2D Euclidean Distance Transform Algorithms: A Comparative Survey

RICARDO FABBRI
Brown University
LUCIANO DA F. COSTA
Instituto de Física de São Carlos, USP
and
JULIO C. TORELLI, ODEMIR M. BRUNO
Instituto de Ciências Matemáticas e de Computação, USP

The Distance Transform (DT) is a general operator forming the basis of many methods in computer vision and geometry, with great potential for practical applications. However, all the optimal algorithms for the computation of the exact Euclidean DT (EDT) were proposed only since the 1990s. In this work, state-of-the-art sequential 2D EDT algorithms are reviewed and compared, in an effort to reach more solid conclusions of their differences in speed and their exactness. Six of the best algorithms were fully implemented and compared in practice.

Categories and Subject Descriptors: I.4.0 [**Image Processing and Computer Vision**]: General; I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling; E.1 [**Data Structures**]: Graphs and networks; I.2.10 [**Artificial Intelligence**]: Vision and Scene Understanding; D.2.4 [**Software Engineering**]: Software/Program Verification; I.5.3 [**Pattern Recognition**]: Clustering

General Terms: Algorithms,Performance,Design

Additional Key Words and Phrases: Distance transform, Exact euclidean distance map, Dijkstra's Algorithm, Shape Analysis, Computational Geometry, Performance evaluation

The work of Ricardo Fabbri was supported by CNPq (200875/2004-3) and FAPESP (03/09834-0); Odemir M. Bruno acknowledges support from FAPESP (03/09834-0) and CNPq (303746/04-1); Luciano da F. Costa thanks FAPESP (99/12765-2 and 05/00587-5) and CNPq (308231/03-1).
Author's address: Ricardo Fabbri, Laboratory for Engineering Man-Machine Systems (LEMS), Brown University, 182 Hope Street, Box D, Providence, RI 02912 USA, email: rfabbri@lems.brown.edu; Odemir M. Bruno and Julio C. Torelli, Instituto de Ciências Matemáticas e de Computação (ICMC), Universidade de São Paulo (USP), Avenida do Trabalhador São-carlense, 400, 13560-970, São Carlos SP, Brazil, email: bruno@icmc.usp.br, jctorelli@yahoo.com.br; Luciano Da F. Costa, Instituto de Física de São Carlos, Universidade de São Paulo, CP 369, 13560-970, São Carlos, SP, Brazil, email: luciano@if.sc.usp.br.
Permission to make digital/hard copy of all or part of this material without fee for personal or classroom use provided that the copies are not made or distributed for profit or commercial advantage, the ACM copyright/server notice, the title of the publication, and its date appear, and notice is given that copying is by permission of the ACM, Inc. To copy otherwise, to republish, to post on servers, or to redistribute to lists requires prior specific permission and/or a fee.
© 20YY ACM 0000-0000/20YY/0000-0001 $5.00

1. OVERVIEW

The distance transform (DT) maps each image pixel into its smallest distance to regions of interest [Rosenfeld and Pfaltz 1966]. It is a fundamental geometrical operator with great applicability in computer vision and graphics, shape analysis, pattern recognition, and computational geometry. DT methods are useful propagation schemes that efficiently construct a solution to the eikonal differential equation [John 1982] in the integer lattice. This, in turn, is related to many other important entities such as medial axes, Voronoi diagrams, shortest-path computation, and image segmentation.

The DT can be defined in terms of arbitrary metrics. The Euclidean distance is often necessary in many applications, as it is the adequate model to numerous geometrical facts of the human-scale world. However, as in pure mathematics, some non-Euclidean metrics are much easier to manipulate and to compute. For this reason, efficient non-Euclidean DT algorithms have been reported since 1966, while fast algorithms for the exact Euclidean DT (EDT) started to appear only in the 1990s. Many others have recently been proposed, as described in Section 7.

Currently, it is still uncertain what is the best exact EDT algorithm, or even whether the recently proposed ones are correct or not. Moreover, validation of EDT methods is scarce and incomplete. In the majority of cases, comparative evaluation is published on the manuscript where an algorithm being judged is also being proposed. The survey therein is targeted at the novel method, with test cases that tend to emphasize its assets and overlook liabilities.

Comprehensive validation and comparison of methodologies are still incipient in image analysis, mainly because the algorithms are complicated, and also due to a general disregard towards this type of activity [Jain and Binford 1991]. In the case of EDTs, thorough evaluation faces a series of particular difficulties. To begin with, interesting EDT methods are numerous, recent and relatively obscure both in theory and implementation. In addition, performance depends on the contents of the input image, not only on its size. Therefore it is not trivial to predict the behavior of an EDT algorithm on a given input. Another difficulty is that there exists a number of factors that can be used to compare the algorithm, such as temporal and spatial performances, exactness and easiness of implementation.

2. OBJECTIVES

The main purpose of this paper is to provide an up to dated study, comparison and validation of EDT algorithms, reaching solid conclusions about the advantages and shortcomings of each. Such conclusions are useful for advancing the theory of EDT, as well as to support the adoption of fast methods in practical applications. This study is also valuable for improving computation of entities related to EDT, such as shape skeletons and discrete Voronoi diagrams, fractal dimension, and segmentation.

The main points to be clarified by the empirical tests are which algorithms are verified to be exact, which are the fastest and in which cases. There are six recent exact EDT algorithms we selected to be compared [Lotufo and Zampirolli 2001; Eggers 1998; Cuisenaire and Macq 1999b; Saito and Toriwaki 1994; Maurer et al. 2003; Meijster et al. 2000].

Another intent of this work is to characterize the performance of the algorithms for different classes of shape. Ideally, given an image type one wishes to determine the best algorithm for it. This analysis is specially relevant since our empirical results show that each EDT algorithm tend to have a very distinct behavior in terms of the input shape.

We restrict the study in this paper to 2D exact EDT algorithms on sequential architectures. For a recent account of general 3D distance transforms (not specific to the exact EDT), please refer to [Jones et al. 2006].

3. ORGANIZATION

This work is organized as follows. The next section presents the formal definitions required for a precise understanding of the ideas. However, it is suggested that the definitions of Section 4.2 be read only as needed. In Section 5 there is an account of DT applications, to illustrate its importance and its relation to other entities. Section 6 illustrates the importance of the Euclidean metric, motivating the need for efficient EDT schemes. The main recent EDT algorithms are organized and explained in Section 7. Section 8 describes the methodology for speed and exactness tests. The results of the tests are shown in Section 9. Finally, Section 10 lists the main contributions of this work, and future activities.

4. DEFINITIONS

4.1 Main concepts

The idea of a distance transform (DT) is quite simple, but it is nevertheless necessary to be established with rigor and clarity, so to avoid confusion caused by slightly different conventions used throughout the literature. Furthermore, some concepts and conventions must be precisely defined for a correct understanding of this work, specially Section 7.

The central problem of a DT is to compute the distance of each point of the plane to a given subset of it. In image processing terminology, this is rephrased in the following way. Let $I : \Omega \subset \mathbb{Z}^2 \to \{0, 1\}$ be a binary image where the domain Ω is convex and, in particular, $\Omega = \{1, \ldots, n\} \times \{1, \ldots, n\}$, unless otherwise stated. By convention, 0 is associated to 'black' and 1 to 'white'. Hence we have, naturally, an object $\mathcal{O}$ represented by all the white pixels:

$$\mathcal{O} = \{p \in \Omega \mid I(p) = 1\}$$

The set $\mathcal{O}$ is called 'object' or '*foreground*' and can consist of any subset of the image domain, including disjoint sets. The elements of its complement, $\mathcal{O}^c$, that is, the set of black pixels in Ω, are called '*background*'. From the DT point of view, the background pixels are called the 'interest points', 'seeds', 'sources', 'feature points', 'sites', or 'Voronoi elements'. In spite of being counter intuitive, this is just a convention frequently adopted in the DT literature.

Definition 4.1. The **distance transform** (**DT**) is the transformation that generates a map D whose value in each pixel p is the smallest distance from this pixel to $\mathcal{O}^c$:

$$D(p) := \min\{\mathrm{d}(p, q) \mid q \in \mathcal{O}^c\} = \min\{\mathrm{d}(p, q) \mid I(q) = 0\} \tag{1}$$

The image D is called the **distance map** of I (or of $\mathcal{O}$, in case I is tacitly understood).

D can also be called itself a distance transform, in case there is no ambiguity between the image D and the transformation (DT) that generated it. The term DT may also refer to a DT algorithm, depending on the context.

It is assumed that $\mathcal{O}^c$ contains at least one pixel, as in [Rosenfeld and Pfaltz 1966], otherwise the output of the DT is undefined. Moreover, $d(p, q)$ is generally taken as the Euclidean distance, given by:

$$d(p, q) = \sqrt{(p_x - q_x)^2 + (p_y - q_y)^2}. \tag{2}$$

Figure 1 shows a numerical example of EDT. For each pixel in (a), the corresponding

0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	1	1	1	0	0	0
0	0	1	1	1	1	1	0	0
0	0	1	1	1	1	1	0	0
0	0	1	1	1	1	1	0	0
0	0	0	1	1	1	0	0	0
0	0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	0	0

(a)

0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	1	1	1	0	0	0
0	0	1	2	4	2	1	0	0
0	0	1	4	8	4	1	0	0
0	0	1	2	4	2	1	0	0
0	0	0	1	1	1	0	0	0
0	0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	0	0

(b)

Fig. 1. Numerical example of distance transform. In (b) there is the Euclidean distance of each pixel to the nearest black pixel. The distance values are squared so that only integers values are stored.

pixel in the DT of Figure (b) holds the smallest Euclidean distance between this pixel and all the other black pixels. The squared Euclidean distance is used for saving storage: since the pixel coordinates are integers, the square of the Euclidean distance $d^2(p, q)$ is also an integer.

The DT can be visualized as a surface whose height is proportional to the distances, as in Figure 2(b), or as an image whose intensity is proportional to the distances, as in Figure 2(c). Another interesting visualization can be obtained by taking the modulo-n operation (i.e. the remainder of the division by n) for each distance value: $D_{mod\,n}(p) = D(p) \mod n$. As the distance increases, the value of $D_{mod\,n}$ repeats itself, falling between 0 and $n - 1$. Hence, there will be abrupt changes from $n - 1$ to 0 emphasizing iso-distance curves, as shown in Figure 2(d).

Various metrics, in addition to the Euclidean, can be used to compute the distance in Equation (1). Frequently used examples are the *city-block* (d_1) and *chessboard* (d_∞), defined by:

$$d_1(\mathbf{x}, \mathbf{y}) = |\mathbf{x}_1 - \mathbf{y}_1| + |\mathbf{x}_2 - \mathbf{y}_2|$$
$$d_\infty(\mathbf{x}, \mathbf{y}) = max\{|\mathbf{x}_1 - \mathbf{y}_1|, |\mathbf{x}_2 - \mathbf{y}_2|\}$$

These metrics are less costly to compute than the Euclidean metric.

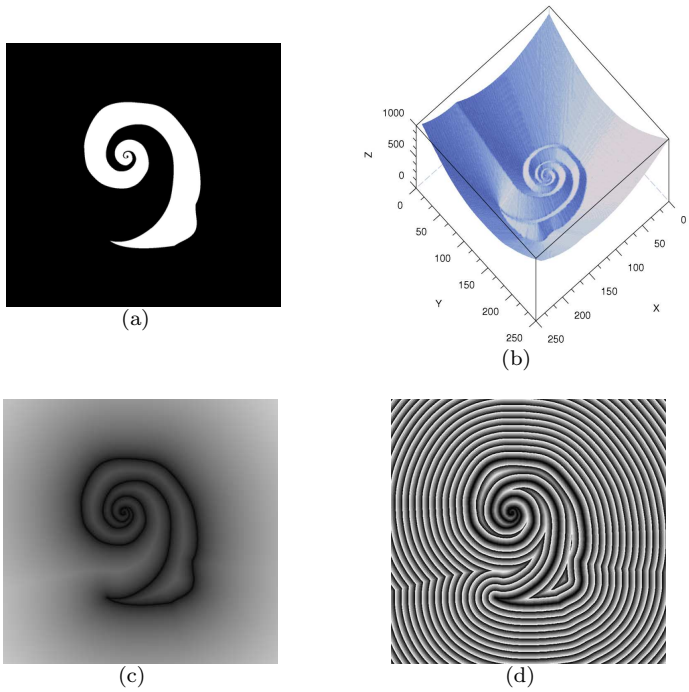

Fig. 2. The image of a spiraled shape (a) and representations of the distance transform of its border, where the height of the surface in (b) or the brightness in (c) are proportional to the smallest distance of each point to the border pixels. In (d) there is a visualization of iso-distance curves.

4.2 Further definitions and conventions for reference

In this text, N denotes the total number of pixels in an image, and n is the number of lines or columns in a square $n \times n$ image. A more general interpretation for n is as $\sqrt{N}$ for a $c \times r$ image of $N = c \cdot r$ pixels, where r and c are the number of lines and columns, resp. The symbol $\mathcal{N}$ denotes a neighborhood, defined as a set of ordered pairs representing relative displacements or vectors. The notation $d(p, q)$ denotes the squared Euclidean distance or the Euclidean distance itself, depending on the context, as long as there is no ambiguity. The norm notation will also be used: $\|p\| := d(p, O)$, where O is the origin, and $d(p, q) = \|p - q\|$.

An EDT (algorithm) is said to be *exact* for a given collection of images if it produces an exact map for all the images. If it is inexact for one of the considered images, the EDT is said to be *inexact*. The EDT is said to be exact (without specifying a collection of images) if it is exact for every possible input image.

The main focus of this work is the temporal complexity of EDT algorithms. The asymptotic upper bound $O(f(n))$, lower bound $\Omega(f(n))$ and equivalence $\Theta(f(n))$ will be expressed in function of n, and not of N.[1] As every EDT algorithm visits

[1]Further information about the $\Omega(f(n))$, $\Theta(f(n))$, and $O(f(n))$ notations can be found in [Knuth 1976]. Recall that the $\Theta(f(n))$ notation is *not* the average-case complexity of an algorithm.

each image pixel at least once, the best possible algorithm will be $O(n^2)$, $\Omega(n^2)$ and, thus, $\Theta(n^2)$. In fact, some authors already proposed exact EDT algorithms that run in $\Theta(n^2)$, as will be seen in Section 7. These algorithms are linear-time in terms of total number of input pixels $N = n^2$.

Another convention is that a boundary pixel p (or contour pixel) is a white pixel that has at least one black pixel in its neighborhood $\mathcal{N}(p)$. Boundary (or contour) is the set of all the boundary pixels.

An important entity throughout this text is the point-wise Voronoi diagram (VD) and related concepts. The definitions adopted in this text is given briefly as follows; a comprehensive account is given in [Aurenhammer 1991; Fabbri et al. 2002; Preparata and Shamos 1990].

The Voronoi region (VR) of an interest point[2] is the set of points strictly closer to it than to any other interest point. Other names for VR are: Voronoi polygon, Voronoi tile, and region of influence. The Voronoi element closer to a given pixel p is denoted by $VS(p)$. In case p has two or more closest sites, one of them is arbitrarily chosen to be $VS(p)$. By definition, the point-wise Voronoi diagram is the set of these points closest to one or more sites, that is, the points not in any Voronoi region. A Voronoi partition is the collection of the VRs of all sites or seeds. To build the partition, each point of the VD is arbitrarily attributed to the VR of one of the sites minimally equidistant to it.

The Voronoi partition can be represented by the label map, where each VS has an associated label (a number) that identifies this VS and pixels of its VR. The label map is formally defined as:

$$\begin{aligned} \mathit{Label} : \Omega &\to \{1, \ldots, n_s\} \\ p &\mapsto \mathit{Label}(p) = \{\mathit{Label}(q) \mid q = VS(p)\} \end{aligned}$$

where n_s is the number of *sites* (and of VRs). A similar map is the nearest-site or nearest-feature map, that to each pixel p associates $VS(p)$. Yet another similar entity is the vectorial DT, that to each pixel p associates a vector pointing to $EV(p)$.

5. APPLICATIONS

The DT is a fundamental operator in shape analysis, having numerous applications, some of which are briefly listed bellow.

—**Separation of overlapping objects**, through watershed segmentation [Soille and Vincent 1991; Cuisenaire and Macq 1999b]. Figure 3 illustrates this application. Consider the task of counting the number of blood cells in a microscope image. A problem that frequently occurs is illustrated in Figure 3(a), where there are two overlapping cells that appear as a single connected component in the binary image of Figure 3(b). To count the number of cells through the number of connected components, those overlapping instances must be properly separated. This can be done by first computing the DT of the connected components, shown in 3(c). Note that the peaks of the distance map locate the centers of the overlapping cells. By performing watershed segmentation on the inverse of the distance

[2]Recall the other names for interest point are: Voronoi site (VS), Voronoi element, seed, source, and that they consist of black or background pixels.

map, the overlapping cells get separated as in Figure 3(d).

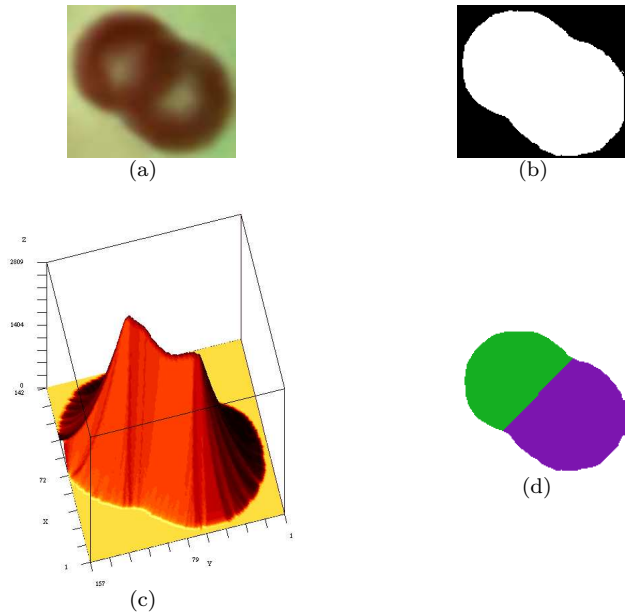

Fig. 3. Identification of overlapping blood cells (a). Steps: (b) initial segmentation; (c) Euclidean distance transform; and (d) watershed segmentation.

