



Euclidean Voronoi labelling on the multidimensional grid

Craig Gotsman^{*}, Michael Lindenbaum

Department of Computer Science, Technion – Israel Institute of Technology, Haifa 32000, Israel

Received 4 July 1994; revised 8 November 1994

Abstract

Given a set of k points (sites) $S \subset \mathbb{R}^d$, and a finite d -dimensional grid $G_n^d = \{1, n\}^d$, we describe an efficient algorithm computing the mapping $D: G_n^d \rightarrow \{1, \dots, k\}$ such that $D(p)$ is the index of the site closest to p under the Euclidean norm. This mapping is called the *Voronoi labelling* of the grid. The algorithm traverses the points of G on a “locality-preserving” space-filling curve, exploiting the correlation between the value of D on adjacent grid points to reduce computation time. The advantage of this algorithm over existing ones is that it works in any dimension, and is general-purpose, in the sense that it is easily modified to accommodate many variants of the problem, such as non-integral sites, and computation on a subset of G . The runtimes of our algorithm in two dimensions compare with those of the existing algorithms, and are better than the few existing algorithms operating in higher dimensions.

Keywords: Voronoi diagram; Grid; Space-filling curve, Hilbert curve

1. Introduction

Given a set of k points (sites) $S = \{s_i\}_{i=1}^k \subset \mathbb{R}^d$, and a finite d -dimensional grid $G_n^d = \{1, \dots, n\}^d$, computation of the mapping $D: G_n^d \rightarrow \{1, \dots, k\}$, such that $D(p)$ is the index of the site closest to p under the Euclidean norm, is useful in a variety of applications. In two dimensions, this is a variant of the *distance transform* $DT(p) = e(p, s_{D(p)})$, a popular image morphological operation (Serra, 1982), where $e(\cdot)$ is the Euclidean norm. Computation of D is also an important ingredient of any discrete vector quantization problem, for which the optimal quantizer mapping is to the closest site. A concrete example (that originally motivated this work) is in

color quantization of the three-dimensional RGB color cube, necessary when a true-color (24-bit) image is to be displayed on a device with lower color resolution (Heckbert, 1982). The image *color representatives* are the sites, and the image histogram buckets are three-dimensional grid points. Computing D also arises in multidimensional random sampling scenarios, in which the set of samples, drawn from a continuous space, must be converted (finally) to a regular set of values on a grid. This is called *resampling* to the grid. The simplest way of doing this (piecewise constant interpolation) is to assign to each of the $N = n^d$ grid points (“pixel” in 2D, “voxel” in 3D), the value of the sample closest to it. Applications are adaptive ray-tracing in computer graphics (Painter and Sloane, 1989), and three-dimensional medical imaging (Kaufman, 1990).

Computing D is equivalent to *scan-converting* a

^{*} Corresponding author.

Voronoi diagram (Shamos and Preparata, 1989, Ch. 5), namely, explicit construction of the Voronoi diagram of S and assigning to each of the N grid points the index of the Voronoi cell containing it. We call this the *Voronoi labelling*. In two dimensions, computing the Voronoi diagram of S requires $O(k \log k)$ operations and performing N nearest-site queries requires $O(N \log k)$ operations. Constructing the Voronoi diagram in more than two dimensions is very complex, not practical in real applications (see e.g. discussion in (Klee, 1980)). In any case, explicitly computing the Voronoi diagram and then using it to perform nearest-site queries for each grid point is not an optimal way to solve the problem at hand. That technique is designed for a scenario where the query points are not known in advance. In our case they are, so more efficient algorithms are possible. Alternatives to Voronoi diagram construction for nearest-site searching are k -d trees (Friedman et al., 1977) and locally-sorted search (Heckbert, 1982). The former is considered too inefficient, and the latter is simple and popular, widely used for three-dimensional color quantization, but suboptimal, as we show later.

