COMPUTER VISION, GRAPHICS, AND IMAGE PROCESSING 34, 344–371 (1986)

Distance Transformations in Digital Images

GUNILLA BORGEFORS

National Defence Research Institute, Box 1165, S-581 11 Linköping, Sweden

Received September 10, 1985; revised February 6, 1986

A distance transformation converts a binary digital image, consisting of feature and non-feature pixels, into an image where all non-feature pixels have a value corresponding to the distance to the nearest feature pixel. Computing these distances is in principle a global operation. However, global operations are prohibitively costly. Therefore algorithms that consider only small neighborhoods, but still give a reasonable approximation of the Euclidean distance, are necessary. In the first part of this paper optimal distance transformations are developed. Local neighborhoods of sizes up to 7×7 pixels are used. First real-valued distance transformations are considered, and then the best integer approximations of them are computed. A new distance transformation is presented, that is easily computed and has a maximal error of about 2%. In the second part of the paper six different distance transformations, both old and new, are used for a few different applications. These applications show both that the choice of distance transformation is important, and that any of the six transformations may be the right choice.

1. INTRODUCTION

Consider a digital binary image, consisting of feature and non-feature pixels. The features can be points, edges, or objects. A distance transformation is an operation that converts this binary image to a grey-level image where all pixels have a value corresponding to the distance to the nearest feature pixel. An example is shown in Fig. 1. The binary image depicts the letter *F*. After the distance transformation all pixels are have a value corresponding to the distance to the *F*. The image can be seen as a series of distance contours, each contour being all pixels equidistant from the feature.

Computing the distance from a pixel to a set of feature pixels is essentially a global operation. Unless the digital image is very small, all global operations are prohibitively costly. Therefore algorithms that consider only a small neighborhood at a time, but still give a reasonable approximation to the Euclidean distance are necessary. Distance transformation algorithms that use small neighborhoods will be denoted DTs henceforth. A number of different DTs, more or less complex, and more or less accurate, have been developed, and will be discussed in this paper.

The paper consists of two parts. In the first part, Section 3, optimal DTs are computed, optimal in the sense that the maximum difference from the Euclidean distance that can occur is minimized. Local neighborhoods of sizes up to 7×7 pixels are investigated. Parts of the results for 3×3 neighborhoods have been published before [1], but in less mathematical detail. An excellent new DT is presented, that is easily computed, and that has a maximal difference from the Euclidean distance of about 2%.

In the second part, Section 4, six different DTs are used in some applications. Two of the DTs are the best integer ones developed in Section 3. The other four ones are previously published DTs, which will be briefly described, with references, in Section 4. The applications show both that DTs in general are useful in a number

0734-189X/86 $3.00

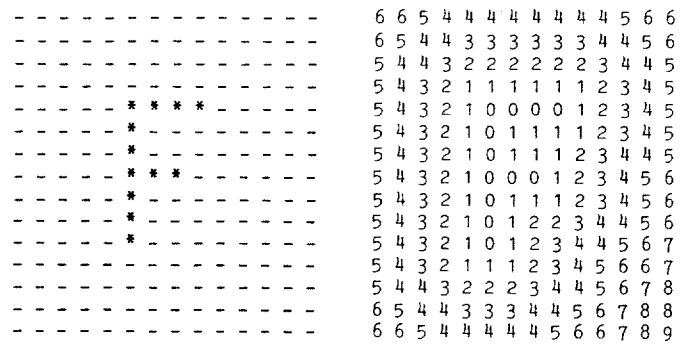

FIG. 1. Example of a distance transformation. To the left is a binary image, with feature (*) and non-feature (—) pixels. To the right is the resulting image: Each pixel has a value corresponding to the distance to the nearest feature pixel. The Euclidean distance has been rounded to the nearest integer.

of contexts, and that the choice of DT is important. One application is new: the computation of pseudo-Dirichlet tessellations in digital images, Section 4.4, where the fact that the images are digital rather than continuous is taken into account.

2. BASIC IDEA AND ALGORITHMS

Digital distance transforms, DTs, that use only a small image neighborhood at a time are based on the following idea: Global distances in the image are approximated by propagating local distances, i.e., distances between neighboring pixels. This propagation can be done either in parallel or sequentially. Sequential DTs was first published in 1966 [2], and parallel ones in 1968 [3]. These papers present the basic idea, and some DTs.

An original binary image, to which the DT is to be applied, consists of feature pixels with the initial value zero, and non-feature pixels with the initial value infinity, i.e., a suitably large number. All DTs will here be described in graphical form as "masks," see Fig. 2. Note that the DT masks are not linear filters! The constants c_n are the local distances that are propagated over the image. The size of the neighborhood can vary. In Fig. 2, a 5×5 neighborhood is illustrated.

The computation of the DT is either parallel or sequential. In the parallel case the center of the mask at the top of Fig. 2 is placed over each pixel in the image. The local distance in each mask-pixel c_n is added to the value of the image pixel "below" it (including the central zero). The new value of the image pixel is the minimum of all the sums. The process is repeated until no pixel value changes, i.e., the number of iterations is proportional to the largest distance in the image. The parallel algorithm is thus:

$$v_{i,j}^{m} = \underset{(k,l)\in \text{mask}}{\text{minimum}} \left(v_{i+k,j+1}^{m-1} + c(k,l) \right) \tag{1}$$

where $v_{i,j}^{m}$ is the value of the pixel in position (i, j) in the image at iteration m, (k, l) is the position in the mask (the center being $(0, 0)$), and $c(k, l)$ is the local distance from the mask. A small example of the parallel algorithm is shown at the top of Fig. 3.

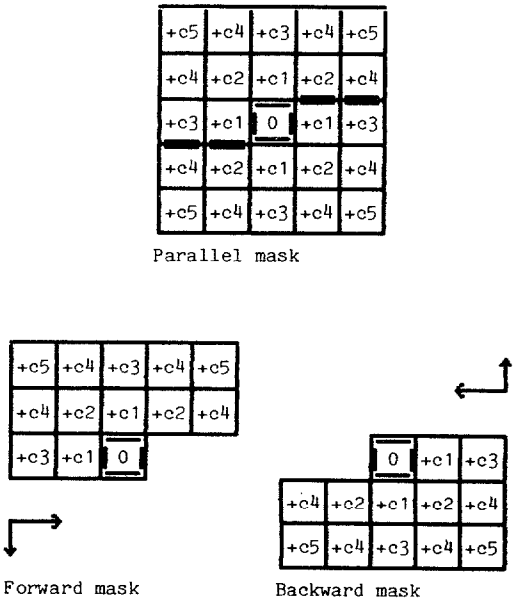

FIG. 2. Masks describing the distance transformations. The upper mask is used in parallel computations. In sequential computations that mask is split at the thick line, resulting in the two lower masks.

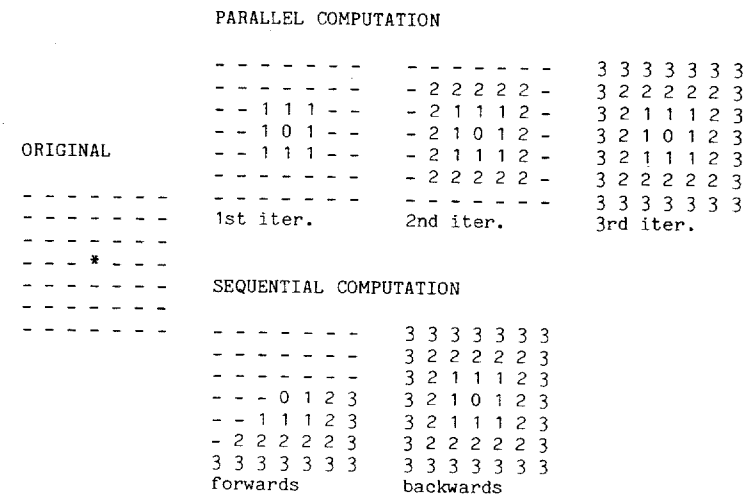

FIG. 3. Computation of a DT. At the left is the original image with one feature point in the middle. The upper images illustrate the parallel algorithm. The lower images illustrate the sequential algorithm, showing the result of the forward and backward passes. (The DT is the chessboard one, see Sect. 4.1).