—**Computation of morphological operators** [Ragnemalm 1992; Parker 1997; Cuisenaire 2006]. If the DT is thresholded at level r, the 'object' pixels of the resulting image are those whose distance to the background is greater than r. This is the same as eroding the original image with a disc of radius r. Therefore, once the DT is computed, simple thresholding provides erosion with a desired radius. Similarly, considering the inverse image, one obtains the DT of the background. Thresholding this images yields the dilation of the original image by an arbitrary radius. The DT is, thus, extremely useful for implementing other operators of mathematical morphology that are based on successive erosions. For example, one can use it to obtain an image representing successive opening operations, as in [Parker 1997]. Moreover, from that image one can obtain the roughness spectrum [Parker 1997], which have been extensively applied for the classification of porous materials and other images having distribution of holes of varying sizes. An example of such images comes from rocky oil reserves [Parker 1997]. The operation of successive dilation is also useful for generating a **morphological scale-space** [Chen and Yan 1989], enabling an intrinsic and hierarchic structural analysis of shape.

—**Computation of geometrical representations and measures**, such as **skeletonization** [Ge and Fitzpatrick 1996; Danielsson 1980; Saúde et al. 2006; Couprie et al. 2006; Coerjolli and Montanvert 2007], **Voronoi diagrams** and **Delaunay**

triangulation [Vincent 1991], **fractal dimension** [Coelho and Costa 1996], **Gabriel graphs** [Vincent 1991], among others [Borgefors and Nyström 1997].

—**Robot navigation**, to find the shortest path from one place to another among obstacles [Chin et al. 2001; Shih and Wu 2004c; Cuisenaire and Macq 1999b].

—**Shape matching** [Borgefors 1986; Paglieroni 1992a; Liu and Srinath 1990; You et al. 1995]. Basically, this application consists in the correlation of a binary template with the DT of the binary image where objects similar to the template are to be found. An advantage of using the DT here is the fact that the resulting correlation surface is smoother than the correlation performed over the original binary image. This is better suited for fast convergence of optimization algorithms. DT-based correlation was also used for stereo feature matching. More details are found in Paglieroni's paper [Paglieroni 1992a]

—**Shape measures related to distance** [Rosenfeld and Pfaltz 1968; 1966; Cuisenaire and Macq 1999b]. For instance, the maximum of the DT of an object is its width; the distribution of the distances in the DT is also a useful shape descriptor.

—Other interesting areas in which the DT has been applied are: **image registration** [Kozinska et al. 1997; Cuisenaire and Macq 1999b], **sub-pixel contour tracing** [Siddiqi et al. 1997], **multidimensional data analysis (classification, clustering)** [Starovoitov 1996], **image enhancement** [Zeng and Hirata 2003], **ray-tracing optimization** [Paglieroni 1997; Sramek and Kaufman 2000], **embedding surface for level set-based methods** [Sethian 1999; Osher and Sethian 1988; Tek and Kimia 2003; Siddiqi et al. 1997; Sebastian et al. 2003; Xia et al. 2004], **efficient belief-propagation** [Felzenszwalb and Huttenlocher 2006], **multi-phase flow simulations** [Ceniceros and Roma 2005], **medical image analysis** [Cuisenaire and Macq 1999b; hao Tseng et al. 1998; Wintermark et al. 2002; Jolesz et al. 1997; Thiran and Macq 1996; Saito and Toriwaki 1994; Sebastian et al. 2003], **analysis of interaction between biological structures** [Travençolo et al. 2007], **botanics** [Travis et al. 1996], and **geology** [Paglieroni 1997; Parker 1997].

There are also evidences that processes linked to the EDT are related to phenomena of perception and biology [Kovács and Julesz 1994; Kimia 2003; Blum 1967; Leyton 1992].

6. THE IMPORTANCE OF THE EUCLIDEAN DT AND ITS EXACTNESS

In spite of being a simple concept, the DT is hard to compute with good efficiency and precision. The difficulty lies in the fact that its definition involves a point-wise distance minimization (cf. Equation 1). One way to improve DT computation is to consider local properties of the metric as to avoid performing global minimization independently for each pixel, as will be explained in Section 7. This type of optimization has been explored as the basis of efficient algorithms for non-Euclidean metrics since the DT itself was first described, in 1966 [Rosenfeld and Pfaltz 1966; 1968; Borgefors 1984; 1986].

However, difficulties involving non-local properties of the Euclidean metric on discrete lattices [Ge and Fitzpatrick 1996; Cuisenaire and Macq 1999b; Cuisenaire 1999; Danielsson 1980], explained in Section 7.3, delayed the discovery of efficient

EDT schemes for a long time. Fast algorithms for computing the exact Euclidean DT in sequential computers have appeared since 1990, and new ones continue to be published (cf. [Lotufo and Zampirolli 2001; Saito and Toriwaki 1994; Maurer et al. 2003; Hirata 1996; Meijster et al. 2000; Boxer and Miller 2000; Eggers 1998; Ge and Fitzpatrick 1996; Gavrilova and Alsuwaiyel 2001; Shih and Liu 1998; Cuisenaire 1999; Vincent 1991; Breu et al. 1995; Mullikin 1992; Ragnemalm 1992; Felzenszwalb and Huttenlocher 2004; Shih and Wu 2004b; 2004a; Lucet 2007]).

The reason for all the quest towards computing the DT with the Euclidean metric is that it has many properties desirable in applications. It is radially symmetric, differently from the other metrics used for DTs. This enables one to generate shape representations and properties that are virtually invariant to rotation, which is crucial for robust recognition. Skeletons, or medial axes, for example, are extremely unstable to rotations when non-Euclidean metrics are used. Furthermore, with non-Euclidean metrics the shortest path or maximum width of shape may not correspond to the expected practical meaning. The Euclidean metric also has assets for shape matching [Cuisenaire and Macq 1999b; Paglieroni 1992a; Borgefors 1986].

Since 1980, very good approximations to the EDT have been proposed [Danielsson 1980], which indeed suffices for many applications. However, a minimum error in the EDT can cause undesirable consequences in some applications. For example the skeleton obtained using approximated EDTs can get disconnected [Ge and Fitzpatrick 1996] in many common cases, violating a very important property of this representation. Furthermore, shortest path computation may accumulate unacceptable levels of error. Another significant problem is due to the inexactness that occur in the implementation of morphological operators [Cuisenaire and Macq 1999b], leading to violations of basic properties most applications take for granted.

7. EDT ALGORITHMS

This section reviews conceptually important DT algorithms and describes state-of-the-art Euclidean algorithms [Lotufo and Zampirolli 2001; Eggers 1998; Cuisenaire and Macq 1999b; Saito and Toriwaki 1994; Maurer et al. 2003; Meijster et al. 2000] in greater detail. It intends to provide an updated conceptual survey in order to organize the area in view of recent advances.

The novel and uniform description of recent EDT algorithms constitutes a major contribution of this work. Generally, the originals describe the algorithms separately from others, in a very general and concise way. There is seldom room for specific examples or comparisons in the papers that are proposing a new method. For instance, a new method is often proposed in arbitrary dimensions, non-convex domains, general metrics and non-orthogonal lattices. Differently, in this text the intent is to present the essence of each algorithm and to overcome difficulties found in reading and coding from the originals. This purpose is sought by restricting the concepts to 2D orthogonal grids and the Euclidean metric. Further details and generalizations can be found in the originals.

The main methods were coded in C by the first author of this paper. The code was released under the GPL free software license, both in a C library [Fabbri a] and in the SIP toolbox for Scilab [Fabbri b].

7.1 Brute-force EDT

The direct application of Definition 4.1 leads to the following EDT algorithm: for each pixel p, its distance to each of the black pixels is computed. The distance map at p is equal to the smallest of these distances. Obviously, if p itself is black, then it already has its final value, i.e., zero distance.

Even for this trivial method, the number of operations depends not only on the image size, but also on its content. Suppose the number of black pixels is k and, thus, the number of white pixels is $n^2 - k$. The number of comparisons performed by the algorithm is $k \cdot (n^2 - k)$, where $0 \leq k \leq n^2$. This function reaches its maximum at $k = n^2/2$ and, hence, the maximum number of operations is $n^2/2 \cdot (n^2 - n^2/2) = n^4/4 = O(n^4)$.

The minimum non-zero number of operations occurs for $k = 1$ or $k = n^2$, that is, when there is only one black or white pixel in the image. Therefore, the brute-force algorithm runs in time $\Omega(n^2)$. Often, however, k is a linear function of n. This occurs for images obtained by sampling a continuous contour, for example. In this case, the number of operations is $\Theta(n^3)$. In summary, the brute-force algorithm is $O(n^4)$ (worst case), $\Omega(n^2)$ (best case), and typically around $\Theta(n^3)$ for nearly one-dimensional objects. However, a brute-force algorithm coded directly with two loops over the image will always run in $\Theta(n^4)$ time. This is dealt with by using two lists of indexes, one for the black pixels, the other for the white ones. This requires $n \cdot n$ large integers or pointers. On the other hand, the naive $\Theta(n^4)$ algorithm can be performed in-place, i.e., it outputs directly in the input image, saving memory.

7.2 Efficient EDT algorithms

The general principle to improve the efficiency of DT algorithms is to explore the 'redundancy' or 'locality' of metric properties. The locality of the nearest site of each pixel, for example is an important property of metrics in the continuous plane:

PROPERTY 1. *For each point p, there is another point q, in a neighborhood of p, with the same closest feature point. In other words, continuous Voronoi regions are always connected in the continuous plane. Formally:*

$$\exists q \in \mathcal{N}(p) \mid EV(p) = EV(q), \quad \forall p \in \mathbb{R}^2, \forall \mathcal{N}(p) \tag{3}$$

where $\mathcal{N}(p)$ denotes a neighborhood of p.

If p and q are restricted to integer coordinates (discrete grid), then Property 1 remains valid for many metrics. Such fact enables the use of discrete neighborhood-based operations to compute the DT for such metrics. A related property that enables one to compute the DT value of a pixel from the value of its neighbors is called *regularity*:

PROPERTY 2. *[Rosenfeld and Kak 1976; Cuisenaire 1999] A metric d is said to be* regular *if, for every p and q such that $d(p,q) \leq 2$, there exists an r, different from p and q, such that $d(p,q) = d(p,r) + d(r,q)$.*

Rosenfeld and Pfaltz [Rosenfeld and Pfaltz 1966] were the first to propose efficient DT algorithms, based on local sequential operations. Other methods of this kind are reviewed in Section 7.4.

7.3 Sources of error in EDT computation

Most of the local algorithms do not compute the exact EDT, basically because Properties 1 and 2 are not valid for the Euclidean metric in discrete grids. The nearest Voronoi site of a pixel p can be different from the nearest site of all its discrete neighbors for this metric. This fact is expressed in other words by Property 3, stated bellow.

PROPERTY 3. *Discrete Euclidean Voronoi regions are not always connected.*

Property 3 is the main reason why exact EDT algorithms first appeared in the 1990's, while the non-Euclidean ones have been around since 1966. Examples of Voronoi regions with more than one 4- and 8-connected component are shown in Figure 4.

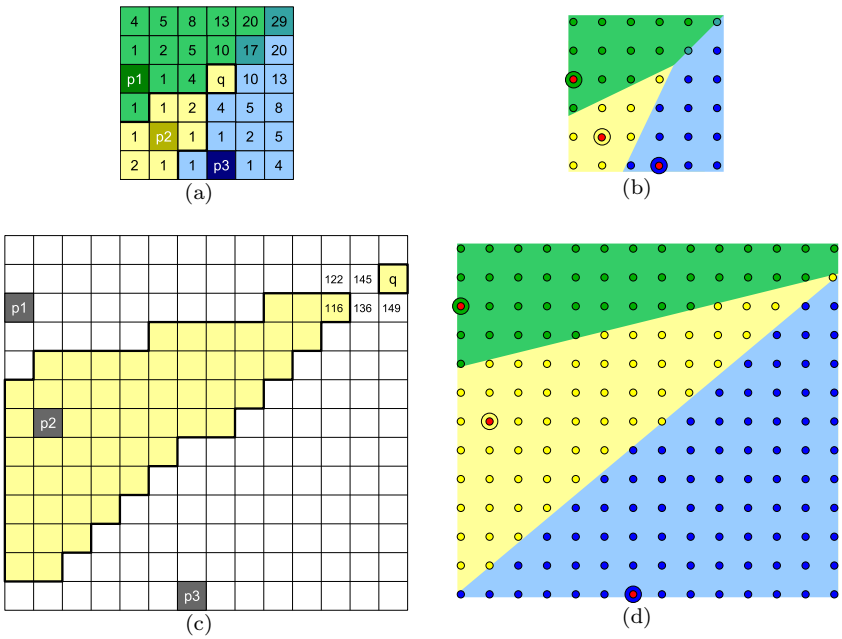

Fig. 4. Disconnected Euclidean Voronoi regions for the 4-neighborhood, (a) and (b), and 8-neighborhood, (c) and (d). Pixel q receives a wrong distance value if the algorithms assume 4-connectivity (a) or 8-connectivity (c) for the discrete regions. In (b) and (d), the continuous and connected regions are shown in lighter colors, with overlapping sample pixels. Pixels equidistant to p_1 and p_2 or to p_1 and p_3 are shown in turquoise.

In Figure 4(a), the VR of pixel p_2 is disconnected with respect to 4-neighborhood. Pixel q is closer to p_2 than to p_1 and p_3, however none of its 4-neighbors is closer to p_2. In order to better illustrate the reason behind the disconnection in the VR, Figure 4(b) shows the continuous Voronoi regions, in pale colors, on top of the sample points in stronger colors. Each color represents the nearest pixel of each point in the plane. Pixels equidistant to p_1 and p_2 are shown in turquoise blue. Figures 4(c) and 4(d) show that a similar problem occurs for a larger 3×3 neighborhood. The

disconnection generally occurs due to sampling of thin and slanted VRs that do not pass through any discrete neighbor of a given pixel, although it passes on this pixel. Property 3 is the cause of error in a class of DT algorithms formally defined in the following.

Definition 7.1. An $\mathcal{N}$-EDT is defined as an EDT algorithm in which the final distance of each pixel is computed with respect to the same seed pixel of some pixel in a fixed neighborhood $\mathcal{N}$. The map generated by such an algorithm is denoted $D_{\mathcal{N}}$, and whenever the subscript $\mathcal{N}$ is dropped, the notation denotes an exact map.

Example of $\mathcal{N}$-EDTs are the algorithms of Danielson [Danielsson 1980] and Cuisenaire's PSN [Cuisenaire and Macq 1999b; Cuisenaire 1999; Falcão et al. 2004]. It is not surprising that these algorithms present errors due to Property 3. For instance, in Figure 4, if the nearest pixel to q is inferred from its neighbors, then a wrong distance will occur for q, since none of them has the same closest pixel as q. In fact, the distance in q will be computed with respect to an interest pixel different to the one that is really the closest to it. Therefore, it is clear that $D_{\mathcal{N}}(q) \geq D(q)$, for every pixel q.

It can be shown that every $\mathcal{N}$-EDT algorithm is not exact for every pixel with exact distance greater to a certain value $d_c(\mathcal{N})$. This distance depends only on the neighborhood $\mathcal{N}$ being used. Formally:

PROPERTY 4. *(based on [Cuisenaire and Macq 1999b; Cuisenaire 1999]) Given a neighborhood $\mathcal{N}$, there is a distance value $d_c(\mathcal{N})$ such that:*

$$D_{\mathcal{N}}(p) \geq d_c(\mathcal{N}) \Rightarrow D_{\mathcal{N}}(p) > D(p),$$

for some pixel p of some input image.

Assuming that the disconnection of Voronoi regions is the only source of error of $\mathcal{N}$-EDTs, then Property 4 can be extended to Property 5 bellow.

PROPERTY 5. *(based on [Cuisenaire and Macq 1999b; Cuisenaire 1999]) Given a neighborhood $\mathcal{N}$, there is a value $d(\mathcal{N})$ such that, for every pixel p:*

$$D_{\mathcal{N}}(p) < d(\mathcal{N}) \Rightarrow D_{\mathcal{N}}(p) = D(p)$$

Furthermore, there is a maximal value $d_c(\mathcal{N})$ for $d(\mathcal{N})$ above which $D_{\mathcal{N}}(p)$ is inexact for some p and some image satisfying $D(p) \geq d_c(\mathcal{N})$. Moreover, $d_c(\mathcal{N})$ increases as $\mathcal{N}$ increases.

An upper bound for $d_c(\mathcal{N})$ can be computed through an exhaustive search algorithm described in p. 60 of Cuisenaire's thesis [Cuisenaire 1999]. There is no open implementation of that procedure, to this date; the one performed by R. Fabbri, author of this monograph, is available on the Internet [Fabbri a].

Property 5 indicates that, given an image, its exact EDT can be computed by an $\mathcal{N}$-EDT with sufficiently large $\mathcal{N}$. However, the direct application of this idea is inefficient for large images, since the size of the required neighborhood can be too large. The complexity of an optimal $\mathcal{N}$-EDT is $O(N \cdot m)$, where m is the number of elements of the neighborhood, and N is the number of pixels in the image.

7.4 Classification of EDT Algorithms

The efficient and sequential EDT algorithms can be classified in terms of the order in which the pixels are processed. In the so-called *ordered propagation* algorithms, the smallest-distance information is computed starting from the seeds (0 distance) and progressively transmitting the information to other pixels in order of increasing distance. On the other hand, the *raster scanning* algorithms use 2D masks to guide the processing of pixels line by line, top to bottom, then bottom to top. *Independent scanning* schemes process each row of the image, independently of the other, and then process each column of the result. This process is similar to separable linear transforms, such as computing the Fourier transform of an image by a sequence of 1D transforms in orthogonal directions [Brigham 1988]. This taxonomy of EDT algorithms into three broad classes is similar to that of mathematical morphology algorithms [Zampirolli and Lotufo 2000].

The EDT categories are not exclusive, specially the raster and independent scanning ones. Many independent scanning algorithms were inspired from related raster scanning methods, frequently through the decomposition of a 2D operation into separable one-dimensional transforms. Furthermore, in a sense all EDT algorithms perform propagation; what distinguishes ordered propagation algorithms is that they tend to visit pixels in order of increasing distance from the source pixels. In what follows, the concepts and algorithms of each class are reviewed, emphasizing Euclidean DTs.

7.4.1 *Propagation algorithms: fundamental concepts.* One way to compute the DT is illustrated by the grass-fire analogy. Imagine that the input binary image represents grass fields, where 1 means 'grass', and 0 'non-grass'. Suppose that fire is startled in the boundary of the grass fields. As the grass gets burned, the fire front gets progressively distant from its initial position, until it extinguishes. It can be said that in time t, the fire will be at some distance d from the regions initially without grass. By marking the distance of the forefront in each pixel where it passes, the DT of the original binary image is generated.

