A Note on "Fast Raster Scan Distance Propagation on the Discrete Rectangular Lattice"

F. Leymarie and M. D. Levine[*]

January 1992[†]

Abstract

The main result of this paper is that simple (raster scan) sequential algorithms for computing *Euclidean Distance Transforms* can be implemented in an *optimized* manner from the point of view of numerical computations. We will show that these fast implementations have computational complexities comparable to the *city block*, *chessboard* and other simple *chamfer* Distance Transforms.

Keywords: Euclidean and pseudo-Euclidean Distance Transforms, discrete rectangular lattice, numerical complexity and accuracy.

[*]This research was partially supported by the Natural Sciences and Engineering Research Council of Canada in the form of a postgraduate scholarship. It was also partially supported by the Medical Research Council of Canada under Grant no. MT-3236.

[†]Appeared in "CVGIP-IU, Vol.55(1), pp. 84-94, Jan. 1992. Academic Press." The authors were with the "McGill Research Center for Intelligent Machines, McGill University, Montréal, Québec, Canada." Reproduced in electronic format on September 24, 2001, while at Brown University, by F. Leymarie.

Contents

1 Introduction

A distance transformation on a binary image I (object O, non-object O') produces a mapping from a double-valued function f_1 in a space S to a multi-valued function f_2 in the same space S. The binary function f_1 is defined as follows:

$$f_1(p) = \begin{cases} M \Leftrightarrow p \in O\,, \\ 0\,, \text{ elsewhere}\,, \end{cases} \tag{1}$$

where M is some real constant ($M \in I\!R$). The values of the function f_2 replace those of f_1 by minimizing a *distance metric*, $dist(p, p')$ over the domain of interest, the object O, where p and p' are two points in the space S such that $p \in O$ and $p' \in O'$. The three following necessary and sufficient conditions for a distance metric [21] are, with p_1, p_2 and $p_3 \in S$:

1. $dist(p_1, p_2) \geq 0$: $dist(p_1, p_2) = 0 \Leftrightarrow p_1 = p_2$ (identity);
2. $dist(p_1, p_2) = dist(p_2, p_1)$ (symmetry);
3. $dist(p_1, p_3) \leq dist(p_1, p_2) + dist(p_2, p_3)$ (triangle inequality).

The *distance values* of $f_2(p)$, for $p \in O$, are given by:

$$f_2(p) = \min_{p' \in O'} \left[dist(p, p') \right] \,. \tag{2}$$

We consider the case of a two-dimensional (2-D) discrete space only, but our results should essentially hold for higher dimensions. We use the traditional Cartesian coordinate system to index image elements (*e.g.*, $p = p(\overline{x}, \overline{y})$). We restrict our analysis to *rectangular* and *isotropic* image grids, but our discussions and results should also hold for other grids (*e.g.*, the *hexagonal* grid). We assume that such a grid or lattice is isotropic in both the horizontal and vertical directions; that is, pixels are equally spaced and of equal length in these two directions (*i.e.*, the usual case of *square shaped* pixels). We specify the image grid (coordinates) by the use of two sets, $\overline{X}$ and $\overline{Y}$, defining indices on I. The sets $\overline{X}$ and $\overline{Y}$ consist of integer values varying from 0 to some maximum indexing values, $M_{\overline{X}}$ and $M_{\overline{Y}}$, respectively: $\{\overline{x} \in \overline{X} \; : \; \overline{X} = \{0, \ldots, M_{\overline{X}}\}\}$ and $\{\overline{y} \in \overline{Y} \; : \; \overline{Y} = \{0, \ldots, M_{\overline{Y}}\}\}$. We make use of barred symbols to emphasize the fact that they represent *discrete* variables.

The main result of this short paper will be to demonstrate that simple sequential algorithms for computing *Euclidean Distance Transforms* (*EDT*'s) can be implemented in an *optimized* manner from the point of view of numerical computations. We will show that these fast implementations have computational complexities comparable to the *city block*, *chessboard* and other simple *chamfer* Distance Transforms [12]. The particular *EDT* algorithms we will consider are based on the work of Danielsson [8].

1.1 The Propagation of Distances in the Discrete Domain

Let us start by identifying the 2-D binary image I with the function f_1, *i.e.*, $I \equiv f_1$. We use the following notation. A distance mapping on I is produced by a distance transform, *DT*. The latter consists of a minimum operation on a distance metric $dist(p, p')$, where $p \in O$ and $p' \in O'$, applied over O (Eq. (2)). Commonly used metrics in discrete image processing are:[1]

$$\begin{aligned} dist_{CB}(p,p') &= \left|\overline{x}_p - \overline{x}_{p'}\right| + \left|\overline{y}_p - \overline{y}_{p'}\right| \;, \\ dist_{Chess}(p,p') &= \max\left(\left|\overline{x}_p - \overline{x}_{p'}\right|, \left|\overline{y}_p - \overline{y}_{p'}\right|\right) \;, \\ dist_{Euclid}(p,p') &= \sqrt{\left(\overline{x}_p - \overline{x}_{p'}\right)^2 + \left(\overline{y}_p - \overline{y}_{p'}\right)^2} \;, \end{aligned}$$

where $p = p(\overline{x}_p, \overline{y}_p)$ and $p' = p'(\overline{x}_{p'}, \overline{y}_{p'})$. The *DT*'s based on the metrics $dist_{CB}$, $dist_{Chess}$ and $dist_{Euclid}$ are called the *city block*, *chessboard* and *Euclidean DT*'s, respectively.

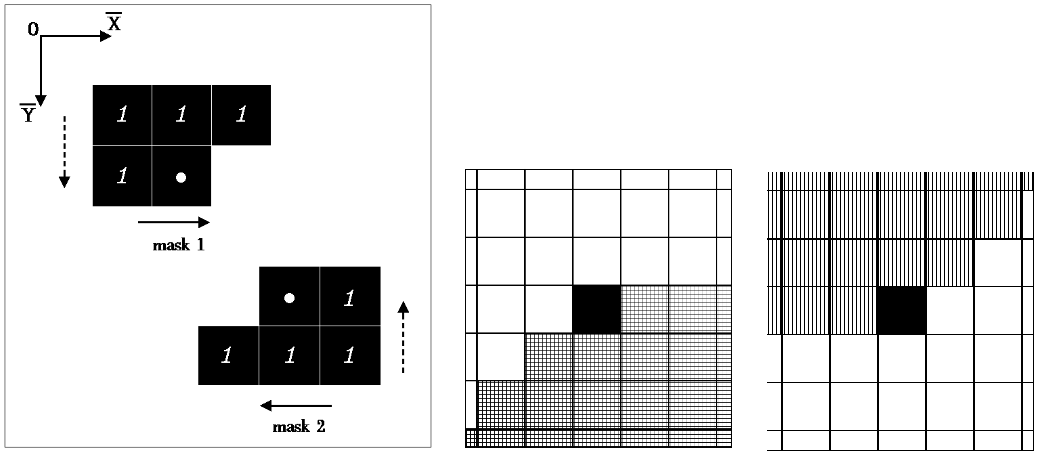
Applying a *DT* to I corresponds to "propagating" distances over I [20]. Points in the background ($p' \in O'$) are seen as sources from which distance values are propagated as waves over the complete image, I; for example, the propagation of circular waves in the case of an Euclidean metric. The first time an object point ($p \in O$) is reached by a given wave it is assigned