A variety of algorithms have been developed to compute the two-dimensional distance transform, which may be adapted to solve the two-dimensional Voronoi labelling problem, running in $O(N)$ time (Borgefors, 1986; Breu et al., 1994; Danielsson, 1980; Kolountzakis and Kiriakos, 1992; Vincent, 1991; Yamada, 1984; Paglieroni, 1992). In the interest of efficiency, some of the algorithms (e.g., Borgefors, 1986; Danielsson, 1980) compute only an *approximation* to the Euclidean Voronoi labelling, as the metric used is not true Euclidean. Many of the algorithms (Breu et al., 1994; Danielsson, 1980) perform only integer computations. Most assume that the sites are themselves grid points, so that extensions to non-integral sites are problematic, and some (Breu et al., 1994) rely on the fact that D is to be computed for the *entire* grid G_n^d , in the sense that computing D for a subset of G_n^d is equivalent in complexity to the computation of D on all G_n^d . All the efficient two-dimensional algorithms either cannot be generalized, or their performance is degraded, in higher dimensions. For example, the performance of the algorithm in (Breu et al., 1994) deteriorates from $O(N)$ to $O(N \log k)$ for $d \geq 3$.

We describe an algorithm which does not suffer from the drawbacks mentioned above. For a start, it computes the exact Euclidean Voronoi labelling. Additionally, it runs efficiently in any dimension, does not rely on the sites being grid points, and performance scales (in a sense) with the size of the subset of the grid for which D is needed. Its only drawback is that it is not an integer algorithm. However, this is to be expected if the sites are not necessarily integral grid points. The only other multidimensional distance transform algorithm known to the authors is that of Ragnemalm (1993). However, it is efficient only in two and three dimensions. In higher dimensions, an exponential (in d) number of operations must be performed for each grid point.

The ideas behind our algorithm are very simple. The multidimensional grid G_n^d is scanned in an order preserving locality, along a space-filling Hilbert curve. The preservation of locality enables the algorithm to use information on distances accumulated in computations for previous grid points when performing the calculation for the current grid point. This significantly reduces the amount of computations required.

2. Space-filling curves

A space-filling curve on the finite d -dimensional grid $G_n^d = \{1, \dots, n\}^d$ is a mapping $C_n^d: \{1, \dots, n^d\} \rightarrow G_n^d$. C_n^d defines a *traversal order* of G_n^d . We are interested in space-filling curves which preserve *locality*, in the sense that if two grid points are close in the traversal order, they will not be too far apart in G_n^d , when measured by the Euclidean metric. A possible measure of this “locality” is

$$L(C_n^d) = \max_{i, j \in \{1, \dots, n^d\}} \frac{e(C_n^d(i), C_n^d(j))^d}{|i - j|}$$

where $e(\cdot)$ is the Euclidean distance. The use of the normalization exponent d is justified, as Euclidean distances are $O(n)$, and distances along the traversal are $O(n^d)$. A family $\mathcal{E}^d = \{C_n^d\}_{n=1}^\infty$ of space-filling curves is called *locality preserving* if $L(C_n^d) \leq \alpha$ for all n , for some constant α (which might depend on d). For a discussion of these concepts, see (Gotsman and Lindenbaum, 1994). Note that the standard d -di-

mensional raster scan on G_n^d , denoted R_n^d , does not preserve locality, as $L(R_n^d) = O(n^{d-1})$. The locality-preservation property enables efficient “local” computations, and preservation of the correlation present in the multidimensional data while using the scan, which is one-dimensional by nature. This has been exploited in a number of image processing algorithms (Stevens et al., 1983; Laurini, 1985; Zhang and Webber, 1993; Lempel and Ziv, 1986).

There is a known family of space-filling curves preserving locality. These curves are called the Hilbert curves (Hilbert, 1891), denoted H_n^d , and a recursive construction of H_n^d exists for $n = 2^m$ (positive integer m) and any dimension d . This construction is described in Fig. 1, and efficient algorithms exist for its generation (Stevens et al., 1983). For grid sizes which are not powers of 2, a construction of Perez et al. (1992) provides curves with good

locality properties, similar to those of Hilbert curves. Hilbert curves preserve locality, because (Gotsman and Lindenbaum, 1994)

$$L(H_n^d) \leq (d + 3)^{d/2} 2^d.$$

3. The algorithm

Our algorithm is based on the following intuitive observation:

If the site s is closest to the grid point p then it is very likely that s is closest to p 's neighbors too.