The grass-fire analogy is the basic idea behind ordered propagated algorithms. Starting from the boundary pixels, they compute the distances of pixels from closest to farthest. The processing is performed only around the narrow band of pixels (fire front) where a change in the current stored distance can occur. Figure 5 describes the fundamental process that performs ordered propagation.

The employment of a contour set restricts the processing only to the narrow band where the distances can change. This is the main reason why this procedure has a good potential for efficiency. Note that no clue was given about *which* pixel must be removed by Step 3a, or *what* neighborhood to use in Step 3b. Basically, the details of these steps control the efficiency and correctness of the method; many pixels can be unnecessarily updated if they are not adequately designed.

The algorithm in Figure 5 is similar to the so-called Dijkstra's algorithm [Dijkstra 1959; Moore 1959; Bellman 1958]. In the latter, Step 3a removes the pixel with smallest current distance among all the pixels of the contour set. In other words, the contour set is a priority queue – a structure in which the 'pop' operation removes the pixel having smallest distance. The priority queue is the most important bottleneck of Dijkstra's algorithm. Its efficiency can be improved for DTs since, in digital

(1) Initialize the distance of every white pixel to a sufficiently high value, also called "infinity".
(2) Initialize an auxiliary set of pixels, the so-called *contour set*, with the boundary pixels, and assign distance of 1 to them.
(3) While the contour set is not empty, do:
  (a) Remove a pixel from the contour set, called the *central pixel*.
  (b) For each white neighbor of the central pixel, do:
    i. Compute a new distance for the neighbor, based on the distance of the central pixel.
    ii. If the new distance is smaller than the current distance of the neighbor:
    —update its current distance to be the smallest one.
    —insert this neighbor in the contour set.

Fig. 5. Fundamental procedure of EDTs by ordered propagation

images, the costs (distances) are limited integers (e.g. squared Euclidean distance values). In this case, it is possible to apply the optimizations of Dial [Dial 1969] and Ahuja et. al. [Ahuja et al. 1993] for Dijkstra's algorithm, using *buckets* for the priority queue. The buckets are sets of pixels with equal cost. This is the basis for numerous fast algorithms that are explained later. It is also the basis of numerical schemes to solve the *Eikonal* PDE – such as the *fast marching method* – which are not covered in this work.

The bucket queue is a vector indexed by an integer distance value. Each bucket i contains a list of the pixels having current distance $d = i$. The advantage of this structure is that the pixels in the queue are naturally ordered, in the same way as in the bucket sort algorithm[Aho et al. 1982]. To determine the least-cost pixel in Step 3a, one simply increments the current distance (index of the bucket) until a non-empty bucket is found.

We denote by **PSN** (Propagation by a Single Neighborhood) the generic Dijkstra's algorithm applied for the Euclidean metric (Figure 5), with a priority queue implemented using buckets. The size of the neighborhood strongly affects the exactness of the results.

As stated in Section 7.3, for each neighborhood size there is a distance value up to which the EDT generated by PSN will be exact. For larger distances, there can be errors cases similar to Figure 4. Cuisenaire proposed a method to correct the PSN using propagations with larger distances only in certain places that possibly need it, as explained in Section 7.11.

There are other optimizations that may be applied to ordered propagation algorithms. First, the orientation of the neighborhood elements with respect to the current pixel may be restrained, in order to minimize unnecessary updates. This is done in [Montanari 1968; Verwer et al. 1989; Ragnemalm 1992; Eggers 1998; Cuisenaire and Macq 1999b], though not in [Falcão et al. 2004]. Second, the queue never stores more than a certain maximum cost difference, so the bucket vector can be circular [Falcão et al. 2004; Dial 1969], the index being the distance value *modulo* the maximum distance increment possible, which is dependent on the size of the neighborhood.

The Euclidean ordered propagation algorithms have the important asset that the EDT can be computed only for pixels up to a given distance. This is particularly suitable for implementing erosion and dilation by a disc of radius r. Propagation

schemes also have the advantage of being potentially more efficient for non-convex domains [Cuisenaire 1999].

7.4.2 *Propagation algorithms: a brief historical account.* Montanari [Montanari 1968] pioneered an idea similar to the one in Figure 5 with *bucket sort* to compute the DT and the medial axis with a metric that approximates the Euclidean. He also devised a relevant result about the direction in which the propagation shall occur as to minimize the deviation from the Euclidean metric.

In 1986, Piper and Granum [Piper and Granum 1987] proposed a FIFO (first-in-first-out) strategy for the contour set, that is, Step 3a in Figure 5 removes the pixel that has been longer in the queue. This is a well-known breadth-first search. Unfortunately pixels can be updated more than once in this scheme, up to once per seed pixel in some cases [Ragnemalm 1992]. The work of Piper and Granum also considered non-convex domains.

Verwer [Verwer et al. 1989], in 1989, proposed a non-Euclidean propagation method that also implements the contour set using buckets. Ragnemalm [Ragnemalm 1992] extended it for the Euclidean metric. Verwer argued that this type of propagation is better suited than raster-scanning, for DTs restricted to non-convex domains.

Ragnemalm [Ragnemalm 1992] used a propagation scheme with a circular queue and yet another strategy for Step 3a of Figure 5. A threshold variable holds an upper bound for the distance values of all the pixels to be processed in the vicinity of the current contour set. Pixels whose distances are greater than this value are kept in the contour set for the next iteration. After each iteration, this upper bound is increased by the maximum increment implied by the neighborhood being used. Ragnemalm states that this scheme removes almost all the multiple updates, as shown by some experiments [Ragnemalm 1992]. However, Cuisenaire [Cuisenaire 1999] says that some pixels can be updated more than once, up to once per feature pixel in some cases, which implies $O(n^3)$ complexity. The reason behind multiple updates, as also happens with other methods such as Eggers' [Eggers 1998], is that the propagation is ordered by the chessboard metric instead of the Euclidean.

Sharaiha and Christofides [Sharaiha and Christofides 1994] proposed in 1994 a DT by propagation, explicitly formulated with graph theory and solved by a variation of Dijkstra's algorithm. This idea was recently extended for the Euclidean metric and generalized to other problems in image analysis, forming a unified approach called the *Image Foresting Tranform* (IFT) [Falcão et al. 2004].

The IFT states various image analysis concepts as graph problems, giving to them a unified formalism and a single efficient solution – Dial's implementation of Dijkstra's algorithm. Reference [Fabbri 2004] provides a didactic explanation of the main concept of the IFT and how to compute EDT by this approach. Up to this date, the EDT by IFT is equivalent to the PSN algorithm and, therefore, is not exact. However, in principle it can be made exact by use of multiple neighborhoods using Cuisenaire's ideas (cf. Section 7.11). The IFT has the advantage of being readily applicable to compute other entities related to EDTs, such as watershed segmentation, skeletonization, and many others. This fact provides a theoretical link between propagation-based EDT and other image analysis algorithms. On the practical side, the multi-purpose nature of the IFT makes it a useful toolbox to

have, by solving many problems at once in addition to the EDT. We have implemented the method based on the original authors' code and made it available to the public [Fabbri a].

In this work, two recent propagation EDTs were empirically evaluated: the methods of Cuisenaire [Cuisenaire and Macq 1999b] and Eggers [Eggers 1998]. They are described in Sections 7.11 and 7.12, respectively.

7.4.3 *Raster-Scanning Algorithms.* Rosenfeld and Pfaltz proposed the first sequential DT algorithms by raster scanning with non-Euclidean metrics [Rosenfeld and Pfaltz 1966; 1968]. In [Rosenfeld and Pfaltz 1968], they proposed cityblock and chessboard metrics, as well as hexagonal and octagonal metrics to approximate the Euclidean DT.

Many authors improved the idea of raster scanning to better approximate the Euclidean metric with little overhead [Montanari 1968; Borgefors 1984; 1986; Akmal Butt and Maragos 1998]. The work of Borgefors [Borgefors 1984] reviews those algorithms and proposes the *chamfer* DTs, which have been in widespread use. Chamfer metrics are regular and are defined by local masks. The weight on these masks are chosen to minimize the deviation from the Euclidean DT. Chamfer DTs require two raster scans in the image, in the same style as the original algorithm by Rosenfeld and Pfaltz. Further improvements on the chamfer DTs of Borgefors were also proposed [Akmal Butt and Maragos 1998].

In 1980, Danielsson [Danielsson 1980] proposed an algorithm to generate the Euclidean DT in a similar way as the raster scanning of the chamfer DTs. However, the propagated information is the absolute value of the relative coordinates of the nearest feature pixel, instead of only the relative distances. Therefore the method propagates two values in the masks, instead of one, which is called *vectorial propagation.* Given these coordinates, the Euclidean distances are easily computed.

Danielsson stated that, although his new EDT produces a correct result for most of the pixels in an image, some errors are produced. To improve precision, he proposed the use of larger masks. His algorithm with 4-neighborhood masks is called 4SED (*Sequential Euclidean Distance map*), and the more precise method that uses 8-neighborhood masks is called 8SED. In Section 7.6, Danielsson's algorithm is explained in further details, given its historical and conceptual importance.

Many improvements have been proposed for Danielsson's algorithm. Ye and Fitzpatrick [Ge and Fitzpatrick 1996] proposed the *signed distance map*, meaning that the propagate information are relative coordinates with sign, and not only their absolute value as in the original SED[3]. These algorithms are called 4SSED(*Signed* SED) and 8SSED.

Leymarie [Leymarie and Levine 1992] proposed an efficient implementation of Danielsson's algorithm, comparable in speed to the fastest chamfer DTs. Another important improvement was published by Ragnemalm [Ragnemalm 1993] – an implementation of 8SSED with only 3 separable raster scans (i.e. without scanning forward and backward in each line). He also demonstrated that this is the least possible number of scans.

[3]The name "signed distance map" is sometimes used in the literature for the distance map of a contour in which inside pixels have negative distances, and outside pixels positive distances.

Most of the exact EDTs by raster scanning are based on corrections of some SED variation. Mullikin's exact correction of the 4SED [Mullikin 1992] is considered by Cuisenaire [Cuisenaire 1999] as $O(n^3)$ in the worst case. Another method, proposed by Shih and Liu [Shih and Liu 1998], performs post-processing of 8SED to correct eventual errors. However, these errors were under-estimated, as pointed by Cuisenaire [Cuisenaire 1999].

Three recent methods seem to be the fastest exact EDTs by raster-scanning [Cuisenaire and Macq 1999a; Shih and Wu 2004b; 2004a]. The first, proposed by Cuisenaire, consists in performing a post-processing step on 4SED. The basic idea is to apply corrections in pixels of the borders of the Voronoi regions generated by 4SED. In these pixels, a simple method is proposed to check if the continuous VR generated disconnected pixels in the discrete grid. This is done by defining borders that include the discrete VR and limits the number of pixels to be verified.

The recent methods by Shih and Wu consist in two raster-scans using a 3×3 neighborhood [Shih and Wu 2004b], and a dynamically adjusted neighborhood size [Shih and Wu 2004a]. The authors supposedly prove the correctness of the former algorithm and show that its complexity is independent of image content, however no comparative experimental study is performed. The latter algorithm is more recent, was proven by the authors to be correct, and was shown in some experiments to be faster than Eggers [Eggers 1998].

Although raster-scanning EDTs seem to be the fastest ones available, there are some disadvantages. First, many pixels might be processed more than once, specially by SED, as pointed by [Ragnemalm 1992]. On the other hand, efficient ordered propagation algorihms should avoid multiple updates and process pixels only when they receive their final value – which can result in less than two passes in the image. Secondly, raster-scanning algorithms are more difficult to extend for non-convex domains, usually requiring multiple passes before achieving the final values.

7.4.4 *Independent scanning algorithms.* Rosenfeld and Pfaltz devised yet another approach for computing DTs, called independent scanning or dimensional reduction. Basically, the 1D DT is first constructed for each row (or column) independently, and then this intermediate result is used in a second phase to construct the full 2D DT. The first stage is common to all Euclidean DTs based on independent scanning:

Transformation 1. *Given an input image F, the first transformation of independent scan generates an image G defined by:*

$$G(i,j) = \min_y \left\{ (j-y)^2 \mid F(i,y) = 0 \right\} \tag{4}$$

This corresponds to compute, for each pixel (i,j), its (squared) distance to the closest black pixel in the same line. This transformation is efficiently implemented by performing a forward scan (i.e. left to right) followed by a backward scan in each line of the image. The result of *transformation* 1 for a simple image is illustrated in Figure 6.

The non-trivial processing lies on the second stage; this is where each algorithm applies its specific strategy to generate the 2D EDT from the 1D line-wise transform.

0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	1	1	1	0	0	0
0	0	1	1	1	1	1	0	0
0	0	1	1	1	1	1	0	0
0	0	1	1	1	1	1	0	0
0	0	0	1	1	1	0	0	0
0	0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	0	0

(a)

0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	1	4	1	0	0	0
0	0	1	4	9	4	1	0	0
0	0	1	4	9	4	1	0	0
0	0	1	4	9	4	1	0	0
0	0	0	1	4	1	0	0	0
0	0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	0	0

(b)

Fig. 6. Example of the 1D transformation common to independent scanning EDTs. (a) The input image F; (b) its 1D EDT G along each row.

The independent scanning approach was generalized by Paglieroni for the Euclidean distance [Paglieroni 1992a; 1992b] in addition to a broad class of distance functions satisfying certain properties. Basically, these important properties assure that it is possible to compose a 2D EDT using independent 1D scans along each direction. The second step of Paglieroni's method consists of up-and-down scans over each column, together with tests to restrict the number of feature pixels to consider for each pixel. However, the algorithm is $O(n^3)$ in the worst-case.

Specific properties of the Euclidean metric were exploited by subsequent methods in order to restrict the number computations for each pixel during the second scan phase. Based on these properties, there is clearly three sub-variants of independent scanning algorithms. One variant uses properties based on parabola intersections; another uses a mathematical morphology approach; and the last class is explicitly based on fast computation of Voronoi diagram intersections with image lines.

The fundamental idea behind the independent-scanning EDT methods is that only $O(n)$ feature pixels influence the distance map values along each line of the image. Therefore, knowing which are the relevant pixels for each of the n lines, the whole 2D EDT is computable in $n \cdot O(n) = O(n^2)$ operations. The determination of the relevant pixels for each row can be done in $O(n)$ time, as shown by some papers to be reviewed shortly in this text. Therefore, in principle two $O(n)$ steps are performed for each row – the determination of relevant sites and the computation of the Euclidean distance from this information. Moreover, many methods intertwine the two $O(n)$ steps to improve non-asymptotic performance.

7.4.4.1 *Methods based on parabola intersections.* An early independent scanning algorithm [Kolountzakis and Kutulakos 1992], which had complexity $O(n^2 log(n))$, was improved by Chen and Chuang [Chen and Chuang 1994], making it linear in the worst case [Hirata 1996]. Saito and Toriwaki [Saito and Toriwaki 1994] also proposed a method based on similar ideas, explicitly using concepts based on parabola intersections to speed up the second scan, as explained in Section 7.7. In spite of being fast for numerous images, the complexity of this method has not been established. Some authors claim that it is $O(n^3)$ [Maurer et al. 2003; Cuisenaire and Macq 1999b; Cuisenaire 1999] based on unpublished experimental observation.

Similar EDT algorithms were more recently proposed by Hirata and Meijster [Hirata 1996; Meijster et al. 2000]. Meijster's method is in general much faster than

Achievement	Seminal References
The concept of DT	[Rosenfeld and Pfaltz 1966]
Efficient approximate EDT ($\mathcal{N}$-EDT)	[Danielsson 1980]
Study of errors in $\mathcal{N}$-EDT	[Danielsson 1980]
Non-trivial exact algorithm	[Yamada 1984]
Linear-time exact algorithm	[Breu et al. 1995; Chen and Chuang 1994] [4]
Independent scanning EDT algorithm	[Paglieroni 1992a]
Propagation algorithm (approximate)	[Montanari 1968]
Propagation algorithm (exact)	[Piper and Granum 1987]
EDT as single erosion	[Shih and Mitchell 1992]
EDT as lower envelope of parabolas	[Saito and Toriwaki 1994]
Linear-time EDT from Voronoi diagram	[Breu et al. 1995]

Table I. Some of the main advances towards efficient sequential EDT computation.

Saito, in the informal opinion of some authors [Lotufo and Zampirolli 2003]. This is confirmed by our experiments in Section 9.

7.4.4.2 *The mathematical morphology approach.* Shih and Mitchell [Shih and Mitchell 1992] showed that the EDT can be computed by a single gray-scale morphological erosion of the input image. Later, Huang and Mitchell [Huang and Mitchell 1994] decomposed the structuring function into a sequence of 3×3 structuring functions as the basis of their method. Lotufo and Zampirolli improved this method by further decomposing the structuring element into 1D elements. As a result, their algorithm perform independent scanning, the first stage being transformation 1. The second stage is performed using simple FIFOs for the 1D propagation. More details are given in Section 7.10 and in their works [Lotufo and Zampirolli 2001; 2003; Zampirolli 2003].

7.4.4.3 *Methods based on Voronoi diagram intersections.* The EDT can be computed using general $O(n^2 \log n)$ Voronoi algorithms [Preparata and Shamos 1990; Fortune 1987]. However, since image points have integer coordinates, this efficiency can be reduced to $O(n^2)$ [Breu et al. 1995; Ogniewicz and Kübler 1995].

A key fact is that the intersection of the VD of the seed pixels with a row or column of the image can be efficiently computed. Breu et. al.[Breu et al. 1995] proposed the first EDT algorithm using the construction of these intersections, and proved that it is $O(n^2)$ and exact. Guan and Ma [Guan and Ma 1998] improved this method using properties of the Euclidean metric while maintaining a slightly different representation for the intersection of the VD with each line of the image. The recent method by Maurer et. al. [Maurer et al. 2003] is an improvement on both of these previous methods, as will be explained in Section 7.8. Its complexity and correctness have been formally proved.

7.5 Summary of main advances

The main advances in the area of exact Euclidean distance transforms are summarized in Table I.

7.6 Danielsson's algorithm

One of the most widely used EDT algorithms is the one proposed by Danielsson in 1980. Although it is simple and efficient, it is not exact. Given a binary image, the method generates the vector[5] EDT in 4 raster scans, using the masks of relative displacement shown in Figure 7(left).

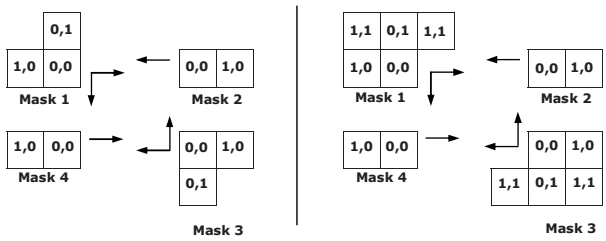

Fig. 7. Masks used in 4SED (left) and 8SED (right).