[1]For extensive surveys of metrics used in the discrete domain see [4, 5].

a new label, that is, a minimum distance value from O'. Obviously such a wave propagation scheme can be naturally implemented in parallel (*e.g.*, [9, 23]). Propagation of distances is then an iterative procedure, where iterations correspond to distances from the point sources, dependent on the maximal width of the object O. Typically, for such parallel algorithms, the time complexity is of order $O(N)$ for an image of size $N \times N$ [9], but these require an expensive architecture with one processor per pixel, that is, a total of N^2 processors. Therefore, the *time-processor complexity* or numerical complexity is of order $O(N^3)$ [9].

However, for such a simple propagation transform, a sequential implementation provides equivalent results [20]. Now the time complexity is of order $O(N^2)$ and only one processor is required, leading to a time-processor complexity of order $O(N^2)$ [9]. Several sequential algorithms exist for computing a distance map on a discrete grid. Essentially two categories of algorithms are available depending on the type of metric used: Euclidean or non-Euclidean. *DT*'s based on the Euclidean metric use *multi-valued* vector elements to propagate distances; we shall refer to them by the symbol *SEDT* for *Sequential Euclidean DT*. *DT*'s based on non-Euclidean metrics use only *single-valued* vector elements (*i.e.*, scalars) to propagate distances; we shall refer to them by the symbol *SWDT* for *Sequential Weighted DT*(adapting Borgefors' notation [6]). Common to both kinds of distance mapping is the fact that they are based on information flow or propagation using small local neighborhoods (*e.g.*, a 3×3 pixel window).

Each of the two categories of algorithms for sequential distance propagation can be further classified into two classes depending on *how* distances are propagated over I: in a *raster scan* or a *contour scan* fashion. Most algorithms found in the literature employ the propagation of distances using masks of fixed size and shape irrespective of the form of the domain of interest of I (*i.e.*, the shape of O). This class of algorithms follows the approach originally proposed by Rosenfeld and Pfaltz [20, 21] where a mask, say a 3×3 window, is broken into two or more submasks which are recursively convolved with I in two or more passes in fixed directions [17]. For example, the chessboard *DT* is usually processed using two symmetrical masks which are recursively convolved with I in a *raster scan* fashion in opposite directions (Fig. 1). Typically *SWDT*'s require a minimum of two passes with two different masks over an image to obtain the distance



(a) (b) (c)

Figure 1: Example of a *raster scan* implementation of a *DT*. In *(a)* are shown the two masks used for the *chessboard DT*. The masks are centered at pixel p (indicated by a white dot). The weights in each mask represent the distance from p. The reference frame (directions indexed by $\overline{X}$ and $\overline{Y}$) is shown to be positioned in the top-left corner of an image, I. The arrows (next to the masks) indicate in which direction a given mask is passed over I. The convolution of these masks over I is performed in two picture scans (indicated by dashed arrows) in opposite directions (here downward for masks labelled 1 and upward for masks labelled 2). In *(b)* and *(c)* are shown the *propagation envelopes* for this *SWDT*. These windows (over I) represent the set of (closeby) pixels whose distance could be updated (shown as shaded pixels) from the central black pixel (adapted from [17, Fig.5, p.604]). In this case, the first scan covers a 135^o interval of propagation angles over I as indicated in *(b)*. The second scan then also covers a 135^o interval of propagation angles, but in the opposite direction as indicated in *(c)*.

map. On the other hand, the *SEDT*category usually requires four passes with four different masks [8].

In the second class of algorithms, distance values are propagated in a *contour scan* fashion, where the shape of the domain to be processed is determined by iteratively *peeling off* contours of O. The shape and connectivity of these contours is governed by the metric $dist$ employed, and by the particular *queuing scheme* adopted for the processing of the pixels on these contours. These contour scan algorithms lead to more complex algorithmic procedures. A complete analysis of this approach is outside the scope of the present communication.

1.1.1 Raster Scan Algorithms

Let us consider more precisely the *raster scan* class of distance propagation. In this case, an object pixel updates its distance value(s) by comparing itself to some of its neighbors which have already been processed. The neighborhood is defined by the mask being used. Propagation occurs "from the [previously processed] neighbors *into* the [presently] processed pixel" [19].

We can rewrite Eq. (2) for the iterative and recursive raster scan process, as follows:

$$\begin{aligned} f_2(p) &= f_1(p)\,,\; j=0\,, p \in O \text{ or } p \in O'\,, && \text{(3a)}\\ f_2\left(p_{(j)}\right) &= \min_{p_N \in N_p(j)} \left[f_2\left(p_{(j-1)}\right)\,,\; f_2\left(p_{N(j)}\right) + w_N \right]\,, j > 0\,,\, p \in O\,, && \text{(3b)} \end{aligned}$$

where j stands for *iteration* or pass, $N_{p(j)}$ is the *neighborhood* of p defined by the mask used in the present pass, p_N is a pixel in that neighborhood, and w_N stands for the vector elements value(s) given by the mask. w_N represents the "distance" between p and p_N, that is, w_N's value(s) is (are) defined by the chosen metric $dist$. Note also that we first initialize f_2 to f_1 (Eq. (3a)) for both object and non-object pixels. Object pixels are thereby assigned the value M (Eq. (1)). We require that M be a *large* integer value greater than the maximum possible width of any binary object that could exist in I. This is necessary to be able to correctly use the minimization operation in Eq. (3b) to propagate distances. Furthermore, in the case of *SEDT*'s, it will be convenient to use a value of M greater than the *square* of the maximum possible object width (see § 2).[2]

From Eq. (3b) we see how distance values at a given object pixel are updated by *reading* the distance values of previously visited neighbors. Also, since iteration corresponds to a (raster scan) pass, the maximal number of iterations is *fixed* independently of the object O, and depends only on the number of masks used. As a consequence, object pixels are processed as many times as there are passes.

1.2 Accuracy and Speed of Propagation

In terms of isotropy of propagation of distances, or equivalently, invariance under rigid transformations of the associated *DT*, such as rotations, the *optimal* distance metric is of course Euclidean, that is, $dist_{Euclid}$. The other metrics provide coarse approximations of circular propagation; that is, they approximate an Euclidean propagation. For example, the city block metric gives diamond-shape propagation, while the chessboard metric gives square-shape propagation [21]. However, *DT*'s based on non-Euclidean metrics, that is, *SWDT*'s, have a long history in picture processing. In most cases they were used because they were believed to be much faster to apply than *SEDT*'s. This should have been recognized to be true only until Danielsson's *SEDT* algorithms were published in 1980 [8]. In fact, only the city block *DT* can be evaluated significantly faster than Danielsson's *SEDT* [12].[3] This is because of its simpler spatial complexity, though the price paid is poor accuracy (see § 4).

