if it coincides with a cell center in $\mathbb{I}$. This algorithm has a global time complexity in $\mathcal{O}(n_X n_Y \log n_Y)$. It can be easily extended to higher dimensions: the Step 1 stands as an initialization step, and for each greater dimension, a mixing process, as Step 2, permits to combine results obtained in the lower dimensions. If we consider a d-dimensional labelled $\mathbb{I}$-grid, the cost of the consecutive steps is in $\mathcal{O}(n^d \log^{d-1} n)$, where the dimension of the irregular matrix $\mathbf{A}$ associated to $\mathbb{I}$ is n^d. The size of $\mathbf{A}$ clearly depends on the organization of the cells of $\mathbb{I}$; a matrix $\mathbf{A}$ built with a regular grid $\mathbb{I}$ would have the same size as $\mathbb{I}$. The more an $\mathbb{I}$-grid has an irregular structure, the more the difference between $n_X n_Y$ and N, the number of cells of $\mathbb{I}$, is important. The space required, in $\mathcal{O}(n_X n_Y)$, is principally occupied by the irregular matrix $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$. Furthermore, when we have implemented this algorithm, we have used only one matrix that stores initial and temporary distance values. Those two elements about the complexity of our contribution will be discussed in the conclusion.

4 Experiments

We first illustrate in Figure 4 the irregular grids we have generated with the sample binary images we have chosen: *canon* which is a big image containing a single binary object, *lena*, a more complex binary image, and finally an image generated with gaussian noise, named *noise*. In Figure 5, we depict the distance maps obtained by our two algorithms processed on those grids and the regular grids. The grey level gl of a cell corresponds to the value of the distance d with a simple modulo operator ($gl = d \mod 255$). We present in Table 2 the time of execution for each algorithm, and in Table 1 the important features for each $\mathbb{I}$-grid (*e.g.* number of background and foreground cells, size of the irregular matrix). We have performed those experiments on a mobile workstation with a 1.5 Ghz Intel Pentium M processor, and 1 Gb RAM. Algorithm 1 is a fast way to compute the $\mathbb{I}$-DT on irregular sparse grids (in our tests, the Quadtree and RLE grids) and is faster than Algorithm 2 on those grids. But, our separable approach is very competitive and fast for the regular grid and the irregular dense grids (Quadtree and RLE grids for the image *noise*). The time of execution of Algorithm 2 is slighlty the same as the original version of [22] in the regular square grid case. Indeed, the irregular matrix and the image that generates the grid have the same size ($n_X \times n_Y = N$), and the only difference

between the two versions of this algorithm is the find operation in the last scan, which is fastly processed (for the image *canon*, the original algorithm takes 0.90 seconds). Thanks to Table 1, we can see that the numbers of background and foreground cells do not significally modify the behaviour of our algorithms. The structure of the considered grid seems to be the only factor that slows down them. Algorithm 1 hardly handles regular grids, and returns the $\mathbb{I}$-DT in more than 1 minute for the image *canon*. On the contrary, irregular sparse grids make the irregular matrix more complex than the initial image structuration for Algorithm 2.

In conclusion, to anticipate the speed of our algorithm, we should consider the size and the density of the irregular grid. If we compare the complexity of our algorithms, it is clear that Algorithm 2 has generally a worse time complexity in $\mathcal{O}(n_X n_Y \log n_Y)$, where n_X and n_Y are the size of the irregular matrix. However, in practice, this approach is faster than Algorithm 1 which time complexity is $\mathcal{O}((n_B + n_F) \log n_B)$, where n_F and n_B are the number of foreground and background cells. This could be explained by the simplicity of our separable algorithm, more precisely two independent scans along the X and Y axis. In Algorithm 1, we have to search for each foreground point p the Voronoi cell where p stands. Even if this operation is bounded at $\mathcal{O}(\log n_B)$, the global execution time suffers from the data structure implemented.