The vector distance map is initialized in the following way: if the pixel (i, j) is black, $D(i, j) = (0, 0)$, and $I(i, j) = (\infty, \infty)$ otherwise. The value ∞ is a number greater than the maximum possible displacement in the image (e.g. number of rows or columns). Starting on top, the map is scanned by moving mask 1 from left to right in each line, and mask 2 from right to left in the same line. In each white pixel p, the vectors in the mask are added to the corresponding values in the distance map and the new value of p is defined as the minimum of these sums. Masks 3 and 4 are subsequently moved line-by-line from right to left and vice-versa, respectively, starting from the bottom. It is recommended for the reader to try each of these steps in a small numerical example.

This algorithm is called 4SED (*Four-point Sequential Euclidean Distance transform*). The name refers to the fact that 4-neighbors are inspected by the masks. Although this algorithm generates an EDT without drastic numerical errors, it is not exact. This is due to Property 3 described in Section 7.3. To improve accuracy, larger masks can be used, such as those in Figure 7(right). For these masks the algorithm is called 8SED. However, it is also not exact for sufficiently large distance values. In fact, as seen in Section 7.3, for any neighborhood size there is a distance value for which an error can occur in the corresponding EDT.

7.7 Saito's algorithm

Saito and Toriwaki [Saito and Toriwaki 1994] devised an algorithm to produce the EDT of a k-dimensional image using k transformations, one for each coordinate direction. For 2D images, distance values are first computed along each row (*1st transformation*), as described in Section 7.4.4. These values are then used for computing minimal distances along each column (*2nd transformation*). Using the notation of Section 7.4.4:

[4]The first manuscript was submitted on July 1993, while the second on August 1993.
[5]see Section 4.2

TRANSFORMATION 2. *Starting from G (of Transformation 1) the 2nd transformation of Saito generates an image S which is the distance map of F:*

$$S(i,j) = \min_x \left\{ G(x,j) + (i-x)^2 \right\} \tag{5}$$

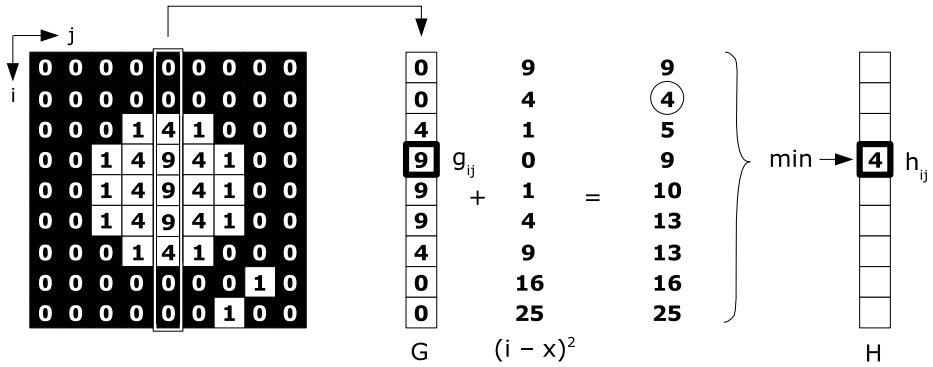

Fig. 8. Example of Transformation 2 at a pixel (i,j).

Figure 8 illustrates Transformation 2. Note that the squared Euclidean distance between two pixels is defined as the squared vertical distance plus the squared horizontal distance. After Transformation 1, every pixel (x,j) of column j has the value $G(x,j)$, which is the squared distance between (x,j) and the nearest black pixel in the same row. Adding to $G(x,j)$ the vertical distance between (i,j) and (x,j), $(i-x)^2$, one finds the 2D distance between (i,j) and the nearest pixel to (x,j) in row x. Taking the minimum of the results for all lines x, $S(i,j)$ will be the distance from (i,j) to the nearest seed pixel.

Saito and Toriwaki implement Transformation 2 using a downward scan followed by an upward scan in each column of G. During the downward scan, for each pixel (i,j) one applies a test to restrict the number of pixels ahead over which distance minimization is performed. The upward scan proceeds in a similar fashion. More details of this algorithm can be found on the original paper [Saito and Toriwaki 1994].

7.8 Maurer's algorithm

The original paper by Maurer et. al. [Maurer et al. 2003] is an improvement of the method by Breu [Breu et al. 1995], and extends it for any dimension and more general metrics. This Section attempts a more conceptual description, since it is not necessary to expose the general theory in detail. Maurer's EDT is summarized in Figure 7.8.

The first step is the same as for the other independent scanning algorithms, as seen in Section 7.4.4. However, for explanation purposes it is convenient to perform 1D EDTs vertically instead of horizontally. The non-trivial computation lies on the second stage.

(1) Compute the 1D EDT for each column
(2) For each row j
  (a) Find the intersection of the VD with the line containing row j.
    i. Restrict computation to the relevant sites using the information of the 1D EDT.
  (b) Sweep row j computing the EDT by querying the Voronoi region containing each pixel.

Fig. 9. High-level description of Maurer's EDT

A key fact of this method is that the intersection of the Voronoi diagram with a row of the image can be efficiently computed, represented, and queried. This is due to the following important fact.

PROPERTY 6. ***Few sites influence the distance map in a row of the image.*** *More specifically, only $O(n)$ Voronoi sites have tiles that intersect any given row of a $n \times n$ image.*

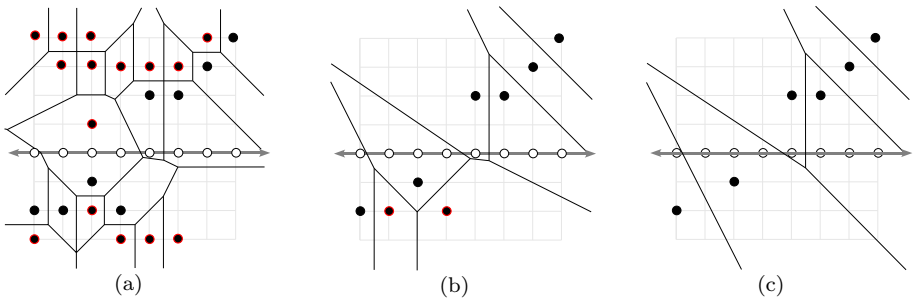

Fig. 10. In (a) there is the VD of all sites represented by black dots. Sites marked in red are not relevant for computing the EDT within the gray line, since their VR do not intercept this line. The sites marked in (a) are discarded to generate (b), keeping only the closest sites along each column. In (b) the marked sites are removed, which do not pass a certain test of bisector intersection. In (c), there is the VD of the relevant sites. The intersection of the complete VD (a) with the line is not altered in the partial VDs (b) and (c).

Figure 10 illustrates this fact. Given the $O(n)$ nearest sites to any given row, then for each pixel of this row one queries the nearest site and computes the distance to it. Doing this for all the n rows results in $O(n^2)$ operations. This is basically how this optimum complexity is achieved by Breu and Maurer.

The determination of *which* sites are relevant for a row $\mathcal{R}$ comprises two restrictions:

(1) Discard any site that is not the nearest of its column to pixels in $\mathcal{R}$. This information is provided by the column-wise EDT. In Figure 10(a), the sites circled in red were removed using this criterion, yielding the VD in (b).
(2) Let u, v and w be three of the remaining sites such that $u_x < v_x < w_x$ (abscissae). Let $\widehat{uv}$ be the intersection of the bisector between u and v with row $\mathcal{R}$, and let $\widehat{vw}$ be defined analogously, as shown in Figure 11. It is easy to see that the Voronoi region of site v will not intercept line $\mathcal{R}$ if $\widehat{uv}$ is to the right of $\widehat{uv}$, that is, $\widehat{uv}_x \geq \widehat{vw}_x$. Figure 10(b) shows the sites that were removed using this property; the VD of the remaining sites are shown in Figure 10(c).

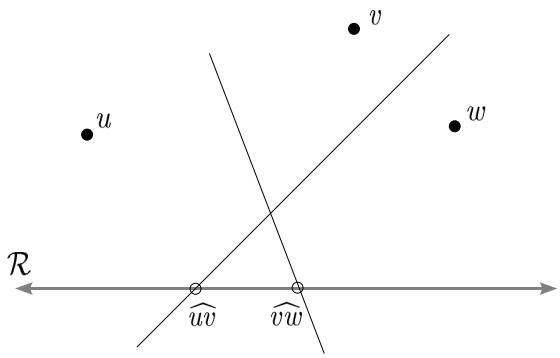

Fig. 11. The Voronoi region of v does not intercept line $\mathcal{R}$ if $\widehat{uv}_x \geq \widehat{vw}_x$.

(1) The current site s_i is equal to s_1 (leftmost site), i.e., $i = 1$.
(2) For each pixel p of line $\mathcal{R}$, from left to right, do:
—if $i < m$, and $d(p, s_i) > d(p, s_{i+1})$
do $i = i + 1$
—EDT(p) = $d(p, s_i)$

Fig. 12. 2D EDT by Maurer for a line of the image, given its relevant sites

In what concerns line $\mathcal{R}$, nothing differs from the VD with all the sites, Figure 10(a), to the partial VD of Figure 10(c). Once the relevant sites are found for $\mathcal{R}$ using the above restrictions, it is easy to determine the final EDT for the pixels in this line.

The relevant sites can be indexed only by the order in which they appear from left to right: $s_1, s_2, \ldots, s_m$, with $m < n_{cols}$ and $s_1.x < s_2.x < \cdots < s_m.x$. Such ordered sites also implicitly represent the intersection of the VD with line $\mathcal{R}$; since s_i is to the left of $s_i + 1$, then $RV(s_i)$ constrained to $\mathcal{R}$ is also to the left of $RV(s_i + 1)$ in $\mathcal{R}$, for $i = 1 \ldots m$.

Given this set of relevant sites ordered by column, the EDT in line $\mathcal{R}$ is generated in $O(n_{cols})$ by the following algorithm:

Running the algorithm in Figure 7.8 for each line, it is clear that this query stage of Maurer is $O(n^2)$ for an $n \times n$ image. Since the site-removal stage is also $O(n^2)$, the final algorithm is $O(n^2)$.

The 1D EDT is used both to eliminate irrelevant sites and to compute $d(p, s_i)$ in the procedure of Figure 7.8 in the following way: $d(p, s_i) = EDT_{1D}(p) + (p_x - i)^2$. It is the same principle used in Transformation 2 of Saito's algorithm described in Section 7.7.

7.9 Meijster's algorithm

The method of Meijster et. al. [Meijster et al. 2000] follows the same concepts as mentioned in the description of Saito's algorithm in Section 7.7, the difference being that distance minimization in the second step is performed in a more efficient way. In a first pass on column j their algorithm represents the lower envelope of parabolas $f_i(y) = G(i, j) + (y - j)^2$ explicitly by their intersections, where G is

defined in Transformation 1 (Section 7.4.4). This is a method for obtaining the boundary of Voronoi regions of the black pixels constrained to column j of the image, as described in Section 7.8.

Meijster's algorithm is similar to Maurer's algorithm, as confirmed by our empirical tests reported in Section 9. In fact, the authors of Meijster's method state in another publication [Hesselink et al. 2005] that they had rediscovered Hirata's method [Hirata 1996], and that in their opinion Maurer's algorithm is a re-discovery of Hirata as well. Maurer, however, clearly derives from the algorithm by Breu et. al. [Breu et al. 1995], as mentioned in Section 7.8. It turns out that Hirata [Hirata 1996] was inspired by (and cites) both Breu's algorithm for constructing the Voronoi diagram and Saito's idea of lower-envelope of parabolas. We also note that Meijster's method differ from Hirata's in that it computes the lower envelope of the parabolas without explicit use of a stack.

7.10 Lotufo-Zampirolli's algorithm

This section describes the method of Lotufo-Zampirolli, whose ideas are based on gray-scale mathematical morphology [Serra 1982]. Shih and Mitchell [Shih and Mitchell 1992] showed that the EDT can be computed using gray-scale erosion of the image by the following structuring function:

$$b_e(x) = -d_e^2(x, O)$$

where O is the origin and $d_e^2(p, q)$ is the squared Euclidean distance. The input image must have an 'infinity' value wherever the distances are to be computed (i.e. white pixels). This infinity value is a number greater than every (squared) distance to be found in the image, e.g. the length of the diagonal plus one.

This result enables the use of fast mathematical morphology algorithms to compute the exact EDT. Two concepts are commonly used in the design of efficient morphological algorithms: decomposition of structuring elements and erosion by propagation. They form the essence of Lotufo-Zampirolli's EDT.

Shih and Mitchel showed that the structuring element b_e can be decomposed in the following way:

$$b_i = \begin{bmatrix} -4i+2 & -2i+1 & -4i+2 \\ -2i+1 & \mathbf{0} & -2i+1 \\ -4i+2 & -2i+1 & -4i+2 \end{bmatrix}$$
$$b_e = b_1 \oplus b_2 \oplus b_3 \cdots$$

where the origin of b_i is marked in boldface and '$\oplus$' is the Minkowski sum [Serra 1982]. Therefore, due to a property of mathematical morphology, the EDT can be written in terms of successive erosions:

$$\varepsilon_{b_e}(f) = \cdots \varepsilon_{b_3}(\varepsilon_{b_2}(\varepsilon_{b_1}(f)))$$

In Lotufo-Zampirolli, the structuring element is further decomposed, resulting in

1D elements in the north (N), south (S), east (E), and west (W) directions:

$$
\begin{aligned}
b_{N_i} &= \begin{bmatrix} -2i+1 \\ 0 \end{bmatrix}, \; b_{E_i} = \begin{bmatrix} 0 & -2i+1 \end{bmatrix}, \\
b_{S_i} &= \begin{bmatrix} 0 \\ -2i+1 \end{bmatrix}, \; b_{W_i} = \begin{bmatrix} -2i+1 & 0 \end{bmatrix} \\
b_e &= \cdots \oplus b_{N_2} \oplus b_{N_1} \oplus \cdots \oplus b_{S_2} \oplus b_{S_1} \oplus \cdots \\
&\quad \cdots \oplus b_{W_2} \oplus b_{W_1} \oplus \cdots \oplus b_{E_2} \oplus b_{E_1}. \qquad (6)
\end{aligned}
$$

For this reason, and by the idempotence property of mathematical morphology [Serra 1982], the distance transform can be computed by first eroding each column by $b_{N_1}, b_{N_2}, \ldots$ until stability (i.e. the column does not change), and subsequently eroding the columns by $b_{S_1}, b_{S_2}, \ldots$ followed by the erosion of the lines by $b_{E_1}, b_{E_2}, \ldots$, and by $b_{W_1}, b_{W_2}, \ldots$. The algorithm uses two queues of pixels, effectively performing 1D propagation to concentrate computation on the pixels that change from erosion to erosion. Further details can be found in the original paper [Lotufo and Zampirolli 2001].

7.11 Cuisenaire's Propagation using Multiple Neighborhoods

The EDT method proposed in 1999 by O. Cuisenaire and B. Macq [Cuisenaire and Macq 1999b; Cuisenaire 1999] basically adds a correction stage over the PSN algorithm described in Section 7.4.1. As seen in Section 7.3, the exact EDT can be computed by the PSN using a sufficiently large neighborhood. Since this can be costly, Cuisenaire proposes that, after a 4-neighborhood PSN, propagation using larger neighborhoods is performed only in places that might need it. As explained bellow, such places are points in the boundaries of Voronoi regions that may be 4-disconnected with other pixels of the same region, causing errors.

7.11.1 *First stage: Propagation using a fixed neighborhood.* The PSN algorithm proposed by Cuisenaire has some peculiarities relative to the description given in 7.4.1. Each node of the contour set stores a point $p = (p_x, p_y)$ and the relative distance $dp = (dp_x, dp_y)$ of the nearest source pixel. The vector dp is used to determine the distance of the neighbor of the current pixel p to the nearest source to p in the following way. For $\mathbf{n} \in \mathcal{N}$, the neighbor of p is given by $q = p + \mathbf{n}$, and its distance to the nearest source of p is $\|dp + \mathbf{n}\|^2 = (dp_x + n_x)^2 + (dp_y + n_y)^2$. Figure 7.11.1 describes Cuisenaire's PSN with these details. For the sake of comprehension, the reader should compare it to the general propagation procedure (Figure 5).

In practice, step 2 initializes the contour set with the boundary pixels, that is, the white pixels having at least one black 4-neighbor. Such pixels are unit distance from the set of white pixels and, therefore, are inserted in *bucket*(1), while the current distance of step 3 starts at 1. Therefore, the trivial propagation of the pixels having zero distance to unit distance is explicitly performed in the initialization step, avoiding one insertion and propagation cycle.

It is possible to determine whether all the buckets are empty in step 4 of Figure 7.11.1 only by testing a small predetermined number of the last buckets up to the bucket d [Cuisenaire 1999]. For the 4-neighborhood it suffices to test if the last

(1) Every white pixel receives distance $d = \infty$.
(2) Insert the white pixels in $bucket(0)$ together with relative coordinates $(0, 0)$.
(3) $d = 0$ {*Current distance starts at 0.*}
(4) While the contour set (*buckets*) is not empty
    While $bucket(d)$ is not empty
        Remove p and dp from $bucket(d)$ {*pixel having smallest cost.*}
        For every $\mathbf{n} \in \mathcal{N}$
            $D_{new} = \|dp + \mathbf{n}\|^2$
            If $D_{new} < D(p + \mathbf{n})$
                {*p propagates to* $p + \mathbf{n}$.}
                $D(p + n) = D_{new}$
                Insert $(p + \mathbf{n}, dp + \mathbf{n})$ in $bucket(D_{new})$.
            end if
    $d = d + 1$

Fig. 13. Cuisenaire's PSN

$2\sqrt{d} + 1$ are empty, since $\|dp + n\|^2 \leq (\sqrt{d} + 1)^2 = d + 2\sqrt{d} + 1$.

7.11.2 *Second stage: correction using multiple neighborhoods.* During PSN, the method verifies if a pixel p has propagated minimum distance information to a neighbor. If p is near to the boundary of a $\mathcal{N}$-connected Voronoi Region, then it will not propagate to any neighbor, since they are closer to other sites and, therefore, are outside of p's VR. It is only at these non-propagating pixels that propagation using larger neighborhoods might be necessary, as noted by Cuisenaire and explained in the following.