In the following sections we will consider different aspects of the implementation and application of raster scan distance mapping to binary images. First, in § 2 we will give a detailed analysis of a *numerically optimized* implementation of Danielsson's *SEDT*algorithm. In § 3 we will consider the *SWDT*'s and discuss their principal characteristics. In § 4 we will

[2]In practice we have used the following simple equation to fix the value of M : $M = (M_{\overline{x}} + 1)^2 + (M_{\overline{y}} + 1)^2$.

[3]This must be contrasted with the generally incorrect belief appearing in the literature (even recently) that most *SWDT*'s are computationally less expensive than *SEDT*'s (see for example [4, 5, 10, 11, 1, 2]).

then compare these optimized implementations to *SWDT*'s. There, we will show that *SEDT*'s are in fact numerically simpler than most *SWDT*'s.

2 Fast Signed Sequential Euclidean Distance Transforms

In this section we present optimized versions, in terms of numerical complexity, of the sequential algorithm for computing the *EDT*. We discuss the raster scan algorithm first proposed by Danielsson [8] and later refined by Ye [24].

Following Danielsson's notation [8], we consider a mapping from a binary image $I(\overline{x},\overline{y})$ (a double-valued function, f_1; see Eq. (1)) to a multivalued image $L(\overline{x},\overline{y})$ ($\equiv f_2$). In order to evaluate L, Danielsson proposes to compute a *multivalued vector image* $\mathbf{L}$, in which each pixel of the image is assigned a vector (a doublet) rather than a singleton, as it is the case with *SWDT*'s, as follows:

$$\mathbf{L}(\overline{x},\overline{y}) = (L_{\overline{x}}, L_{\overline{y}})\,, \tag{4}$$

where $L_{\overline{x}}$ and $L_{\overline{y}}$ will contain the minimum integer distance values from the background O' in the $\overline{x}$ and $\overline{y}$ directions, respectively. Note that these distance values can be *signed*, giving positive or negative directions with respect to the origin of the Cartesian coordinate system defined by $\overline{X}$ and $\overline{Y}$. L is then simply obtained as the Euclidean norm (or 2-norm) $\|\mathbf{L}\|_2$ as follows:

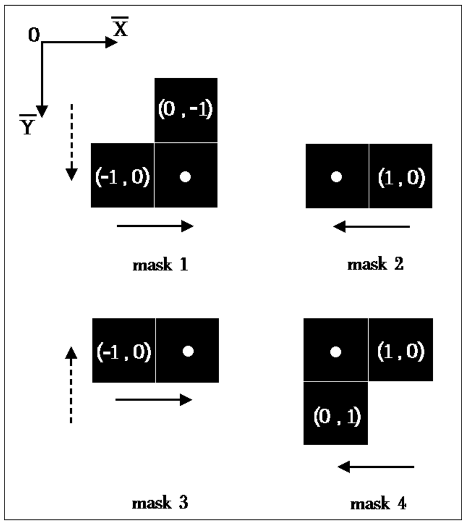
$$L(\overline{x},\overline{y}) = \sqrt{L_{\overline{x}}^2 + L_{\overline{y}}^2} = \|\mathbf{L}\|_2\ . \tag{5}$$

The propagation of distances over the complete image is obtained with a four-pass algorithm (see [18] for a recent three-pass version). This is best visualized as a recursive convolution of I with four masks (Fig. 2). This recursive convolution is usually performed in two complete picture scans in opposite directions. Masks 1 and 2 are passed downwards over the image, while masks 3 and 4 are passed upwards (Fig. 2). Note how each row is scanned in both horizontal directions to ensure an isotropic propagation of the distance values [18]. Depending on the required accuracy and numerical complexity, two versions of the *SEDT*algorithm were described by Danielsson (see also Ye [24]). The simpler one, the 4SSEDT, which stands for the "four-points signed *SEDT*", requires a visit to only the four direct neighbors (*i.e.*, horizontal and vertical neighbors) at each object pixel $p \in O$ (Fig.2.(a)); the 8SSEDT, which stands for the "eight-points signed *SEDT*", requires a visit to the eight closest neighbors around each object pixel $p \in O$ (*i.e.*, direct plus diagonal neighbors; see Fig. 2.(b)). Obviously, the 8SSEDT is numerically more complex than the 4SSEDT, but it is also more accurate.

As we saw in § 1.1, when scanning the image with masks, every object pixel, $p \in O$, is updated by *comparing* it to some of its neighboring pixels. For the 4SSEDTand 8SSEDT, a neighborhood is defined by the masks of Fig. 2. Comparing a pixel p to its neighbors implies first seeking the neighbor (denoted by p_{min}) already visited at least once and which minimizes the distance, as indicated by Eq. (3b). If such a neighbor p_{min} exists, that is, one of the p_N (of Eq. (3b)) minimizes the distance, then a second step will involve assigning the pair $(L_{\overline{x}}(p_{min}), L_{\overline{y}}(p_{min}))$ to $\mathbf{L}(p)$ plus the weights (w_N's) indicating the relative position of p_{min} with respect to p. We note here that the addition of these local distances given by the w_N's reduces to a simple increment or decrement in the $\overline{X}$ and $\overline{Y}$ directions for the *SEDT*masks (Fig. 2). Thus, the complete updating procedure generates the vector image $\mathbf{L}$. At first, it would seem that seeking the minimum distance requires numerically expensive computations for every visited pixel $p \in O$.[4] Furthermore, floating point numbers, rather than integers, are required. For example, if we wish to compare p with a neighbor p_+ in the positive $\overline{x}$ direction, then we must compare $L(p)$ with $L_{new}(p_+)$, where $L_{new}(p_+)$ is $L(p_+)$ incremented as follows:

$$L_{new}(p_+) = \sqrt{(L_{\overline{x}}(p_+) + 1)^2 + L_{\overline{y}}^2(p_+)}\,.$$

[4]That is, two integer adds, two integer squares (multiplications), one floating square root and, possibly, two assignments and one or two increments/decrements.