The sequential algorithm also starts from the zero/infinity image. The symmetrical parallel mask is split into two masks, shown at the bottom of Fig. 2. The masks are passed over the image once each: the forward one from left to right, and from top to bottom, and the backward one from right to left and from bottom to top. The new value of the "central" image pixel is the minimum of the sums of the image pixel values and the local distances c_n, as before. After these two passes the distance transform is computed. The sequential algorithm is thus:

Forward:

for i = (size + 1)/2, . . . , lines do

for j = (size + 1)/2, . . . , columns do

$$v_{i,j} = \min_{\substack{(k,l)\in \\ \text{forward mask}}} \left(v_{i+k,j+l} + c(k,l)\right) \tag{2}$$

Backward:

for i = lines − (size − 1)/2, . . . , 1 do

for j = columns − (size − 1)/2, . . . , 1 do

$$v_{i,j} = \min_{\substack{(k,l)\in \\ \text{backward mask}}} \left(v_{i+k,j+l} + c(k,l)\right)$$

where "size" is the side-length of the mask and the rest of the notation is the same as in (1). A small example of the sequential algorithm is found at the bottom of Fig. 3.

From now on each DT will be described only by its parallel mask (with a thick line indicating where it should be split in the sequential case). The final DT result is exactly the same whether the parallel (1) or sequential (2) computation method is used.

3. OPTIMAL DISTANCE TRANSFORMATIONS FOR DIFFERENT NEIGHBORHOOD SIZES

In this section optimal local distances in the DT will be derived. Optimality is here equivalent to minimizing the **maximum** difference between the DT and the Euclidean distance that can possibly occur. Other definitions of optimality could of course be used, e.g., minimizing the **average** difference. Then the values of the optimal local distances would be (slightly) different.

The Euclidean distance transformation, that gives the correct real valued Euclidean distance between pixel centers, is abbreviated EDT henceforth. Algorithms that compute EDT have been published, but they are rather computationally complex, see Section 4.1. (If EDT could be easily computed there would be no need for approximations.)

In early papers about distance transforms, [2, 3], almost no attempt was made to optimize the local distances used in the DT. This optimization was accomplished for a 3 × 3 neighborhood in [4 and 1]. To make the presentation in this paper complete the relevant results from [1] are repeated here, but extended and in greater

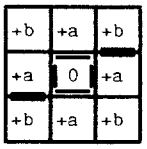

FIG. 4. The 3 × 3 neighborhood mask. The local distances a and b are to be optimized.

mathematical detail. The mathematics of the optimization is straightforward. The greatest difficulty is to keep track of all the different equations.

Naturally the approximation to the EDT becomes better the larger the size of the neighborhood that is used in the algorithm is. The neighborhood sizes 3 × 3, 5 × 5, and 7 × 7 are analyzed.

In many digital image processing applications real-valued pixels are undesirable. This is especially the case when using dedicated hardware, and when simplicity and speed are essential. Therefore, the true goal of optimization of the local distances is to find as good integer approximations as possible. Such integer approximations are discussed in Section 3.4.

3.1. 3 × 3 Neighborhood

The general 3 × 3 neighborhood mask is found in Fig. 4. The two local distances a and b are to be determined, where a is the distance between horizontal/vertical neighbors, and b is the distance between diagonal neighbors. Values that have been suggested for a and b are: $a = 1$, $b =$ infinity, and $a = 1$, $b = 1$ in [2]; $a = 1$, $b = \sqrt{2}$ in [5]; $a = 2$, $b = 3$ in [6]; and $a = 3$, $b = 4$ in [1].

Consider two pixels with the horizontal distance x units, and the vertical distance y units. Assume $y \leq x$. This is not a restriction as the mask is symmetric. In Fig. 5 the geometry is illustrated. The two pixels under consideration are marked with black dots. Montanari has proved, [5, Theorem 1], that there always exists a minimal path between the pixels that consists of (at most) two straightline segments. A minimal path is the shortest path between the pixels, when the distance is

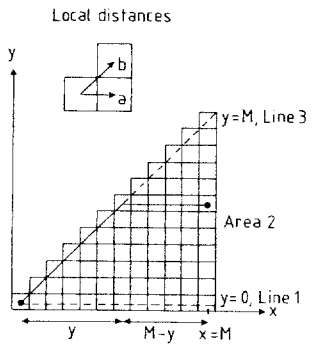

FIG. 5. The geometry of the DT in the 3 × 3 neighborhood case. The two local distances in the small upper diagram are used. The distance between the lower left-hand pixel and all pixels with $x = M$ are computed. The solid line is an example of a minimal path.

measured in the small steps allowed by the mask. Montanari's theorem is true for all neighborhood sizes, as long as the local distance values are constrained in some very natural ways (discussed below).

The minimal path between the points in Fig. 5 is the solid line. The length of the diagonal piece is $y*b$, and the length of the horizontal piece is $(x - y)*a$ (remember $x \geq y$). These expressions are valid for the minimal path between the pixel in the lower left-hand corner and any other pixel in the area illustrated in Fig. 5. Consider the DT along the vertical line $x = M$ (i.e., not only at the pixel centers). The DT value is a function of y, $T_2(y)$, where

$$T_2(y) = y*b + (M - y)*a = y(b - a) + Ma. \tag{3}$$

Function (3) is valid only if a and b are constrained as

$$b < 2a \qquad \text{and} \qquad b > a. \tag{4}$$

The first inequality ensures that one diagonal step is "shorter" than one horizontal + one vertical step, and the second inequality ensures that two diagonal steps (e.g., up right + down right) is "longer" than two horizontal steps. These constraints are the ones necessary for the validity of Montanari's theorem.

The difference Diff(y) between DT and EDT along the line $x = M$ is, using (3),

$$\text{Diff}(y) = y(b - a) + Ma - \sqrt{M^2 + y^2}, \qquad 0 \leq y \leq M. \tag{5}$$

The absolute maximum of (5) is the measure of optimality of the DT. The aim is now to determine a and b so that this maximum is minimized. The maximum occurs either when the derivative of Diff(y) is zero, or at the ends of the interval, $y = 0$ or $y = \text{M}$.

The maximum for many functions of the same type as (5), but with other constants, will be needed in the subsequent sections. Therefore a general formula is derived. For a function

$$F(y) = y*S + M*T - \sqrt{M^2 + y^2} \tag{6}$$

the derivative becomes

$$F'(y) = S - \frac{y}{\sqrt{M^2 + y^2}}. \tag{7}$$

The extremal value occurs for $F'(y) = 0$, i.e., for

$$y = \frac{M*S}{\sqrt{1 - S^2}}, \tag{8}$$

if $|S| < 1$, which is true if the constraints (4) hold. Inserting (8) into (6) and simplifying the resulting expression gives the value of the maximum:

$$F_{\max}(y) = (T - \sqrt{1 - S^2})*M. \tag{9}$$

This formula will be used frequently.

Using (9), the maximum of the difference (5) within the interval $0 < y < M$, i.e., in Area 2 of Fig. 5, becomes

$$\mathrm{Diff}_2 = \left(a - \sqrt{1 - (b-a)^2}\right) * M. \tag{10}$$

The ends of the interval are the points where $x = M$ crosses the lines $y = 0$ and $y = x$, drawn as dashed lines in Fig. 5. For $y = 0$, i.e., on Line 1, the difference (5) becomes

$$\mathrm{Diff}_1 = (a - 1) * M, \tag{11}$$

and for $y = M$, i.e., on Line 3, (5) becomes

$$\mathrm{Diff}_3 = (b - \sqrt{2}) * M. \tag{12}$$

First fix the local distance between horizontal/vertical pixels to one, $a = 1$. This is a natural assumption, as a can be interpreted as a scaling factor. Then from (11): $\mathrm{Diff}_1 = 0$; and from (4): $1 < b < 2$. The optimal b is the value that minimizes the absolute values of Diff_2, (10), and Diff_3, (12), i.e., minimizes

$$\max\left(1 - \sqrt{2b - b^2}, |b - \sqrt{2}|\right), \qquad 1 < b < 2, \tag{13}$$

where the constant factor M is disregarded. Expression (13) is illustrated in Fig. 6. Diff_2 and the absolute value of Diff_3 are drawn as functions of b. The optimal b is clearly the value for which the two curves cross, at $b \approx 1.35$.

The exact value of b can be found by solving $\mathrm{Diff}_2 = -\mathrm{Diff}_3$, i.e.,

$$1 - \sqrt{2b - b^2} = \sqrt{2} - b. \tag{14}$$

The solution of (14) is

$$b_{\mathrm{opt}} = \frac{1}{\sqrt{2}} + \sqrt{\sqrt{2} - 1} \approx 1.35070. \tag{15}$$

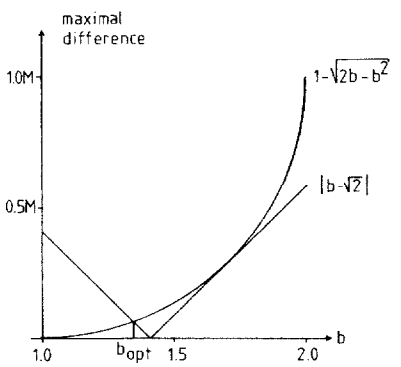

FIG. 6. The maximal difference between the DT and the EDT as a function of the local distance b (13). The two curves are the difference for $y = M$ (12), and for the maximal difference in $0 < y < M$ (10).