Disconnected Voronoi regions cause distance errors, for example in the simple 4-neighborhood propagation, as discussed in Section 7.3 and illustrated in Figure 4(a). To correct this type of error one can use a larger neighborhood on the boundary of the 4-connected VR as to recover longer-range connectivity. In Figure 4(a) it is sufficient to propagate the pixel between p_2 and q with an 8-neighborhood to correct the map. However, for other images an error like in Figure 4(c) can occur. Again, the idea is to use a larger neighborhood, for example 5×5, in the pixel marked with distance 116 in Figure 4(c). The method by Cuisenaire is an efficient realization of this idea [Cuisenaire and Macq 1999b; Cuisenaire 1999].

As described in Section 7.3, the EDT for a neighborhood $\mathcal{N}$ ($\mathcal{N}$-EDT) is correct for every distance smaller than a predetermined value $d_c(\mathcal{N})$. Furthermore, the distance value $d_{np}(p)$ of a non-propagating pixel p leading to an error is well-defined for a given distance $\mathcal{N}$. In other words:

PROPERTY 7. *For each neighborhood $\mathcal{N}$, there exists a distance $d_{np}(\mathcal{N})$ such that: $D(p) \geq d_{np}(\mathcal{N})$ if and only if for some image, p is not propagated during the $\mathcal{N}$-EDT and has generated a disconnection error in its Voronoi region.*

In simple terms, $d_{np}(\mathcal{N})$ is the minimum distance value for which a non-propagating pixel leading to an error can occur in the $\mathcal{N}$-EDT. Thus, not every non-propagating pixel in a $\mathcal{N}$-EDT need a larger neighborhood – only those with a distance greater than or equal to $d_{np}(\mathcal{N})$. Therefore multiple propagation can stop for every pixel with sufficiently small distance.

Neighborhood		Closest error		Closest non-propagator	
k	$\mathcal{N}_{\mathbf{k}}$	relative position	d_c	relative position	d_{np}
1	4-neighbors	(2,2)	8	(1,1)	2
2	3×3	(12,5)	169	(10,4)	116
3	5×5	(25,7)	674	(22,6)	520
4	7×7	(48,10)	2404	(44,9)	2017
5	9×9	(72,12)	5328	(67,11)	4610
6	11×11	(108,15)	11889	(102,14)	10600
7	13×13	(143,17)	20738	(136,16)	18752
8	15×15	(192,20)	37264	(184,19)	34217
9	17×17	(238,22)	57128	(229,21)	52882
10	19×19	(300,25)	90525	(290,24)	84676
11	21×21	(357,27)	128178	(346,26)	120392
12	23×23	(420,29)	177241	(408,28)	167248
13	25×25	(500,32)	251024	(487,31)	238130
14	27×27	(574,34)	330632	(560,33)	314689
15	29×29	(667,37)	446258	(652,36)	426400
16	31×31	(768,40)	591424	(752,39)	567025
17	33×33	(841,41)	708962	(824,40)	680576
18	35×35	(972,45)	946809	(954,44)	912052
19	37×37	(1054,46)	1113032	(1035,45)	1073250
20	39×39	(1200,50)	1442500	(1180,49)	1394801
21	41×41	(1312,52)	1724048	(1291,51)	1669282
22	43×43	(1452,55)	2111329	(1430,54)	2047816
23	45×45	(1575,57)	2483874	(1552,56)	2411840
24	47×47	(1680,58)	2825764	(1656,57)	2745585
25	49×49	(1862,62)	3470888	(1837,61)	3378290

Table II. Nearest errors and non-propagating positions for various neighborhood sizes.

The value of d_{np} for a neighborhood $\mathcal{N}$ can be computed by an exhaustive search algorithm, in the same way that d_c is determined (see Section 7.3). It is worth noting that this algorithm for $d_{np}(\mathcal{N})$ was vaguely described by Cuisenaire [Cuisenaire and Macq 1999b; Cuisenaire 1999]. A C language implementation was devised for this survey and is now part of an open source C library [Fabbri a]. Table II shows some values of d_c and d_{np} for various neighborhoods. The relative positions in which an error can occur and of the non-propagating pixel are also shown. Moreover, $\mathcal{N}_1$ denotes the 4-neighborhood, and $\mathcal{N}_k$ is the $2k-1 \times 2k-1$ square neighborhood, $k > 1$, listed in the first two columns of Table II.

The values in Table II were generated by our exhaustive search implementation. The values are identical to those published by Cuisenaire in his paper [Cuisenaire and Macq 1999b] for all lines up to $k = 16$. The lines $k > 16$ are published for the first time in the present manuscript. It is worth noting that d_{np} in line 16 equals the value in Cuisenaire's paper [Cuisenaire and Macq 1999b] but differs from the

Input:

$\mathcal{N}_1$-EDT: output of PSN (Figure 7.11.1);
buckets: contains the pixels that did not propagate during PSN;
d_{max}: maximum distance evaluated during PSN;
k_{max}: Number of neighborhoods for which $d_{np}(\mathcal{N}_k)$ has been precomputed.

Output:

D: $\mathcal{N}_k$-EDT with $k = k_{max}$. This EDT is exact if the image has only distances smaller than $d_c(\mathcal{N}_k)$, but may contain errors otherwise.

Explanation of variables:

d: current distance; indexes into the buckets vector.
k: indexes neighborhood, i.e. refers to the neighborhood $\mathcal{N}_k$

begin

For $k = 2$ to k_{max}
  For $d = d_{np}(\mathcal{N}_{k-1})$ to the minimum of $d_{np}(\mathcal{N}_k)$ and d_{max}
    For all $\mathbf{n} \in \mathcal{N}_k$
      $D_{new} = \|dp + \mathbf{n}\|^2$
      If $D_{new} < D(p + \mathbf{n})$
        $D(p + \mathbf{n}) = D_{new}$
        Insert $(p + \mathbf{n}, dp + \mathbf{n})$ in $bucket(D_{new})$.
      end if

end

Fig. 14. Cuisenaire's multiple neighborhood propagation (PMN)

value in Cuisenaire's thesis [Cuisenaire 1999]. Since both Cuisenaire's code and his paper use the same value as ours (567025), the thesis value (567027) is probably a typo.

Cuisenaire's PMN algorithm consists in successive propagations with increasing neighborhoods $\mathcal{N}_k$ where necessary. Given a non-propagating pixel p, which remained in the priority queue after the $\mathcal{N}_1$-EDT, p will be propagated using the smallest necessary neighborhood to correct the associated errors. For example, if $D_{\mathcal{N}_1}(p) = 3000$, one can verify in Table II that p shall be propagated again using a 9×9 neighborhood for an exact EDT, since 3000 is between 2017 and 4610. In other words, of all non-propagating pixels p in an $\mathcal{N}_k$-EDT, the ones to be propagated with $\mathcal{N}_{k+1}$ are those satisfying $d_{np}(\mathcal{N}_k) \leq D_{\mathcal{N}_k}(p) < d_{np}(\mathcal{N}_{k+1})$. Figure 14 describes the multiple-neighborhood EDT in detail.

Cuisenaire states that his algorithm apparently remains $O(n^2)$ even in the worst case, based on experiments [Cuisenaire and Macq 1999b; Cuisenaire 1999]. However, this conjecture has not been proved to this date. It is important to note that the pre-computation of distances $d_{np}(\mathcal{N}_i)$ is time-consuming even for moderate values of i. However, these values need only be tabulated once and for all.

In a variant of PMN, called PMON (Propagation by Oriented Neighborhoods), propagation is restrained only to the necessary directions [Cuisenaire 1999]. It is not yet clear if this is more efficient than PMN, due to the added cost of having to compute the propagation directions. Some empirical tests by Cuisenaire suggest

(1) Initialize the distance of all white pixels to a sufficiently large value.
(2) Initialize the *Contour Set* doing, for each pixel p:
—for $k = 0, 2, 4, 6$, if $N_k(p)$ is white, store p into $\mathcal{L}_1$ with direction index equal to $k + 1$.
(3) While the contour set $\mathcal{L}_1 \bigcup \ell_1$ is not empty:
(4) for each pixel $p \in \ell_1$
  (a) Compute a new distance for the direct neighbor $N_k(p)$ based on the distance at p.
  (b) If the new distance of the neighbor is less than its current:
    —Update its current distance.
    —Put $N_k(p)$ in ℓ_2 with direction index k.
(5) For each pixel $p \in \mathcal{L}_1$:
  (a) Compute a new distance for the indirect neighbor $N_k(p)$ based on the distance at p.
  (b) If this new distance of $N_k(p)$ is less than its current distance:
    —Update its current distance.
    —Store $N_k(p)$ into $\mathcal{L}_2$ with direction index k.
  (c) Compute a new distance for both the direct neighbors $N_{k+1}(p)$ and $N_{k-1}(p)$ based on the value at p.
  (d) If the new distance of $N_{k+1}(p)$ is smaller than its current distance:
    —Update its current distance.
    —Store $N_{k+1}(p)$ into ℓ_2 with direction index $k + 1$.
  (e) Do the same for item $N_{k-1}(p)$, with direction $k - 1$ when inserted in ℓ_2.
(6) Do $\mathcal{L}_1 = \ell_2$, $\mathcal{L}_2 = \emptyset$, $\ell_1 = \ell_2$, and $\ell_2 = \emptyset$.

Fig. 15. Eggers' d_∞ propagation

that PMON is faster than PMN for sufficiently large images.

7.12 Eggers' algorithm

Eggers introduced two ordered propagation techniques, denoted d_∞ and d_1, that restrict the propagation to Euclidean shortest paths [Eggers 1998]. Only d_∞ is described in this monograph, since it is analogous to d_1 while having a better average performance.

Eggers separates the contour set in two lists: $\mathcal{L}$ storing the so-called *main contour pixels*, and ℓ storing the *minor contour pixels*. The neighbors of a given pixel are ordered as bellow:

$$\begin{array}{ccc} N_3(p) & N_2(p) & N_1(p) \\ N_4(p) & p & N_0(p) \\ N_5(p) & N_6(p) & N_7(p) \end{array}$$

For each contour pixel, one associates a *direction index* k which points to the neighbor $N_k(p)$ to which p shall propagate information. Instead of propagating distance information for all eight neighbors, the main contour pixels propagate only to three neighbors $N_k(p)$, $N_{k+1}(p)$ e $N_{k-1}(p)$, where k is the direction index of the pixel. The minor contour pixels propagate only to $N_k(p)$. Furthermore, the direction indexes of the main contour pixels are odd (indicating indirect neighbors), while the indexes of the minor pixels are even.

Figure 15 describes the d_∞ propagation, the overall steps being the same as in Figure 5 of Section 7.4.1. In practice, four dynamic lists are used, two for each iteration. The lists $\mathcal{L}_1$ and ℓ_1 store the main and minor contour pixels, respectively, while $\mathcal{L}_2$ and ℓ_2 store the same information but for the next iteration.

Figure 16 illustrates the d_∞ propagation process for a single pixel of interest.

Note that the information in p is propagated to a pixel q through the shortest Euclidean path from p to q, consisting in $d_1(p,q) - d_\infty(p,q)$ indirect neighbors followed by $2 \cdot d_\infty(p,q) - d_1(p,q)$ direct neighbors. This propagation process is called d_∞ since, starting from a single black pixel p, the contour set at iteration $m+1$ is the circle with radius m in the d_∞ metric.

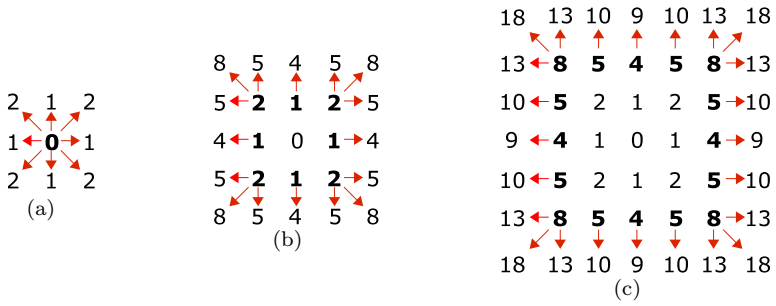

Fig. 16. First three iterations of Eggers' sufficient propagation d_∞. Bold values indicate pixels in the contour set. Main contour pixels are those propagating to three pixels, and the minor are those propagating only to a single one. An indirect pixel $N_k(p)$ of a main pixel p having directional index k will be a main pixel during the next iteration. The remaining ones will be minor pixels.

8. METHODOLOGY

This work tested and compared the main state-of-the-art EDT algorithms, denoted by: Maurer [Maurer et al. 2003], PMN (or Cuisenaire) [Cuisenaire and Macq 1999b], Saito [Saito and Toriwaki 1994], Lotufo-Zampirolli [Lotufo and Zampirolli 2001], Meijster [Meijster et al. 2000], and Eggers [Eggers 1998]. Some of these algorithms depend on the size of the image and on the number of feature pixels, others do not depend on the number of feature pixels, or depend on some other factor such as maximum distance to be computed and geometric factors such as the configuration of Voronoi regions.

8.1 Test images

In order to study the speed and exactness relative to the content and size of the image, tests using the following images are proposed in this work:

(1) **A single black or white pixel in an image corner**. The EDT, in this case, produces the largest and smallest possible distances for a given image size, i.e. $n\sqrt{2}$ (the diagonal) and 1, respectively. Moreover, the number of pixels receiving a non-zero distance is also the largest and smallest possible, respectively. For scanning algorithms, not all the scans are necessary to compute the EDT for this type of image, depending on the chosen corner having the distinct pixel. This test case was also used by Maurer et.al. [Maurer et al. 2003].

(2) **A white disk inscribed in the image**. This test was suggested by Saito [Saito and Toriwaki 1994] and also by Cuisenaire [Cuisenaire and Macq 1999b]. It is a good test for exactness due to the following fact. The Voronoi diagram of the pixels along a circle is very regular in the continuous plane, and consists

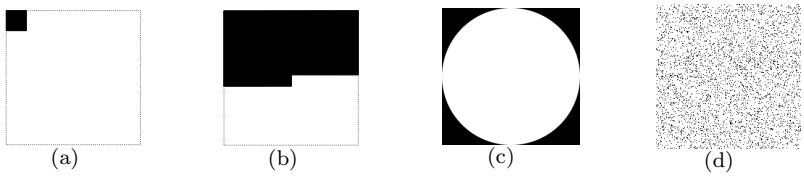

Fig. 17. Some of the test images used in the paper. (a) one pixel in the corner; (b) half-filled image; (c) white circle inscribed in the image; and (d) images with varying number of randomly placed white pixels (this particular example has 90% white pixels).

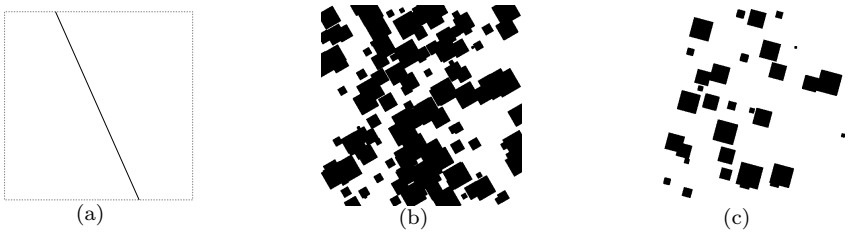

Fig. 18. Orientation tests: (a) digital line in different directions; and (b), (c) randomly generated squares with varying numbers, sizes and orientations.

in a series of radial lines, and the influence zones are very thin triangular regions, whose tips meet at the center of the circle. However, the discrete Voronoi regions in this case are irregular, specially near the center of the disk. Furthermore, the largest the circle, the denser is the sampling of the Voronoi regions, however these regions will be thinner.

(3) **Half-filled image.** This is the worst case of the brute force algorithm, as described in Section 7.1. This image has also a uniform distance distribution, i.e., there is approximately the same number of pixels for each different distance.

(4) **Random pixels, comprising 1%, 2%, 5, 10, 20,. . . , 90, 95, 98 and 99% of the image**. This shall provide an idea of the performance of the algorithms relative to the number of feature pixels. The brute force EDT will take longer for 50% feature (white) pixels, as analyzed in Section 7.1. Therefore, this is the general expected behavior to be compared to the other algorithms.

(5) **A line of black pixels turning from** $0°$ **to** $90°$ **through the center of the image**. The worst case of the propagation algorithms occurs for non-horizontal, non-vertical and non-diagonal orientations, as explained by [Ragnemalm 1992]. This test case was also used by Cuisenaire in some of his comparisons.

(6) **Random squares**. These images are generated by randomly choosing the centers and sizes of black squares rotated by $\theta \in [0, 90°]$. The squares are filled and plotted into the image until the black pixels add up to a percentage value p. An image of this type shall be denoted by the pair (p, θ). This test was proposed by Eggers, since it is based on a synthetic image having more similarity to real images with some orientation. For assessing performance relative to orientation, this test is more realistic than a single rotating line.

(a) (b)

Fig. 19. Test image composed of edge maps of real images.

(1) Run the algorithm, measuring time. If the image is too small, repeat this step many times until a good measurement precision is obtained.
(2) If the execution time is too high (e.g. more than $30s$), store the resulting map in a file for use in further exactness tests.
(3) Compute a reference EDT for exactness comparisons:
—If the image is sufficiently small, run the brute-force algorithm. Save the result in a file for further tests.
—If the image is too large, the reference will be an EDT computed by a fast algorithm that has shown to be exact in preliminary tests. However, if the image is very large, check if there is a file having a precomputed reference image. If there is no such file, compute and save the reference EDT in a file for future tests.
(4) Compare the distance map of the algorithm being tested with the reference map. If there is an error, output the following information:
—Number of incorrect pixels and $\%err = \frac{\text{number of incorrect pixels}}{\text{total number of white pixels}}$
—Maximum error $\%err_{max} = \frac{\text{maximum error}}{\text{maximum distance}}$

Fig. 20. Main procedure for performance and exactness evaluation

(7) **Binary images of real objects**. Borders from the Lenna image obtained by thresholding the response of an edge detector were used. Lenna was chosen since it has been universally used as an impartial *benchmark* for image analysis algorithms.

In addition to tests described above, a possibility for a future work would be to extract measures from the images to characterize the performance of the algorithms with respect to the shape of the input object, as described in the objectives of this work (Section 2).

8.2 Test procedure

The tests were performed as described in Figure 20. Test results were stored in text files in readable and well-structured form, as to enable data extraction using utilities based on regular expressions. Performance plots were generated in Scilab, using tables assembled from such files.

In Step 4 of Figure 20, the reference DT for large images was Maurer's algorithm, which proved to be consistently exact in preliminary tests.

