Computer Vision and Image Understanding
Vol. 76, No. 2, November, pp. 163–172, 1999
Article ID cviu.1999.0783, available online at http://www.idealibrary.com on IDEAL®

NOTE

Fast Euclidean Distance Transformation by Propagation Using Multiple Neighborhoods

O. Cuisenaire* and B. Macq

Communications and Remote Sensing Laboratory, Université Catholique de Louvain, Belgium

Received September 8, 1998; accepted June 28, 1999

We propose a new exact Euclidean distance transformation (DT) by propagation, using bucket sorting. A fast but approximate DT is first computed using a coarse neighborhood. A sequence of larger neighborhoods is then used to gradually improve this approximation. Computations are kept short by restricting the use of these large neighborhoods to the tile borders in the Voronoi diagram of the image. We assess the computational cost of this new algorithm and show that it is both smaller and less image-dependent than all other DTs recently proposed. Contrary to all other propagation DTs, it appears to remain $o(n^2)$ even in the worst-case scenario.

1. INTRODUCTION

From a binary image made of an object O and its background O', a distance transformation [1] makes an image, the distance map, in which the value of any pixel is the distance from this pixel to the object O, i.e. the distance to the nearest pixel of O,

$$D(p) = \min\{\text{dist}(p, q), q \in O\}.$$

Approximations of the Euclidean distance transformation (DT), were proposed by Danielsson [2] and Borgefors [3, 5, 7]. Danielsson uses four raster scans on the image to get a result that is exact on most points but which can produce small errors with some configurations of the object pixels. Borgefors proposes a chamfer DT using two raster scans, but only provides a much coarser approximation of the Euclidean metric. Leymarie [13] showed that, if implemented carefully, both approximations have similar computational cost. Ragnemalm [14] proposed an ordered propagation version of Danielsson's algorithm, as well as a raster scan implementation [17] using a minimal number of scans. While these approximations are good enough for many applications, there are cases for which the exact Euclidean DT is needed. For instance, the mathematical morphology dilation operator can be implemented as the threshold of a Euclidean DT, as shown in [25]. If an approximate DT such as Danielsson's was used, the occasional errors in the distance computation could lead to pixels missing from the dilated object. Thus, the morphological closing, a dilation followed by an erosion with the same structuring element, could actually remove pixels from the original object, which is in contradiction with the basic properties of mathematical morphology.

Many algorithms providing exact Euclidean maps have been proposed in the past. They can be divided into three categories, according to the order used to scan the pixels. First, parallel algorithms were presented by Yamada [4], Mitchell [16, 18], or Embrechts [20], but they cannot be efficiently implemented on a general-purpose computer. Second, raster scanning algorithms were proposed by Mullikin [12] or Saito [19]. Third, propagation or contour-processing algorithms were introduced by Vincent [9], Ragnemalm [14], and Eggers [24].

In these algorithms, the information is transmitted from each pixel to its neighbors, starting from the contours of the object and using a dynamic list to store the pixels in the propagation front. For a Euclidean DT, the information propagated is usually a vector pointing to the nearest object pixel. As shown by Eggers [21], this can be seen as an efficient implementation of the parallel algorithms of Yamada or Mitchell on general purpose computers.

Saito's algorithm and the propagation methods are the fastest exact Euclidean DT for general purpose computers. Nevertheless, their computational cost is highly image dependent. For some images, the computational complexity reaches $o(n^3)$ for $n \times n$ images.

We propose a faster Euclidean DT by propagation that works in two steps. First, we perform an approximate DT using ordered propagation by bucket sorting. It produces a result similar to Danielsson's. Then, this approximation is improved by using neighborhoods of increasing size. The computational cost is kept small by restricting the use of these neighborhoods to the points where they are really needed: the edges of tiles in a Voronoi diagram of the object pixels.

* Corresponding author. E-mail: Cuisenaire@tele.ucl.ac.be.

1077-3142/99/ $30.00

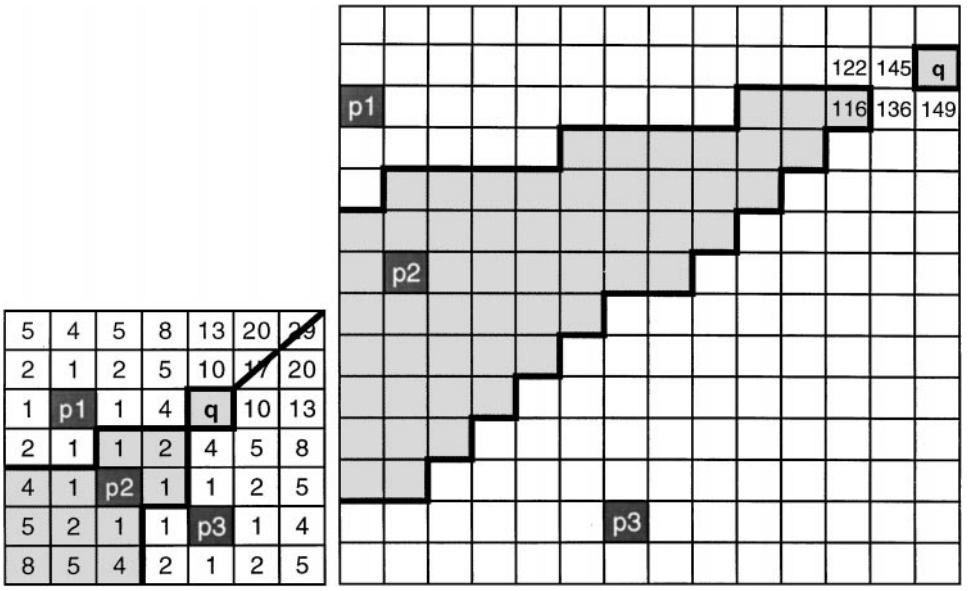

FIG. 1. Pixel q is closer to object pixel p_2 than to p_1 or p_3. On the left (right) image, in grey, the Voronoi polygon $VP(p_2)$ is disconnected with the 4-direct (8-direct) neighborhood. Thus, Danielsson's 4SED (8SED) algorithm assigns $D(q) = \text{dist}_E(p_1, q) = 9$ (170), instead of $\text{dist}_E(p_2, q) = 8$ (169).

In Section 2, we show some relations between Voronoi diagrams and distance transformations. In Section 3, we present the bucket sorting propagation, while the use of larger neighborhoods is explained in Section 4. In Section 5 we compare the computational cost of our DT to the algorithms of Eggers and Saito and analyse its dependency to image features. Finally, Section 6 briefly addresses the extension of the DT to 3D or anisotropic data.