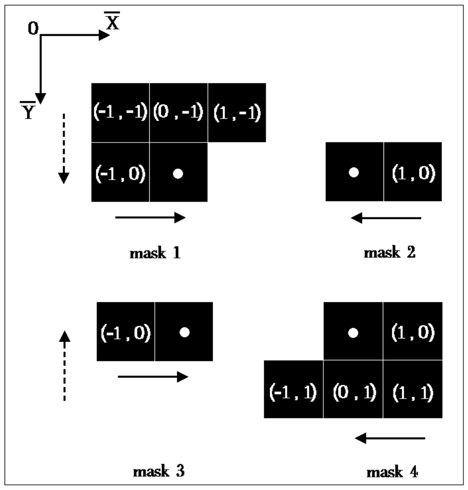


Figure 2: The four masks used in the raster-scan signed *SEDT*. Two sets of masks are shown. In (a) are shown the four masks for the 4SSEDT. In (b) are shown the four masks for the 8SSEDT. The masks are centered at pixel p (indicated by a white dot). The pairs of weights in each mask ($w_N = (\overline{x}, \overline{y})$ pairs) represent the distances from p in horizontal and vertical steps. The reference frame (directions indexed by $\overline{X}$ and $\overline{Y}$) is shown to be positioned in the top-left corner of an image, I. The arrows (next to the masks) indicate in which direction a given mask is passed over I. The recursive convolution of these masks over I is performed in two picture scans (indicated by dashed arrows) in opposite directions (here downward for masks 1 and 2, and upward for masks 3 and 4). For each row of I the pairs of masks (1,2) and (3,4) are moved in opposite horizontal directions so that the propagation of distance values is isotropic (see text).

However, there is a much better way to perform these computations using integers only, getting rid of the multiplications and bypassing the square root operations completely.[5] Since we only wish to compare the amplitudes of $L(p)$ and its neighbors (*e.g.*, $L_{new}(p_+)$), it is equivalent and sufficient to compare the squares of their amplitudes (*e.g.*, $L(p)^2$ and $L_{new}(p_+)^2$), thereby eliminating the floating point operation. Furthermore, we can *augment* the vector image $\mathbf{L}$ by also assigning the sum $L_{\overline{x}}^2 + L_{\overline{y}}^2$ $(= L^2)$ to each pixel p [7, 24], thereby obtaining a new vector image representation, $\mathbf{L}_+$, as follows:

$$\mathbf{L}_+(\overline{x},\overline{y}) = (L_{\overline{x}}\,,\, L_{\overline{y}}\,,\, L_{\overline{x}}^2 + L_{\overline{y}}^2) = (L_{\overline{x}}\,,\, L_{\overline{y}}\,,\, L^2)\,. \tag{6}$$

Then, $L_{new}(p_+)$ can be easily evaluated from the stored data as follows:

$$\begin{aligned} L_{new}(p_+)^2 &= (L_{\overline{x}}(p_+) + 1)^2 + L_{\overline{y}}^2(p_+) \\ &= (L_{\overline{x}}^2(p_+) + L_{\overline{y}}^2(p_+)) + 2L_{\overline{x}}(p_+) + 1 \\ &= L^2(p_+) + 2L_{\overline{x}}(p_+) + 1\,. \end{aligned}$$

Indeed, since now L^2 is stored in $\mathbf{L}_+$, we can in this example (and similarly in all other cases; see below) evaluate $L_{new}(p_+)^2$ using only one add, one left-shift (*i.e.*, a multiplication by two) and one increment.[6] We summarize all possibilities when evaluating $\mathbf{L}_+$ for both the 4SSEDT and 8SSEDT in the following four equations:

$$\begin{aligned} (L_{\overline{x}} \pm 1)^2 + L_{\overline{y}}^2 &= L^2 \pm 2L_{\overline{x}} + 1\,, && \text{(7a)} \\ L_{\overline{x}}^2 + (L_{\overline{y}} \pm 1)^2 &= L^2 \pm 2L_{\overline{y}} + 1\,, && \text{(7b)} \\ (L_{\overline{x}} \pm 1)^2 + (L_{\overline{y}} \pm 1)^2 &= L^2 + 2(L_{\overline{x}} + L_{\overline{y}} \pm 1)\,, && \text{(7c)} \\ (L_{\overline{x}} \pm 1)^2 + (L_{\overline{y}} \mp 1)^2 &= L^2 + 2(\pm L_{\overline{x}} \mp L_{\overline{y}} + 1) && \text{(7d)} \\ &= L^2 \pm 2(L_{\overline{x}} - L_{\overline{y}} \pm 1)\,. \end{aligned}$$

[5]This numerically optimized implementation was first brought to our attention in an unpublished work of one of our colleagues at McGill University [7]. Since then a similar idea has been presented by Ye [24], but without much detail.

[6]An increment (or decrement) is usually faster to compute (or at least as fast to compute) on most machines than an add (or subtract).

The first two equations, (7a) and (7b), are used for comparisons of a pixel with its immediate horizontal or vertical neighbors, that is, along the two axes specifying the rectangular image grid. The other two equations, (7c) and (7d), are used for comparisons with the immediate diagonal neighbors of a pixel, that is, along a 45^{o} and a 135^{o} orientation, respectively.

2.1 Numerical Complexity

We can now evaluate the numerical complexity of both the 4SSEDT and 8SSEDT. Let us define the integer constant N^2 to represent the total number of object pixels found in an image I.[7] In the following, we consider that updates are performed only at object pixels. As we scan through the image, non-object pixels will also, in general, be encountered. By testing if the distance value of a pixel is null or not we can easily avoid doing any more computations with it. We note that this still requires one comparison operation (or maybe a logical AND) to be performed at each pixel in both O and O'. We will not consider this test operation in our evaluation of the numerical complexity of raster scan DT's since it is rather simple to perform in comparison to the other operations used to evaluate distance values.

Let us first consider the case of the 4SSEDT. Then, only Eq. (7a) and (7b) need be considered. The complete updating procedure will require six comparison operations for each object pixel, which we refer to by using the symbol CO_{4SSEDT} . The six neighbors are defined by the masks of the 4SSEDT (Fig. 2.(a)). A CO_{4SSEDT} operation represents one left-shift, one add (or subtract), one increment and one compare. Therefore, finding the minimum distance values in the case of the 4SSEDT will require a total of $6\mathrm{N}^2$ CO_{4SSEDT} operations. In the case of the 8SSEDT all four Eq. are used. The complete updating procedure will require ten comparison operations for each object pixel. Six of them are just the CO_{4SSEDT} operations as before. The four other comparisons are derived from Eq. (7c) and (7d) where both $L_{\overline{x}}$ and $L_{\overline{y}}$ are incremented/decremented simultaneously. They are associated with the four diagonal neighbors (with respect to p) found in the masks of Fig. 2.(b). We refer to these four comparison operations using the symbol CO_{8SSEDT} . A CO_{8SSEDT} operation represents one

[7] $\mathrm{N}^2 = \sum_{(\overline{x},\overline{y})} i$, where $i = 1 \Leftrightarrow I(\overline{x},\overline{y}) \in O$ and $i = 0 \Leftrightarrow I(\overline{x},\overline{y}) \in O'$.

left-shift, two add(s)/subtract(s), one increment/decrement and one compare. Therefore, finding the minimum distance values in the case of the 8SSEDT will require a total of $6\mathrm{N}^2$ CO_{4SSEDT} operations together with $4\mathrm{N}^2$ CO_{8SSEDT} operations; or in other words, $10\mathrm{N}^2$(left-shift, increment/decrement, compare) plus $14\mathrm{N}^2$(add/subtract) operations.