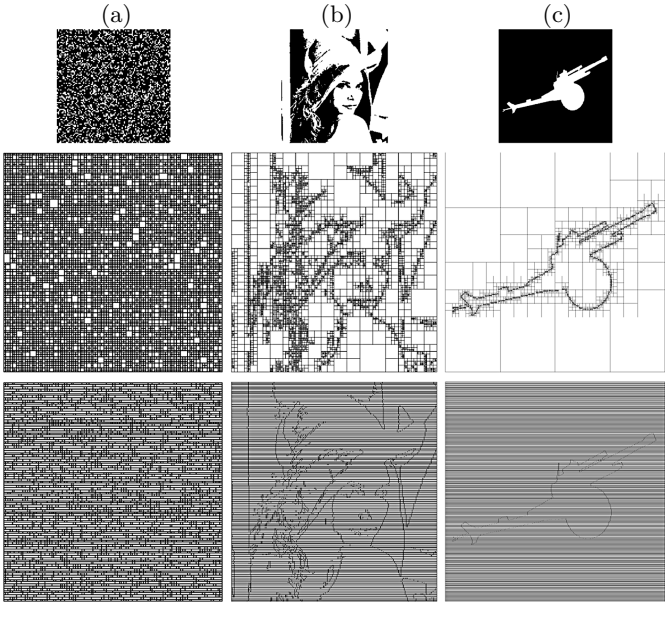

Fig. 4. We depict here the input irregular grids - *(top)*: Quadtree, *(bottom)*: RLE along X axis- we consider for each sample image *(a): noise, (b): lena, (c): canon*

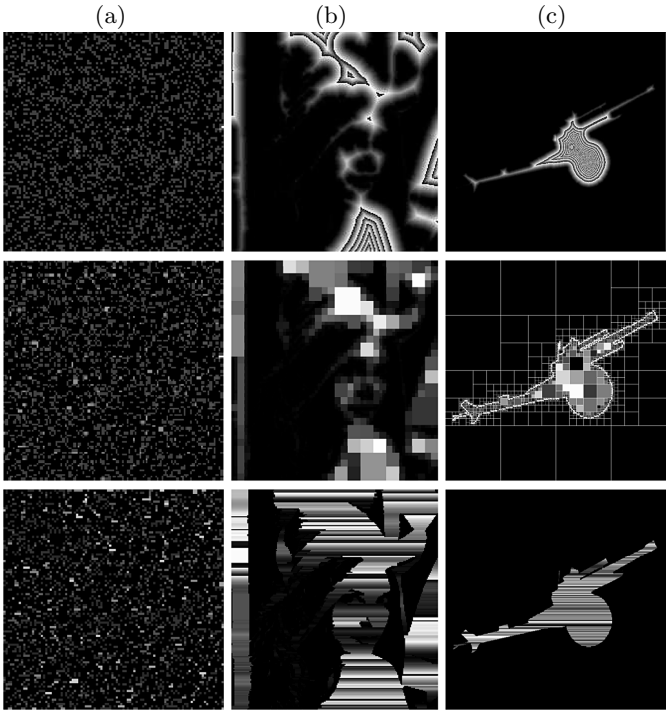

Fig. 5. Distance maps for our sample images, for the square grid *(top)*, the Quadtree grid *(middle)*, and the RLE grid along X axis *(bottom)*

Table 1. For each $\mathbb{I}$-grid, we illustrate the image size, the number of cells, the number of background and foreground cells, and the size of the associated irregular matrix

Image	Image size	$\mathbb{I}$-grid	N	n_B	n_F	$n_X \times n_Y$	$\frac{N}{n_X \times n_Y}$
noise	100×100	Square	10 000	7414	2586	100×100	1.000
		Quadtree	7 457	5 003	2 454	138×138	0.390
		RLE	3 903	1 973	1 930	199×100	0.200
lena	208×222	Square	46 176	18 789	27 387	208×222	1.000
		Quadtree	8 171	4 143	4 028	308×365	0.070
		RLE	3 462	1 695	1 767	392×222	0.040
canon	512×512	Square	262 144	234 302	27 842	512×512	1.000
		Quadtree	4 177	2 162	2 015	920×527	0.010
		RLE	1 432	972	460	577×512	0.005