Note that b_{opt} is smaller than the corresponding Euclidean distance between diagonal neighbors: $\sqrt{2} \approx 1.41$.

The maximum difference possible, denoted maxdiff, is found by inserting (15) into (12) (or (10)):

$$\text{maxdiff} = \left(1/\sqrt{2} - \sqrt{\sqrt{2} - 1}\right) * M \approx 0.06351*M. \tag{16}$$

The difference from EDT has now been minimized along $x = M$. As M is arbitrary the difference have been minimized everywhere. In reality maxdiff is not proportional to the size of the image, but rather to the longest distance between feature and non-feature pixels that occurs in the image. As y actually only takes integer values (the pixel centers) maxdiff may never occur. It is thus an upper limit rather than a maximum.

In the previous optimization $a = 1$. Now let a be any real value (allowed by (4)). Then the optimal a and b can be found by minimizing the absolute value of the two-parametric functions Diff_1, Diff_2, and Diff_3, i.e., (10), (11), and (12). Some study reveals that this minimum occurs when $-\text{Diff}_1 = \text{Diff}_2 = -\text{Diff}_3$. Solving these equations the results become

$$a_{opt} = \left(\sqrt{2\sqrt{2} - 2} + 1\right)/2 \approx 0.95509$$

and

$$b_{opt} = \sqrt{2} + \left(\sqrt{2\sqrt{2} - 2} - 1\right)/2 \approx 1.36930. \tag{17}$$

The maximum difference from EDT is found by inserting a_{opt} and b_{opt} in (10) (or (11) or (12)); maxdiff becomes

$$\text{maxdiff} = \left(\sqrt{2\sqrt{2} - 2} - 1\right) * M/2 \approx 0.04491*M. \tag{18}$$

As before maxdiff is proportional to the size of the image (or rather the longest distance), but with a smaller factor, 0.045 instead of 0.064.

In the computations of a_{opt} and b_{opt} only the maximum difference from the EDT have been considered. But having minimized maxdiff it is interesting to see how the "error" varies along the line $x = M$, $0 \le y \le M$ (Fig. 5 again). In Fig. 7 the difference from the EDT (5), is plotted as a function of y, with the optimal local distances inserted.

When both a and b are free variables, (17), the difference function is the solid curve marked OPT in Fig. 7. The maximum absolute difference occurs at both ends of the interval, and also at a point within the interval (the function is not symmetric). Expressed in another way: the maximum difference from the EDT occurs along the lines angled $0° + n*45°$, $24.5° + n*90°$, and $65.5° + n*90°$ from the horizontal, where n is any integer.

When $a = 1$ and only b is optimized, (15), the difference function is the solid curve marked OPT_1 in Fig. 7. The difference is 0 for $y = 0$, grows to its maximal value, and then decreases to the same value with negative sign at the end of the

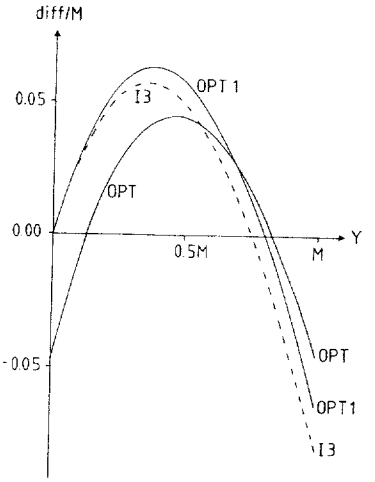

FIG. 7. The difference between some DTs and EDT along the line $y = M$ in the 3×3 neighborhood case. See the text.

interval. The maximum difference occurs along the lines angled $20.5° + n*90°$, $45° + n*90°$, and $69.5° + n*90°$ from the horizontal.

The dashed curve marked *I3* in Fig. 7 is the difference function for the recommended integer approximation in the 3×3 neighborhood case, $a = 3$ and $b = 4$, (see Sect. 3.4). The maximum difference here occurs for $y = M$, i.e., $45° + n*90°$.

3.2. 5×5 *Neighborhood*

When the local neighborhood is extended to a 5×5 pixels the general DT mask becomes the one in Fig. 8. Some of the mask-pixels are not used (marked —), as that would be pointless: Consider the mask-pixel in the middle of the leftmost column, and call its value x. If $x \geq 2a$, then x will never be "used" in the algorithm, as using two steps of length a will be shorter or equally long. If $x < 2a$ then the value a will never be used (except for the very first step from the feature pixels), and in practice the effect is a new a value, $a = x/2$. As each adding and comparison in the DT algorithms (1) or (2) take time, the number of mask-pixels should be as small as possible, and thus these unnecessary mask-pixels are excluded.

–	+c	–	+c	–
+c	+b	+a	+b	+c
–	+a	0	+a	–
+c	+b	+a	+b	+c
–	+c	–	+c	–

FIG. 8. The 5×5 neighborhood mask. The local distances a, b, and c are to be optimized. The empty mask-pixels are not needed in the computations (see the text).

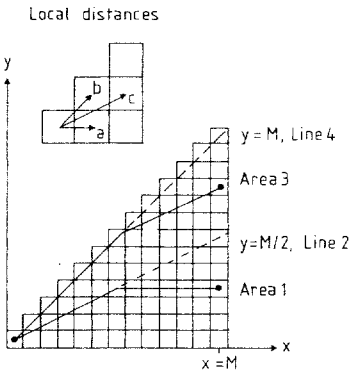

FIG. 9. The geometry of the DT in the 5 × 5 neighborhood case. The three local distances in the small upper diagram are used. The distance between the lower left-hand pixel and all pixels with $x = M$ are computed. The solid lines are examples of the two occurring types of minimal paths.

As in the 3 × 3 neighborhood case the maximum difference between the DT and the EDT along the line $x = M, 0 \leq y \leq M$ is minimized. The geometry is illustrated in Fig. 9. Montanari's theorem still holds (ensuring that the minimal path consists of two straight lines), if the values of the local distances are constrained in natural ways (cf. (4)). As there are three local distances involved, the minimal path between the lower left-hand pixel and a pixel on the line $x = M$ can be of two different types. The types are illustrated as solid lines in Fig. 9. The change between the two types occurs for $y = M/2$, marked with a dashed line in the figure.

For $y < M/2$ the path consists of the steps c and a, and for $y > M/2$ the path consists of the steps b and c. The value of the DT in the two intervals along the line is, for $0 \leq y \leq M/2$:

$$T_1(y) = y * c + (M - 2y) * a = y(c - 2a) + Ma, \tag{19}$$

and for $M/2 \leq y \leq M$:

$$T_3(y) = (2y - M) * b + (M - y) * c = y(2b - c) + M(c - b). \tag{20}$$

The maximum difference occurs either inside one of the two intervals, where the derivatives of T_1 − EDT and T_3 − EDT are zero, or at the ends of the intervals.