Once we have completely updated the vector image $\mathbf{L}_+$, the real Euclidean distance map is recovered by computing the square roots of the $\mathbf{L}_+$'s third element (Eq. (6)). This accounts for N^2 floating point operations. However, in many practical cases the (integer) squared distance values may be sufficient. Furthermore, since we know the maximum possible size of an object ($\leq \sqrt{M} = \sqrt{(M_{\overline{x}}+1)^2 + (M_{\overline{y}}+1)^2}$), for fixed size images I, we can store all possible distance values L in a double-index lookup table [24]. This table can be indexed using the absolute values of the first two elements of the vector image $\mathbf{L}_+$, that is, $|L_{\overline{x}}|$ and $|L_{\overline{y}}|$. Such an approach is particularly useful for fixed size images if distance maps are to be evaluated often [12].

In summary, we have two versions of the raster scan signed *SEDT* algorithm: the 4SSEDT and the 8SSEDT, the latter being roughly twice as expensive numerically. Besides computations, the essential difference between these two *DT*'s relates to accuracy. This is briefly considered in the following subsection.

2.2 Accuracy

Both *SEDT* algorithms will possibly generate some errors in computing distance values when distance propagation on the discrete rectangular grid needs to go through very thin areas. This is the case when "propagation needs to pass *between* pixels" [19]. However, the errors generated by both the 8SSEDT and 4SSEDT algorithms have maximal amplitudes smaller than the half size of a pixel, that is, 0.09 and 0.29 pixel units,[8] respectively [8]. Therefore, these errors are negligible in [some] practical applications.

This completes our detailed analysis of the two versions of Danielsson's algorithm optimized for numerical performance. In a later section we will

[8]A pixel unit equals the distance between two horizontal or vertical pixel centroids on the discrete rectangular image grid.

show how these algorithms compare to the non-Euclidean ones, that is, the *SWDT*'s. In the following section, we first analyze the *SWDT*'s.

3 Sequential Weighted Distance Transforms

In this section we present the main characteristics of some *SWDT* algorithms. We will consider the four simplest *SWDT*'s: the city block *DT*, the chessboard *DT*, and the *chamfer DT*'s [4] for the two smallest neighborhood sizes (*i.e.*, 3×3 and 5×5).

In the case of the *SWDT*'s, rather than computing a *vector* image $\mathbf{L}$ or $\mathbf{L}_+$ as was the case for the *SEDT*'s, we compute a *scalar* image. We denote this image by the symbol $\mathbf{L}_-$, defined as follows:

$$\mathbf{L}_-(\overline{x}, \overline{y}) = (L_w)\,, \tag{8}$$

where L_w represents the approximated, unsigned and scaled distance value. Once $\mathbf{L}_-$ is computed, the best approximation to the Euclidean distance, $\hat{L}$, is obtained as follows:

$$\hat{L} \; = L_w\, r\,, \tag{9}$$

where r represents a multiplication factor used to scale the weights w_N of the masks employed to perform a *SWDT*. The value of r is a function of the selected metric $dist$, but is generally fixed at the value of the smallest weight.

Fig. 3 shows the masks used for the *SWDT*'s. Note that only two masks are sufficient for each *SWDT* while four were used for the *SEDT* (compare with Fig. 2). This implies that the propagation of distances will not be isotropic (*e.g.*, see the propagation envelopes of Fig. 1, (b) and (c), which do not cover a full 360° orientation range) and will generate errors in the distance map. However, this type of error is in general negligible "compared to [the errors due to] the differences between the Euclidean distance and the non-Euclidean approximations" [18].[9]

[9]Note that in certain atypical cases this type of error will *not* be negligible (*e.g.*, see [17, §4.2]).

The chamfer *DT*'s have different sizes and weights that can be chosen to satisfy various needs. A 3×3 chamfer *DT*is called chamfer-a-b where a and b represent the mask's weights (Fig. 3.(a)).[10] In order to propagate distances that approximate circular or Euclidean propagation, the following *natural* constraints are imposed on the weights a and b [5]:

$$0 < a \leq b \leq 2a\,. \tag{10}$$

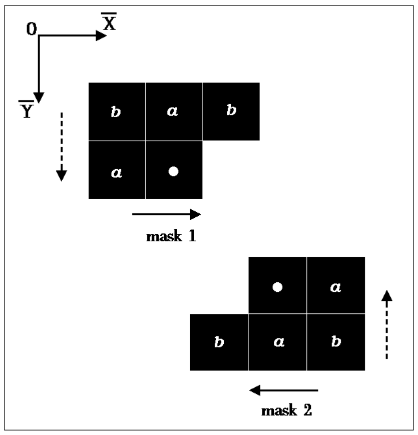
These constraints are called natural since they emphasize the desirable property that diagonal steps should never be smaller than horizontal or vertical ones on the discrete rectangular lattice.[11] Under these constraints, the weights a and b are chosen to take into account accuracy and numerical efficiency. The "optimal" weights, for accuracy, are $a \approx 0.95509$ and $b \approx 1.36930$ [5]. However, for practical applications, the weights $a = 23$ and $b = 33$ are preferred to perform integer computations [22]. Furthermore, the weights $a = 2$ and $b = 3$ may be preferred over $a = 23$ and $b = 33$ when the scaling by $r = a$ is required. In this case $r = 2$ and the divide operation is simply replaced with a right-shift operation. If one permits floating point operations, another interesting compromise between accuracy and numerical efficiency is obtained with the weights $a = 1$ and $b \approx 1.351$, where no scaling is required.

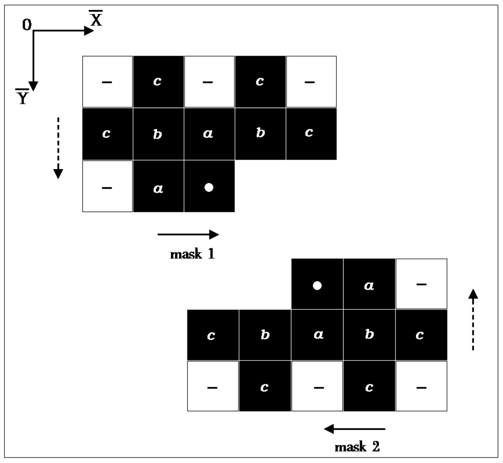
A 5×5 chamfer *DT* is called chamfer-a-b-c where an additional weight c is needed. For this neighborhood size, certain mask pixels do not contribute to the distance value propagation and are redundant. A simple definition for those mask pixels which are not redundant is obtained by observing the ratio of the vertical to the horizontal steps necessary to go from the central mask pixel to any other peripheral mask pixels. If this ratio consists of integer components having a *largest common divisor of one* [19], then the peripheral mask pixel is not redundant.[12] Redundant mask pixels are

[10]Note that the city block and the chessboard *DT*'s can be interpreted as the chamfer-1-2 and chamfer-1-1 *DT*'s, respectively (Fig. 3.(a)). For the city block *DT*, the weight b becomes redundant and the masks can be further reduced in size.