The following notations are used. Letters p, q, r are used for pixels with indexes p_i, where needed. Those pixels belong either to the object O or the background O' of the image. The coordinates of pixel p are (p_x, p_y). $\text{dist}_M(p, q)$ is the distance between pixels p and q using metric M; dist_e stands for the Euclidean metric and dist_E for its square; i.e., $\text{dist}_E(p, q) = (p_x - q_x)^2 + (p_y - q_y)^2$. In what follows, all DTs are computed using dist_E, because this metric requires no floating-point computations, contrary to dist_e. In the algorithms, we also write $\text{dist}_E(dp) = dp_x^2 + dp_y^2$, with $dp = p - q$. $D(p)$ is the value of the distance map at pixel p. $VP(p)$ is the Voronoi polygon surrounding pixel p of the object. $N(p)$ is the set of neighbors of p; i.e., $N(p) = \{q = p + n \mid n \in N\}$; $N = N((0, 0))$ is a neighborhood. The neighborhoods we consider here are balls; i.e., $N = B_d = \{n \mid \text{dist}_M(n, (0, 0)) < d\}$ with some metric M.

2. VORONOI DIAGRAMS AND DISTANCE TRANSFORMATIONS

Let us consider, in a continuous plane, the discrete set of points made of the centres of the pixels in the object O. The division of the plane into polygonal areas containing the part of the plane that is nearer to one object pixel than to any other is called the Voronoi diagram [10]. The polygonal areas are called tiles. Voronoi diagrams and distance transformations are closely related problems. On one hand, given a set of points and its Voronoi division, computing the distance map to this set of point is a straightforward operation. On the other hand, DTs are an efficient way of computing Voronoi division, as shown by Borgefors [5] and Saito [19].

On a continuous plane, the tiles of the Voronoi diagram are connected sets. As illustrated at Fig. 1, this is not always true on a discrete lattice. The tile around object pixel p_2 is disconnected in Fig. 1a (1b) for 4-direct (8-direct) connectivity. In other words, belonging to a given Voronoi tile is not a local property: the tile to which a pixel belongs cannot always be deduced from the tiles to which its neighbors belong. Because they propagate the information locally from neighbor to neighbor, both raster scanning and propagation DT algorithms [2, 13, 14, 17] will provide a wrong value for $D(q)$ at Fig. 1. The information from pixel p_2 cannot reach pixel q, p_2 is hidden from q by pixels p_1 and p_3.

Parallel algorithms do not suffer from this limitation because they allow multiple propagation fronts to follow each other. Hence, the information flow from p_2 reaches q before the flows from p_1 and p_3, and thus, there is no such hiding effect. Authors of sequential algorithms have proposed methods to emulate this multiple fronts behaviour with a sequential propagation. Ragnemalm [14] explicitly introduces a delayed updating mechanism. Mullikin [12] propagates lists of ties and near ties. Eggers [24] orders the propagation according to the chessboard metric instead of the Euclidean one.

Regardless of the way it is emulated, the multiple propagation mechanism common to these exact Euclidean DTs requires performing many unnecessary computations while successive propagation fronts reach the same pixel and update its value.

For some images (see Section 5), this leads to a $o(n^3)$ complexity for $n \times n$ images.

As pointed out by Ragnemalm [14] to justify ordered propagation, a pixel only needs to be updated once—when it gets its final value. The algorithm we propose approaches this behavior in two steps. First, the propagation from each object pixel is restricted to pixels of its tile by using an ordered propagation scheme similar to Ragnemalm's approximate DT [14]. Second, we produce the exact Euclidean DT by restoring the connectivity of the tiles as we consider larger neighborhoods on the tile boundaries. These two steps are developed in Sections 3 and 4, respectively.

3. PROPAGATION DT USING BUCKET SORTING

In order to restrict the propagation from one object pixel to the pixels of its tile, the propagation order should be the same as the metric order. In other words, the pixels in the propagation front should be sorted by increasing distance values before being propagated.

As shown by Verwer [8] for constrained chamfer DT and Ragnemalm [14] for Euclidean DT, this can be accomplished by bucket sorting the pixels in the propagation front. Instead of using a single list, pixels to be propagated are stored in a number of buckets, one for each possible distance value. The square of the Euclidean metric, $\text{dist}_E(p, q)$ is always an integer if p and q are located on an integer grid. Thus, we use it as the index d for the lists, which are processed by increasing index values, as illustrated at Fig. 2. For each pixel p in the propagation front, we store its coordinates (p_x, p_y) and its coordinates (dp_x, dp_y) relative to the nearest pixel of the object. This gives the following algorithm, which we call "Propagation using a single neighborhood," or PSN.

INITIALIZATION: let M be an upper bound for $D(p)$
  for all $p \in O$
    $\{D(p) = 0; (p, (0, 0)) \rightarrow \text{bucket}(0)\}$
  for all $p \in O'$
    $D(p) = M$
  $d = -1$;

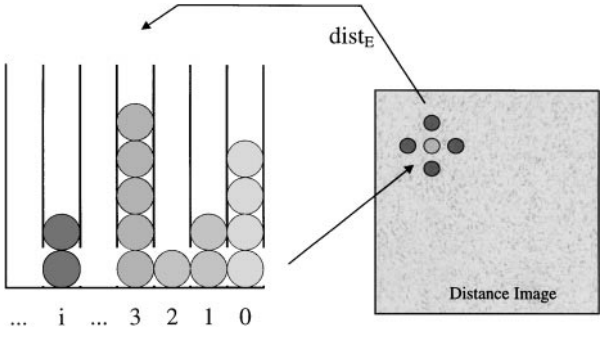

FIG. 2. Ordered propagation with bucket sorting. Pixels are taken from the nonempty bucket with the lowest index, their neighbors are tested, and, if needed, entered in the bucket whose index is the square of their Euclidean distance.

ITERATIONS:
  while all buckets are not empty
    $\{d = d + 1$;
     for all (p, dp) in bucket(d)
       if $D(p) = d$
         for all $n \in N$
           $\{newD = \text{dist}_E(dp + n)$
            if $newD < D(p + n)$
           $\{D(p + n) = newD$; $(p + n, dp + n)$
            $\rightarrow$ bucket($newD$);$\}$
    $\}$ $\}$

The termination condition—that all lists are empty—is true if the last $(2d + 1)$ processed are empty, with d the current distance, d^2 the current index.

An efficient implementation of the buckets' data structure requires a memory allocation in chunks for the dynamic lists. Verwer [8] discusses the optimal chunk size for memory/speed optimization. It should be large enough so that the dynamic memory allocation becomes a negligible part of the computation time, but small enough in order not to waste too much memory in partially filled buckets. Practically, we chose chunks of 64 elements, but a large range of values would be acceptable.