Let $a = 1$. Then the difference becomes zero for $y = 0$. The maximum in the interval $0 < y < M/2$ (Area 1 in Fig. 9) is found using the general formula (9) on T_1 − EDT (19):

$$\text{Diff}_1 = \left(1 - \sqrt{1 - (c - 2)^2}\right) * M. \tag{21}$$

For $y = M/2$ (Line 2) the DT value is $c/2*M$ (use (20)), EDT is $\sqrt{5}/2*M$, and the difference becomes

$$\text{Diff}_2 = (c - \sqrt{5})*M/2. \tag{22}$$

In the interval $M/2 < y < M$ (Area 3) the maximum is again found using (9) on T_3 – EDT, (20):

$$\text{Diff}_3 = \left(c - b - \sqrt{1 - (2b - c)^2}\right) * M. \tag{23}$$

Finally for $y = M$ (Line 4) the DT value is $b*M$ (use (20)), EDT is $\sqrt{2}*M$, and the difference becomes

$$\text{Diff}_4 = (b - \sqrt{2}) * M. \tag{24}$$

The optimal local distances b and c are the values that minimize the largest of the absolute values of the four difference expressions (21)–(24). When these expressions are studied it soon becomes apparent that only the value of c is critical, i.e., as long as b is within a certain (small) interval only c changes the maximal difference because Diff_1 and Diff_2 (which are dependent only on c) are larger than Diff_3 and Diff_4. Thus the approximation to EDT is worst in Area 1 (as can be expected when a is fixed). The optimal c is found by solving $\text{Diff}_1 = \text{Diff}_2$:

$$c_{\text{opt}} = \left(6 + \sqrt{5} + \sqrt{32\sqrt{5} - 64}\right)/5 \approx 2.19691. \tag{25}$$

Inserting the c_{opt} into (22) (or (21)), the maximum difference between the EDT and the DT becomes

$$\text{maxdiff} = \left(\sqrt{5} - c_{\text{opt}}\right) * M/2 \approx 0.01958*M. \tag{26}$$

With $c = c_{\text{opt}}$ the small interval within which b can vary can be computed. The ends of the interval are computed by finding the Diff_n, $n = 3$ or 4, that becomes larger than maxdiff at each end, and then solving the corresponding equation $\text{Diff}_n = \text{maxdiff}$ ($= -\text{Diff}_2$). The minimal b is found by solving $-\text{Diff}_4 = -\text{Diff}_2$:

$$b_{\min} = \left(2\sqrt{2} - \sqrt{5} + c_{\text{opt}}\right)/2 \approx 1.39463, \tag{27}$$

and the maximal b is found by solving $\text{Diff}_3 = -\text{Diff}_2$:

$$b_{\max} = \left(7c_{\text{opt}} - \sqrt{5} + 4\sqrt{\sqrt{5}\,c_{\text{opt}} - c_{\text{opt}}^2}\right)/10 \approx 1.43155. \tag{28}$$

The DT values will be different for different b, but as long as b is within the interval (27) to (28), maxdiff is unchanged. One attractive choice is $b = \sqrt{2} \approx 1.41$, which is within the allowed interval. The local distance is then equal to the corresponding Euclidean distance, and the difference from EDT becomes zero for $y = M$.

The local distances have not been optimized for $a \neq 1$. Undoubtedly maxdiff would be smaller, but the computations are complex and the most important aim of these real-valued optimizations is to, eventually, find good integer DTs. For integer DTs a becomes a scale factor, and the distance between horizontal/vertical neighbors is thus (in some sense) always one. The optimal local distances are used as

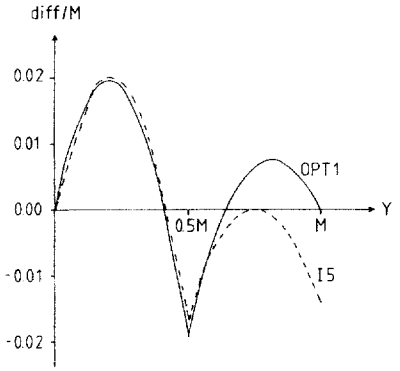

FIG. 10. The difference between some DTs and EDT along the line $y = M$ in the 5×5 neighborhood case (see the text).

indications of which integer values to use, but the choices are always confirmed by the resulting values of the difference expressions Diff_n.

As in the 3×3 neighborhood case the difference from the EDT along the line $x = M$, $0 \le y \le M$ have been plotted as a function of y, Fig. 10. The solid curve, marked OPT1, represents the difference when $a = 1$, $b = \sqrt{2}$, and $c = c_{\text{opt}}$. The function depends on a and c for $y < M/2$, and on b and c for $y > M/2$. The maximum absolute difference occurs in the first interval, and at $y = M/2$.

The dashed curve in Fig. 10, marked *I5*, represents the difference function for the recommended integer approximation in the 5×5 neighborhood case, $a = 5$, $b = 7$, and $c = 11$ (see Sect. 3.4).

3.3. 7×7 *Neighborhood*

Now consider a 7×7 pixel neighborhood. The general DT mask is the one in Fig. 11. There are five local distances to be determined. As in the 5×5 neighborhood case some of the mask-pixels can be excluded.

The geometry of this case is shown in Fig. 12. As before the difference between the DT and EDT is minimized along $x = M$, $0 \le y \le M$. A minimal path still

-	+e	+d	-	+d	+e	-
+e	-	+c	-	+c	-	+e
+d	+c	+b	+a	+b	+c	+d
-	-	+a	0	+a	-	-
+d	+c	+b	+a	+b	+c	+d
+e	-	+c	-	+c	-	+e
-	+e	+d	-	+d	+e	-

FIG. 11. The 7×7 neighborhood mask. The local distances a, b, c, d, and e are to be optimized. The empty mask-pixels are not needed in the computations (see the text).

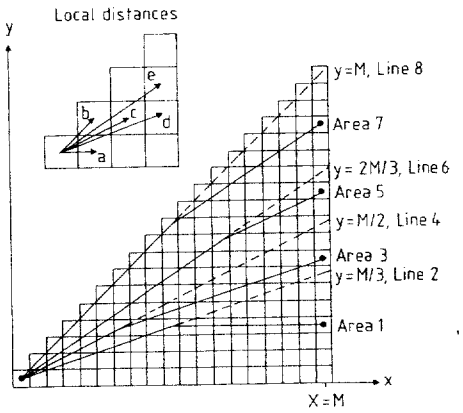

FIG. 12. The geometry of the DT in the 7 × 7 neighborhood case. The local distances in the small upper diagram are used. The distance between the lower left-hand pixel and all pixels with $x = M$ are computed. The solid lines are examples of the four occurring types of minimal paths.

consists of two straight segments. However, there are now four different types of minimal paths, exemplified by the four solid lines in Fig. 12. The changes between the different types of minimal paths occur at $y = M/3$, $y = M/2$, and $y = 2M/3$.

If the local distance values are suitably constrained (cf. (4)), then the value of the DT in the four different intervals is, for $0 \le y \le M/3$:

$$T_1(y) = y * d + (M - 3y) * a = y(d - 3a) + Ma; \tag{29}$$

for $M/3 \le y \le M/2$:

$$T_3(y) = (3y - M) * c + (M - 2y) * d = y(3c - 2d) + M(d - c); \tag{30}$$

for $M/2 \le y \le 2M/3$:

$$T_5(y) = (2y - M) * e + (2M - 3y) * c = y(2e - 3c) + M(2c - e); \tag{31}$$

and for $2M/3 \le y \le M$:

$$T_7(y) = (3y - 2M) * b + (M - y) * e = y(3b - e) + M(e - 2b). \tag{32}$$

If the local distance e is excluded (the reason for excluding it will be explained later) (31) and (32) are replaced by, for $M/2 \le y \le M$:

$$T_z(y) = (2y - M) * b + (M - y) * c = y(2b - c) + M(c - b). \tag{33}$$

The maximum difference between the DT and the EDT occurs either within the four intervals, where the derivative of the difference T_n − EDT is zero, or at the ends of the intervals.

Let $a = 1$ as in the 5 × 5 neighborhood case. Then the difference between DT and EDT is zero for $y = 0$. There are now eight different expressions that will

determine the maximum absolute difference. The expressions for the maxima within the intervals are computed using (29)–(32) and the general formula (9). The expressions for the ends of the intervals can also be found using (29)–(32). The eight expressions become

$$\text{Diff}_1 = \left(1 - \sqrt{1 - (d-3)^2}\right) * M, \tag{34}$$

$$\text{Diff}_2 = (d - \sqrt{10}) * M/3, \tag{35}$$

$$\text{Diff}_3 = \left(d - c - \sqrt{1 - (3c-d)^2}\right) * M, \tag{36}$$

$$\text{Diff}_4 = (c - \sqrt{5}) * M/2, \tag{37}$$

$$\text{Diff}_5 = \left(2c - e - \sqrt{1 - (2e-3c)^2}\right) * M, \tag{38}$$

$$\text{Diff}_6 = (e - \sqrt{13}) * M/3, \tag{39}$$

$$\text{Diff}_7 = \left(b - e - \sqrt{1 - (e-2b)^2}\right) * M, \text{ and} \tag{40}$$

$$\text{Diff}_8 = (b - \sqrt{2}) * M, \tag{41}$$

where the number n in Diff_n corresponds to the area and line numbers in Fig. 12.