Memory usage is not assessed in the present work. However, this task is not very difficult to be performed in Unix systems. One way to measure memory performance at run-time is using system calls or, more easily, using specialized utilities. Another way is to use Linux's `'/proc'` filesystem. Yet another option

is to libraries or utilities that substitute `malloc` calls by routines that track the memory usage of a process. Examples of these are the libraries and utilities from Ccmalloc [Biere] and Valgrind [Seward].

As the empirical results are being obtained, theoretical studies were performed, and possible proofs to observed properties were attempted; and conversely, theoretical results and conjectures were experimentally verified. Some examples of specific questions answered by this work are:

—Does the recent method of Cuisenaire [Cuisenaire and Macq 1999b] have linear complexity even for images that had not been tested by the original author? For questions of this kind, if the experiments show evidence of positive answers, it is suggested that they be mathematically proven in a future work. For negative answers, errors in the implementation and theory were sought.

—Is Saito's algorithm (Figure [Saito and Toriwaki 1994]) $O(n^3)$? Is the worst-case frequent in practice? Is it really slower than Meijster's algorithm?

—Is the recent algorithm by Maurer [Maurer et al. 2003] really fast and exact as the author claims?

8.3 Other properties assessed in this work

In addition to time performance and exactness, another important criterion to compare EDT algorithms regards the difficulty of implementations. A worst-case $O(n^3)$ method may be preferable if it is reasonably fast in the average case and easier to code. In order to assess the ease of implementation, the number of lines of the implementation of each EDT by the same person can be used, as well as a listing of the data structures being used. However, the real analysis of this criterion has to be personal, with arguments reasonably rooted in the programming experience.

9. EMPIRICAL RESULTS

9.1 Initial considerations

The tests were performed on a laptop having a Pentium M 2GHz processor, 1GB RAM, and a Gentoo Linux OS with kernel v2.6.17. The programs were built using GCC v4.1.1, and the source code is freely available on the Internet [Fabbri a] as a companion to this paper. For the automation of the tests, Bash scripts were used to coordinate the execution of C programs and to generate text files reporting test results. The display of empirical data was also programmed in the Scilab environment [INRIA]. Moreover, timing was coded using the ANSI-C `clock` function, which measures the CPU time, yielding reliable test results even under timesharing.

In this section, the most relevant results will be analyzed, focusing on large images, i.e. $n \times n$ with n in the order of 10^3. The complete results and test framework are available on the web [Fabbri et al. 2006].

9.2 Running time results

9.2.1 *Images with a point in the corner.* As shown in Figure 21, the best result occurs for Maurer, while Saito and Eggers perform similar to each other, followed by Lotufor Zampirolli and Meijster. Meijster was the slowest independent scan-

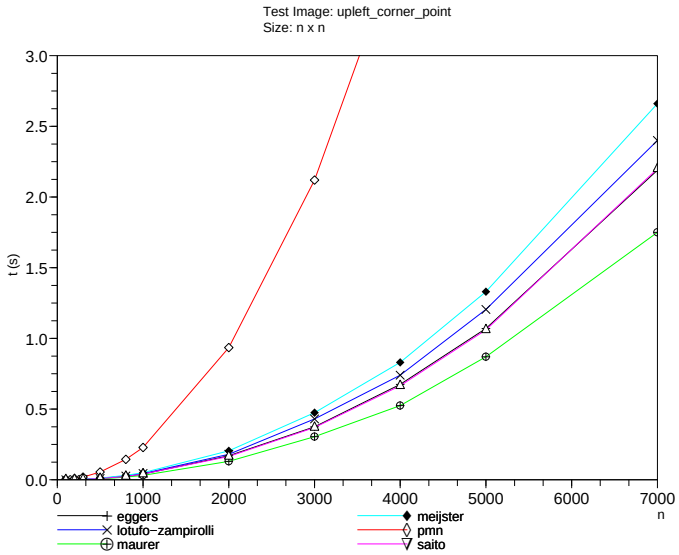

Fig. 21. Performance for images with a single black pixel in the upper-left corner.

ning algorithm for this case (about 50% slower than Maurer[6]). Cuisenaire is the slowest method, being 6 times slower than Maurer. Another interesting fact is that Eggers does well in this test, being an $O(n^3)$ propagation algorithm outperforming Cuisenaire, which is also a propagation algorithm supposedly $O(n^2)$.

9.2.2 *Circle images.* The best methods in this case were Meijster and Maurer, which performed very close to each other ($0.062s$ and $0.063s$ resp. for the 1000x1000 image), as shown in Figure 22. Cuisenaire and Saito come not too far behind. Eggers and Lotufo-Zampirolli are much slower, both performing around 8 times slower than Meijster.

9.2.3 *Lenna's contours.* As shown in Figure 23, again Meijster and Maurer display the best timing, followed by Saito and Lotufo-Zampirolli. The slowest methods were Eggers and Cuisenaire, both approximately 2 times slower than Meijster.

9.2.4 *Half-filled images.* As shown in Figure 24, this is the best relative performance of Eggers and the worst of Maurer and Meijster, among all types of image content we considered. Eggers is followed by Lotufo-Zampirolli and Saito, which are close to each other, then comes Cuisenaire, Meijster and Maurer, also close to each other. The propagation algorithms were better than raster-scanning algorithms for this image. Although Maurer and Meijster were among the best methods for many different types of input, in this case they are both around 2 times slower than Eggers.

9.2.5 *Random pixels.* The percentage of interest pixels (black pixels) proved to be of great impact to the performance of the methods. As shown in Figure 25,

[6]In this text, this kind of figure refers to the largest image of the test, unless otherwise stated.

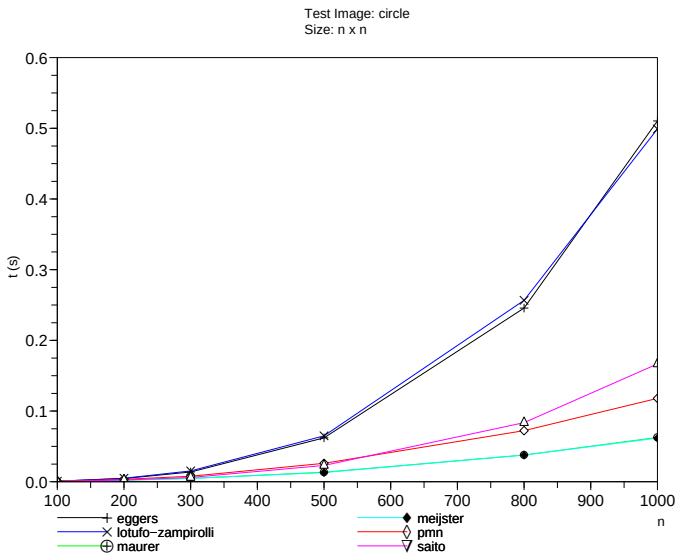

Fig. 22. Performance for images containing an inscribed white circle.

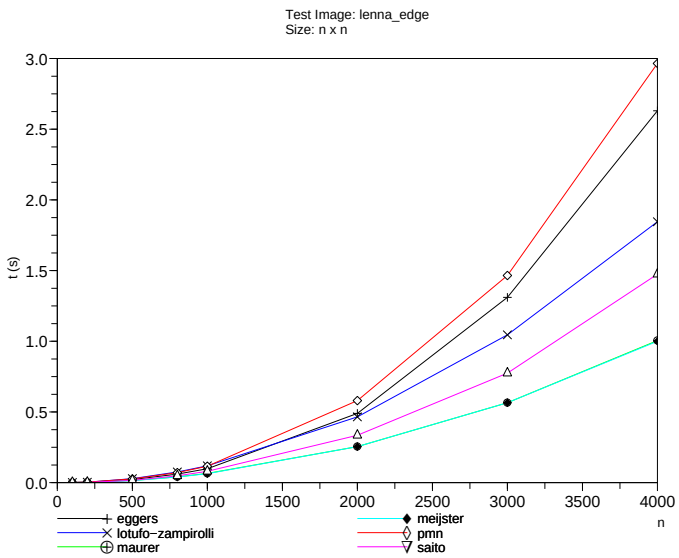

Fig. 23. Performance for Lenna's contours.

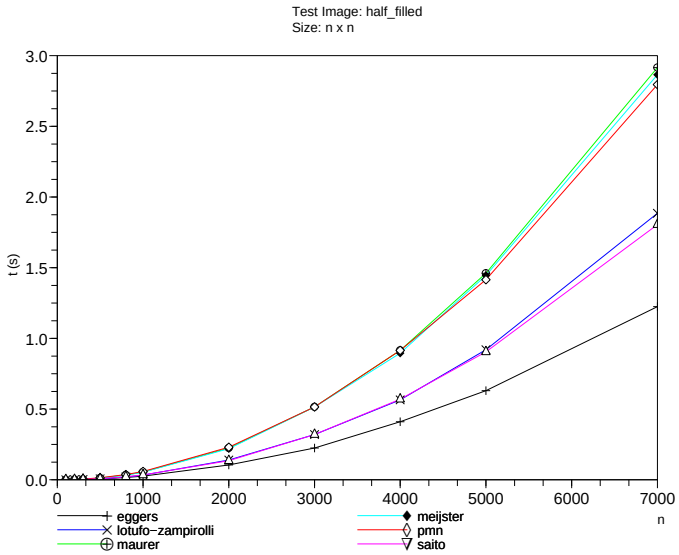

Fig. 24. Performance for the half-filled image.

the speed of almost every algorithm is roughly proportional to the number of white pixels (i.e. where distance values are to be computed). Eggers is the only exception, presenting peak time around 60%, with its best performance still for the case of a single white pixel. This peak around 60% is similar to the behavior of the brute-force algorithm, which has its peak in 50%.

For 100×100, the fastest algorithms are Saito and Lotufo-Zampirolli, above 20%, and PMN and Saito, below 20%. Saito remains very close to Lotufo-Zampirolli and Meijster very close to Maurer, as the image size increases.

Maurer's algorithm (and technically Meijster's) is the slowest below 20%, while Cuisenaire is the slowest above 70% for all images larger than 100×100. However, Meijster and Maurer are the steadiest (slight advantage to Meijster), followed by Lotufo-Zampirolli. Eggers and PMN were the most unstable of all, slightly improving for large images.

9.2.6 *Turning line.* All the methods were shown to depend on orientation. As shown in Figure 26, Meijster, Maurer and PMN were the most stable (i.e. independent of orientation). The least stable methods were Eggers, Lotufo-Zampirolli, and Saito. As the image size increases, PMN and Maurer apparently get stabler, since they get a lot faster compared to the other methods. However, in reality the stability remains the same, as shows the zoom of the plot of Figure 26(d) in Figure 27.

The performance disparity between each of the methods is larger for slopes different from 0° and 90°, specially for angles between 60° and 70°. As can be expected, the time curves are symmetric with respect to 90°. Clearly, Saito has its worst performance at 60°. Eggers has peaks of time around 20° and 75°, and peaks around 0° and 45°. PMN, in turn, is slowest for 45° (differently than Eggers), and

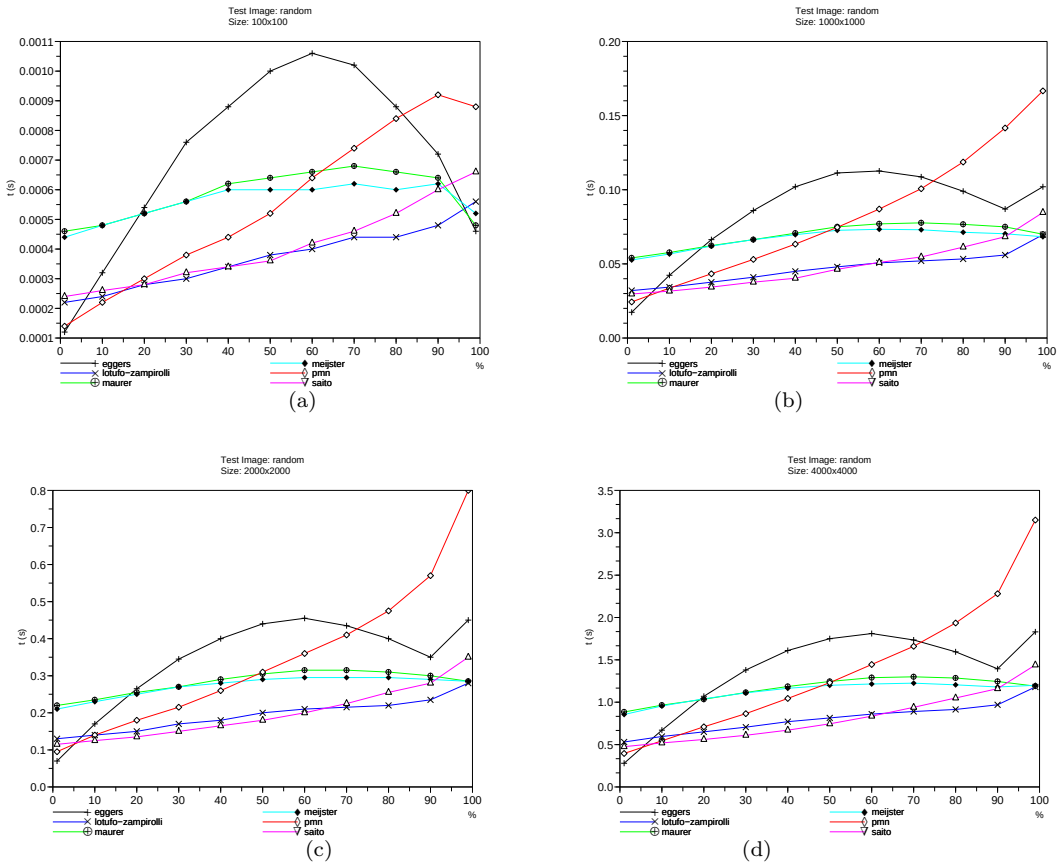

Fig. 25. Performance of EDT methods for images with varying percentage of white pixels at random locations. Plots are shown for sizes 100×100 (a), 1000×1000 (b), 2000×2000 (c), and 4000×4000 (d).

is fastest at 0°. Lotufo-Zampirolli displayed maximum time around 55° and best performance at 0°. Curiously, Maurer has a minimum around 90° and a constant maximum between 0° and 40°. This asymmetry is due to the choice of processing columns first than lines, instead of lines first. The other raster-scanning algorithms also present this line/column asymmetry, but did not show considerable performance difference between horizontal and vertical orientations. Meijster proved to be more stable than Maurer, displaying a virtually constant profile with respect to orientation.

For all sizes greater than $100x100$, Meijster and Maurer were the fastest. The evolution of time curves relative to the image size shows evidence that Lotufo-Zampirolli, Eggers, and Saito are *not* linear-time.

9.2.7 *Random squares.* All the plots for this test, in Figures 28 and 29, show strong dependency of EDT algorithms with respect to the percentage of pixels. The

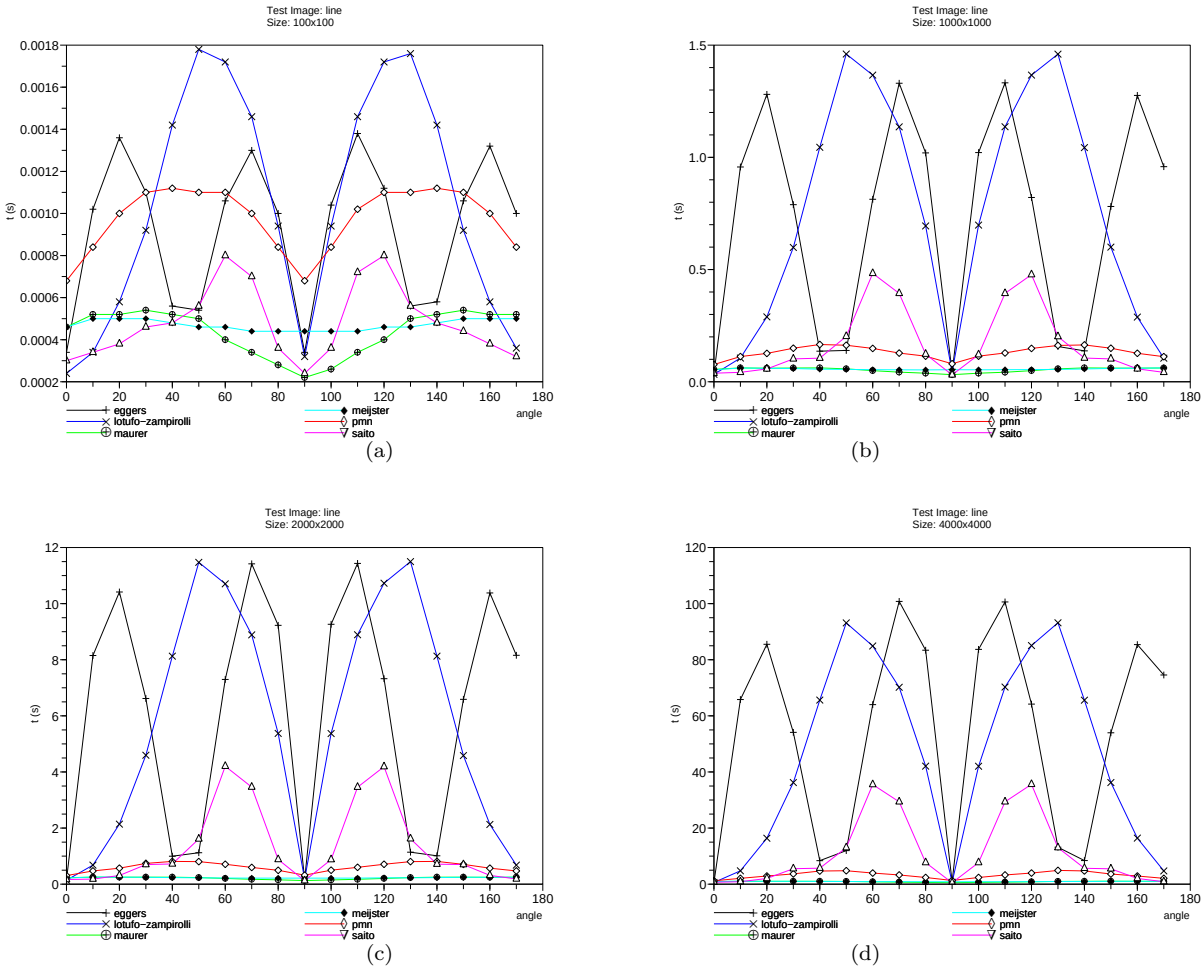

Fig. 26. Performance of EDT algorithms for images with a single line in different angles. Plots are shown for sizes 100×100 (a), 1000×1000 (b), 2000×2000 (c), and 4000×4000 (d).

only exception is Meijster's and Maurer's methods, which perform similarly to each other for large enough images and have displayed much more stability than the other methods, under any orientation. However, Meijster and Maurer were seldom the fastest. For high percentages of black pixels, Maurer is the slowest method, followed closely by Meijster, irrespective of orientation. This can be verified, for example, from the plots of 95% percent in Figure 29. Under any fixed orientation, all the other methods are faster for a larger amount of black pixels, since the fixed-angle curves are decreasing.