Also, special care should be given to an efficient computation of $\text{dist}_E(dp + n)$. Leymarie [13] recommends $\text{dist}_E(dp + n) = \text{dist}_E(dp) + 2 \cdot dp_x + 1$ if $n = (1, 0)$, $\text{dist}_E(dp + n) = \text{dist}_E(dp) + 2 \cdot dp_y + 1$ if $n = (0, 1)$, etc. which only requires using additions, one shift and one increment. Instead, we use $\text{dist}_E(dp + n) = sq[dp_x + n_x] + sq[dp_y + n_y]$, where $sq[x]$ is a precomputed lookup table for x^2. This has a similar computational cost and is more general when n belongs to a large neighborhood, such as in the next section.

As shown by Ragnemalm [14], not all neighbors in N need to be considered, but only those neighbors n in the same direction as dp, i.e. neighbors for which $dp_x \cdot n_x \geq 0$ and $dp_y \cdot n_y \geq 0$. Apart from the first two iterations, these inequalities can even be considered in the strict sense. The neighborhoods for the first, second, and all successive iterations are illustrated in Fig. 3. Contrary to Ragnemalm's [14], those are not based on the 8-direct neighborhood, but the simpler 4-direct one. Indeed, the few additional errors made at this step do not matter since they will be corrected in the next section anyway.

4. USE OF MULTIPLE NEIGHBORHOODS

As pointed out in Section 2, the above algorithm is not an exact Euclidean DT, because the tiles in the Voronoi diagram are not connected sets. Actually, it makes the same errors as Danielsson's 4SED algorithm [2]. In order to overcome this limitation, we should consider neighborhoods larger than those of Fig. 3. Unfortunately, we cannot apply this simple idea directly, for two reasons. First, the computation time is proportional to the size of the neighborhood and becomes prohibitive with large ones. Second, there is no upper bound on the size of

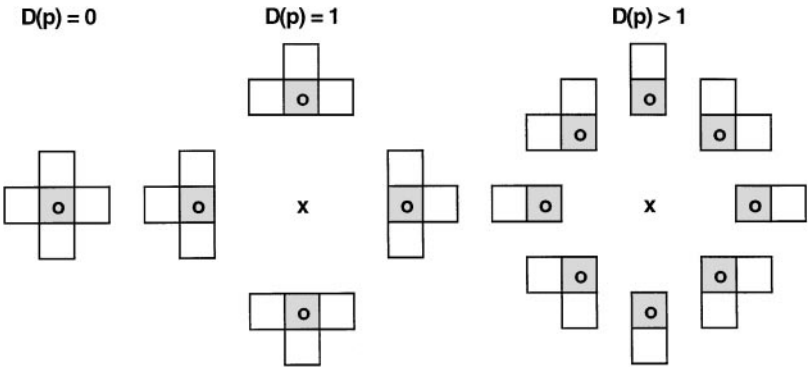

FIG. 3. Neighborhoods used by PSN to propagate pixel p.

the neighborhood to be used to ensure the connectivity of all tiles regardless of the image.

Nevertheless, it is possible to restrict the number of pixels for which larger neighborhoods should be considered and the size of these neighborhoods, thanks to the following properties.

PROPERTY 1. *Let N_1 and N_2 be two neighborhoods such that $N_1 \subset N_2$. Let D_1 and D_2 be the resulting distance maps generated using N_1 and N_2, respectively. If there is a pixel p such that $D_1(p) \neq D_2(p)$, i.e. $D_1(p)$ is inexact and $D_1(p) > D_2(p)$, then p has a N_2 neighbor q such that either*

—$D_1(q) \neq (D_2)(q)$
—$D_1(q)$ was not propagated using N_1.

Proof. Let us first consider that pixels belonging to a Voronoi polygon $VP(q)$ can only propagate to other locations belonging to $VP(q)$ with the same q, which is what ordered propagation aims to achieve. Let us consider q such that $p \in VP(q)$, i.e. let q be the nearest object pixel to p. Let us consider the set $S = VP(q) \cap (N_2(p) \backslash N_1(p))$ (see Fig. 4). We know this set is not empty since at least one pixel propagated to p using N_2, but none did using N_1. Let us suppose that all pixels $r \in S$ have a correct value $D(r)$ and were propagated using N_1. Since propagation implies a strict increase of the distance value, at least one pixel $r \in S$ must propagate to a pixel $r' \notin$ S, i.e. a pixel $r' \in VP(q) \cap N_1(p)$. This is impossible since r' would then have been propagated to p using N_1, and $D_1(p)$ would then be correct. ■

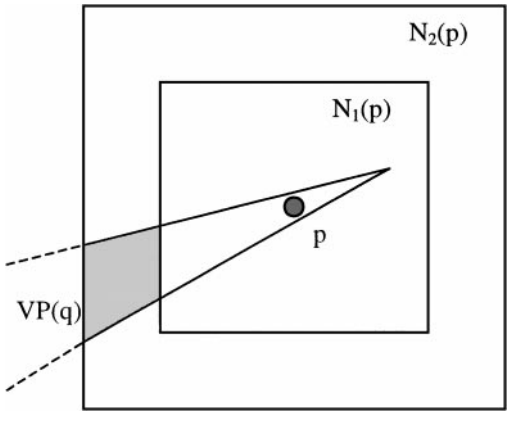

FIG. 4. Proof of Property 1. The area in grey is $VP(q) \cap (N_2(p) \backslash N_1(p))$.

In practice, the ordered propagation using bucket sorting—the PSN algorithm—allows propagation to locations that are either in the same VP or in the neighborhood of this VP. Nevertheless, the above proof remains correct since propagated pixels located in the wrong VP are corrected before being propagated and do not propagate themselves.

With this property, only pixels that did not propagate using the 4-direct neighborhood in the PSN algorithm need to be considered with a larger neighborhood. The size and shape of those neighborhoods is determined by Properties 2 and 3, respectively.