If e is not used then Diff_5, Diff_6, and Diff_7, (38)–(40), are replaced by the single expression (see (33)):

$$\text{Diff}_z = \left(c - b - \sqrt{1 - (2b-c)^2}\right) * M. \tag{42}$$

As before, the task is to determine b, c, d, and e so that the absolute maximum of all the expressions Diff_n is minimized. When (34)–(41) are studied it becomes apparent that the **only** critical local distance is d, i.e., the approximation is worst in area 1, as in the 5×5 case (and for the same reason). The optimal d is the solution of $\text{Diff}_1 = -\text{Diff}_2$, (34) and (35), which is

$$d_{\text{opt}} = \left(24 + \sqrt{10} + \sqrt{108\sqrt{10} - 324}\right)/10 \approx 3.13487. \tag{43}$$

The absolute maximum difference then becomes (substituting (43) into (35)),

$$\text{maxdiff} = \left(9\sqrt{10} - 24 - \sqrt{108\sqrt{10} - 324}\right)/30 \approx 0.00914 * M. \tag{44}$$

With $d = d_{\text{opt}}$ the other three constants, b, c, and e, can be allowed to vary within certain small intervals, without increasing maxdiff. The intervals are determined by finding out which Diff_n that becomes critical at each end of the interval, and then solving the equation $\text{Diff}_n = \text{maxdiff}$.

The lower limit for b is determined by $-\text{Diff}_8 = -\text{Diff}_2$ (35) and (41), which gives

$$b_{\min} = \left(3\sqrt{2} - \sqrt{10} + d_{\text{opt}}\right)/3 \approx 1.40508. \tag{45}$$

The lower limit for c is determined by $-\text{Diff}_4 = -\text{Diff}_2$ (35) and (37), which gives

$$c_{\min} = \left(3\sqrt{5} - 2\sqrt{10} + 2d_{\text{opt}}\right)/3 \approx 2.21780. \tag{46}$$

The lower limit for e is determined by $-\text{Diff}_6 = -\text{Diff}_2$ (35) and (39), which gives

$$e_{\min} = \sqrt{13} - \sqrt{10} + d_{\text{opt}} \approx 3.57814. \tag{47}$$

The upper limits are a little more complex to compute. The easiest is the upper limit for c, which is determined by $\text{Diff}_3 = -\text{Diff}_2$. These expressions, (35) and (36), contain only d and c, and solving for c gives

$$c_{\max} = \left(22d_{\text{opt}} - \sqrt{10} + 6\sqrt{\sqrt{10}\,d_{\text{opt}} - d_{\text{opt}}^2}\right)\Big/30 \approx 2.25212. \tag{48}$$

The upper limits for b and e unfortunately depend on each other, as they are both determined by $\text{Diff}_7 = -\text{Diff}_2$. The constants b and e corresponds to the Euclidean distances $\sqrt{2}$ and $\sqrt{13}$, locally. It would be nice if these Euclidean values were included in the respective allowed intervals. To achieve this the following constraining equations are introduced:

$$b_{\max} = \sqrt{2} + x \qquad \text{and} \qquad e_{\max} = \sqrt{13} + x. \tag{49}$$

Substituting (49) into $\text{Diff}_7 = -\text{Diff}_2$ (35) and (40), and solving for x gives the following rather daunting expression:

$$\begin{aligned} x = \Big(& d_{\text{opt}} - 24\sqrt{2} - \sqrt{10} + 9\sqrt{13} \\ & + \sqrt{-4d_{\text{opt}}^2 + 4(3\sqrt{2} + 2\sqrt{10} - 3\sqrt{13})\,d_{\text{opt}} - 6(2\sqrt{20} - 3\sqrt{26} - 2\sqrt{130}) - 130}\Big)/15 \\ \approx\ & 0.00417. \end{aligned} \tag{50}$$

Thus the upper limits become (use (49) and (50))

$$b_{\max} = \sqrt{2} + x \approx 1.41839 \tag{51}$$

and

$$e_{\max} = \sqrt{13} + x \approx 3.60972. \tag{52}$$

Summing up the above expressions (43), (45)–(48), and (51)–(52), the optimal values of the four local distances b, c, d, and e are

$$\begin{aligned} &1.40508 \lessapprox b_{\text{opt}} \lessapprox 1.41839, \\ &2.21780 \lessapprox c_{\text{opt}} \lessapprox 2.25212, \\ &d_{\text{opt}} \approx 3.13487, \text{ and} \\ &3.57814 \lessapprox e_{\text{opt}} \lessapprox 3.60972. \end{aligned} \tag{53}$$

As has been hinted at before, the local distance e can be excluded without increasing maxdiff. The reason is that if e is excluded the local distance value of the

vector (3, 2) is approximated by $b + c$ instead of by e, see Fig. 12. Values for b and c can be chosen within their allowed intervals, (53), so that their sum is within the allowed interval for e, and thus maxdiff is unchanged. (Obviously the actual DT **values** do become different.)

If e is excluded Diff_z is valid instead of Diff_5, Diff_6, and Diff_7. The optimal value of d is not affected, as Diff_1 and Diff_2 still are the same. The lower limits for b and c are also unchanged. However, the upper limits of b and c now becomes determined by $\mathrm{Diff}_z = -\mathrm{Diff}_2$. As before these upper limits are dependent on each other. The allowed intervals are now much smaller (as could be expected), so that $\sqrt{2}$ and $\sqrt{5}$ cannot be included in the respective intervals. Extra arbitrary constraints must be added as before. In this case the constraint is that both intervals should be equally long:

$$b_{\max} = b_{\min} + x \qquad \text{and} \qquad c_{\max} = c_{\min} + x. \tag{54}$$

Substituting eqs. (54) into $\mathrm{Diff}_z = -\mathrm{Diff}_2$ (35), and (42), and solving for x gives the value of x as

$$\begin{aligned} x &= c_{\min} - 2b_{\min} \\ &\quad + \sqrt{6\left(\sqrt{10} - d_{\mathrm{opt}}\right)\left(c_{\min} - b_{\min}\right) - 9\left(c_{\min} - b_{\min}\right)^2 - d_{\mathrm{opt}}^2 + 2\sqrt{10}\, d_{\mathrm{opt}} - 1}\,/3 \\ &\approx 0.00284. \end{aligned} \tag{55}$$

Inserting (55) into (54) give the upper limits for b and c:

$$b_{\max} \approx 1.40791 \tag{56}$$

and

$$c_{\max} \approx 2.22063. \tag{57}$$

Summing up the new results (56) and (57) and using the old results in (53), the optimal values of the local distances b, c, and d becomes

$$\begin{aligned} &1.40508 \lessapprox b_{\mathrm{opt}} \lessapprox 1.40791, \\ &2.21780 \lessapprox c_{\mathrm{opt}} \lessapprox 2.22063, \text{ and} \\ &d_{\mathrm{opt}} \approx 3.13487. \end{aligned} \tag{58}$$

To exclude the local distance e will speed up the DT computation, without increasing the maximum difference from the EDT. There will then be 25 instead of 33 pixels in the DT mask (Fig. 11), and thus eight sums less to be computed in each step.

As in the previous cases the difference from the EDT along the line $x = M$, $0 \le y \le M$ have been plotted as a function of y, Fig. 13. The solid curve marked OPT1, shows the difference when $a = 1$, $b = \sqrt{2}$, $c = \sqrt{5}$, $d = d_{\mathrm{opt}}$, and $e = \sqrt{13}$. The maximum absolute difference occurs in Area 1 and on Line 2 (Fig. 12). The difference is zero for $y = 0$, $y = M/2$, $y = 2M/3$, and $y = M$.

The solid curve marked OPT1M (Minus e) shown the difference when $a = 1$, $b = 1.407$, $c = 2.220$, and $d = d_{\mathrm{opt}}$. This curve OPT1M is exactly the same as OPT1

TABLE 3
Integer Approximations of the Optimal Local Distances for a 7 × 7 Neighborhood

a	b	c	d	e	maxdiff
14	20	31	44	—	0.0197
15	21	33	47	—	0.0180
17	24	38	53	—	0.0147
19	27	42	60	—	0.0142
⋮	⋮	⋮	⋮	⋮	⋮
1	opt	opt	opt	—	0.0091
12	17	27	38	43	0.0140
19	27	42	60	68	0.0128
⋮	⋮	⋮	⋮	⋮	⋮
1	opt	opt	opt	opt	0.0091

Note. In the upper part of the table the local distance e is not used. Even though the optimal maxdiff is the same in both cases, the approximations become better using e.

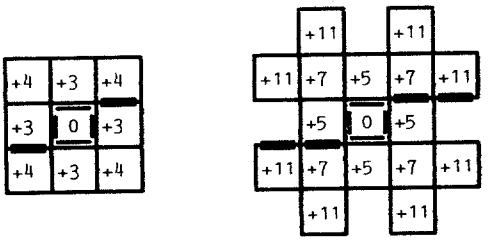

FIG. 14. The two recommended integer DTs with neighborhood size at most 7 × 7.

of Table 3 the local distance e is included. With $n = 12$ maxdiff does become somewhat lower than in the 5 × 5 case, but the difference is still too small to justify the additional computational complexity. Therefore no 7 × 7 integer approximation is recommended. However, the two "best" approximations are shown in Fig. 13.