Table 2. The execution time in seconds for every $\mathbb{I}$-grids

Image	Algorithm 1 (complete VD algorithm)		
	Square	Quadtree	RLE
noise	0.77	0.34	0.22
lena	2.8	**0.36**	**0.22**
canon	104.4	**0.38**	**0.27**

Image	Algorithm 2 (proposed approach)		
	Square	Quadtree	RLE
noise	**0.15**	**0.21**	**0.15**
lena	**0.26**	0.40	0.32
canon	**1.0**	1.2	1.3

5 Conclusion and Future Works

In this article, we have proposed two completing algorithms to compute the $\mathbb{I}$-DT on $\mathbb{I}$-grids. The execution time of our approaches mainly depends on the structure of the grid and on the size of the image treated. The sparser the grid is, the slower the $\mathbb{I}$-DT will be performed thanks to our separable approach (Algorithm 2), and inversely, Algorithm 1 (based on the Voronoi diagram of the background cells) hardly handles dense grids.

In future works, we would like to study the optimization of Algorithm 2 and propose an optimal time process in $\mathcal{O}(N)$ time complexity (instead of $\mathcal{O}(n_X n_Y \log n_Y)$). Indeed, we compute the distance in every nodes of the matrix to propagate the distance values in the rest of the matrix. Since the size of this structure increases with the complexity of the $\mathbb{I}$-grid (see Table 1, last column), we propose to use a list-based data structure to reduce the computational time and memory space of the $\mathbb{I}$-DT [2,12]. We have also depicted applications of our algorithms in GIS, since we compute distance between grid cell centers. We would like to develop the $\mathbb{I}$-DT given in Equation 3. In this case, the representation of the background does not impact the result obtained with the $\mathbb{I}$-DT. We are interested in investigating three dimensions (3-D) $\mathbb{I}$-DT, since we can easily extend our approaches to higher dimensions. To build the medial axis of binary irregular objects, we should make the $\mathbb{I}$-DT reversible and compute the REDT (Reversible Euclidean Distance Transformation) on $\mathbb{I}$-grids, as it was proposed in [6] for the regular square grid. This REDT algorithm is separable and is naturally adapted to handle 3-D objects. Finally, since the irregular matrix permits to conserve a constant number of neighbors for each node, a chamfer mask-based approach (like in [10]) may be adapted to this structure. The main problem is to change the mask and consider the configuration of each node of the matrix during the scan (non-stationary distance computation).

References

1. Bentley, J.L.: Multidimensional Binary Search Trees Used for Associative Searching. Communications of the ACM 18(9), 509–517 (1975)
2. Breu, H., Gil, J., Kirkpatrick, D., Werman, M.: Linear Time Euclidean Distance Algorithms. IEEE Transactions on Pattern Analysis and Machine Intelligence 17(5), 529–533 (1995)
3. CGAL, Computational Geometry Algorithms Library, `http://www.cgal.org`
4. Chehadeh, Y., Coquin, D., Bolon, P.: A Skeletonization Algorithm Using Chamfer Distance Transformation Adapted to Rectangular Grids. In: 13th International Conference on Pattern Recognition (ICPR 1996), vol. 2, pp. 131–135 (1996)
5. Coeurjolly, D.: Supercover Model and Digital Straight Line Recognition on Irregular Isothetic Grids. In: Andrès, É., Damiand, G., Lienhardt, P. (eds.) DGCI 2005. LNCS, vol. 3429, pp. 311–322. Springer, Heidelberg (2005)
6. Coeurjolly, D., Montanvert, A.: 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), 437–448 (2007)
7. Cuisenaire, O.: Distance Transformations: Fast Algorithms and Applications to Medical Image Processing. PhD Thesis, Université Catholique de Louvain, Louvain-La-Neuve, Belgium (October 1999)