[11]Note that equality in Eq. (10) holds only for two particular cases of the city block *DT*and chessboard *DT*, that is, when $a = 1$ and $b = 2$ and $a = 1 = b$, respectively. Therefore, for any other chamfer-a-b *DT*'s, inequalities in Eq. (10) can be replaced by *strict* inequalities.

[12]An equivalent definition is obtained in terms of *Farey sequences* of order n, when





(a) (b)

Figure 3: Masks used for *SWDT*'s for the two smallest neighborhood sizes (*i.e.*, 3×3 and 5×5). The masks are centered at pixel p (indicated by a white dot). The weights in each mask represent the (approximated, unsigned and scaled) distance from p. The reference frame (directions indexed by $\overline{X}$ and $\overline{Y}$) is shown to be positioned in the top-left corner of an image, I. The arrows (next to the masks) indicate in which direction a given mask is passed over I. The recursive convolution of these masks over I is performed in two picture scans (indicated by dashed arrows) in opposite directions (here downward for masks labelled 1 and upward for masks labelled 2). In (a) are shown the masks for the, 3×3 chamfer *DT*'s where a and b are weights to be optimized (see text). The city block and the chessboard *DT*'s can be seen as particular cases of 3×3 chamfer *DT*'s; that is, the chamfer-1-2 and chamfer-1-1 *DT*'s, respectively. In (b) are shown the masks for the 5×5 chamfer *DT*'s where a, b and c are three weights to be optimized, and where some mask pixels, shown by a minus symbol $(-)$, are suppressed (see text).

therefore suppressed from the local search for a minimum distance value (marked with a minus symbol ($-$) in Fig. 3.(b)). As with the case of the chamfer-a-b DT's, the following *natural* constraints on the weights a, b and c are required:

$$0 < a < b < 2a < c < a + b\,. \tag{11}$$

Under these constraints, the weights a, b and c are chosen to satisfy the needs of accuracy and numerical efficiency. The optimal accuracy is obtained with $a \approx 0.98128$, $b \approx 1.40314$ and $c \approx 2.19529$ [22]. For practical applications, the weights $a = 5$, $b = 7$ and $c = 11$ are preferred [5, 22].[13] Furthermore, the weights $a = 4$, $b = 6$ and $c = 9$ may be preferred over $a = 5$, $b = 7$ and $c = 11$ when the scaling by $r = a$ is required. In this case $r = 4$, and the divide operation is simply replaced by a double right-shift operation. If one permits floating point operations, another interesting compromise between accuracy and numerical efficiency is obtained with the weights $a = 1$, $b \approx 1.41$ and $c \approx 2.197$, where no scaling is required.

3.1 Numerical Complexity

As before, we use the constant N^2 to represent the total number of object pixels found in an image. For the city block DT (Fig. 3.(a)), a complete updating procedure will require four comparison operations, CO_{CB} , for each object pixel, $p \in O$. A CO_{CB} operation involves one increment and one compare. Therefore, finding the minimum distance values in the case of the city block DTrequires $4\mathrm{N}^2$ CO_{CB} operations. For the chessboard DT (Fig. 3.(b)), each object pixel will require eight comparison operations, CO_{CB} (*i.e.*, same operations as for the city block DT). Therefore, finding the minimum distance values in the case of the chessboard DT requires $8\mathrm{N}^2$ CO_{CB} operations.

considering neighborhoods of size $(2n + 1)$x$(2n + 1)$ [16]. The sequence of valid ratios (*i.e.*, corresponding to mask pixels which are not redundant) is the Farey sequence of order n, that is, "the ordered sequence of all rational numbers between 0 and 1 with denominators less than or equal to n " [16].

[13]Vossepoel has made the interesting observation that for practical applications, the integer valued weights satisfy the relation: $c = a + b - 1$ [22].

The chamfer-a-b *DT*'s (Fig. 3.(a)) require eight comparison operations, per object pixel, $CO_{chamfer}$. A $CO_{chamfer}$ comparison consists of one add and one compare. Therefore, finding the minimum distance values in the case of the chamfer-a-b *DT*'s requires $8\mathrm{N}^2$ $CO_{chamfer}$ operations. The chamfer-a-b-c *DT*'s (Fig. 3.(b)) requires sixteen comparison operations, $CO_{chamfer}$, per object pixel (*i.e.*, the same operations as the chamfer-a-b *DT*'s). Therefore, finding the minimum distance values in the case of the chamfer-a-b-c *DT*'s requires $16\mathrm{N}^2$ $CO_{chamfer}$ operations.

Once we have completely updated the scalar image $\mathbf{L}_{-}$, the best approximation to the Euclidean distance map is recovered by computing the ratios, as indicated by Eq. (9), for all processed object pixels. This accounts for N^2 floating point operations. However, note that certain *SWDT*'s will not need this last step (*e.g.*, the city block *DT* and chessboard *DT*) or will require simpler operations (*e.g.*, left-shifts for divisions by powers of two, such as for the chamfer-2-3 *DT*).

3.2 Accuracy

The accuracy of *SWDT*'s are a function of the size of the neighborhood N_p, the values taken by the weights, w_N, and the value taken by the scaling factor, r. As a general principle, the larger the neighborhood, the better the accuracy (although the weights' values can counteract this effect). Essentially, with larger neighborhoods, circular propagation can more easily be approximated.

The weights w_N, which represent an approximation of the local distance from a pixel to its surrounding neighbors, can be selected on the basis of different error criteria. Essentially, an error measure with respect to the Euclidean distance is selected and sequences of weights are then evaluated on this basis. For example, Borgefors has proposed the *maximum difference* between the chamfer *DT* result and the *EDT* [5] and Vossepoel, the *minimum average difference* [22].

The scaling factor r may be fixed to values other than a. This may be useful for some sequences of weights as it permits having maximal errors symmetrically distributed over all directions of propagation [22].

4 Comparison of Sequential Distance Transforms

On the basis of the discussions in § 2 and § 3, we will in this section summarize our results by comparing *SEDT*'s with *SWDT*'s. Comparisons will be made in terms of accuracy and numerical complexity.

An important remark has first to be made concerning our evaluation of numerical complexity and its relation to the *speed* of processing of a *DT*. We will take as a measure of speed the numerical complexity we have evaluated previously. However, other measures are also possible. For example, Ragnemalm only considers the *number of memory accesses* [19]. Note that our measures of numerical complexity also embody the number of memory accesses (*i.e.*, number of pixels processed, updated or simply accessed). Furthermore, we also account for the complexity of the type of operation performed on each accessed pixel (*e.g.*, adds, increments, square roots, sorting, etc.) and distinguish whether integer or floating point arithmetic is used. We also discuss the effects of including or not rescaling operations (a step not always necessary in practice). Significantly, we assume that memory requirements are not of concern in our evaluation of the different *DT*'s.[14] Finally, following Ragnemalm [19], we will assume that the bus used to read or write values from the "distance image" $\mathbf{L}_+$, in the case of *SEDT*'s, is large enough to process a whole three-component vector (*i.e.*, $(L_{\overline{x}}, L_{\overline{y}}, L^2)$) with only one memory access.