The masks for the two recommended DTs are shown in Fig. 14. The 3 × 3 neighborhood algorithm gives an "error" of at most 8%, and the 5 × 5 neighborhood algorithm gives an "error" of at most 2%. The algorithms can be implemented either in parallel (1), or sequentially (2). There is no reason to use a 7 × 7 neighborhood if the local distances are integers!

Before discussing these results in more detail the two recommended DTs will be compared to four other well-known DTs. The six DTs will be applied to several image processing tasks. The optimal real-valued DTs will also be illustrated.

4. EXAMPLES AND COMPARISONS

Here six different distance transformations will be presented, Section 4.1. These will be applied to different digital image processing tasks, Sections 4.2–4.4. The results will be illustrated and compared, mostly as pictures.

4.1. The Six Distance Transformations

City Block. This is the simplest and fastest of all DTs. It is, however, also the worst approximation of the Euclidean distance. It is, among other places, found in [7]. The city block DT is described by the general 3×3 neighborhood mask, Fig. 4, with $a = 1$ and $b =$ infinity, i.e., the diagonal neighbors are ignored. As for all algorithms described by the mask, the computation can be either parallel, (1), or sequential, (2).

Chessboard. This DT, which is also found in [7], is described by the mask in Fig. 4, with $a = 1$ and $b = 1$.

Octagonal. This DT is a mix of the city block and the chessboard DTs. The underlying idea is to use the two DTs alternately, as city block distances are always too large, and chessboard distances always too small. Presentation of it, and a parallel algorithm for computing it are found in [3]. In the simplest case each DT is used every other time. Other octagonal algorithms are also described in [3], where the city block and chessboard DTs are mixed in more complex ways, to get a better approximation of EDT. A sequential algorithm for computing the simplest octagonal distance using, four passes over the image, is found in [8] and described with DT masks in [1].

Chamfer 3–4. This is the 3×3 neighborhood integer DT from Section 3.4. It is called a "chamfer" DT because the distance values are "chamfered out" in two passes over the image when the computation is sequential [6, 4, 1].

Chamfer 5–7–11. This is the 5×5 neighborhood integer DT from Section 3.4. This new DT is the key result of Section 3.

Euclidean. This is the true Euclidean distance, i.e., EDT. For sequential computation of EDT the best published algorithm is probably [8], even though the results are not always quite correct. Due to complex feature geometry errors can occur, but

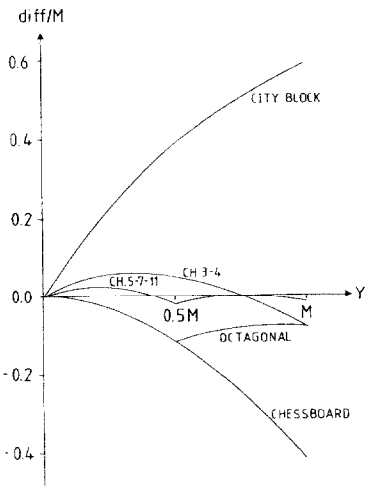

FIG. 15. The difference between the six DTs to be used in section 4 and EDT. The chessboard and octagonal curves coincide in the first half of the interval.

they are always small. This sequential algorithm uses four passes of a 3×3 neighborhood over the image; the sum of two squares must be computed for each of the nine mask-pixels; and it needs two extra images of the same size as the original one to store intermediate results (the number of vertical and horizontal steps to the nearest feature pixel). A parallel EDT algorithm that always gives correct results has been published, [9]. This algorithm also uses a 3×3 neighborhood; the sum of two squares must be computed for each mask-pixel; and it needs two extra images to store intermediate results (the signed number of steps to the nearest feature pixel). The number of iterations is proportional to the longest distance in the image.

The difference from the EDT for all the six DTs is shown in Fig. 15. The curves show the difference along the line $x = M$, $0 \leq y \leq M$, cf. Fig. 5. All DTs give the correct distance value along horizontal and vertical lines, and thus all curves start at zero. The city block and chessboard DTs are the worst approximations (and the

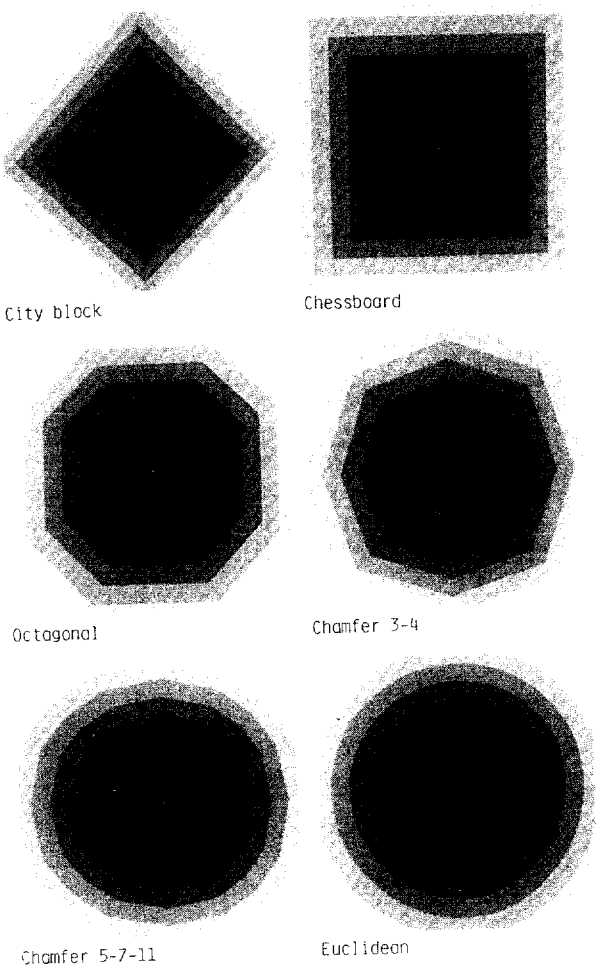

FIG. 16. The distances from a point for the six DTs. The lighter the color the larger the distance.

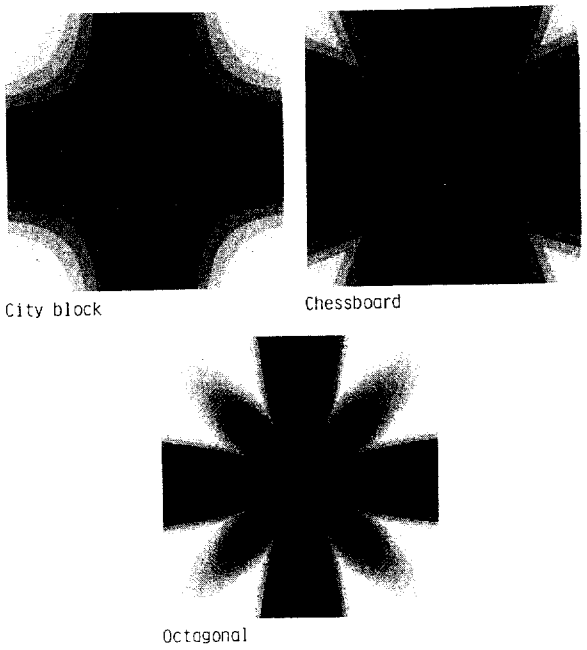

FIG. 17. The difference between the city block, chessboard, and octagonal DTs and the EDT (see the text).

most used DTs!). The octagonal, 3–4, and 5–7–11 are progressively better. The EDT is of course always correct.

4.2. *"Circles" and Difference from Euclidean Distance*

In Fig. 16 the distance from a point, i.e., a single pixel, have been computed using the different DTs. The distance from the central pixel to the edge is 100 pixels. The distance values have been grey-level coded: the larger the distance the lighter the color. The images illustrate the "circles" of the different DTs. The city block and chessboard circles are square. The octagonal circle is an octagon (as the name implies). The chamfer 3–4 circle is also an octagon, but one that is a better approximation of the Euclidean circle than the octagonal one. However, it is not quite regular, which may be a problem in applications where rotation occurs. The chamfer 5–7–11 circle is a hexadecagon, i.e., a polygon with 16 sides.

Previously the difference between the DTs and the EDT have been measured by its maximum, and it has also been illustrated in a number of diagrams, Figs. 7, 10, 13, and 15. Here the difference is illustrated as grey-level images: the lighter the color, the greater the absolute difference. The difference images have been computed from the images in Fig. 16, so that the difference "fields" around a point are illustrated.