8. Devillers, O.: Improved Incremental Randomized Delaunay Triangulation. In: 14th Annual ACM Symposium on Computational Geometry, 106–115 (1998)
9. Fouard, C., Malandain, G.: 3-D Chamfer Distances and Norms in Anisotropic Grids. Image and Vision Computing 23(2), 143–158 (2005)
10. Fouard, C., Strand, R., Borgefors, G.: Weighted Distance Transforms Generalized to Modules and their Computation on Point Lattices. Pattern Recognition 40(9), 2453–2474 (2007)
11. Golomb, S.W.: Run-length Encodings. IEEE Transactions on Information Theory 12(3), 399–401 (1966)
12. Guan, W., Ma, S.: A List-Processing Approach to Compute Voronoi Diagrams and the Euclidean Distance Transform. IEEE Transactions on Pattern Analysis and Machine Intelligence 20(7), 757–761 (1998)
13. Hesselink, W.H., Visser, M., Roerdink, J.B.T.M.: Euclidean Skeletons of 3D Data Sets in Linear Time by the Integer Medial Axis Transform. In: Proceedings of 7th International Symposium on Mathematical Morphology, pp. 259–268 (2005)
14. Jung, D., Gupta, K.K.: Octree-Based Hierarchical Distance Maps for Collision Detection. In: IEEE International Conference on Robotics and Automation, vol. 1, pp. 454–459 (1996)
15. Karavelas, M.I.: Voronoi diagrams in CGAL. In: 22nd European Workshop on Computational Geometry (EWCG 2006), pp. 229–232 (2006)
16. Maurer, C.R., Qi, R., Raghavan, V.: 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), 265–270 (2003)
17. Meijster, A., Roerdink, J.B.T.M., Hesselink, W.H.: A General Algorithm for Computing Distance Transforms in Linear Time. In: Mathematical Morphology and its Applications to Image and Signal Processing, pp. 331–340 (2000)
18. Paglieroni, D.W.: Distance Transforms: Properties and Machine Vision Applications. In: CVGIP: Graphical Models and Image Processing, vol. 54, pp. 56–74 (1992)
19. Preparata, F.P., Shamos, M.I.: Computational Geometry - An Introduction. Springer, Heidelberg (1985)
20. Rosenfeld, A., Pfaltz, J.L.: Sequential Operations in Digital Picture Processing. Journal of the ACM 13(4), 471–494 (1966)
21. Rosenfeld, A., Pfalz, J.L.: Distance Functions on Digital Pictures. Pattern Recognition 1, 33–61 (1968)
22. Saito, T., Toriwaki, J.: New Algorithms for n-dimensional Euclidean Distance Transformation. Pattern Recognition 27(11), 1551–1565 (1994)
23. Samet, H.: The Design and Analysis of Spatial Data Structures. Addison-Wesley Longman Publishing Co., Inc, Amsterdam (1990)
24. Schouten, T., Broek, E.: Fast Exact Euclidean Distance (FEED) Transformation. In: 17th International Conference on Pattern Recognition (ICPR 2004), vol. 3, pp. 594–597 (2004)
25. Sintorn, I.M., Borgefors, G.: Weighted Distance Transforms for Volume Images Digitized in Elongated Voxel Grids. Pattern Recognition Letters 25(5), 571–580 (2004)
26. Vörös, J.: Low-Cost Implementation of Distance Maps for Path Planning Using Matrix Quadtrees and Octrees. Robotics and Computer-Integrated Manufacturing 17(6), 447–459 (2001)
27. Wang, X., Bertrand, G.: Some Sequential Algorithms for a Generalized Distance Transformation Based on Minkowski Operations. IEEE Transactions on Pattern Analysis and Machine Intelligence 14(11), 1114–1121 (1992)