The time curves were all symmetric with respect to 45°. Moreover, for a high percentage of interest pixels (i.e. black pixels), all methods get less dependable on orientation. This reflects the fact that varying the orientation of an image having

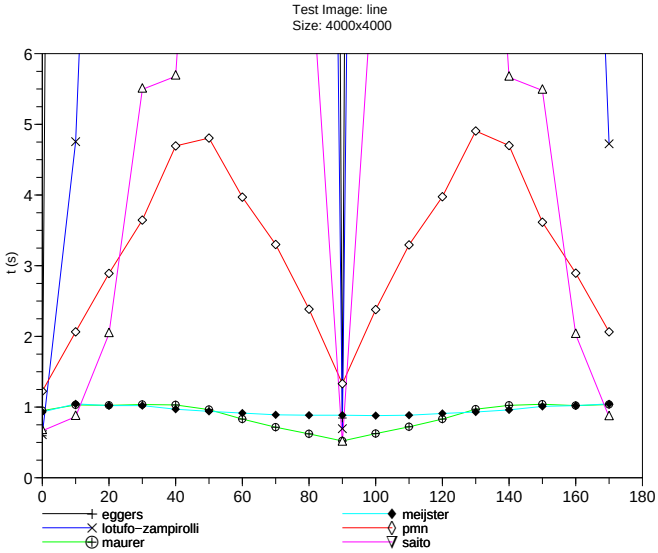

Fig. 27. Zoom of the plot from Figure 26(d).

few white pixels has little effect on the distribution of distances.

As in the line test, Lotufo-Zampirolli and Eggers were the most dependent methods on orientation. Cuisenaire was strongly dependent on percentage, specially for angles near 0°. For large images, Lotufo-Zampirolli displayed the worst mean at 45° and Eggers at 15°, except for high percentages of black pixels.

9.2.8 *Exactness.* All methods were confirmed to be exact for all test images and test procedure of Figure 20 from Section 8.

9.3 Discussion

The results are very diverse, no method being the fastest in all tests. An example of this diversity is that, although the test case of Lenna's edges is in a sense similar to random images having a low percentage of black pixels, the empirical results are quite different. This can be seen by comparing the plots of the Lenna test (Figure 23) with those of random pixels and squares, shown in Figures 25 and 28. Moreover, it is curious that the peak of Eggers at 60% white pixels in Figure 25 does not occur for the image of random squares for any fixed angle (Figure 28).

The speed of the methods is proportional to the number of interest pixels (black pixels), i.e. inversely proportional to the number of white pixels (where the distances are computed). This can be seen, for instance, in the plot for random pixels from Figure 25, where all curves have a tendency to increase with the percentage of white pixels. This behavior can also be verified in Figure 28, where the curves are decreasing with the number of black pixels.

The constant factors in the time function of an algorithm can be reduced by optimizing the implementation. Therefore, the plots have to be analyzed mainly through the shape of the curves, and less through absolute timing values. Further-

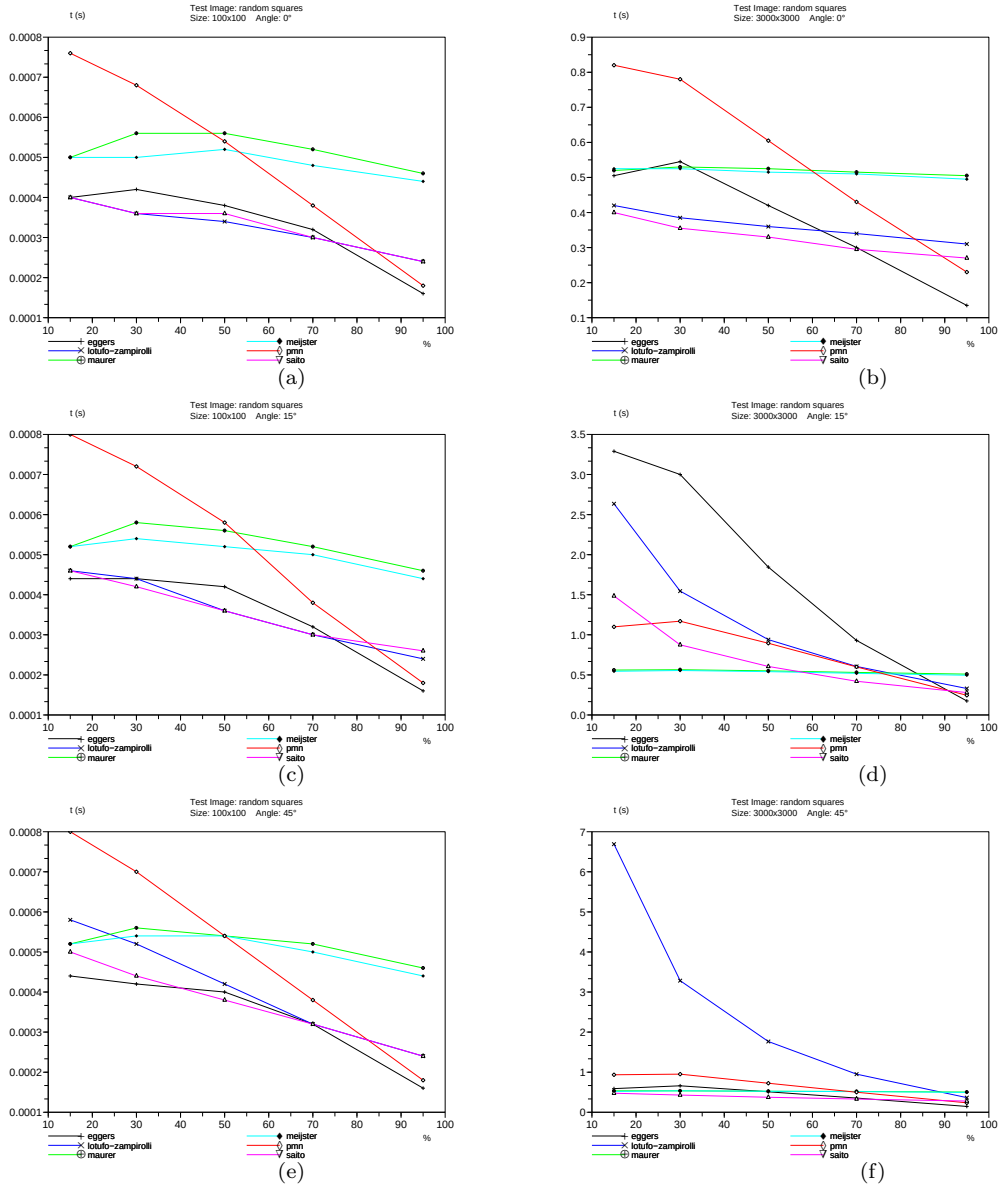

Fig. 28. Performance of EDT methods for random squares, varying percentage of black pixels (abscissas) and fixing orientation at 0° in (a) and (b), 15° in (c) and (d), and 45° in (e) and (f), for 100×100 images (left column) and 3000×3000 (right column).

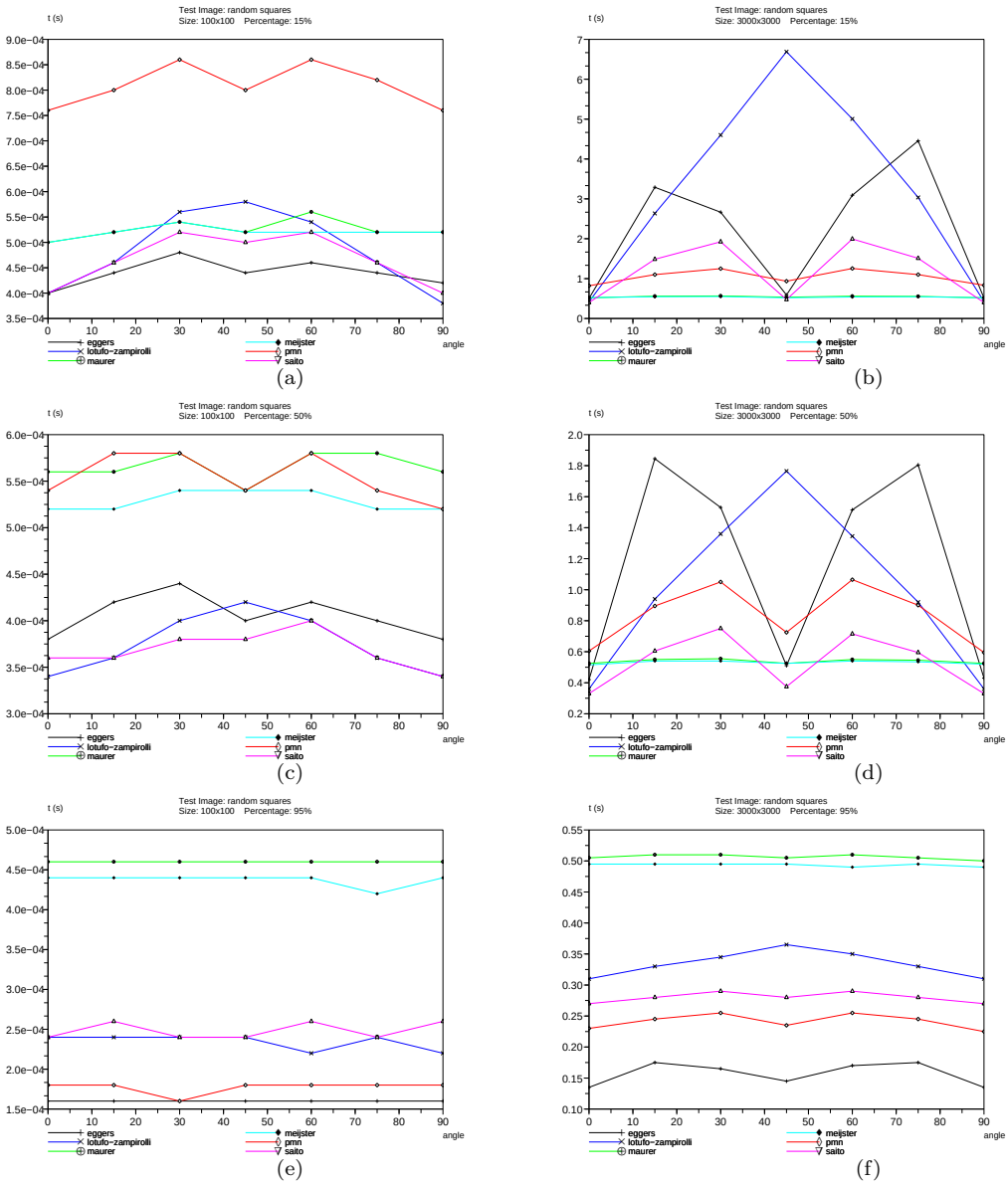

Fig. 29. Performance of EDT methods for images of random squares, varying the angle (abscissas), and fixing the percentage of black pixels at 15% in (a) and (b), 50% in (c) and (d), and 95% in (e) and (f), for 100×100 images (left columnt) and 3000×3000 (right column).

more, special attention has to be given to performance for large images, in which case constant factors have less influence when comparing algorithms with different complexities.

All angle tests were symmetric with respect to 90°, being unnecessary to test for angles outside the 0°–90° range. However, no method displays symmetry with respect to 45° in the 0°–90° range, except for Eggers, which displayed such symmetry for the cases of a rotating straight line and random squares.

9.4 Overall performance of each algorithm

The performance of each algorithm is discussed in the following.

9.4.1 *Performance of Cuiseinaire.* The worst case for Cuisenaire, among all tests, occurs for images having many white pixels. This fact is visible from the results for images with a single black pixel (Figure 21), and in the plots for varying percentage (Figures 25 and 28): the more white pixels, worst is the performance of Cuisenaire. This algorithm seems to be the most dependent on the number of interest pixels.

The method has shown to be relatively stable with respect to orientation, only being behind of Meijster and Maurer. As shown in Figures 26 and 29, PMN performs best for 0° or 90°, getting worst as the angle gets far from these values. Moreover, it is the only method besides Maurer and Meijster that displays linear complexity relative to the size of the line image and of the random squares.

Cuisenaire's algorithm was relatively hard to implement due to the complicated data structures involved. This is generally true of propagation algorithms.

9.4.2 *Performance of Maurer.* Maurer's algorithm (as well as Meijster's) proved to be the most stable in terms of image content, as shown in Figures 25, 26, 28 and 29. Still, the method does display some dependency on image content, specially in the test of turning line. As shown in Figure 27, its best performance occurs at 90°, curiously better than at 0°. However, this dependency on orientation is not verified in the test of random squares, as shown in Figure 29.

Maurer's algorithm is also technically the fastest for the majority of the images: Lenna's borders (Figure 23); inscribed circle (Figure 22); corner pixel (Figure 21); and the majority of orientations for the line test (Figure 26). It was among the top in the squares test with low percentage of black pixels (Figures 28 and 29). However, for some cases it can do worst than all the other algorithms. This occurs for random images with high percentages of interest (black) pixels, as shown in Figures 25 and 28, and for the half-filled image as well, though in these cases Maurer was not far behind the others.

Implementing Maurer was relatively time-consuming, as it requires one to spend digest the paper before extracting a specific 2D EDT implementation out of the general pseudocode given there.

9.4.3 *Performance of Meijster.* Meijster was virtually the best performing method of all, together with Maurer. Technically, the only real difference between Meijster and Maurer is for the image with a single black pixel (Figure 21), where Maurer superseded Meijster. For all the other tests, the methods performed similarly, with a slight advantage to the current implementation of Meijster in terms of speed and

stability relative to image content. Moreover, this algorithm, together with Saito's, was the easiest to implement.

9.4.4 *Performance of Saito.* Saito's method performs well on average, never being the worst method in any test, and was the easiest to code (together with Meijster's). It was among the top three fastest algorithms in many cases, performing especially well on the random pixels test as shown in Figure 25. However, in its worst case – the slanted line at 60° – it gets 40 times slower than Maurer.

The line test (Figure 26) shows that Saito's time-complexity is really worst than $O(n^2)$. This is clear in large images, for which Saito's curve gets far from Maurer and Meijster. Nevertheless, for all other tests the method seemed to be $O(n^2)$, since the evolution of its time curve relatively to image size follows those of the other $O(n^2)$ algorithms.

The method is very dependent on the orientation of image content, if compared to Maurer, Meijster and Cuisenaire. The worst performance occurs for angles around 60° for both the straight line test (Figure 26) and the squares test (Figure 29). Saito is also very dependent on the number of feature pixels. In the random pixels test, the larger the image, the sharper the time curve gets relative to the percentage of white pixels (Figure 25), approximating Cuisenaire's curve. The performance was superior for many black pixels. A similar behavior is verified for the random squares test, shown in Figure 28.

9.4.5 *Performance of Lotufo-Zampirolli.* The method displayed an average performance compared to the others. It is reasonably stable to percentage of interest pixels, as shown in Figure 25. The exception is the random squares test around 45°, as shown in Figure 28(f).

Lotufo-Zampirolli's algorithm is clearly not linear for the line test under non-orthogonal directions. Moreover, it was the second most dependent method on the orientation of the line and squares tests, the worst performance occurring around 45°. In this case, the performance was worst also compared to the other methods, as shown in Figure 28(f), except above 85% black pixels. The observed tendency in similar plots for every tested size leads to the conclusion that, for images larger than 3000×3000 at 45°, Lotufo-Zampirolli was the worst method for practically all percentages of black pixels.

Implementing Lotufo-Zampirolli's algorithm was relatively harder than the remaining independent scanning algorithms, even though their pseudocode is quite clear.

9.4.6 *Performance of Eggers.* Eggers is the fastest algorithm for the half-filled image, and is second best for the similar image of random squares with 50% of feature pixels at 0°. For the image with a single black pixel, the algorithm is not far from the best ones. However, it is the worst method (along with PMN) for Lenna's contours, for the inscribed circle, and for some images of the other tests, as analyzed in the following.

Eggers was the most heavily dependent method on image content, for both the amount of interest pixels and orientation. In the test of random pixels, the algorithm displays peaks of time around 50% white pixels, but greatly improving for low percentages and for percentages around 90%. However, above 90% white pixels

the method displays another drop in efficiency, as shown in Figure 25. The same behavior was not confirmed in the squares test: for every fixed angle the curve was monotonous, the time being lower for low percentages of white pixels. The method was also the less stable relatively to orientation. For the straight line test, it displayed maximum speed around 0° and 45°, being very slow outside this interval, specially for angles between 60° and 80°. This fact was also confirmed in the squares test.

The method of Eggers is easier to implement in our opinion than Cuisenaire's PMN, but harder than the other methods.

10. CONCLUSION

In this work, the main Euclidean Distance Transform algorithms in the literature were reviewed, tested, coded, and compared. The empirical tests clearly reflect the large performance variability of EDT algorithms in terms of image content. No method has shown to be definitely faster in all cases.

The methods of Meijster [Meijster et al. 2000] and Maurer [Maurer et al. 2003] proved to be the best options overall among the considered methods. Perhaps Meijster's method should be preferred, being slightly faster in most cases and easier to implement. Both Meijster's and Maurer's algorithms displayed excellent performances in the majority of tests, and clearly were the least dependent methods on image content. They also work in any dimension, and readily enable label propagation. The devised open-source implementation will also contribute to the widespread adoption of these EDT algorithms. The novel conceptual description of Maurer's theory, provided in this paper, will hopefully contribute to its understanding.

Saito's algorithm can be considered a good alternative to Meijster's. Although it is $O(n^3)$, it has shown an excellent relative performance on all tests different than the worst case. Furthermore, it is easy to code (slightly easier than Meijster's in our opinion), and its extension to 3D is immediate.

The big surprise in this study was the performance of Cuisenaire's PMN, which was bellow the expected. However, an important advantage of Cuisenaire's algorithm is the immediate possibility of generating the EDT up to a maximum distance only.

The algorithms of Eggers and Lotufo-Zampirolli were inferior to Saito. Both are heavily dependent on image content, and all tests confirm their $O(n^3)$ complexity. In general, the independent-scanning algorithms proved to be faster and easier to implement than ordered propagation.