The difference images for the city block, chessboard, and octagonal distances are found in Fig. 17. The grey-level scale is the same for the city block and chessboard DTs, but different for the octagonal distance, because otherwise the image would have been almost completely black. For the city block and chessboard DTs the

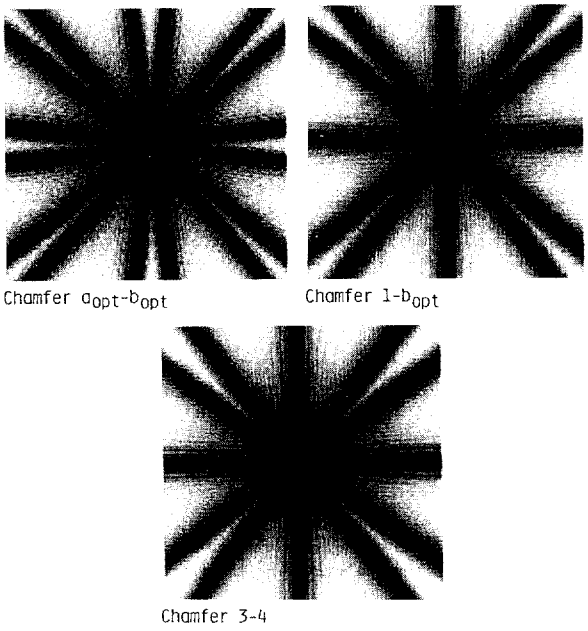

FIG. 18. The difference between the two optimal and the integer 3 × 3 neighborhood DTs and the EDT (see the text).

difference is greatest along the diagonals. The maximal differences in the images are −58.6 and +41.4, respectively. For the octagonal DT the difference is greatest at the corners of the associated octagonal circle. The maximal difference is +11.8.

Difference images have been computed for three 3 × 3 neighborhood DTs, Fig. 18. The first image shows the difference when both local distances a and b have their optimal values (17). The difference "field" is then very regular. The second image shows the difference when $a = 1$ and b is optimal (15). Finally the third image is the difference for the 3–4 integer approximation. The difference from the optimal image is not too great, but the difference is larger, especially along the diagonals. The maximal differences are ±4.5, ±6.4, and −8.1, respectively.

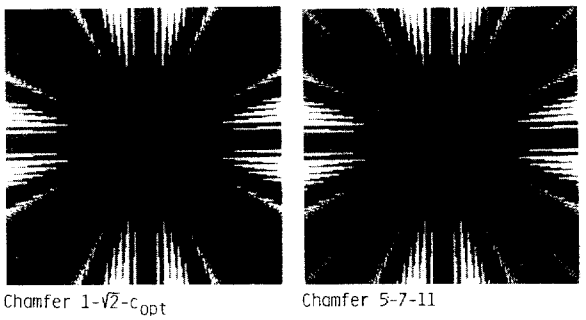

FIG. 19. The difference between the two 5 × 5 neighborhood DTs and the EDT (see the text).

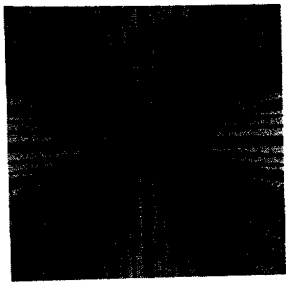

FIG. 20. The difference between the optimal 7 × 7 neighborhood DT and the EDT (see the text).

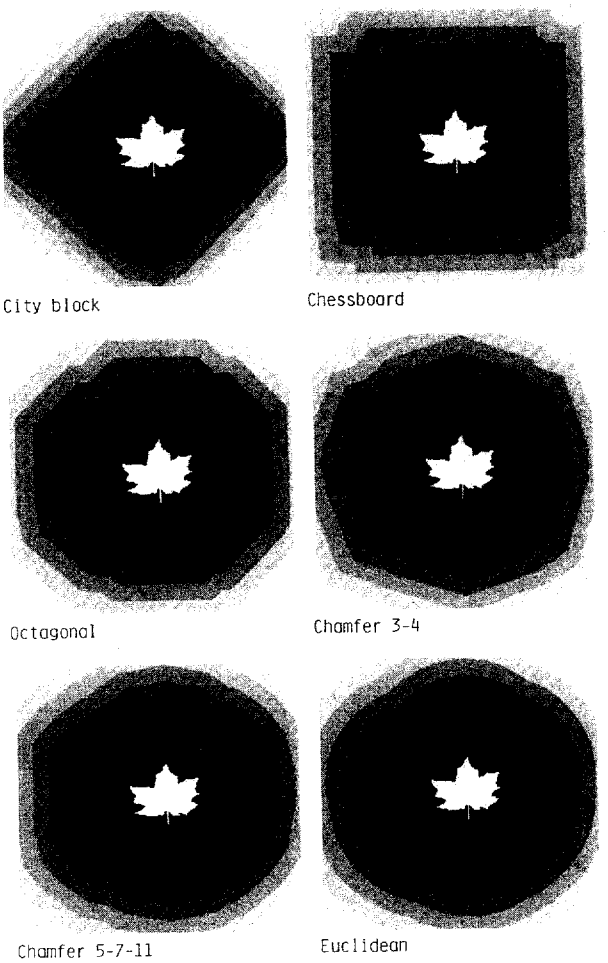

FIG. 21. The distances from an object for the six DTs. The lighter the color the larger the distance.

For 5×5 neighborhoods two difference images have been computed, Fig. 19. The first is the optimal case, $a = 1$, $b = \sqrt{2}$, and $c = c_{opt}$, (26); and the second is the 5–7–11 integer approximation. The images are all but indistinguishable, but if you look closely, there is a discernible difference along the diagonals. The maximal absolute differences are ± 1.96 and $+2.02$.

Finally the difference image for the optimal 7×7 neighborhood DT have been computed, Fig. 20. The local distances are those where e has been excluded, (58), with b and c about in the middle of their allowed intervals. The maximal absolute difference is only ± 0.91, and thus the image is very dark. (The grey-levels are the same as in Fig. 19.)

4.3. Distance from an Object

In some applications the distance from an object, or object contour, must be computed. One such case is when the distance values are used to compute a matching measure, i.e., a value that measures how close to each other the shape of two different objects or contours are, [10]. The matching algorithm presented there performed better the better the DT approximated the EDT. (The city block, chamfer 3–4, and Euclidean DTs were tested.)

As an illustration the distance from a maple leaf have been computed for the six DTs, Fig. 21. The distance is grey-level coded as before: the larger the distance, the lighter the color. The size of the images are 200×200 pixels. It is very obvious that even when the distance is computed from a complex shape, the "flavor" of the distance is preserved. Note also how close the chamfer 5–7–11 DT is to the EDT.

4.4. Pseudo-Dirichlet Tessellations

Consider a set of points in the plane. Then divide the plane into polygonal areas such that each area contains one point, and the part of the plane that is nearer to that point than to any other. Such a division of the plane is called a Dirichlet, or Voronoi, tessellation. The polygonal areas are called tiles. An example of such a tessellation is found in Fig. 22. Dirichlet tessellations in the image processing context are described in [11].

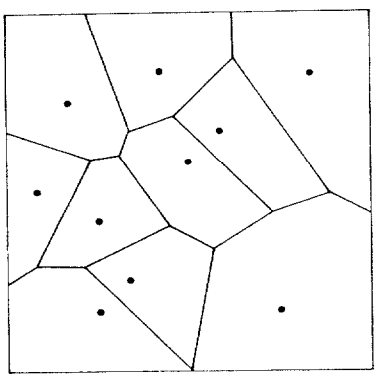

FIG. 22. A Dirichlet tessellation with ten kernel points. Each tile consists of the area that is closer to its kernel point than to any other point.

There are several uses of Dirichlet tessellations in image processing. In [11] they are mostly used in the context of texture descriptions. Another application is when objects in an image must be ordered in such a way that neighbors in the image are close to each other in the list, [12]. However, in neither application the fact that the image is actually digital rather than continuous is taken into account. The line dividing the tiles are computed analytically (see, e.g., [13]).

In a digital image there is another way of computing the Dirichlet tessellation: Compute a DT from all the kernel points of the tessellation, while at the same time keeping track of from which point the distance is computed. The computation must be done in parallel, (1).