This is especially true if the distance of p to s is much less than its distance to all the other sites, as

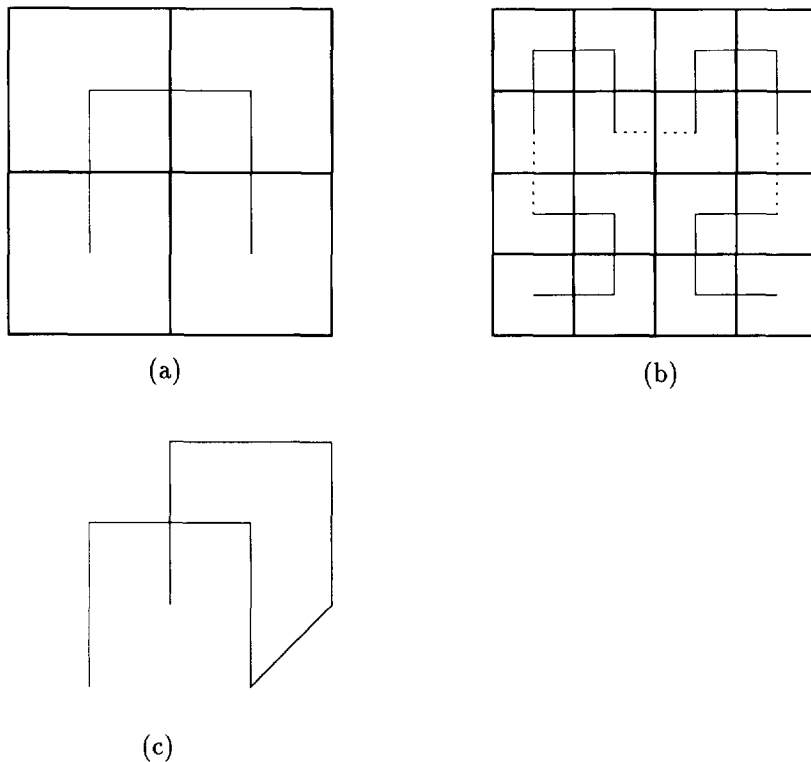


Fig. 1. Recursive construction of multidimensional Hilbert curves H_n^d . (a) Two-dimensional “seed” curve H_2^2 on a 2×2 grid. (b) Curve H_2^4 on a 4×4 grid obtained by rotating, translating and joining four copies of the seed curve H_2^2 . (c) Three-dimensional seed curve H_2^3 on a $2 \times 2 \times 2$ grid obtained by joining two copies of the two-dimensional seed curve H_2^2 . A d -dimensional seed curve H_2^d is obtained in a similar fashion from two $(d - 1)$ -dimensional seed curves H_2^{d-1} .

these distances will not change by much when p is replaced with its neighbor.

By now the essence of our algorithm is probably clear to the reader. Traverse G_n^d along a locality-preserving space-filling curve. For the first grid point of the traversal, sort the sites according to their distances from this point. At each step of the traversal, bound the distances of the current grid point from the sites using information accumulated from the previous grid points treated. Explicitly compute Euclidean distances only when needed to resolve uncertainties.

A dynamic list of k records, $\{R_i\}_{i=1}^k$, is maintained for the sites. Each record R_i is occupied by a site s . R_i contains the coordinates of a reference grid point r_i , to which the exact Euclidean distance $d_i = e(r_i, s)$ is known (having been computed sometime in the past). This distance is also stored in R_i . At the beginning we initialize by calculating the distances of all sites to the first scan grid point and sorting the list according to these distances, so that grid point becomes the initial reference of all sites. During traversal, the references r_i , and corresponding distances d_i , are updated when needed. The value of D on the first grid point is the index of the site occupying R_1 .

We say that a site is *safe* relative to grid point p if it cannot possibly be the site closest to p , based on all the information computed until now. At the beginning (after the initialization stage), all sites occupying records R_i , for $i > 1$, are safe. A dynamic pointer L marks the record in the list occupied by the first safe site, beyond which all other sites are also safe (see next paragraph for an explanation why this is true). The dynamic pointer B marks the record occupied by the site closest to the previous grid point treated, who is the primary candidate to be closest to the current grid point under consideration. Obviously, at the beginning $B = 1$, $L = 2$, and $B < L$ throughout the algorithm. The main objectives of our algorithm are to determine safety efficiently, and keep L close to the head of the list. Obviously, all safe sites need not be considered, when trying to determine the closest site.