In Table 1 we compare *SEDT*'s with *SWDT*'s from the point of view of both numerical complexity and accuracy. *DT*'s are ranked according to their accuracy.

4.1 Accuracy

Let us first note that both the 8SSEDT and the 4SSEDT have a maximal possible error (MaxDiff in Table 1) expressed in absolute pixel unit val-

[14]We have avoided memory requirements in our discussions since these days memory is considered to be more or less an "infinite" resource. However, if memory does become an issue for a particular application, other data structures could be considered for both *SEDT*'s or *SWDT*'s to ease the memory requirements. For example, *pyramid* structures can be used for such a purpose (*e.g.*, see [6]).

Distance Transform (raster scan)	Numerical Complexity		Accuracy (MaxDiff)
	Comparison op.	Rescaling	
8SSEDT	$10\mathrm{N}^2(<<, ++/--, <)$ $+41N^2(+/-)$	$\mathrm{N}^2(\sqrt{\ })$	-0.09 *pu* (1)
4SSEDT	$6\mathrm{N}^2(<<, ++/--, <)$	$\mathrm{N}^2(\sqrt{\ })$	-0.29 *pu* (1)
chamfer-*a*-*b*-*c* *DT*'s:	$16\mathrm{N}^2(+, <)$	$\mathrm{N}^2(\div)$	
$a \approx 0.981, b \approx 1.403, c \approx 2.195$	″	″	1.872% (2)
$a \approx 1, b \approx 1.41, c \approx 2.197$	″	None	1.96% (3)
$a \approx 5, b \approx 7, c \approx 11$	″	$\mathrm{N}^2(\div)$	2.175% (2)
$a \approx 4, b \approx 6, c \approx 9$	″	$2\mathrm{N}^2(>>)$	$\approx$ 4% (2)
chamfer-*a*-*b* *DT*'s	$8\mathrm{N}^2(+, <)$	$\mathrm{N}^2(\div)$	
$a \approx 0.955, b \approx 1.369$	″	″	4.491% (2)
$a \approx 1, b \approx 1.351$	″	None	6.351% (3)
$a \approx 20, b \approx 27$	″	$\mathrm{N}^2(\div)$	6.42% (3)
$a \approx 8, b \approx 11$	″	$3\mathrm{N}^2(>>)$	7.30% (3)
$a \approx 2, b \approx 3$	″	$\mathrm{N}^2(>>)$	13.40% (4)
chessboard *DT*	$8\mathrm{N}^2(++, <)$	None	41.4% (4)
city block *DT*	$4\mathrm{N}^2(++, <)$	None	58.6% (4)

Table 1: Comparison of *SEDT*'s and *SWDT*'s for both numerical complexity and accuracy. *DT*'s are ranked according to decreasing accuracy. For chamfer *DT*'s, useful (*i.e.*, for accuracy or speed) real and integer weights are given. For numerical complexity, both comparison and rescaling operations are shown. The significance of the symbols is: left-shift ($<<$), right-shift ($>>$), increment/decrement ($++/--$), add(s)/subtract(s) ($+/-$), compare ($<$), square root ($\sqrt{\ }$), divide ($\div$). N^2 stands for the number of object pixels being processed. For accuracy, the maximum possible difference (MaxDiff) or error with respect to the true Euclidean distance is given in percent (%) or in absolute pixel units, *pu*. References for accuracy are: (1) [8], (2) [22], (3) [5], (4) [4].

ues, pu, which is less than the sampling error of the digital grid ($\pm 0.5pu$). Therefore, these errors [may] be considered negligible [18]. Moreover, as the distance values are augmented, these possible errors become more and more negligible (in relative percentage), that is, their effect tends rapidly toward zero. Also, the errors can only occur in a few rare and sparsely distributed locations [8] and therefore they do not propagate over large portions of the distance map.

In the case of the *SWDT*'s, all errors are expressed as a percentage of the real Euclidean distance value. Therefore, in distinction to the *SEDT*'s, the absolute errors increase with the distance values, L. Finally, these errors generally occur in groups of pixels (*i.e.*, are not isolated) and propagate rapidly to corrupt large portions of the distance map.

4.2 Numerical Complexity

Let us now consider the numerical complexity of the different raster scan *DT* algorithms. We first ignore rescaling and any floating point operations; this eliminates the chamfer *DT*'s that use real-valued weights. Thus, the fastest *DT* is the city block *DT* (because it uses the smallest masks), but it is also the least accurate. In second place we find the chessboard *DT* and the 4SSEDT which have similar numerical complexity, with a slight advantage for the chessboard *DT*. The 4SSEDT is slightly slower than the chessboard *DT*, even though the latter requires a visit to 25% more pixels, because the comparison operations for the 4SSEDT are more complex than in the chessboard case. In fourth place follows the chamfer-a-b *DT*'s with integer-valued weights. These have comparison operations as complex as the ones for the 4SSEDT (if we neglect the left-shift and the increment operations in front of the add/subtract operation), but they require a visit to 25% more pixels. In fifth place comes the 8SSEDT, followed by the chamfer-a-b-c *DT*'s with integer-valued weights (*e.g.*, the chamfer-5-7-11 *DT*). Finally, other chamfer *DT*'s defined on larger neighborhood follow; *e.g.*, 7×7 chamfer *DT*'s.

If we include rescaling operations, but still consider integer-valued weights only, the city block *DT* is still the fastest, followed by the chessboard *DT* (which do not require any rescaling operations). In third place come chamfer-a-b *DT*'s with a scaling factor of power two (*e.g.*, in Table 1, chamfer-

2-3 *DT* with $r = a = 2$, followed by chamfer-8-11 *DT* with $r = 8$). Then comes either the 4SSEDT or the chamfer-20-27 *DT* (or any chamfer-a-b *DT*'s with integer-valued weights but requiring a (floating point) divide operation) depending on whether or not a square root is faster then a divide operation on a given machine.[15] Then come chamfer-a-b-c *DT*'s with integer-valued weights having a scaling factor of power of two (*e.g.*, chamfer-4-6-9 *DT*), followed by the 8SSEDT and chamfer-a-b-c *DT*'s with integer-valued weights but requiring a (floating point) divide operation (*e.g.*, chamfer-5-7-11 *DT*). Finally, other chamfer *DT*'s defined on larger neighborhoods follow "paired" as above; that is, first chamfer *DT*'s of a given size with r a power of two, followed by those with other scaling factors.

If we now consider real-valued weights, we get a similar classification as above with some additions. After the chamfer-a-b *DT*'s with integer-valued weights and the 4SSEDT we now have to include chamfer-a-b *DT*'s with real-valued weights (*i.e.*, chamfer-1-1.351 *DT* with no scaling followed by other chamfer-a-b *DT*'s). Similarly, after chamfer-a-b-c *DT*'s with integer-valued weights and the 8SSEDT we now have to include chamfer-a-b-c *DT*'s with real-valued weights (*i.e.*, chamfer-1-1.41-2.197 *DT* with no scaling followed by other chamfer-a-b-c *DT*'s). Then, similar comments apply to chamfer *DT*'s defined on larger neighborhoods.