First create a new image of the same size as the original one, and mark each kernel point pixel with a unique number identifying it. At each iteration of the DT, the pixels in the new image that corresponds to the pixels that get new values in the

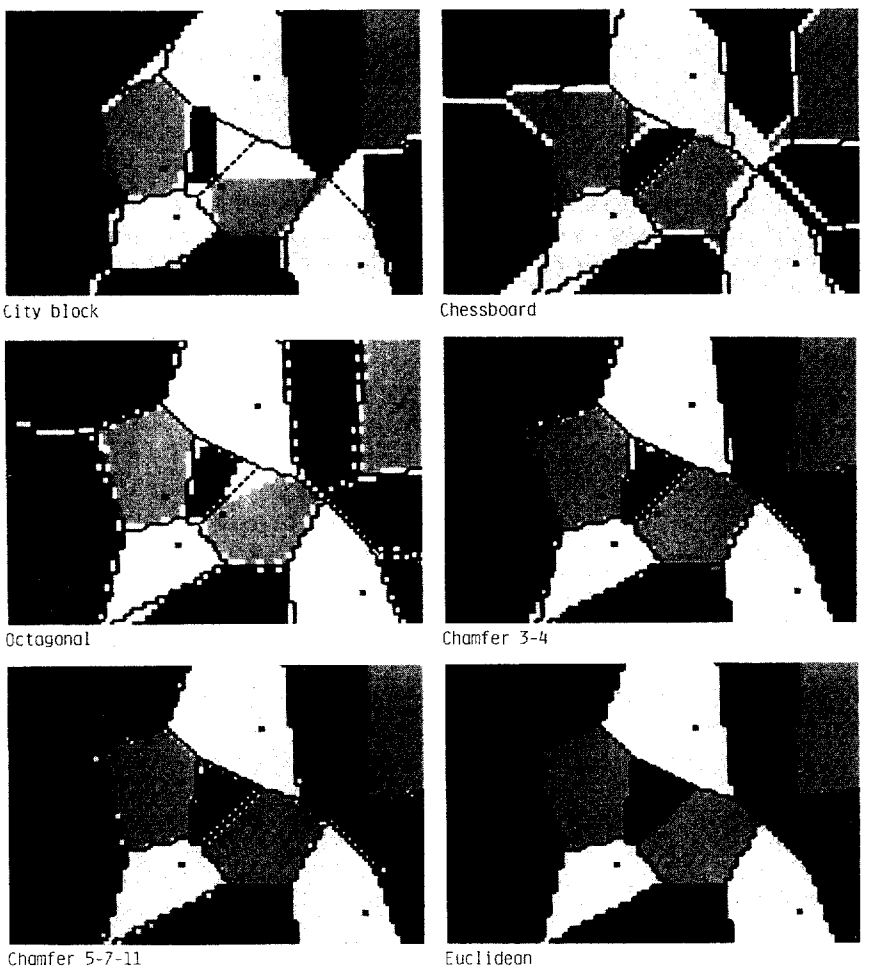

FIG. 23. Pseudo-Dirichlet tessellations computed for the same set of points, using different DTs.

distance image are marked with the number of the kernel point from which the distance is computed. Note that this number may in some cases change at the next iteration! When the algorithm stops, all pixels will have been identified as belonging to a certain tile, except those that have the same distance to more than one kernel point. The resulting tessellation is here called a pseudo-Dirichlet tessellation, as the tessellation is digital and the result depends on the DT used.

The pseudo-Dirichlet tessellation have been computed for 12 kernel points in a 50×70 pixel image. The results are shown in Fig. 23. Each tile is identified by a certain grey-level. The kernel points are black. The pixels that do not belong to a unique tile are white. The outlines of the tiles for the correct Euclidean tessellation, i.e., the digital EDT, are overlayed in black.

The main difficulty with the city block DT is that large areas are not assigned to any tile. Note the two large white areas center right in the image. For the chessboard DT many of the pixels get assigned to the wrong tile. Note, e.g., the long "arm" of the dark grey tile in the lower right corner. The results of the octagonal DT are much better, even though many pixels are unassigned. For the chamfer 3–4 DT the result is almost correct, even if a few pixels between the tiles are unassigned, and a few are assigned to the wrong tile. Finally for the chamfer 5–7–11 DT the results are almost correct, i.e., almost equal to the results for the EDT.

5. CONCLUSIONS AND RECOMMENDATIONS

A digital distance transformation converts a binary image to a distance image. A distance transformation that gives the correct Euclidean distances have here been called EDT. Suggested EDT algorithms are too computationally complex to be really attractive (Sect. 4.1). Thus distance transformation algorithms that use only a small image neighborhood and work within the image itself, i.e., do not need any extra memory, are needed. They have here been called DTs. Several DTs have been suggested in the literature (Sect. 4.1). In [1] the 3×3 neighborhood DTs were optimized and analyzed. This paper gives some new results for these DTs and, more importantly, extends the results to larger neighborhoods.

A number of DTs are now available. However, the problem of choosing the best distance transformation seems seldom to get much consideration. As the examples in Section 4 show, this is probably a mistake. The DT used may influence the results of the application to a large degree, and not in an advantageous way. Therefore some thought should be spent on the choice. Note that any of the six DTs in Section 4 may be the right choice, depending on the application.

In some digital image processing applications, where a distance transformation is needed, it is necessary to use the correct Euclidean distance. In other cases the distance should be as close to EDT as possible, but need not be quite exact. Then the new chamfer 5–7–11 DT is an ideal choice. If the accuracy needed is somewhat less, then the chamfer 3–4 DT, which is less computationally complex, may be the best choice. One common case when the distances need not be quite exact is when the images to which the DT is applied are somewhat noisy. Computing exact distances from inexact features is not reasonable, at least not when the exact distances are more computationally costly than adequate approximations.

If the DT computations can be performed in parallel, then the octagonal DT may be good enough; (the sequential algorithm uses four passes over the image and is thus too computationally complex compared to its resulting accuracy). The octago-

nal DT has the advantage of only using the local distance one, and also that the distance values are not multiplied by any scale factor. The chessboard DT can be advantageous for images consisting mainly of rectangles with the sides parallel to the coordinate axis, e.g., house or street scenes. Finally the city block distance is the fastest to compute, and where speed is essential rather than accuracy it may be the best choice.

The two most important results in this paper are thus:

*The new chamfer 5–7–11 distance transformation, illustrated in Fig. 14.

*The insight that for all applications using a distance transformation some effort should be spent on choosing the "right" one. The results may differ considerably for different distance transformations.

REFERENCES

1. G. Borgefors, Distance transformations in arbitrary dimensions, *Comput. Vision, Graphics Image Process.* **27**, 1984, 321–345.
2. A. Rosenfeld and J. Pfaltz, Sequential operations in digital picture processing, *J. Assoc. Comput. Mach.* **13**, 1966, 471–494.
3. A. Rosenfeld and J. Pfaltz, Distance functions on digital pictures, *Pattern Recognit.* **1**, No. 1, 1968, 33–61.
4. G. Borgefors, Chamfering: A fast method for obtaining approximations of the Euclidean distance in N dimensions, in *3rd Scand. Conf. on Image Analysis*, Copenhagen, Denmark, 1983, pp. 250–255.
5. U. Montanari, A method for obtaining skeletons using a quasi-Euclidean distance, *J. Assoc. Comput. Mach.* **15**, 1968, 600–624.
6. A. Rosenfeld and A. C. Kak, *Digital Picture Processing*, 2nd ed., Vol. 2, Chap. 11, Academic Press, New York, 1982.
7. H. G. Barrow, J. M. Tenenbaum, R. C. Bolles, and H. C. Wolf, Parametric correspondence and chamfer matching: Two new techniques for image matching, in *Proc. 5th Int. Joint Conf. on Artif. Intell.*, Cambridge, Mass., 1977, pp. 659–663.
8. P. E. Danielsson, Euclidean distance mapping, *Comput. Graphics Image Process.* **14**, 1980, 227–248.
9. H. Yamada, Complete Euclidean distance transformation by parallel operation, in *Proc. 7th Int. Conf. on Pattern Recognit.* Montreal, Canada, 1984, pp. 69–71.
10. G. Borgefors, An improved version of the chamfer matching algorithm, in *Proc. 7th Int. Conf. on Pattern Recognit.*, Montreal, Canada, 1984, pp. 1175–1177.
11. N. Ahuja and B. J. Schachter, *Pattern Models*, Chap. 1, Wiley, New York, 1983.
12. E. Jungert and J. R. Eriksson, *Organizing Geographical Objects Derived from a Digitized Map*, National Defence Research Institute Report D 30388, 1985; submitted for publication.
13. P. J. Green and R. Sibson, Computing Dirichlet tessellations in the plane, *Computer J.* **21**, No. 2, 1978, 168–173.