This survey is expected to be widely useful for the computational geometry and vision communities, since the EDT is the foundation for many other operators, techniques, and applications.

ACKNOWLEDGMENT

We would like to thank Angela Guzman for assisting us with Figure 2.

REFERENCES

Aho, A. V., Hopcroft, J. E., and Ullman, J. 1982. *Data Structures and Algorithms.* Addison Wesley.

Ahuja, R. K., Magnanti, T. L., and Orlin, J. B. 1993. *Network Flows: Theory, Algorithms, and Applications.* Prentice Hall.

Akmal Butt, M. and Maragos, P. 1998. Optimum design of chamfer distance transforms. *IEEE Transactions On Image Processing 7,* 10 (oct), 1477–1484.

Aurenhammer, F. 1991. Voronoi diagrams - a survey of a fundamental data structure. *ACM Computing Surveys 23,* 3, 345–405.

Bellman, R. 1958. On a routing problem. *Quaterly of Applied Mathematics 16*, 87–90.

Biere, A. The ccmalloc memory-leak tracing tool. `http://www.inf.ethz.ch/personal/biere/projects/ccmalloc`.

Blum, H. 1967. *A transformation for extracting new descriptors of shape.* Models for the Perception of Speech and Visual Forms. MIT Press, Cambridge, Mass., 362–380.

Borgefors, G. 1984. Distance transformations in arbitrary dimensions. *Computer Vision, Graphics, and Image Processing 27*, 321–345.

Borgefors, G. 1986. Distance transformations in digital images. *Computer Vision, Graphics, and Image Processing 34*, 344–371.

Borgefors, G. and Nyström, I. 1997. Efficient shape representation by minimizing the set of centres of maximal discs/spheres. *Pattern Recognition Letters*, 465–472.

Boxer, L. and Miller, R. 2000. Efficient computation of the Euclidean distance transform. *Computer Vision and Image Understanding Notes 80*, 379–383.

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

Brigham, E. O. 1988. *The Fast Fourier Transform and Its Applications.* Prentice Hall.

Ceniceros, H. D. and Roma, A. M. 2005. A multi-phase flow method with a fast, geometry-based fluid indicator. *Journal of Computational Physics 205,* 2 (May), 391–400.

Chen, L. and Chuang, H. Y. H. 1994. A fast algorithm for Euclidean distance maps of a 2D binary image. *Information Processing Letters 51*, 25–29.

Chen, M. and Yan, P. 1989. Multiscaling approach based on morphological filtering. *IEEE Patt. Anal. Mach. Intel. 11,* 7, 694–700.

Chin, Y., Wang, H., Tay, L. P., , and Soh, Y. C. 2001. Vision guided AGV using distance transform. In *Proceedings of 32nd International Symposium on Robotics.* 1416–1421.

Coelho, R. and Costa, L. 1996. On the application of the Bouligand-Minkowski fractal dimension for shape characterisation. *Applied Signal Processing 3*, 163–176.

Coerjolli and Montanvert. 2007. Optimal separable algorithm to compute the reverse Euclidean distance transformation and discrete medial axis in arbitrary dimension. *IEEE Transactions on Pattern Analysis and Machine Intelligence 29,* 3 (march), 437–448.

Couprie, M., Saúde, A., and Bertrand, G. 2006. Euclidean homotopic skeleton based on critical kernels. In *Proccedings of SIBGRAPI, 19th Brazilian Symposium on Computer Graphics and Image Processing.* IEEE Computer Society, 307–314.

Cuisenaire, O. 1999. Distance transformations: Fast algorithms and applications to medical image processing. Ph.D. thesis, Université Catholique de Louvain, Belgique.

Cuisenaire, O. 2006. Locally adaptable mathematical morphology using distance transformations. *Pattern Recognition 39,* 3, 405–416.

Cuisenaire, O. and Macq, B. 1999a. Fast and exact signed Euclidean distance transformation with linear complexity. In *ICASSP'99 - IEEE Intl Conference on Acoustics, Speech and Signal Processing.* Vol. 6. Phoenix, USA, 3293–3296.

Cuisenaire, O. and Macq, B. 1999b. Fast Euclidean distance transformation by propagation using multiple neighborhoods. *Computer Vision and Image Understanding 76,* 2, 163–172.

Danielsson, P.-E. 1980. Euclidean distance mapping. *Computer Graphics and Image Processing 14*, 227–248.

Dial, R. B. 1969. Shortest path forest with topological ordering. *Communications of the ACM 12*, 632–633.

Dijkstra, E. W. 1959. A note on two problems in connection with graphs. *Numerische Math. 1*, 269–271.

Eggers, H. 1998. Two fast Euclidean distance transformations in $\mathbb{Z}^2$ based on sufficient propagation. *Computer Vision and Image Understanding 69,* 1 (jan), 106–116.

Fabbri, R. Animal – An Imaging Library. `http://animal.sourceforge.net`.

Fabbri, R. SIP – the Scilab Image Processing toolbox. `http://siptoolbox.sourceforge.net`.

Fabbri, R. 2004. Design and evaluation of Euclidean distance transform algorithms and applications. M.S. thesis, USP, Sao Carlos, Brasil.

Fabbri, R., Bruno, O. M., Torelli, J. C., and da F. Costa, L. 2006. Complete results of the benchmark between exact EDT algorithms. `http://distance.sourceforge.net`.

Fabbri, R., Estrozi, L., and Costa, L. 2002. On Voronoi diagrams and medial axes. *Journal of Mathematical Imaging and Vision 17,* 1 (jul), 27–40.

Falcão, A., Stolfi, J., and Lotufo, R. A. 2004. The image foresting transform: theory, algorithms, and applications. *IEEE Transactions on Pattern Analysis and Machine Intelligence 26,* 1 (jan), 19–29.

Felzenszwalb, P. F. and Huttenlocher, D. P. 2004. Distance tansforms of sampled functions. Tech. rep., Cornell Computing and Information Science.

Felzenszwalb, P. F. and Huttenlocher, D. P. 2006. Efficient belief propagation for early vision. *International Journal of Computer Vision 70,* 1, 41–54.

Fortune, S. J. 1987. A sweepline algorithm for Voronoi diagrams. *Algorithmica* 2, 153–174.

Gavrilova, M. L. and Alsuwaiyel, M. H. 2001. Two algorithms for computing the Euclidean distance transform. *International Journal of Image and Graphics 1,* 4, 635–645.

Ge, Y. and Fitzpatrick, J. 1996. On the generation of skeletons from discrete Euclidean distance maps. *IEEE Transactions on Pattern Analysis and Machine Intelligence 18,* 11, 1055–1066.

Guan, W. and Ma, S. 1998. A list-processing approach to compute Voronoi diagrams and the Euclidean distance transform. *Pattern Analysis and Machine Intelligence 20,* 7 (jul), 757–761.

hao Tseng, Y., neng Hwang, J., and Sheehan, F. H. 1998. 3D heart modeling and motion estimation based on continuous distance transform neural networks and affine transform. *The Journal of VLSI Signal Processing–systems For Signal, Image, and Video Technology 18,* 3 (apr).

Hesselink, W. H., Visser, M., and Roerdink, J. B. 2005. Euclidean skeletons of 3D data sets in linear time by the integer medial axis transform. Computational Imaging and Vision, vol. 30. Springer-Verlag, Dordrecht, 259–268.

Hirata, T. 1996. A unified linear-time algorithm for computing distance maps. *Inf. Process. Lett. 58,* 3, 129–133.

Huang, C. and Mitchell, O. 1994. A Euclidean distance transform using grayscale morphology decomposition. *Pattern Analysis and Machine Intelligence, IEEE Transactions On 16,* 4 (apr), 443–448.

INRIA. Scilab numerical programming environment. `www.scilab.org`.

Jain, R. and Binford, T. 1991. Dialogue: Ignorance, myopia, and naivete in computer vision systems. *CVGIP 53,* 1, 112–117.

John, F. 1982. *Partial Differential Equations*, 4th ed. Universitext. Springer-Verlag.

Jolesz, F., Loresen, W., Shinmoto, H., Atsumi, H., Nakajima, S., Kavanaugh, P., Saiviroonporn, P., Seltzer, S., Silverman, S., Phillips, M., and Kikinis, R. 1997. Interactive virtual endoscopy. *American Journal of Radiology 169*, 1229–1237.

Jones, M. W., Baerentzen, J. A., and Sramek, M. 2006. 3D distance fields: A survey of techniques and applications. *IEEE Transactions on Visualization and Computer Graphics 12,* 4, 581–599.

Kimia, B. B. 2003. On the role of medial geometry in human vision. *Journal of Physiology*. Accepted for publication.

Knuth, D. E. 1976. Big Omicron and big Omega and big Theta. *Sigact News 8,* 2, 18–24.

Kolountzakis, M. N. and Kutulakos, K. N. 1992. Fast computation of the Euclidean distance maps for binary images. *Inf. Process. Lett. 43,* 4, 181–184.

Kovács, I. and Julesz, B. 1994. Perceptual sensitivity maps within globally defined visual shapes. *Nature 370,* 6491, 644–646.

Kozinska, D., Tretiak, O. J., Nissanov, J., and Ozturk, C. 1997. Multidimensional alignment using the Euclidean distance transform. *Graphical Models and Image Processing 59,* 6 (nov), 373–496.

Leymarie, F. and Levine, M. D. 1992. Fast raster scan distance propagation on the discrete rectangular lattice. *Computer Vision and Image Understanding 55,* 1 (jan).

Leyton, M. 1992. *Symmetry, Causality, Mind.* MIT Press.

Liu, H. and Srinath, M. 1990. Partial shape classification using contour matching in distance transformation. *IEEE Transactions on Pattern Analysis and Machine Intelligence 12,* 11, 1072–1079.

Lotufo, R. and Zampirolli, F. 2001. Fast multi-dimensional parallel Euclidean distance transform based on mathematical morphology. In *Proccedings of SIBGRAPI, XIV Brazilian Symposium on Computer Graphics and Image Processing.* IEEE Computer Society, 100–105.

Lotufo, R. A. and Zampirolli, F. A. 2003. Multidimensional parallel EDT using 1D erosions by propagation. Submitted to IEEE Trans. Image Processing.

Lucet, Y. 2007. New sequential exact Euclidean distance transform algorithms based on convex analysis. *Image and Vision Computing.* Accepted.

Maurer, C., Qi, R., and Raghavan, V. 2003. A linear time algorithm for computing the Euclidean distance transform in arbitrary dimensions. *IEEE Transactions on Pattern Analysis and Machine Intelligence 25,* 2 (feb), 265–270.

Meijster, A., Roerdink, J., and Hesselink, W. 2000. A general algorithm for computing distance transforms in linear time. In *Mathematical Morphology and its Applications to Image and Signal Processing (Proc. 5th Int. Conf).*

Montanari, U. 1968. A method for obtaining skeletons using a quasi-Euclidean distance. *Journal of the Association For Computing Machinery 15,* 4, 600–624.

Moore, E. F. 1959. The shortest path through a maze. In *Proc. Internat. Symp. Switching Th., Part II.* Harvard University Press, Cambridge, MA, 285–292.

Mullikin, J. 1992. The vector distance transform in two and three dimensions. *Graphical Models and Image Processing 54,* 6, 526–535.

Ogniewicz, R. L. and Kübler, O. 1995. Voronoi tessellation of points with integer coordinates: Time-efficient implementation and online edge-list generation. *Pattern Recognition, Pergamon Press 28.*

Osher, S. and Sethian, J. A. 1988. Fronts propagating with curvature-dependent speed. *Journal of Computational Physics 79,* 1, 12–49.

Paglieroni, D. W. 1992a. Distance transforms: Properties and machine vision applications. *Graphical Models and Image Processing 54,* 1, 56–74.

Paglieroni, D. W. 1992b. A unified distance transform algorithm and architecture. *Mach. Vision Appl. 5,* 1, 47–55.

Paglieroni, D. W. 1997. Directional distance transforms and height field preprocessing for efficient ray tracing. *Graphical Models and Image Processing 59,* 4 (jul), 253–264.

Parker, R. 1997. *Algorithms for image processing and computer vision.* Wiley.

Piper, J. and Granum, E. 1987. Computing distance transforms in convex and non-convex domains. *Pattern Recognition 20,* 6, 599–615.

Preparata, F. and Shamos, M. 1990. *Computational Geometry.* Springer-Verlag.

Ragnemalm, I. 1992. Neighborhoods for distance transformations using ordered propagation. *CVGIP - Image Understanding 56,* 3, 399–409.

Ragnemalm, I. 1993. The Euclidean distance transformation in arbitrary dimensions. *Pattern Regocgnition Letters 14*, 883–888.

Rosenfeld, A. and Kak, A. C. 1976. *Digital Picture Processing.* Academic Press.

Rosenfeld, A. and Pfaltz, J. 1966. Sequential operations in digital picture processing. *Journal of the ACM 13,* 4.

Rosenfeld, A. and Pfaltz, J. 1968. Distance functions on digital pictures. *Pattern Recognition 1,* 1, 33–61.

Saito, T. and Toriwaki, J. 1994. New algorithms for Euclidean distance transformations of an n-dimensional digitised picture with applications. *Pattern Recognition 27,* 11, 1551–1565.

Saúde, A. V., Couprie, M., and Lotufo, R. A. 2006. Exact Euclidean medial axis in higher resolution. In *Discrete Geometry for Computer Imagery*. Lecture Notes in Computer Science. Springer, 605–616.

Sebastian, T., Crisco, J., and Kimia, B. 2003. Segmentation of carpal bones from CT images using skeletally coupled deformable models. *Medical Image Analysis 7,* 1 (March), 21–45.

Serra, J. 1982. *Image Analysis and Mathematical Morphology*. Vol. 1. Academic Press.

Sethian, J. 1999. *Level Set Methods and Fast Marching Methods: Evolving Interfaces in Computational Geometry*. Cambridge University Press.

Seward, J. Valgrind: a GPL'd system for debugging and profiling x86-Linux programs. `http://valgrind.kde.org`.

Sharaiha, Y. and Christofides, N. 1994. A graph-theoretic approach to distance transformations. *Pattern Recognition Letters 15*, 1035–1041.

Shih, F. Y. and Liu, J. 1998. Size-invariant four-scan Euclidean distance transformation. *Pattern Recognition 31,* 11, 1761–1766.

Shih, F. Y. and Wu, Y.-T. 2004a. The efficient algorithms for achieving Euclidean distance transformation. *IEEE Transactions On Image Processing 13,* 3, 1078–1091.

Shih, F. Y. and Wu, Y.-T. 2004b. Fast Euclidean distance transformation in two scans using a 3×3 neighborhood. *Computer Vision and Image Understanding 93,* 2 (feb), 195 – 205.

Shih, F. Y. and Wu, Y.-T. 2004c. Three-dimensional Euclidean distance transformation and its application to shortest path planning. *Pattern Recognition 37,* 1, 79 – 92.

Shih, F. Y. C. and Mitchell, O. R. 1992. A mathematical morphology approach to Euclidean distance transformation. *IEEE Transactions on Image Processing 1,* 2, 197–204.

Siddiqi, K., Kimia, B. B., and Shu, C. 1997. Geometric shock-capturing ENO schemes for subpixel interpolation, computation and curve evolution. *Graphical Models and Image Processing 59,* 5 (September), 278–301.

Soille, P. and Vincent, L. 1991. Watersheds in digital spaces: An efficient algorithm based on imersion simmulations. *IEEE T. Patt. Anal. Mach. Intel. 13,* 6, 583–598.

Sramek, M. and Kaufman, A. 2000. Fast ray-tracing of rectilinear volume data using distance transforms. *IEEE Transactions on Visualization and Computer Graphics 06,* 3, 236–252.

Starovoitov, V. 1996. A clustering technique based on the distance transform. *Pattern Recognition Letters 17,* 3, 231–239.

Tek, H. and Kimia, B. B. 2003. Symmetry maps of free-form curve segments via wave propagation. *International Journal of Computer Vision 54,* 3 (August), 35–81.

Thiran, J. and Macq, B. 1996. Morphological feature extraction for the classification of digital images of cancerous tissues. *IEEE Transactions on Biomedical Engineering 43*, 1011–1019.

Travençolo, B. A. N., Martínez Debat, C. ; Beletti, M. E., Sotelo Silveira, J., EHRLICH, R., and COSTA, L. d. F. 2007. A new method for quantifying three-dimensional interactions between biological structures. *Journal of Anatomy 210*, 221–231.

Travis, A. J., Hirst, D. J., and Chesson, A. 1996. Automatic classification of plant cells according to tissue type using anatomical features obtained by the distance transform. *Annals of Botany 78*, 325–331.

Verwer, B., Verbeek, P., and Dekker, S. 1989. An efficient uniform cost algorithm applied to distance transforms. *IEEE Transactions on Patt. Anal. and Mach. Intel. 11,* 4 (apr), 425–429.

Vincent, L. 1991. Exact Euclidean distance function by chain propagations. In *Proceedings of the IEEE Computer Vision and Pattern Recognition*. 520–525.

Wintermark, M., Reichhart, M., Cuisenaire, O., Maeder, P., Thiran, J., Schnyder, P., Bogousslavsky, J., and Meuli, R. 2002. Comparison of admission perfusion computed tomography and qualitative diffusion- and perfusion-weighted magnetic resonance imaging in acute stroke patients. *Stroke 8,* 33 (aug), 2025–2031.

Xia, R., Liu, W., Wang, Y., and Wu, X. 2004. Fast initialization of level set method and an improvement to Chan-Vese model. In *International Conference on Computer and Information Technology*. 18–23.

Yamada, H. 1984. Complete Euclidean distance transformation by parallel operation. In *Proceedings of the 7th International Conference on Pattern Recognition*. Vol. 1. 69–71.

You, J., Pissaloux, E., Zhu, W. P., and Cohen, H. A. 1995. Efficient image matching: A hierarchical chamfer matching scheme via distributed system. *Real-time Imaging 1,* 4 (oct), 245–259.

Zampirolli, F. A. 2003. Transformada de distância por morfologia matemática. Ph.D. thesis, UNICAMP, Campinas, Brasil.

Zampirolli, F. A. and Lotufo, R. A. 2000. Classification of the distance transformation algorithms under the mathematical morphology approach. In *Proccedings of SIBGRAPI, XIII Brazilian Symposium on Computer Graphics and Image Processing*. Gramado, RS, Brasil.

Zeng, P. and Hirata, T. 2003. Distance map based enhancement for interpolated images. In *Geometry, Morphology and Computational Imaging*. Lecture Notes in Computer Science, vol. 2616. Springer-Verlag, 86–100.