An effective way to check the safety of the site s_1 occupying the record R_L , relative to an arbitrary grid point p , is based on the triangle inequality (see Fig. 2). For any $1 \leq i \leq k$, denote by Δ_i the l_1 distance

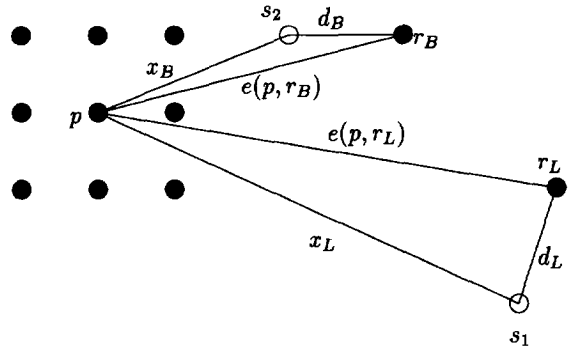


Fig. 2. Bounding grid point-site distances during grid traversal.

between p and r_i (which is easy to calculate), and by x_i the (unknown) Euclidean distance between p and the site occupying R_i . Since always $\Delta_i \geq e(p, r_i)$, if $d_L - \Delta_L > d_B + \Delta_B$, this implies that $d_L - e(p, r_L) > d_B + e(p, r_B)$, in turn implying (by the triangle inequality) $x_L > x_B$. This means that at least s_2 , the site occupying R_B , is closer to p than s_1 , hence s_1 is safe. Throughout the algorithm we maintain the list such that all sites occupying the records R_i for $i \geq L$ have the same reference grid point r at increasing distances from it, i.e. $\Delta_i = \Delta_{i+1}$ and $d_i \leq d_{i+1}$ for $L \leq i \leq k - 1$. It is easy to see that in this case, since $d_i - \Delta_i \geq d_L - \Delta_L > d_B + \Delta_B$, for $i > L$, the safety of the site occupying R_L implies the safety of all sites occupying the records R_i , $i > L$.

A site s occupying R_i is said to be *reevaluated* if we calculate its exact Euclidean distance x_i to the current grid point (in this case, the current grid point becomes the reference r_i). For all sites occupying the first L records, we test for safeness using the condition mentioned above. Any site failing the test is said to be *unsafe*, and is reevaluated. If, after reevaluation, $d_i (= x_i)$ is less than $d_B (= x_B)$, B points to R_i . If the site occupying R_L was found to be unsafe, all sites in the records beyond it are checked for safeness and reevaluated if needed. B is updated as above. This continues, L being shifted as we go, until a safe site is encountered (beyond which, again, all the remaining sites are safe).

The hope is that most sites will be safe, so that L stays relatively close to the list head throughout the process. If L strays too far, it is better to reevaluate the entire site list, sort it, set $B = 1$, $L = 2$, and “start over”. This threshold value for L (called

limit) is a heuristic parameter. Experimentally, we found $\text{limit} = k/10$ to be close to optimal in two dimensions, and $\text{limit} = k/3$ in three dimensions. Pseudo-code of our algorithm appears in Fig. 3.

4. Implementation details

When calculating distances for sorting purposes, the squares of the distances suffice, so no square root operation is required. However, testing for safeness requires exact Euclidean distances, which involves the use of the square root function. This is used sparingly, deferred until really needed.

If a subset of the grid points do not require labelling, it is simply skipped over during the Hilbert scan, the process continuing as described in the previous section.

In most applications (such as two-dimensional image processing and three-dimensional color quantization), the dimension d and size $N = n^d$ of the grid are fixed, only the sites changing between application of the algorithm. In this case, the Hilbert curve H_n^d may be precomputed once and stored.

5. Experimental results

When comparing our algorithm to others, a few words of caution are in order. All the existing two-dimensional algorithms claim to run in $O(N)$ time. However, in practical implementations (finite values of N), this statement is almost meaningless, as most image morphological operations run in $O(N)$ time, except that some run in seconds, and others in hours (Vincent, 1991). The only concrete published information on serial algorithm runtimes is supplied by Vincent (1991), who mentions that the average runtime of his *distance transform* algorithm is 2 seconds on a 256×256 pixel image on a SparcStation 1, whereas the Danielsson (1980) algorithm (which is not even Euclidean) requires double that time, and the Yamada (1984) algorithm is even slower. Vincent does not mention what the effective values of k were in his experiments. Gil (1993) has communicated that the algorithm of (Breu et al., 1994) runs in 1.3 seconds on a Sparc-Station 1 for a 256×256 pixel image with $k = 100$ sites. Paglieroni (1992)

```

SITE_RECORD = record
  x,y,id; // site coordinates and ID
  rx,ry; // reference grid point coordinates
  d; // reference distance from site
end
SITE_LIST: array[0..MAX] of SITE_RECORD;
reevaluate(p,cx,cy)
begin
  SITE_LIST[p].d :=
    sqr_Euclid_dist((cx,cy),SITE_LIST[p].site);
  SITE_LIST[p].rx := cx;
  SITE_LIST[p].ry := cy;
end
sort_sites(x,y)
begin
  for i=0 to k-1 do
    reevaluate(i,x,y);
  sort SITE_LIST in increasing order of field d.
  SITE_LIST[0].d = sqrt(SITE_LIST[0].d);
  SITE_LIST[1].d = sqrt(SITE_LIST[1].d);
end
voronoi_label()
begin
  limit := k/10;
  (cx,cy) := first pixel of Hilbert scan;
  while (Hilbert scan not finished) do
  begin
    sort_sites(cx,cy);
    B := 0;
    L := 1;
    while (L not past limit) do
    begin
      D[cx][cy] := SITE_LIST[B].id;
      if (Hilbert scan finished) break;
      (cx,cy) := next pixel along Hilbert scan;
      B_bd := SITE_LIST[B].d + |SITE_LIST[B].rx-cx|
        + |SITE_LIST[B].ry-cy|;
      p := 0;
      while (p < L) do
      begin
        if (p > B)
          if (SITE_LIST[p].d - |SITE_LIST[p].rx-cx|
            - |SITE_LIST[p].ry-cy| < B_bd)
          begin
            reevaluate(p,cx,cy);
            reevaluate(B,cx,cy);
            if (SITE_LIST[p].d < SITE_LIST[B].d) then
              B := p;
            end
            p := p+1;
          end
        while (L not past limit and
          SITE_LIST[L].d - |SITE_LIST[L].rx-cx|
            - |SITE_LIST[L].ry-cy| < SITE_LIST[B].d)
        begin
          reevaluate(L,cx,cy);
          reevaluate(B,cx,cy);
          if (SITE_LIST[L].d < SITE_LIST[B].d) then
            B := L;
            L := L+1;
            SITE_LIST[L].d := sqrt(SITE_LIST[L].d);
          end
        end
      end
    end
  end
end

```

Fig. 3. The two-dimensional version of the Voronoi-labelling algorithm.

Table 1
Algorithm runtimes (seconds) on a 256×256 grid on SparcStation 1 for Hilbert and raster scans

k	Hilbert		Raster	
	limit	Time	limit	Time
50	10	2.1	20	4.9
100	15	3.1	40	8.1
150	20	3.9	60	11.3
200	20	4.8	100	14.5
250	25	5.4	140	17.6
300	35	6.3	180	20.6

reports that the parallel version of his algorithm runs in 15 seconds using 10 MIP Sun 4 processors when computing the distance transform of a 512×512 pixel image.

To compare our algorithm, we implemented and ran it on a 256×256 grid on a SparcStation 1. The k sites were chosen randomly. The runtimes are summarized in Table 1, along with the optimal values of `limit` for those values of N and k . The runtime varies with k between 2.1 and 6.3 seconds, which is competitive with the Vincent algorithm.

For three-dimensional grids, where we believe our algorithm has a significant advantage over others, we compared with the popular locally sorted search (Heckbert, 1982), which we implemented. The results are summarized in Table 2. It shows that our algorithm is on the average twice as fast.

To determine the contribution of the Hilbert scan to the speedup, we compared the runtimes with an equivalent algorithm using a raster scan. The results (detailed in Tables 1 and 2) show that the Hilbert scan is responsible for a speedup by an average

Table 2
Algorithm runtimes (seconds) on a $32 \times 32 \times 32$ grid on SparcStation 1 for Hilbert and raster scan versions of our algorithm vs. the locally sorted search method

k	Hilbert		Raster		Local Sort
	limit	Time	limit	Time	
50	20	3.8	50	8.8	8.1
100	40	5.9	60	16.3	11.7
150	50	7.9	80	23.4	14.4
200	70	9.3	100	30.5	17.5
250	80	11.1	120	39.5	20.3
300	100	12.8	140	44.8	22.3

factor of 3 in two dimensions, and up to 3.5 in three dimensions. The advantage is more significant in three dimensions probably because the notion of locality-preservation becomes stronger as d increases.

6. Conclusion

We have described a general purpose Voronoi-labelling algorithm for multidimensional grids. Apart from being extremely simple to implement (50 lines of C code), it produces exact Euclidean Voronoi diagrams, accomodates non-integral sites, and scales gracefully if the computation is required for only a subset of the grid.

Our algorithm may be modified trivially to compute the Voronoi labelling under any l_p norm, for $p \geq 1$. The only modification required is the computation of the true l_p distance instead of the Euclidean l_2 distance during reevaluation of a site. Since this computation may be cheaper than the square root operation required for $p = 2$ (especially for the popular cases $p = 1, \infty$), the algorithm is expected to run much faster in those cases than the times presented here.

A Voronoi diagram of order p of a set of sites $S = \{s_i\}_{i=1}^k \subset \mathbb{R}^d$ is the partitioning of \mathbb{R}^d into convex cells such that all points in a cell have the same subset of S for their p closest sites (the standard Voronoi diagram is first order $p = 1$). Higher-order Voronoi diagrams, especially *furthest-neighbor* diagrams ($p = k - 1$), arise naturally in a variety of applications (see Shamos and Preparata, 1989, Ch. 6). Our algorithm may be extended easily (and naturally) to compute Voronoi diagrams of order p . A list of sites is maintained, along with p dynamic pointers $(L_i)_{i=1}^p$ marking the positions in the list beyond which safety, defined analogously to the first-order case, is guaranteed for the i th closest site.

References

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

- Breu, H., J. Gil, D. Kirkpatrick and M. Werman (1994). Linear time Euclidean distance transform and Voronoi diagram algorithms. *IEEE Trans. Pattern Anal. Mach. Intell.*, to appear.
- Danielsson, P.E. (1980). Euclidean distance mapping. *Computer Graphics and Image Processing* 14, 227–248.
- Friedman, J.H., J.L. Bentley and R.A. Finkel (1977). An algorithm for finding best matches in logarithmic expected time. *ACM Trans. Math. Software* 3 (3), 209–226.
- Gil, J. (1993). Personal Communication.
- Gotsman, C. and M. Lindenbaum (1994). On the metric properties of space-filling curves. *Proc. Internat. Conf. on Pattern Recognition*. IEEE, Press, New York, Vol. 3, 98–102.
- Heckbert, P. (1982). Color image quantization for frame buffer display. *Computer Graphics (SIGGRAPH)* 16 (3), 297–307.
- Hilbert, D. (1891). Ueber die stetige Abbildung einer Linie auf ein Flächenstück. *Math. Ann.* 38, 459–460.
- Kaufman, A. (1990). *Volume Visualization*. IEEE Computer Society Press, Los Alamitos, CA.
- Klee, V. (1980). On the complexity of d -dimensional Voronoi diagrams. *Archiv Math.* 34, 75–80.
- Kolountzakis, M.N. and N.K. Kiriakos (1992). Fast computation of the Euclidean distance maps for binary images. *Inform. Process. Lett.* 43 (4), 181–184.
- Laurini, R. (1985). Graphical databases built on Peano space-filling curves. *Proc. of Eurographics '85*, North-Holland, Amsterdam, 327–338.
- Lempel, A. and J. Ziv. Compression of two dimensional data. *IEEE Trans. