COMPUTER GRAPHICS AND IMAGE PROCESSING 14, 227–248 (1980)

Euclidean Distance Mapping

PER-ERIK DANIELSSON

Department of Electrical Engineering, Linköping University, Linköping S-581 83, Sweden

Received June 5, 1979; revised September 28, 1979; accepted February 6, 1980

Based on a two-component descriptor, a distance label for each point, it is shown that Euclidean distance maps can be generated by effective sequential algorithms. The map indicates, for each pixel in the objects (or the background) of the originally binary picture, the shortest distance to the nearest pixel in the background (or the objects). A map with negligible errors can be produced in two picture scans which has to include forward and backward movement for each line. Thus, for expanding/shrinking purposes it may compete very successfully with iterative parallel propagation in the binary picture itself. It is shown that skeletons can be produced by simple procedures and since these are based on Euclidean distances it is assumed that they are superior to skeletons based on d_4-, d_8-, and even octagonal metrics.

1. INTRODUCTION

Distance mapping is frequently used in picture processing. Usually, it is based on one of the metrics

$$d_4((i,j),(h,k)) = |i-h| + |j-k|,$$

which is called the "city block distance," or

$$d_8((i,j),(h,k)) = \max(|i-h|,|j-k|),$$

which is called the "chessboard distance," or a combination of these, e.g., the so-called octagonal distance. Obviously, we assume that i,j,h,k are integers in the two-dimensional rectangular space.

Given a binary image with two set of pixels,

$$S = \text{set of 1's, the objects}$$

$$\bar{S} = \text{set of 0's, the background}$$

a *distance map* $L(S)$ is an image such that for each pixel

$$(i,j) \in S$$

there is a corresponding pixel in $L(S)$ where

$$L(i,j) = \min\left(d\left[(i,j),\bar{S}\right]\right),$$

i.e., each pixel in S has been assigned a *label* in $L(S)$ that amounts to the distance to the nearest background $\bar{S}$. Obviously we can define a similar map $L(\bar{S})$ for the background.

0146-664X/80/110227-22$02.00/0

The computational procedure

$$L: S \to L(S)$$

is called *distance mapping*.

Distance maps can be used for several purposes [1, 3]. Among these are expansion/shrinking of the objects S (by thresholding $L(\bar{S})/L(S)$), construction of shortest paths between points, skeletonizing and shape factor computation [6].

Two principally different methods can be used for distance mapping: sequential (recursive) algorithms or parallel algorithms. In *sequential* algorithms the picture is scanned and $L(i,j)$ is computed using recently computed values from the *present* scan in the neighborhood, e.g., $L(i-1,j)$ and $L(i,j-1)$. The net effect is that distance labels may propagate across the entire picture in one scan. Such algorithms were first presented by Rosenfeld and Pfaltz [2] for the d_4-metric.

In *parallel* algorithms a sequence of distance maps are generated.

$$S \to L_1(S) = L_1 \to L_2(L_1) = L_2 \to \ldots \to L_n(L_{n-1}) = L_n = L(S),$$

where each computed label $L_k(i,j)$ in the map L_k depends on a neighborhood, e.g.,

$$L_{k-1}(i,j), \qquad i,j \leq 1,$$

in the *previous* map L_{k-1} only. Thus, for each iteration from L_{k-1} to L_k, labels can only propagate at a distance that equals the size of the neighborhood. This fact makes parallel distance mapping rather slow compared to sequential algorithms unless physically parallel logic is employed for each pixel.

Distance maps based on the metrics d_4 and d_8 deviate quite substantially from the ideal *Euclidean metric*

$$d_e((i,j),(h,k)) = \sqrt{(j-i)^2 + (k-h)^2}\ .$$

In fact, the errors are so large that d_4 and d_8 are rarely used. Instead, their combination, "the octagon," is employed. A distance map

$$L_{oct}(\bar{S}),$$

where S is a single point is shown in Fig. 1.

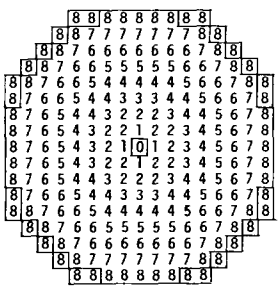


FIG. 1. Octagonal expansion.

The best approximation of a circular disc with radius 8 is seen to be somewhat smaller than the object covered by the octagon expanded area up to distance 8. Relative errors above 10% are abundant in octagonal maps and absolute errors far above the pixel units occur for larger distances.

The inaccuracy (even in the octagon case) of distance mapping has recently led a few authors to take an interest in the problem of "circular propagation" of binary images [7–9]. This is essentially equivalent to the problem of Euclidean distance mapping since the goal is to expand the objects a certain Euclidean distance (or the "correct" approximation thereof) in all directions. Correct circular propagation in binary images seem to be extremely time consuming. Probably, a better choice is octagonal propagation interlaced with a "rounding" operation [9, 10]. This method produces a rather good circular approximation for a single point expansion even at larger radii. However, some minor unpredictable results occur when many-point objects are expanded.

Montanari [4] has investigated a type of quasi-Euclidean distance mapping. The distance between two points is defined as the length of the shortest chain-coded path and each step of the path can, in the simplest case (order 1), be selected from the 4 possible steps in the d_4 neighborhood. In this case the distance map is equivalent to a d_4-map. The quasi-Euclidean map of order 2 selects the steps from the 8 possible cases in the d_8-neighborhood. The resulting map is not equivalent to a d_8-map, however, since each step contributes with its true Euclidean distance. The quasi-Euclidean distance map of order 3 select the steps from a 5×5 neighborhood which gives 24 possibilities, etc. It is proved that a shortest path between two points always consists of no more than two different types of steps, namely those having slopes that are closest to the slope of the Euclidean vector between the points. The distance maps can be produced by sequential algorithms in two raster scans. The map of order 2 gives relative errors of 8% in the 22.5° directions which makes it comparable in accuracy to octagonal maps. So, to get a improved accuracy one has to use rather large neighborhoods. Even then, the skeletons seem to suffer from the deviations from the exact Euclidean metric.

Barrow *et al.* [11] uses a distance mapping which is essentially the same as the one of Montanari of order 2 although the relative lengths of the steps $\sqrt{2} : 1$ are approximated to 3 : 2. A sequential algorithm for this case is given in [11].

Algorithms for true Euclidean distance mapping are not to be found in the literature. The advantages of such maps are quite obvious, however, especially if they can be computed efficiently.

2. FOUR-POINT SEQUENTIAL EUCLIDEAN DISTANCE MAPPING

The picture L is a two-dimensional array with the elements

$$L(i,j) \qquad 0 \le i \le M-1, 0 \le j \le N-1.$$

Each element is a *two-element vector*

$$\bar{L} = (L_i, L_j), \qquad L_i, L_j \text{ being positive integers,}$$

$$\bar{L}(i,j) = \big(L_i(i,j), L_j(i,j)\big).$$

The size of a vector $L(i,j)$ is defined by

$$|L(i,j)| = \sqrt{L_i^2 + L_j^2}\ .$$

The four-point sequential Euclidean distance mapping algorithm 4SED is defined as follows.

4SED

Initially:

$$\bar{L}(i,j) = (0,0) \qquad \text{if } (i,j) \in S,$$
$$\bar{L}(i,j) = (Z,Z) \qquad \text{if } (i,j) \in \bar{S},$$

where Z is the largest integer that can be stored without inconvenience and where S is the object and $\bar{S}$ is the background in the original $M \times N$ binary picture.

First picture scan: For $j = 1, 2, \ldots, N-1$,

$$\text{for } i = 0, 1, 2, \ldots, M-1,$$
$$\bar{L}(i,j) = \min\begin{cases} \bar{L}(i,j) \\ \bar{L}(i,j-1) + (0,1); \end{cases}$$

$$\text{for } i = 1, 2, \ldots, M-1,$$
$$\bar{L}(i,j) = \min\begin{cases} \bar{L}(i,j) \\ \bar{L}(i-1,j) + (1,0); \end{cases}$$

$$\text{for } i = M-2, M-3, \ldots, 1, 0,$$
$$\bar{L}(i,j) = \min\begin{cases} \bar{L}(i,j) \\ \bar{L}(i+1,j) + (1,0). \end{cases}$$

Second picture scan: For $j = N-2, N-3, \ldots, 1, 0$,

$$\text{for } i = 0, 1, 2, \ldots, M-1,$$
$$L(i,j) = \min\begin{cases} \bar{L}(i,j) \\ L(i,j+1) + (0,1); \end{cases}$$

$$\text{for } i = 1, 2, \ldots, M-1,$$
$$\bar{L}(i,j) = \min\begin{cases} \bar{L}(i,j) \\ \bar{L}(i-1,j) + (1,0); \end{cases}$$

$$\text{for } i = M-2, M-3, \ldots, 1, 0,$$
$$\bar{L}(i,j) = \min\begin{cases} \bar{L}(i,j) \\ \bar{L}(i+1,j) + (1,0). \end{cases}$$

The algorithm first assigns a maximum label to all background pixels. The real assignments take place in two scannings of the complete picture where each picture

74	64	54	44	34	24	14	23	13	03	13	23	33	43	53	63
73	63	53	43	33	23	13	22	12	02	12	22	32	42	52	62
64	62	52	42	32	22	12	02	11	01	11	21	31	41	51	61
63	52	43	41	31	21	11	01	10	[00]	10	20	30	40	50	60
62	52	42	32	22	20	10	[00]	10	20	30	40	50	60	70	80
61	51	41	31	21	11	01	11	21	31	41	51	61	71	81	91
60	50	40	30	20	10	[00]	10	20	30	40	50	60	70	80	90
ZZ	ZZ	ZZ	ZZ	ZZ	ZZ	ZZ	ZZ	ZZ	ZZ	ZZ	ZZ	ZZ	ZZ	ZZ	ZZ
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

FIG. 2. After first picture scan.

scan involves for each line j first a one-step propagation in the j-direction, then a sequential propagation in *both* i-directions. In total, each label $\bar{L}(i,j)$ is involved in six comparisons.

An illustration of the behavior of the algorithm is shown in Figs. 2 and 3 which show the labels after the first and second scan, respectively, for a simple picture that consists of three object points, labeled 00 (the commas between the two vector components have been dropped to avoid typographical overcrowding). Labels ZZ are left blank.

During the process, new labels propagate from "sources of change" (which are the three original object pixels) in all directions. Each picture scan provides a 180° propagation angle thanks to the fact that for each vertical step, the new line is scanned in both directions. The propagated "influence" from one object pixel stops when it encounters a propagation wave from another object pixel at a closer distance. Thus, *watersheds* can be observed in the picture. (The ideal watersheds are drawn as heavy lines in Fig. 3.) The distance map can be considered as correct as long as each distance label has a size that amounts to the distance from the center of the pixel to the object pixel in the watershed area of which the pixel is found.

In the (deliberately revealing) example of Fig. 3, we may first note that to the lower right there are diagonally adjacent pixels which happen to lie exactly on the watershed line. Without further ado we have labeled them with 41 (instead of 14), 52 (instead of 25) etc., following the rule that between two labels of equal size we choose the one with largest first component L_i.

A more important fact is that the pixel labeled 30, lying in the very head of the arrow-shaped watershed area, ought to have been labeled 22. Here we have struck on a deficiency of the algorithm which will be analyzed in the following section.

74	64	54	44	34	24	14	23	13	03	13	23	33	43	53	63
73	63	53	43	33	23	13	22	12	02	12	22	32	42	52	62
64	62	52	42	32	22	12	02	11	01	11	21	31	41	51	61
63	53	43	41	31	21	11	01	10	[00]	10	20	30	40	50	60
62	52	42	32	22	20	10	[00]	10	01	11	21	31	41	51	61
61	51	41	31	21	11	01	01	11	02	12	22	32	42	52	62
60	50	40	30	20	10	[00]	10	20	30	13	23	33	43	53	63
61	51	41	31	21	11	01	11	21	31	41	24	34	44	54	64
62	52	42	32	22	12	02	12	22	32	42	52	35	45	55	65
63	53	43	33	23	13	03	13	23	33	43	53	62	46	56	66

FIG. 3. After second scan.

3. ERROR ANALYSIS. THE EIGHT-POINT ALGORITHM

The 4SED-algorithm allows for propagation in all four quadrants but only the four closest neighbors compete in the process of assigning a new label to a given pixel. Since these four neighbors form a nonclosed lattice it is possible that the center pixel ought to be labeled from an object pixel different from any of the object pixels that are nearest to the four neighbors. See Fig. 4.

Here we have three object pixels, A, B and C, for which the (ideal) watersheds are drawn as heavy lines. The question is whether all pixels with center-point in the watershed area of C will be labeled from C. As can be seen, while C is more distant than A and B from the two neighbors to the left and below of P, respectively, P itself, lying in the head of the watershed arrow is closer to C and will not be correctly labeled. Thus, watershed corners in the form of arrowheads constitute an error risk.

It is readily understood that such a risk will be relaxed if the lattice is made more dense, for instance by letting the eight nearest neighbors participate in the propagation processes. Also, since the six neighbors in a hexagonal digitization pattern are more dense then the four adjacent pixels in the Cartesian grid, the errors will be less severe in such a case. On the other hand, since we can never close the box around a pixel completely there is always a (small) chance for the ideal watershed configuration to "sneak into the box and snatch the center pixel."

It is also understood that for the 4SED-algorithm the worst case is the one depicted in Fig. 4 where the watershed arrow penetrates into the box along a 45° line. The maximum error in the distance labeling is calculated as follows. The two neighboring pixels to P are at distance r from pixels A and B, respectively. Thus, P will be assigned a label equivalent to a distance $r + 1$ (the pixel unit distance is set to 1).

The correct distance (from C) is d and the error is

$$e = r + 1 - d.$$

Maximum error occurs when d is as small as possible (without causing the watersheds to expand over the two neighbor pixels) which means that C is at distance $r + \delta = r$ from the neighbor pixels. Simple geometry reveals that

$$r^2 = \left(\frac{d}{\sqrt{2}}\right)^2 + \left(\frac{d}{\sqrt{2}} - 1\right)^2,$$

$$r^2 = d^2 - d\sqrt{2} + 1 = \left(d - \frac{1}{\sqrt{2}}\right)^2 + \frac{1}{2},$$

$$r = \sqrt{\left(d - \frac{1}{\sqrt{2}}\right)^2 + \frac{1}{2}}, \qquad d = \frac{1}{\sqrt{2}} + \sqrt{r^2 - \frac{1}{2}},$$

$$r \approx d - \frac{1}{\sqrt{2}} \qquad \text{for larger } r, d.$$

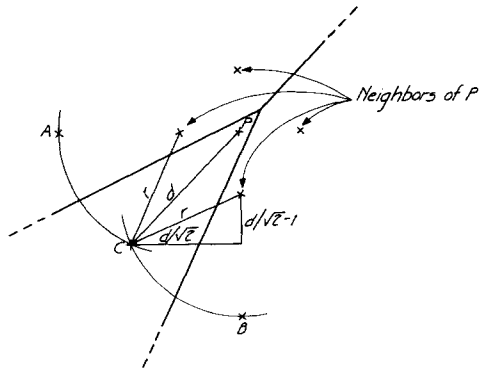


FIG. 4. Worst-case error for 4SED.

The error then becomes

$$e = r + 1 - \sqrt{r^2 - \frac{1}{2}} - \frac{1}{\sqrt{2}},$$

which simplifies to

$$e = r + 1 - r - \frac{1}{\sqrt{2}} = 1 - \frac{1}{\sqrt{2}} = 0.29 \text{ pixel units for larger } r.$$

e_{max} occurs (in principle) for $r = 1$

$$e_{max} = 1 + 1 - \sqrt{\frac{1}{2}} - \frac{1}{\sqrt{2}} = 2 - \sqrt{2} = 0.58 \text{ pixel units.}$$

Because of the limited raster positions actually available for point C, e_{max} will never occur. By trial and error the author has complied the following list of actual errors for various r:

$r = 2$	$e = 3 - \sqrt{8} = 0.172$	or	6.1%
$r = \sqrt{4^2 + 1^2}$	$e = \sqrt{26} - 5 = 0.100$	or	2.0%
$r = 5$	$e = 6 - \sqrt{34} = 0.169$	or	2.9%
$r = 7$	$e = 8 - \sqrt{61} = 0.190$	or	2.4%
$r = 10$	$e = 11 - \sqrt{117} = 0.184$	or	1.7%
$r = 12$	$e = 13 - \sqrt{162} = 0.262$	or	2.05%

The conclusion is that the actual error is in fact bounded by $1 - 1/\sqrt{2} = 0.29$, which is the theoretical distance error for larger r and d if the point C could be positioned anywhere.

Thus, the 4SED algorithm produces a distance map which is error-free except for very sparsely scattered pixels which may be assigned a distance label with an *absolute error* less than 0.29 pixel units. Note that, in contrast to (for instance) octagonal maps, the *relative* error quickly approaches zero for larger distances.

The eight-point sequential Euclidean distance mapping 8SED is defined as follows.

8SED

Initially:

$$L(i,j) = (0,0) \text{ for } (i,j) \in S$$
$$L(i,j) = (Z,Z) \text{ for } (i,j) \in \bar{S}$$

First picture scan: For $j = 1, 2, \ldots, N-1$,

$$\text{for } i = 0, 1, 2, \ldots, M-1,$$
$$\bar{L}(i,j) = \min\begin{cases} \bar{L}(i,j) \\ \bar{L}(i-1,j-1) + (1,1) \\ \bar{L}(i,j-1) + (0,1) \\ \bar{L}(i+1,j-1) + (1,1); \end{cases}$$

$$\text{for } i = 1, 2, \ldots, M-1,$$
$$\bar{L}(i,j) = \min\begin{cases} \bar{L}(i,j) \\ \bar{L}(ib\text{-}1,j) + (1,0); \end{cases}$$

$$\text{for } i = M-2, M-3, \ldots, 1, 0,$$
$$\bar{L}(i,j) = \min\begin{cases} \bar{L}(i,j) \\ \bar{L}(i+1,j) + (1,0). \end{cases}$$

Second picture scan: For $j = N-2, N-3, \ldots, 1, 0$,

$$\text{for } i = 0, 1, 2, \ldots, M-1,$$
$$\bar{L}(i,j) = \min\begin{cases} \bar{L}(i,j) \\ \bar{L}(i-1,j+1) + (1,1) \\ \bar{L}(i,j+1) + (0,1) \\ \bar{L}(i+1,j+1) + (1,1); \end{cases}$$

$$\text{for } i = 1, 2, \ldots, M-1,$$
$$\bar{L}(i,j) = \min\begin{cases} \bar{L}(i,j) \\ \bar{L}(i-1,j) + (1,0); \end{cases}$$

$$\text{for } i = M-2, M-1, \ldots, 1, 0,$$
$$\bar{L}(i,j) = \min\begin{cases} \bar{L}(i,j) \\ \bar{L}(i+1,j) + (1,0). \end{cases}$$

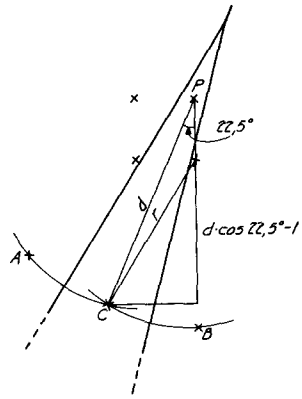


FIG. 5. Worst-case error for 8SED.

In accordance with Fig. 4 we can now draw a worst-case watershed configuration for the eight-point case. See Fig. 5. The label for the pixel at P will incorrectly equal $r + 1$ (the distance from B) and the error e is

$$e = r + 1 - d.$$

Simple geometry gives us

$$r^2 = (d \cdot \cos 22.5° - 1)^2 + (d \cdot \sin 22.5°)^2,$$

$$d = \cos 22.5° + \sqrt{r^2 - 1 + \cos 22.5°},$$

which becomes

$$d = r + \cos 22.5° \qquad \text{for larger } r$$

so that

$$e = 1 - \cos 22.5° = 0.076.$$

For $r = 1$ the error is twice as large but due to the constraints in available raster points such cases will never occur.

In summary, the 8SED-algorithm uses 10 comparisons/pixel instead of 6 for the 4SED-algorithm. The errors will become even more sparsely distributed and each error is bounded to be less than 0.076 instead of 0.29 for the 4SED-algorithm.[1]

Extensive search for errors in the range $1 \leq r \leq 20$ (approx. 360 different cases) has resulted in finding only five erroneous cases. In all these d_{assigned} and d_{correct} can be expressed by the formula

$$d_{\text{assigned}} = \sqrt{d_{\text{correct}}^2 + 1}.$$

[1]The reviewer of this paper has pointed out that the worst-case direction for the watershed arrow is 24.4698... degrees rather than 22.5. So, the correct upper bound for 8SED errors is 0.090 pixel units rather than 0.076.

The largest error in this set of five was for

$$d_{\text{assigned}} = \sqrt{338} = \sqrt{13^2 + 13^2},$$

$$d_{\text{correct}} = \sqrt{337} = \sqrt{16^2 + 9^2}$$

resulting in an absolute error of

$$e = \frac{1}{2\sqrt{338}} = \frac{1}{26\sqrt{2}} \approx 0.027 \text{ pixel units}$$

and a relative error of

$$e = 0.15\%,$$

which is negligible for all practical purposes.

4. EUCLIDEAN DISKS AND SKELETONS

Using the notation of [1], a disk $D_t(i,j)$ is the set of all pixels that are at distance $\leq t$ from (i,j). In what follows we will make a minor change in this definition and define

$$D_t(i,j) = \{(i + h, j + k) | h^2 + k^2 < t^2\},$$

i.e., we do not include the pixels that are exactly at distance t. Let us now use the vectorial distance label notation used previously so that

$$D_{ab} = D_{\sqrt{a^2+b^2}} = D_{ba}.$$

Because of the minor but still existing errors in the distance mappings the 4SED-map of some disks will be slightly different from the (almost error-free) 8SED-maps. The disks from D_{01} to D_{60} mapped by 4SED and 8SED are found in Figs. 6 and 7 respectively.

The smallest disk is D_{01} followed by D_{11}, D_{20}, etc. Only one quadrant is shown, It is worth noting that in contrast to the d_4-, d_8-, and octagonal metrics, Euclidean distances result in disks D_t for other cases than integer values of t. And except for a few cases (e.g., $(D_{51})_1$ mapped by 4SED in Fig. 6) they are all best approximations of circles, each one for its own radius. Thus, by using Euclidean distance maps it is perfectly meaningful to perform expansion of a binary image over noninteger distances.

The previously compiled list of errors for the 4SED-algorithm display themselves in Fig. 6 as ambiguities. For instance, a disk which, correctly mapped by 8SED, has a (2, 2)-label at its center will turn up with a (3, 0)-label when mapped by 4SED. On the other hand, the true disk D_{30} will also get the (3, 0)-label as its center pixel. If one should prefer the 4SED-disks (because of the faster calculations) these ambiguities must be resolved. One way out seems to be to avoid the use of all the disks of

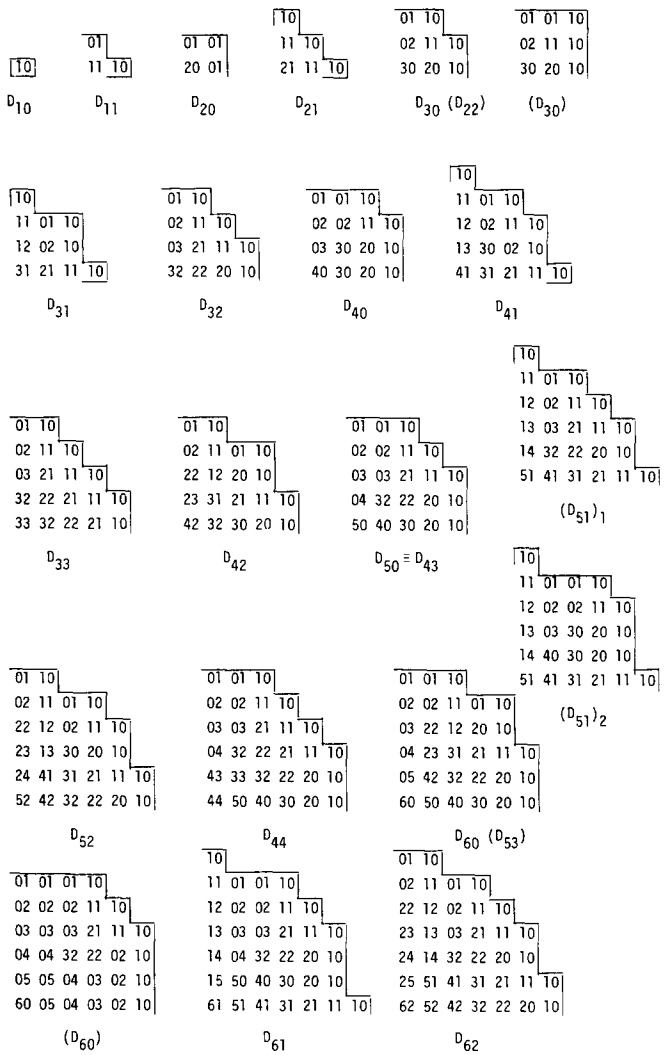


FIG. 6. Disks with 4SED-mapping.

Fig. 6 that are set between parentheses. At the same time, we declare the following corrected sizes for the labels in question:

$$|(3,0)| = |(2,2)| = \sqrt{8},$$
$$|(5,1)| = |(5,0)| = 5,$$
$$|(6,0)| = |(5,3)| = \sqrt{34},$$
$$|(8,0)| = |(6,5)| = \sqrt{61},$$
etc.

That is, we deliberately decrease the distances to the least possible true distance that may exist in a 4SED-map for these labels.

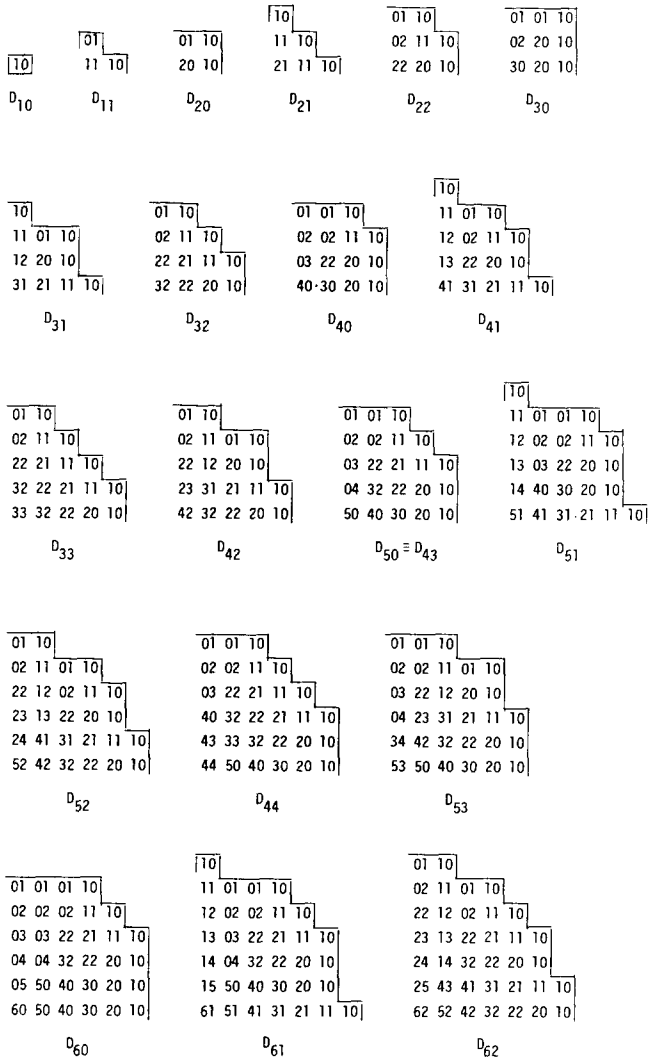


FIG. 7. Disks with 8SED-mapping.

In what follows we will deal with skeletons and the inversion of distance mapping, i.e., reconstruction of objects defined by disks. Although not shown here these procedures could use 4SED-maps with the ambiguity-suppressing constraints just discussed. However, to make the situation more comprehensive we will in the rest of this section assume that all distance maps are produced with the (almost) error-free 8SED-algorithm.

The skeleton S^* of an object can be defined as the set of pixels (i,j) at which the distance map has local maxima. Another way to define the skeleton is to say that it consists of those points (i,j) for which maximum-sized disks

$$D_t(I,j) \in S$$

can be found that are not completely covered by any other (larger) disk [1, 4].

This skeleton is nonredundant in the weaker sense that it includes no disk that is completely covered by another single disk. Note, however, that the skeleton *is* redundant in the stronger sense that a union of two or more disks may cover another disk.

With the help of the disk shapes in Fig. 7, the 8SED-mapped object of Fig. 8 can now be skeletonized, first by intuitive trial-and-error. We start with the largest $|L(i,j)|$ which in this case is 05. Evidently, the three

$$D_{50} \in S$$

and since there are no larger disks these three pixels belong to S^*. From Fig. 7 we can infer that a D_{50} disk has the label 50 in the center and following neighbors:

32 04 32
40 50 40
32 04 32

Now, there are two 40-labels along the diagonals in the vicinity of the leftmost 50-label and since

$$4^2 + 0^2 \neq 3^2 + 2^2$$

these labels indicate that they are centers of D_{40}'s which are not covered by the closest D_{50} and consequently these pixels belong to S^*. Then, around the center of a D_{40} we expect to find the neighbors

22 03 22
30 40 30
22 03 22

But the two ringed 40's have one 30-label each along the diagonals which then also belong to the skeleton, etc.

We can now imagine how the total skeleton S^* can be locally detected by a local 3×3-operator in one picture scan. The skeleton algorithm consists of the following *skeleton operator* SKED for Euclidean distance maps.

10	01	01	01	01	01	01	01	01	01	01	01	01
10	20	02	02	02	02	02	02	02	02	(02)	(20)	10
10	20	(30)	03	03	03	03	03	(03)	(22)	(21)	(20)	10
10	20	30	(40)	04	04	04	(04)	(31)	21	11	01	10
10	20	30	40	(50)	(05)	(50)	40	30	20	10		
10	20	30	(40)	04	04	04	(04)	(31)	21	11	10	
10	20	(30)	03	03	03	03	03	(03)	(22)	20	10	
10	20	02	02	02	02	02	02	02	02	(20)	10	
10	01	01	01	01	01	01	01	01	01	01	10	

FIG. 8. A Euclidean skeleton.

SKED

If and only if for all $h, k \leq 1$, i.e., for all eight neighbors

$$|\bar{L}(i,j)| \neq |\bar{g}_{i-h,j-k}(\bar{L}(i+h,j+k))|$$

then $\bar{L}(i,j)$ is a local maximum and $(i,j) \in S^*$. The function $g_{i-h,j-k}$ is a mapping that translates the label $\bar{L}(i+h,j+k)$ into the label $\bar{L}(i,j)$, should i,j belong to a disk centered at $i+h, j+h$. The functions $g_{i-h,j-k}$ are of two types only, g_8 for the diagonal neighbors and g_4 for the four nearest neighbors.

From the previously discussed D_{40} and D_{50} disks we can see that, e.g.,

$$\bar{g}_8(50) = 32, \quad \bar{g}_4(50) = 40,$$

$$\bar{g}_8(40) = 22, \quad \bar{g}_4(40) = 30.$$

With reasonably short labels g_8 and g_4 can be stored in a look-up table. Furthermore, instead of computing absolute distances from the labels we could simply compare the label $\bar{L}(i,j)$ directly (twice, with the two components shifted) with the output from the g_8- and g_4-tables.

The skeleton S^*, being the set of centers for disks that exactly cover S, is a compact description of the binary object (or objects) S. As has been demonstrated

$$S^* = \text{SKED}(\text{SED}(S))$$

The inverse of this operation, that is to generate SED(S) from S^*, will be called SKED^{-1} and is defined as follows, using a propagation scheme identical to the 8SED-algorithm.

$SKED^{-1}$

Initialize:

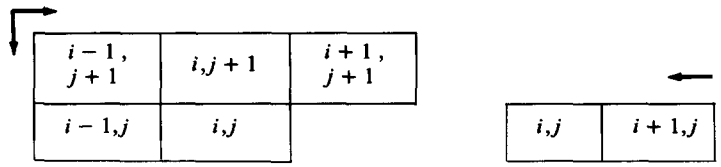
$$L(i,j) = \begin{cases} 00 & \text{if } (i,j) \notin S^* \\ \text{SED}(S) & \text{for } (i,j) \in S^*. \end{cases}$$

First scan:

$$\bar{L}(i,j) = \max \begin{cases} \bar{L}(i,j) \\ \bar{g}_4(\bar{L}(i-1,j)) \\ \bar{g}_4(\bar{L}(i,j+1)) \\ \bar{g}_8(\bar{L}(i-1,j+1)) \\ \bar{g}_8(\bar{L}(i+1,j+1)), \end{cases}$$

$$\bar{L}(i,j) = \max \begin{cases} \bar{L}(i,j) \\ \bar{g}_4(\bar{L}(i+1,j)), \end{cases}$$

which can be symbolized by

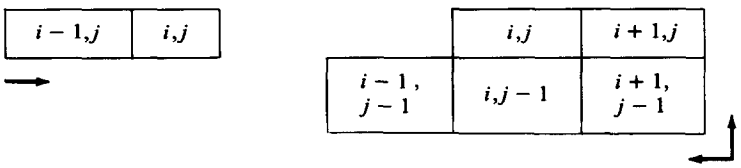


Second scan:

$$\bar{L}(i,j) = \max\begin{cases} \bar{L}(i,j) \\ \bar{g}_4(\bar{L}(i+1,j)) \\ \bar{g}_4(\bar{L}(i,j-1)) \\ \bar{g}_8(\bar{L}(i+1,j-1)) \\ \bar{g}_8(\bar{L}(i-1,j-1)), \end{cases}$$

$$\bar{L}(i,j) = \max\begin{cases} \bar{L}(i,j) \\ \bar{L}(\bar{g}_4(i-1,j)), \end{cases}$$

which can be symbolized by



We will now try to analyze and motivate the SKED- and SKED^{-1}-algorithms. First, the reader is urged to check that for instance the single skeleton point labeled 51 will be transformed to D_{51} by the SKED^{-1} operator. In what follows we claim that

$$\text{SKED}^{-1}(S^*) = 8\text{SED}(S).$$

See Fig. 9. One might suspect that some pixel $A_0 \in S$ may not be reached by the SKED^{-1}-propagation and may incorrectly be labeled (0, 0). However, $A_0 \notin S^*$ which means that $\bar{L}(A_0)$ has a neighbor $\bar{L}(A_1)$ that covers A_0 with its disk. If $A_1 \notin S^*$, then also $\bar{L}(A_1)$ has a neighboring pixel A_2 such that the disk of $\bar{L}(A_2)$ covers A_1, etc. until we reach a point $A_5 \in S^*$. There are several such paths from A_0 to A_5 and all of them will be recovered by the SKED^{-1}-algorithm. From 53 SKED^{-1} will reconstruct 42 diagonally for the very same reason (namely that $\bar{g}_8(53) = 42$) that made the SKED-operator drop this pixel from the skeleton. So, all points up to the boundary of the disk will be labeled with nonzero labels by the SKED^{-1}-algorithm.

The opposite effect that SKED^{-1} could propagate outside the disks of S^* is considered to be self-evidently not existent.

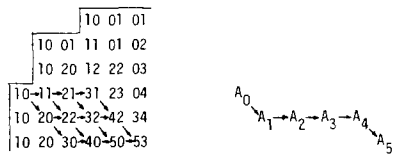


FIG. 9. Paths recovered by SKED^{-1} in reverse direction.

Let us summarize the discussion so far. The skeleton S^* = SKED(8SED(S)) defines a number of overlapping Euclidean disks and by applying the SKED^{-1}-algorithm we can reconstruct 8SED(S) without errors.

What remains to be discussed is whether S^* is redundant in the weaker sense that the disk of one skeleton point may completely cover the disk of another.

If this could occur we would find the same extra skeleton point by applying the SKED-operator to 8SED(S) where S is a Euclidean disk. As an example of a situation where such a false skeleton point may possibly occur consider the D_{44}-disk of Fig. 7. Evidently, the 8 pixels nearest to the center never run the risk of ending up in S^* since they are directly checked in the $g_4(44)$ and $g_8(44)$ look-up tables. Now, the 8SED-algorithm has produced 32-labels in the next layer. What guarantee do we have that this label is not of greater size than either $g_4(33)$ or $g_8(50)$?

To answer that we have to move to the continuous space again. See Fig. 10, where in Fig. 10a we see the worst case depicted without considering the fact that only a few discrete points are available as pixel centers. The geometry proves to be identical to the previous error analysis picture in Fig. 5. The center pixel 1 has a disk with a certain radius. Its nearest neighbors 2 and 3 are centers of somewhat smaller disks D_2 and D_3 that are completely covered by D_1 and are touching D_1 at the horizontal line-crossing and the 45°-line-crossing, respectively. The disk D_4 of point 4, finally, must not contain any point outside the union of D_2 and D_3 which forces D_4 to pass the crossing of D_2 and D_3 in the 22.5°-direction.

Now, assume that D_1 includes a pixel in the tiny area outside the union of D_3 and D_2. SKED^{-1} will reconstruct first D_2 and D_3 and then D_4. But the distance mapping 8SED applied to the binary object D_1 will produce a label $\bar{L}(4)$ corresponding to the distance between point 4 to a point in this "tiny area". Hence

$$\bar{L}_4 \neq g_4\big(\bar{L}(3)\big)$$

and

$$\bar{L}_4 \neq g_8\big(\bar{L}(2)\big)$$

and consequently, the SKED-operator will include point 4 in the skeleton S^*.

Because of the raster points available, this kind of error will be extremely rare. In fact, not one single case has yet been found.

Figure 10b is an illustrative case that motivates our optimism. We see that after disk D_{44} follows D_{50} which boldly (and correctly) extends *beyond* D_{44}. And the critical disk D_{32} is in fact absolutely as large as possible without covering pixels outside D_{44}.

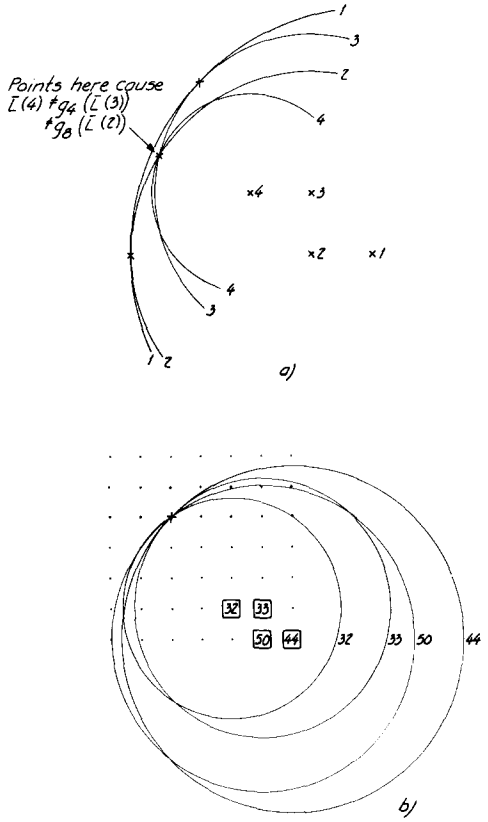


FIG. 10. Do subdisks of one layer always cover the original disk?

Skeleton processing is a subject which is out of the scope of this paper. However, it should be noted that ever since the original papers on "medial axis transformations" were published [5] the application of this elegant concept in digital picture processing has been hampered by the lack of true Euclidean distance maps. Distance maps based on the metrics d_4 and d_8 are almost useless since they produce vastly disconnected skeletons even for reasonably compact objects. Also, the skeleton undergoes drastic changes when an object is rotated. The octagonal distance map is better, but since its disks are octagons and not optimally approximated circles, octagonal skeletons suffer from the same insufficiencies as d_4- and d_8-skeletons, although not to the same extent. Several examples of "skeleton cleaning" can be found in [4]. Such operations are expected to be very effective on Euclidean skeletons.

5. SEQUENTIAL OCTAGONAL MAPPING

As was mentioned in the introduction, octagonal distances and octagonal propagation are almost standard procedures in picture processing. Although parallel algorithms for octagonal distance mapping were published very early [3], the sequential (and much faster) version seems to be lacking in the literature. We can now take advantage of the general insight into the propagation problem gained

from the discussion in Section 3. The Sequential Octagonal Distance mapping (SOD) runs as follows. Each element $L(i,j)$ of the map is now a positive integer.

SOD

Initially:

$$L(i,j) = 0 \qquad \text{if } (i,j) \in S,$$
$$L(i,j) = Z \qquad \text{if } (i,j) \in \bar{S}.$$

First picture scan: For each line, first

$$L(i,j) = \min\begin{cases} L(i,j) \\ L(i-1,j)+1 \\ L(i,j+1)+1 \\ L(i-1,j+1)+1 & \text{if } L(i-1,j+1) \text{ is odd} \\ L(i+1,j+1)+1 & \text{if } L(i+1,j+1) \text{ is odd} \end{cases}$$

then

$$L(i,j) = \min\begin{cases} L(i,j) \\ L(i+1,j)+1. \end{cases}$$

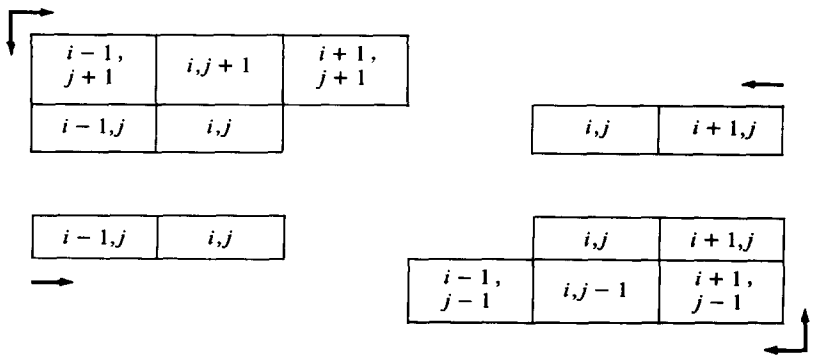
Second picture scan: For each line, first

$$L(i,j) = \min\begin{cases} L(i,j) \\ L(i+1,j)+1 \\ L(i,j-1)+1 \\ L(i+1,j-1)+1 & \text{if } L(i+1,j-1) \text{ is odd} \\ L(i-1,j-1)+1 & \text{if } L(i-1,j-1) \text{ is odd} \end{cases}$$

then

$$L(i,j) = \min\begin{cases} L(i,j) \\ L(i-1,j)+1. \end{cases}$$

The algorithm makes use of following neighborhoods:



The distance errors in octagonal maps are on the order of 10%. Very often this can be tolerated and if so the octagonal mapping seems to be a good choice since the labels $L(i,j)$ are shorter and the comparisons are much simpler than in the Euclidean case. Note, however, that a total of 10 comparisons has to be done while the 4SED-algorithm required only 6, thanks to the smaller neighborhood dependence in the recursion.

Skeletons based on the octagonal distance function can be expected to give many more spurious points and artifacts than skeletons derived from Euclidean distances for two reasons. Firstly, octagonal skeletons represent a set of overlapping octagons instead of circular disks. The octagon shape can be considered much more artificial and rare than a circular boundary. Hence, an original object S is rather cumbersome to match with overlapping octagons. Secondly, Euclidean disks are available for all radii between and including the integer pixel unit distances while octagons only come in unit step sizes.

6. PARALLEL EUCLIDEAN DISTANCE MAPPING

As was said in the introduction, the sequential few-scan algorithms are vastly superior to parallel procedures unless physically parallel hardware exists. Today such true parallel hardware seems rather remote. Still it could be of some interest to see what could be performed in case a Parallel Euclidean Distance Mapping (a PED-algorithm) should be implemented on such a machine. Thus, we assume the existence of a cellular network where each cell corresponds to one pixel (i,j) and has, perhaps rudimentary but still general, logic and arithmetic capability. The basic operation in the distance mapping algorithms is to compare the present label $\bar{L}(i,j)$ to a neighbor, say $\bar{L}(i-1,j)$, and assign the next label at (i,j)

$$L(i,j) = \min\begin{cases} \bar{L}(i,j) \\ \bar{L}(i-1,j) + (1,0). \end{cases}$$

The efficiency of the algorithms is most conveniently measured as the total number of comparison steps (executed in parallel at each cell) to perform a certain task. One comparison propagates the labeling one step in one direction and we can call the above operation

$$\mathrm{PED}/i+$$

since it propagates labels one step in the positive i-direction. It is readily seen that the sequence

$$\mathrm{PED}/i+\,,\mathrm{PED}/i-\,,\mathrm{PED}/j+\,,\mathrm{PED}/j-$$

produces propagation of all labels in the array one step in each direction up to a chessboard distance of 1. That is, each label has the chance to influence its eight neighbors (not only its four nearest neighbors). The relative order of the operations is insignificant.

If the above sequence is iterated t times it is seen that the labeling has propagated so that all pixels up to a distance of t have been labeled and we conclude that to produce a 4SED-map over the chessboard distance t requires $4t$ comparison steps. The more accurate 8SED-map employs diagonally oriented comparisons so that $8t$ comparisons are required.

Like many other cases of propagation through a network it is, at least in principle, possible to reduce this to

$$2_{\log\lceil t+1\rceil}$$

where $\lceil x \rceil$ means a power of 2, $\lceil x \rceil/2 < x < \lceil x \rceil$. To achieve this, we have to use connections (which hopefully exist in our parallel machine) so that one label at $(i+n,j)$ can be transferred to the logic cell at (i,j) and execute PED/ni +

$$L(i,j) = \min\begin{cases} |L(i,j)| \\ |L(i-n,j) + (n,0)| \end{cases}$$

and also PED/ni −

$$L(i,j) = \min\begin{cases} |L(i,j)| \\ |L(i+n,j) - (n,0)|. \end{cases}$$

The operations PED/nj + and PED/nj − are defined in the same manner. We call this operation *multistep* PED-mapping and the idea is to employ a sequence of these with step sizes which are falling powers of 2:

PED/n
PED/$n/2$
⋮
PED/2
PED/1

The principal effect is shown in Fig. 11, which shows the steps PED/2i, PED/2j, PED/1i, PED/1j.

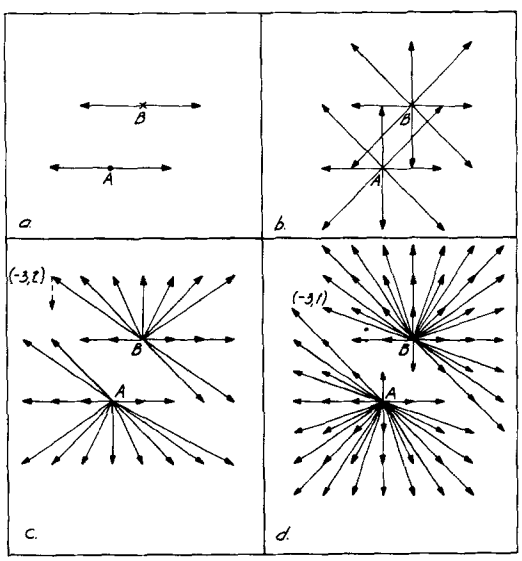


FIG. 11. Multistep labeling represented by vectors.

Unlike the previous continuous propagation from the objects outwards, label vectors are now planted like seeds in the first two operations. From here, new labels are induced to the adjacent pixels. It may now happen that a final correct label is attained only if a longer vector is allowed to induce a shorter one. Thus, the vector $(-3, 2)$ in Fig. 11c induces the vector $(-3, 1)$ in Fig. 11d and this assignment requires a *signed* vector addition.

7. CONCLUSIONS

The most important result of this paper is that Euclidean distances can be produced efficiently by sequential algorithms. The rather simple key is to use a two-component label which makes it possible to propagate in the usual main directions and still keep track of all distances encountered. Distance maps are an alternative to pure binary propagation expansion/shrinking in binary pictures. The cost is of course that one has to use a many-valued (= nonbinary) image memory, but this disadvantage is rapidly disappearing because of new extremely powerful memory components. A distance map is produced with the 4SED algorithm in two-picture double line-scans regardless of expansion distances and gives a much greater versatility than the pure binary image. The expansion/shrinking distance t achieved after thresholding the map is correctly Euclidean except for a few enumerable cases. Errors in the 8SED map are virtually nonexistent. Somewhat amazingly one can obtain an expanded binary picture for other than integer values of t.

Skeletons are a simple byproduct of distance mapping. Although not treated extensively in this paper, Euclidean skeletons can possibly revive interest in this method for image coding and processing.

In many cases, the octagonal distances are sufficiently accurate and it is possible to produce an octagonal distance map in the same manner as a Euclidean map, namely with sequential propagation in two picture scans. Octagonal distance mapping requires less arithmetic capability and shorter labels.

The parallel algorithms are only interesting if truly parallel hardware is available. In the absence of such hardware they are merely a curiosity.

ACKNOWLEDGMENTS

The author is grateful to Mr. Peter Lord for the final error-checking of the 4SED- and 8SED-algorithms. Also, the anonymous reviewer has had quite an influence on the final version of the paper. His in-depth penetration of the content supports his statement that the algorithms in this paper "are essentially identical to those I have been using since Spring 1977, but have not yet published."

REFERENCES

1. A. Rosenfeld and A. C. Kak, *Digital Picture Processing*, Academic Press, New York, 1976.
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 Recognition* **1**, 1968, 33–61.
4. U. Montanari, A method for obtaining skeletons using a quasi-Euclidean distance, *J. Assoc. Comput. Mach.* **15**, 1968, 600–624.
5. H. Blum, A transformation for extracting new descriptors of shape, in *Models for the Perception of Speech and Visual Form* (W. Wathen-Dunn, Ed.), pp. 362–380, MIT Press, Cambridge, Mass., 1967.
6. P. E. Danielsson, A new shape factor, *Computer Graphics Image Processing* **7**, 1978, 292–299.

7. Z. Kulpa, On the properties of discrete circles, rings, and discs, *Computer Graphics Image Processing* **10**, 1979, 348–365.
8. M. Doros, Algorithms for generation of discrete circles, rings, and discs, *Computer Graphics Image Processing* **10**, 1979, 366–371.
9. Z. Kulpa and B. Kruse, Methods of effective implementation of circular propagation in discrete images, Internal Report LiTH-ISY-I-0274, Dept. of Electrical Engineering, Linköping Univ., Sweden, 1979.
10. B. Kruse, "PICAPO, Description and User's Manual," Internal Report LiTH-Isy-I-0171, Dept. of Electrical Engineering, Linköping Univ., Sweden, 1977.
11. H. G. Barrow, J. M. Tenenbaum, R. C. Bolles, and H. C. Wolf, Parametric correspondence and chamfer matching: Two new techniques for image matching, Proc., 5th International Joint Conference on Artificial Intelligence, 1977, pp. 659–663.