PROPERTY 2. *For any neighborhood N and a distance transformation D_N using this neighborhood, there is a value $d(N)$ such that for all pixels p with $D_N(p) < d(N)$, $D_N(p)$ is exact.*

Proof. The existence of such a $d(N)$ is obvious, since any neighborhood N would at least give a perfect Euclidean DT for $d(N) = 1$, i.e. $D_N(p) = 0$. For a neighborhood N, the upper bound for $d(N)$ can be computed by extensive search. For an error to occur at the location dp_0 of a pixel p relatively to its nearest object pixel q_0, there must be, for all pixels $p + n$ in its N neighborhood, an object pixel q_1 closer to $p + n$, but further away from p. Mathematically, there can be an error at the relative location dp_0 if

$$\forall n \in N \; \exists dp_1 \mid \mathrm{dist_E}(dp_0) < \mathrm{dist_E}(dp_1) \quad \text{and}$$

$$\mathrm{dist_E}(dp_0 + n) \geq \mathrm{dist_E}(dp_1 + n) \quad ■$$

Applying this test for all possible couples (dp_{0x}, dp_{0y}) with $dp_{0i} < D$, an $N \times N$ neighborhood, and restricting the possible couples (dp_{1x}, dp_{1y}) to a circle on which $\mathrm{dist_E}(dp_1)$ is just above $\mathrm{dist_E}(dp_0)$ is an algorithm of at most $o(D^3 \cdot N)$ complexity. The results for the 4-direct neighborhood and $N \times N$ neighborhoods are found in Table 1.

TABLE 1
Smallest Errors for the 4-Direct and Square Neighborhoods

Neighborhood	Smallest error		Nonpropagating pixel	
	(dp_x, dp_y)	dist_E	(dp_x, dp_y)	dist_E
4-direct	(2, 2)	8	(1, 1)	2
8-direct $= 3 \times 3$	(12, 5)	169	(10, 4)	116
5×5	(25, 7)	674	(22, 6)	520
7×7	(48, 10)	2404	(44, 9)	2017
9×9	(72, 12)	5328	(67, 11)	4610
11×11	(108, 15)	11889	(102, 14)	10600
13×13	(143, 17)	20738	(136, 16)	18752
15×15	(192, 20)	37264	(184, 19)	34217
17×17	(238, 22)	57128	(229, 21)	52882
19×19	(300, 25)	90525	(290, 24)	84676
21×21	(357, 27)	128178	(346, 26)	120392
23×23	(420, 29)	177241	(408, 28)	167248
25×25	(500, 32)	251024	(487, 31)	238130
27×27	(574, 34)	330632	(560, 33)	314689
29×29	(667, 37)	446258	(652, 36)	426400
31×31	(768, 40)	591424	(752, 39)	567025

Note. The first two lines correspond to the errors illustrated at Fig. 1.

In Table 1, the right column, we also find the smallest relative location where a pixel could fail to propagate using each neighborhood. Using this table, we know the size of the neighborhood to consider for which nonpropagating pixel selected by Property 1. For instance, if its value is $\text{dist}_E(dp) = 1000$, we find in Table 1 that 1000 is between 512 and 2017. Therefore, propagating p with a 7×7 neighborhood is enough to ensure an exact result, but a 5×5 neighborhood may be too small.

PROPERTY 3. *For a pixel p, whose relative location from the nearest object pixel is (dp_x, dp_y), the only neighbors (n_x, n_y) that need to be considered are those "in the same direction" as (dp_x, dp_y). More precisely, if we consider dp_x and dp_y positive, without loss of generality the neighbors (n_x, n_y) to consider are such that*

$$(n_x + 1) \cdot \frac{dp_y}{dp_x} \leq n_y \leq 1 + n_x \cdot \frac{dp_y}{dp_x}.$$

Proof. In Fig. 5, we wish to determine which neighbors (n_x, n_y) should be considered when propagating pixel p, whose location relative to the nearest object pixel q is (dp_x, dp_y). Let us first consider the upper bound. If pixel $p_1 = p + (0, 1)$ belongs to the same $VP(q)$, the upper bound for neighbors of p_1 is higher than for those of p, which guarantees no neighbor will be missed.

On the other hand, we consider $p_1 \in VP(q')$ with $q' \neq q$. The limit between the tiles $VP(q)$ and $VP(q')$ is on the midperpendicular of $q'q$. This midperpendicular would be a good upper bound for neighbors of p, since the information from q only needs to be propagated to pixels belonging to $VP(q)$. This midperpendicular itself is bounded by the second inequality of Property 3. Indeed, it intersects pp_1 between p and p_1, thus below p_1. Also, its angle must be lower than dp_y/dp_x. Otherwise, it would cross qq_1 between q and q_1, which is impossible for the midperpendicular of qq' with q' on an integer location. The proof for the lower bound is similar. ■

Applying this property reduces the computational cost of a $n \times n$ neighborhood from $o(n^2)$ to $o(n)$. From these three properties, we propose a new algorithm for the exact Euclidean DT. First, we apply the above PSN algorithm and store all nonpropagating pixels into a new bucket data structure bucket 2(). This approximation is corrected using a family of neighborhoods N_i made of balls of increasing sizes, for which the limits of validity valid(N_i) have been precomputed. For instance, the right column in Table 1 gives valid(N_i) for the 4-direct and $n \times n$ neighborhoods.

1. Initialization
2. PSN algorithm with nonpropagating pixels stored in bucket2().
3. Propagation with larger neighborhoods N_i:

```
i = 1; d = valid(N_4-direct) − 1;
while all buckets in bucket2() are not empty
  {d = d + 1;
   if d > valid(N_i)
     {i = i + 1;}
   for all (p, dp) in bucket2(d)
     for all n ∈ N_i with direction within bounds
     of property 3
       {newD = dist_E(dp + n);
        if newD < D(p + n)
          {D(p + n) = newD;
           (p, dp) → bucket2(newD);
}      }  }
```

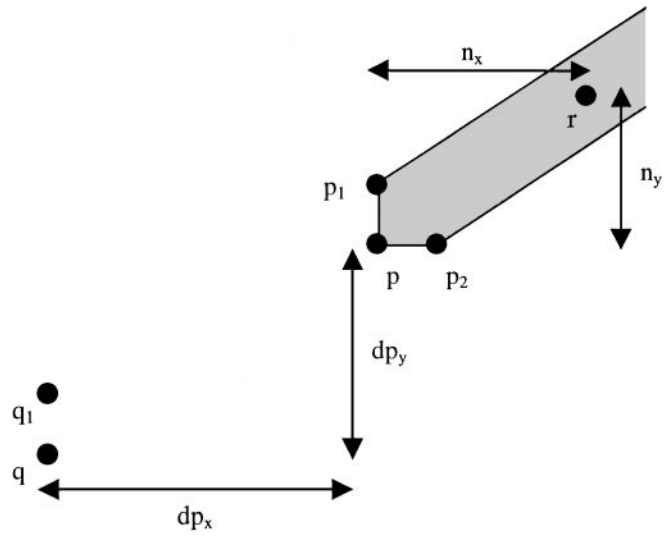

FIG. 5. Proof of Property 3. In PMN, p only needs to propagate to neighbors within the grey area.

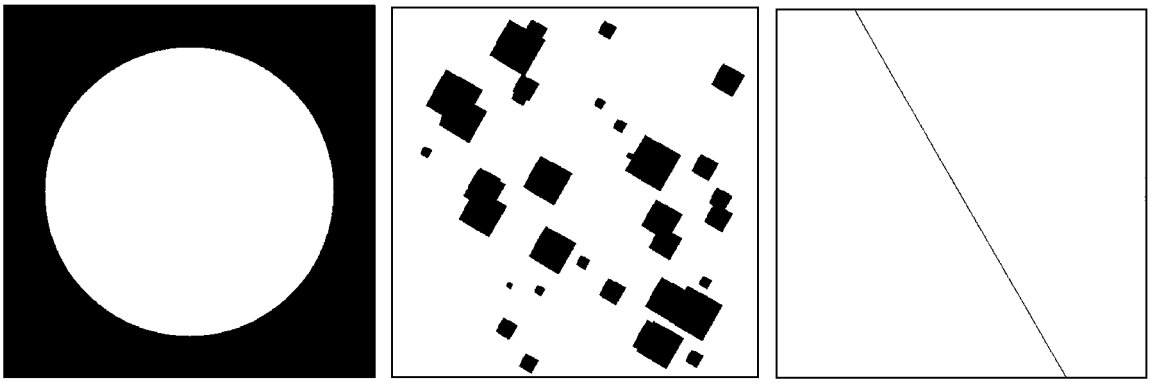

FIG. 6. Test images. Object pixels are black; distances are computed in the white areas.

5. COMPLEXITY AND COMPUTATIONAL COST

Comparing the complexity and computational costs of DT algorithms is a complex task. For $n \times n$ images, simple approximate algorithms such as Danielsson [2], chamfer DT [3] or Leymarie [13] have a fixed $o(n^2)$ cost, regardless of the image content. Yamada's parallel algorithm [4] has a $o(d \cdot n^2)$ cost proportional to the maximum distance d found in the image. More complex exact algorithms such as Ragnemalm [14], Eggers [24], and Saito [19] have costs that are highly dependent of the image content and can vary between $o(n^2)$ and $o(n^3)$. For those, experiments give a better knowledge of their complexity than theoretical considerations.

Among the methods published so far, those of Eggers and Saito seem to be the fastest exact Euclidean DTs. Therefore, we compare our algorithm to those and to the chamfer 3-4 DT, a commonly used approximatation of the Euclidean DT. Two versions of our algorithm were used for the comparison: the approximate single neighborhood (PSN) algorithm of Section 3 and the exact multiple neighborhood (PMN) algorithm of Section 4.

The choice of images on which the tests are performed is subjective and strongly affects the results. Thus, we perform three tests, illustrated at Fig. 6. Test 1, suggested by Saito, is an empty disk of variable diameter. Test 2, suggested by Eggers, is made of random squares covering 15% of the image in total,

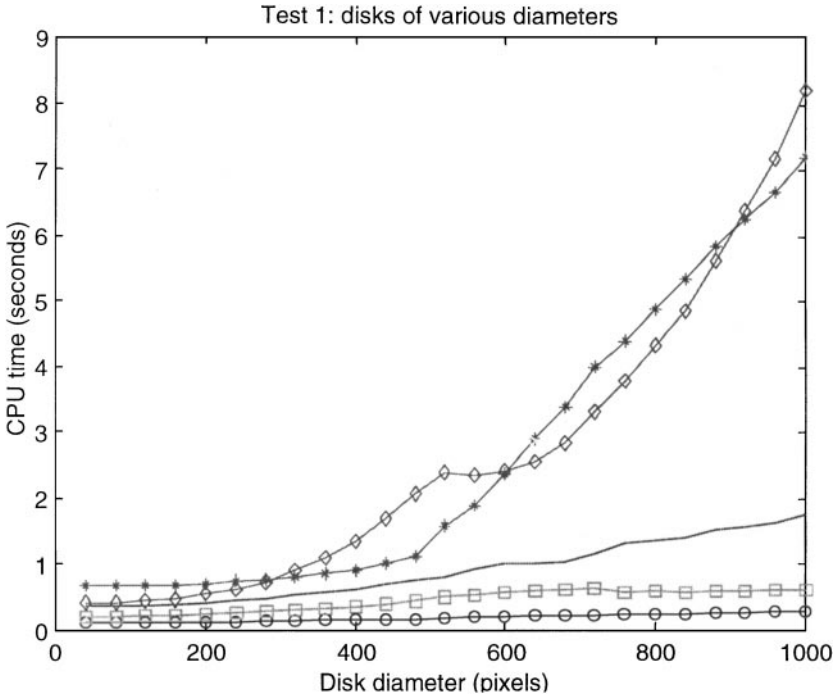

FIG. 7. Test 1: diameters vary from 40 to 1000 pixels: "○," Chamfer 3, 4; "□," PSN; "–," PMN; "◇," Eggers; "∗," Saito.

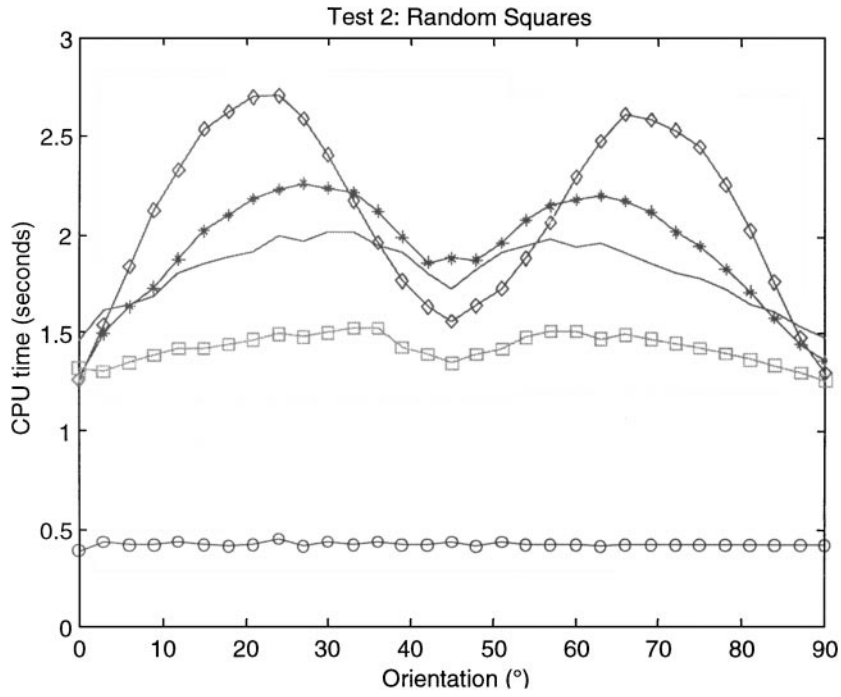

FIG. 8. Test 2: orientations varying from 0° to 90°: "○," Chamfer 3, 4; "□," PSN; "–," PMN; "◇," Eggers; "∗," Saito.

with an orientation varying between 0° and 90°. Test 3, our suggestion, is the worst-case scenario foreseen by Eggers and Ragnemalm for propagation DTs: a straight line across the image with an orientation varying between 0° and 90°. All images are 1024×1024 pixels.

The algorithms were implemented in C on a Sun Sparc Ultra. We compare the CPU time required by each algorithm. Similar results could be drawn comparing the number of comparisons per pixel using each algorithm. All results are shown at Figs. 7 to 9 for tests 1 to 3, respectively.

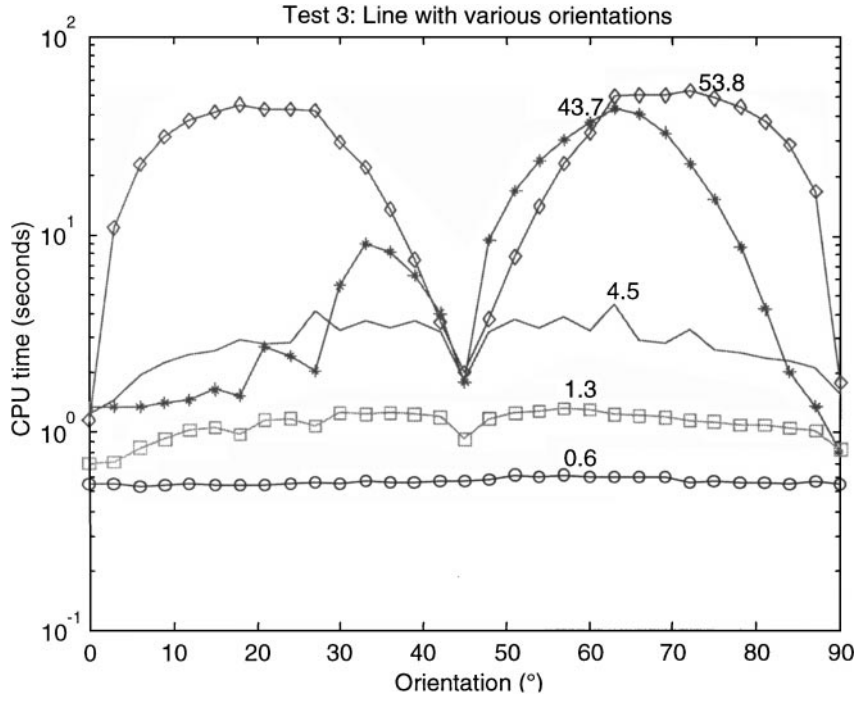

FIG. 9. Test 3: orientation vary from 0° to 90°. Maxima for each curve are displayed: "○," Chamfer 3, 4; "□," PSN; "–," PMN; "◇," Eggers; "∗," Saito.

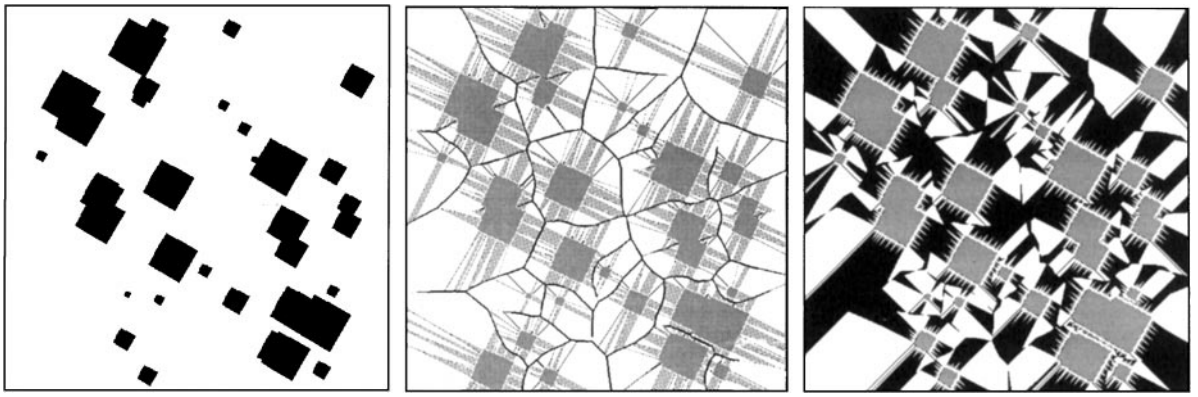

FIG. 10. *Left:* object image. *Centre:* pixels propagated more than once by PMN. *Right:* pixels propagated more than once by Eggers' DT.

Test 1 shows similar results for Eggers and Saito's algorithms, while both PSN and PMN perform much better. For a disk of diameter 1000 pixels, both Eggers and Saito are more than 4 times slower than PMN; PSN is between 1.5 and 3 times faster than PMN.

Test 2 gives more complex results. PSN is always faster than all exact algorithms. Those perform similarly with an optimal performance around 0°, 45°, and 90° orientations and poorer results for 22.5° and 67.5°. The average performance of the algorithms over the range of orientations is quite similar, with Saito 5% and Eggers 10% slower than PMN. Finally, PMN is less orientation dependent. The ratio between maximum and minimum CPU time requirements is 1.37 for PMN, compared to Saito's 1.78 and Eggers' 2.15.

Test 3 shows the worst-case scenario where the methods of Ragnemalm and Eggers are known to have a $o(n^3)$ complexity. The disparity between CPU costs is such that the results are here displayed using a logarithmic scale. Eggers' DT performs very poorly for any orientation but the horizontal, vertical, and 45° diagonal. Saito's DT performs quite well for orientations between 0° and 45°, but very poorly for orientations around 60°. On average, Eggers and Saito are respectively 10 and 4 times slower than PMN. In the worst cases, Eggers and Saito are respectively 12 and 10 times slower than PMN. Compared to approximate DTs, PMN is only 2.5 times slower than PSN and 5 times slower than chamfer DT. Finally, our algorithms are less orientation dependent since PSN varies from 1 to 2, PMN from 1 to 4 while both Eggers and Saito vary from 1 to 50. Chamfer DT is of course orientation independent.

As a general conclusion of those tests, we see that—apart from a few special cases—PMN is always faster than the fastest published DTs, up to 10 times faster for worst-case images. The additional cost of using the error-free PMN, compared to the already good approximation of PSN is to multiply the CPU time by a factor between 1 and 3. These conclusions would be even more favorable to PMN if the tests were performed with larger images. An explanation of the good behavior of PMN is illustrated in Fig. 10, where the pixels propagated more than once are displayed for PMN and Eggers' DT.

Furthermore, PSN and PMN treat the data in a fully progressive way; the computation can be stopped at any time and still provide a sensible result. If stopped during the first propagation, it will provide the DT in a region surrounding the objects only. If stopped during the second propagation, it will provide an error-free DT up to a given value, and a good approximation of it for larger values. We can also create a class of PMN-i algorithms, where i is the size of the largest neighborhood used in the second propagation. This provides us with a variety of trade-offs between PSN and PMN, which can be seen as the PMN-0 and PMN-∞ extrema of the class.

This property makes our algorithm suitable for applications where the computation time is fixed, such a real-time applications, even though its cost is not entirely image-independent.

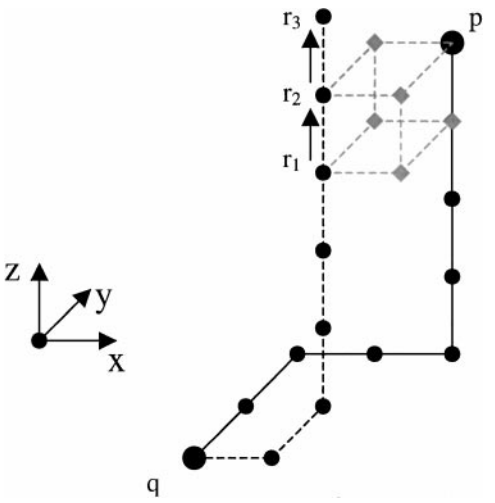

FIG. 11. Counterexample of Property 1 in three dimensions. p belongs to $VP(q)$ and has a relative position of (2, 2, 4). With the 6-direct neighborhood, p is disconnected from the rest of $VP(q)$ if there are object pixels for which the relative position of p is (0, 0, 5), (4, 0, 3), and (0, 4, 3), respectively. The pixels in grey on the figure are then members of another tile. In a 3D PMN, p should get its correct value from either pixel r_1 or r_2, using a $3 \times 3 \times 3$ neighborhood. Unfortunately, both are propagated with the 6-direct neighborhood, to r_2 and r_3, respectively.

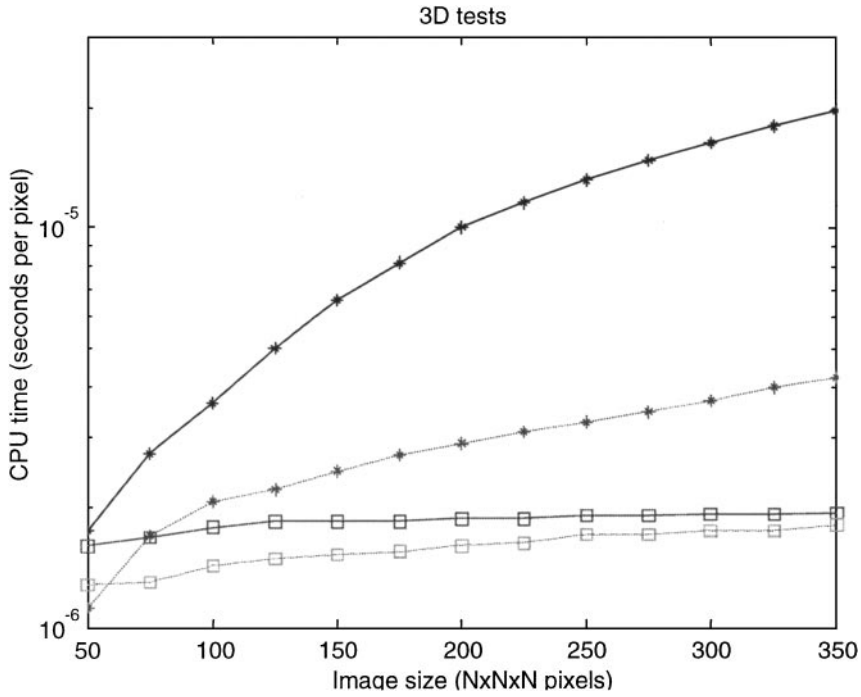

FIG. 12. CPU time required per pixel for two sets of test images of sizes between $50 \times 50 \times 50$ and $350 \times 350 \times 350$. Test image 1 (dashed line) is an eighth of a sphere. Test image 2 (plain line) is a plane oriented at 60°, the worst case for Saito. The DTs considered are PSN (□) and Saito (∗). The *y*-axis has a logarithmic scale.

6. EXTENSION TO 3D AND ANISOTROPY

PSN can easily be extended to multidimensional or anisotropic data. The only restriction is that the metric—or a simple function of it such as its square for the EDT—should only include integer values, providing an appropriate indexing of the lists in the bucket sorting structure.

Extending PMN to anisotropic data also requires to re-evaluate the limits of validity of each neighbourhood at Table 1. Extending PMN to three or more dimensions is unfortunately not straightforward. Indeed, Property 1 of Section 4 is not valid in more than two dimensions. As illustrated at Fig. 11, it is in this case possible for a pixel to propagate in one direction, while it fails to propagate in another. Therefore, extending PMN to 3D or more would require defining a new rule to detect the pixels that should be considered for larger neighborhoods.

In Fig. 12, we compare the performances of Saito's 3D algorithm to the 3D extension of PSN, which only gives an approximation of the Euclidean metric. Two tests are performed. The first one is similar to test 1 in 2D and the second one is similar to test 3. PSN is faster in all cases and should be used if an exact DT is not absolutely needed. The CPU time needed per pixel increases with the size of the image for Saito's algorithm, while it is nearly constant for PSN. On the other hand, the difference of performance between Saito's algorithm and PSN is smaller in 3D than in 2D, because the image sizes considered are smaller ($n_{\max} = 350$ instead of 1024 in 2D).

7. CONCLUSION

We have developed a new Euclidean DT algorithm using ordered propagation. First, a fast approximation is computed. Then, larger neighborhoods are used to correct potential errors. This algorithm is significantly faster and less image dependent than all previously published error-free DTs. Since results are produced in a progressive order, it can be used with a fixed computation time and provide significant results, which makes it usable for real-time applications.

ACKNOWLEDGMENT

The work of Olivier Cuisenaire was supported by Belgium's FRIA fund.

REFERENCES

1. A. Rosenfeld and J. L. Pfaltz, Distance functions on digital pictures, *Pattern Recognit.* **1**, 1, 1968, 33–61.
2. P. E. Danielsson, Euclidean distance mapping, *Comput. Graphics Image Process.* **14**, 1980, 227–248.
3. G. Borgefors, Distance transformations in arbitrary dimensions, *Comput. Vision Graphics Image Process.* **27**, 1984, 321–345.
4. H. Yamada, Complete Euclidean distance transformation by parallel operation, in *Proc. 7th International Conference on Pattern Recognition, Montreal, 1984*, pp. 69–71.
5. G. Borgefors, Distance transformations in digital images, *Comput. Vision Graphics Image Process.* **34**, 1986, 344–371.

6. J. Piper and E. Granum, Computing distance transformations in convex and nonconvex domains, *Pattern Recognit.* **20**, 6, 1987, 599–615.
7. G. Borgefors, Hierarchical chamfer matching: A parametric edge matching algorithm, *IEEE Trans. Pattern Anal. Mach. Intell.* **10**, 6, 1988, 849–865.
8. B. J. H. Verwer, P. W. Verbeek, and S. T. Dekker, An efficient uniform cost algorithm applied to distance transforms, *IEEE Trans. Pattern Anal. Mach. Intell.* **11**, 4, 1989, 425–429.
9. L. Vincent, Exact Euclidean distance function by chain propagation, in *Proc. Computer Vision and Pattern Recognition Conference, Hawaii, June 1991*, pp. 520–525.
10. F. Aurenhammer, Voronoi diagrams—A survey of a fundamental geometric data structure, *ACM Comput. Surveys* **23**, 3, 1991, 345–405.
11. D. W. Paglieroni, Distance transforms: Properties and machine vision applications, *CVGIP: Graph. Models Image Process.* **54**, 1, 1992, 56–74.
12. J. Mullikin, The vector distance transform in two and three dimensions, *Comput. Vision Graphics Image Process.* **54**, 6, 1992, 526–535.
13. F. Leymarie and M. L. Levine, Fast raster-scan distance propagation on the discrete rectangular lattice, *CVGIP: Image Understanding* **55**, 1992, 85–94.
14. I. Ragnemalm, Neighborhoods for distance transformations using ordered propagation, *CVGIP, Image Understanding* **56**, 3, 1992, 399–409.
15. I. Ragnemalm, Fast erosion and dilation by contour processing and thresholding of distance maps, *Pattern Recognit. Lett.* **13**, 1992, 161–166.
16. F. Y.-C. Shih and O. R. Mitchell, A mathematical morphology approach to Euclidean distance transformation, *IEEE Trans. Image Process.* **1**, 2, 1992, 197–204.
17. I. Ragnemalm, The Euclidean distance transform in arbitrary dimensions, *Pattern Recognit. Lett.* **14**, 1992, 883–888.
18. C. T. Huang and O. R. Mitchell, A Euclidean distance transform using greyscale morphology decomposition, *IEEE Trans. Pattern Anal. Mach. Intell.* **16**, 4, 1994, 443–448.
19. T. Saito and J. I. Toriwaki, New algorithms for Euclidean distance transformations of an n-dimensional digitised picture with applications, *Pattern Recognit.* **27**, 11, 1994, 1551–1565.
20. H. Embrechts and D. Roose, A parallel Euclidean distance transformation algorithm, *Comput. Vision Image Understanding* **63**, 1996, 15–26.
21. H. Eggers, Parallel Euclidean distance transformations in Z^n, *Pattern Recognit. Lett.* **17**, 1996, 751–757.
22. O. Cuisenaire, Region growing Euclidean distance transforms, in *Proceedings, 9th International Conference on Image Analysis and Processing (ICIAP'97), Florence, September 1997*, Vol. 1, pp. 263–270.
23. O. Cuisenaire and B. Macq, Applications of the region growing Euclidean distance transform: Anisotropy and skeletons, in *Proceedings, 1997 International Conference on Image Processing (ICIP'97), Santa Barbara, CA, October 1997*, Vol. 1, pp. 200–203.
24. H. Eggers, Two fast Euclidean distance transformations in Z^2 based on sufficient propagation, *Comput. Vision Image Understanding* **69**, 1, 1998, 106–116.
25. O. Cuisenaire and B. Macq, Fast Euclidean morphological operators using local distance transformation by propagation and applications, in *Proceedings, Image Processing and its Applications (IPA99), Manchester, UK, July 12–15, 1999*.

Statement of ownership, management, and circulation required by the Act of October 23, 1962, Section 4369, Title 39, United States Code: of

COMPUTER VISION AND IMAGE UNDERSTANDING

Published monthly by Academic Press, 6277 Sea Harbor Drive, Orlando, FL 32887-4900. Number of issues published annually: 12. Editor: Dr. Avinash C. Kak, Robot Vision Lab, 1285 EE Building, Purdue University, West Lafayette, IN 47907.

Owned by Academic Press, 525 B Street, Suite 1900, San Diego, CA 92101-4495. Known bondholders, mortgagees, and other security holders owning or holding 1 percent or more of total amount of bonds, mortgages, and other securities: None.

Paragraphs 2 and 3 include, in cases where the stockholder or security holder appears upon the books of the company as trustee or in any other fiduciary relation, the name of the person or corporation for whom such trustee is acting, also the statements in the two paragraphs show the affiant's full knowledge and belief as to the circumstances and conditions under which stockholders and security holders who do not appear upon the books of the company as trustees, hold stock and securities in a capacity other than that of a bona fide owner. Names and addresses of individuals who are stockholders of a corporation which itself is a stockholder or holder of bonds, mortgages, or other securities of the publishing corporation have been included in paragraphs 2 and 3 when the interests of such individuals are equivalent to 1 percent or more of the total amount of the stock or securities of the publishing corporation.

Total no. copies printed: average no. copies each issue during preceding 12 months: 1601; single issue nearest to filing date: 1575. Paid circulation (a) to term subscribers by mail, carrier delivery, or by other means: average no. copies each issue during preceding 12 months: 366; single issue nearest to filing date: 362. (b) Sales through agents, news dealers, or otherwise: average no. copies each issue during preceding 12 months: 573; single issue nearest to filing date: 584. Free distribution (a) by mail: average no. copies each issue during preceding 12 months: 55; single issue nearest to filing date: 55. (b) Outside the mail: average no. copies each issue during preceding 12 months: 20; single issue nearest to filing date: 20. Total no. of copies distributed: average no. copies each issue during preceding 12 months: 1014; single issue nearest to filing date: 1021. Percent paid and/or requested circulation: average percent each issue during preceding 12 months: 93%; single issue nearest to filing date: 93%.

(Signed) Stephanie Smith, Asst. Manager, Journal Business Office