In the worst possible case with respect to the numerical complexity of the *SEDT*'s, that is, where increment/decrement operations are not significantly faster than add/subtract and square root operations are slower than (floating point) divisions, we observe that 4SSEDT is faster than any *SWDT* defined over neighborhoods larger than 3×3. Similarly, 8SSEDT is faster than any *SWDT* defined over neighborhoods larger than 5×5.

5 Conclusion

In this paper we have described and compared raster scan sequential distance transforms implemented on the discrete rectangular lattice. In par-

[15]For example, on our SUN 3/50 workstation, square root operations are faster than (floating point) divide operations. However, if we use a look-up table to bypass the square root computations (§ 2), the 4SSEDT is definitely faster than any chamfer-a-b *DT* that requires floating point divisions.

ticular, by a careful analysis of two versions of the algorithm originally proposed by Danielsson, we have shown how the numerical complexity of signed *SEDT*'s can be simplified (*e.g.*, complexity of the 4SSEDT comparable to the chessboard *DT*'s) so that the need for other pseudo-Euclidean *DT*'s, that is *SWDT*'s, becomes unnecessary.

Aside from low numerical complexity, *SEDT*'s possess crucial advantages over *SWDT*'s. They provide "exact" results in the discrete domain. They also give the *orientation* towards the *source* from which the shortest distance was propagated (points of the background, p').

Considering the inaccuracies of *SWDT*'s and their relative lack of speed compared to *SEDT*'s, it seems that for practical applications they have little to offer.

One of the few possible advantages of *SWDT*'s over *SEDT*'s is that they have lower memory requirements. An efficient implementation of an *SEDT* algorithm like the one we have described in § 2 requires three times more space than the *SWDT*'s (to store the vector image $\mathbf{L}_+$). However, the extra information we obtain might be very useful (other than the fact that $\mathbf{L}_+$ gives us exact Euclidean distances at a low numerical complexity). For example, because we store both $\overline{x}$ and $\overline{y}$ distance values and their signed orientation (given by $L_{\overline{x}}$ and $L_{\overline{y}}$), we always know the exact position of the nearest background pixel for any object point. This information is particularly useful when employed in the computation of the *skeleton* of an object [12]. It permits us to directly recover the complete boundary from the skeleton, thereby yielding the *Inverse Grassfire Transform* [3, 12]. Also, when combined with an *active contour model*, this information provides a useful framework for simulating the *Grassfire Transform* [13, 14, 12].[16]

[16]See also [15] (note added by F.Leymarie, on Sept. 24, 2001).

References

[1] C. Arcelli and G. Sanniti di Baja. Computing Voronoi diagrams in digital pictures. *Pattern Recognition Letters*, 4:383–390, Oct 1986.

[2] C. Arcelli and G. Sanniti di Baja. Finding local maxima in a pseudo-Euclidean distance transform. *Computer Vision, Graphics and Image Processing*, 43:361–367, 1988.

[3] H. Blum. Biological shape and visual science. *Journal of Theoretical Biology*, 38:205–287, 1973.

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

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

[6] G. Borgefors. Time: Time and memory efficient distance transformations for parallel and pyramid machines. Technical Report FOA Report (ISSN 0347-3708) C 30531-3.4, Swedish Defense Research Establishment, Dept. of Information Technology, Linköping, Sweden, May 1989.

[7] H. Cox. Distance transformations in binary images. Project Report in Image Processing and Communications, Dept. of Elec. Eng., McGill University, Montreal, QC, Canada, Dec. 1986.

[8] P. E. Danielsson. Euclidean distance mapping. *Computer Graphics and Image Processing*, 14:227–248, 1980.

[9] P. E. Danielsson and S. Tanimoto. Time complexity for serial and parallel propagation in images. In *Proceedings of the SPIE Conference on Architecture and Algorithms for Digital Image Processing*, volume 435, pages 60–67. Society of Photo-Optical Instrumentation Engineers, 1983.

[10] L. Dorst. Pseudo-Euclidean skeletons. In *Proceedings of the 8th International Conference on Pattern Recognition*, pages 286–288, Paris, France, October 1986. IEEE Computer Society Press.

[11] L. Dorst and P. W. Verbeek. The constrained distance transformation: A pseudo-Euclidean, recursive implementation of the Lee-algorithm. In I. Young, editor, *Signal Processing III: Theories and Applications*, pages 917–920. Elsevier Science, Amsterdam, The Netherlands, 1986.

[12] F. Leymarie. Tracking and describing deformable objects using active contour models. Mcrcim technical report cim-90-9, McGill University, Elec. Eng. Dept., Montreal, QC, Canada, Feb 1990.

[13] F. Leymarie and M. D. Levine. New method for shape description based on an active contour model. In *Proceedings of the SPIE "Visual Communications and Image Processing" Conference*, volume 1199 of *Part 1*, pages 390–401, Philadelphia, PA, U.S.A., November 1989.

[14] F. Leymarie and M. D. Levine. Skeletons from snakes. In V. Cantoni et al., editors, *Progress in Image Analysis and Processing*, pages 186–193. World Scientific, Singapore, 1989.

[15] F. Leymarie and M. D. Levine. Simulating the grassfire transform using an active contour model. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(1):56–75, January 1992.

[16] U. Montanari. A method for obtaining skeletons using a quasi-Euclidean distance. *Journal of the Association for Computing Machinery*, 15:600–624, Oct 1968.

[17] J. Piper and E. Granum. Computing distance transformation in convex and non-convex domains. *Pattern Recognition*, 20(6):599–615, 1987.

[18] I. Ragnemalm. The Euclidean distance transform and its implementation on SIMD architectures. In *Proceedings of the 6th Scandinavian Conference on Image Analysis*, pages 379–384, Oulu, Finland, June 1989.

[19] I. Ragnemalm. Generation of Euclidean distance maps. Licentiate thesis no. 206, Linkoping University, Elec. Eng. Dept., Linkoping, Sweden, January 1990.

[20] A. Rosenfeld and J. L. Pfaltz. Sequential operations in digital picture processing. *Journal of the Association for Computing Machinery*, 13:471–494, Oct 1966.

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

[22] A. M. Vossepoel. A note on "Distance transformations in digital images". *Computer Vision, Graphics and Image Processing*, 43:88–97, 1968.

[23] H. Yamada. Complete Euclidean distance transformation by parallel operation. In *Proceedings of the 7th International Conference on Pattern Recognition*, pages 336–338, Montreal, QC, Canada, July 1984. IEEE Computer Society Press.

[24] Q.-Z. Ye. The signed Euclidean distance transform and its applications. In *Proceedings of the 9th International Conference on Pattern Recognition*, volume 1, pages 495–499, Rome, Italy, Nov 1988. IEEE Computer Society Press.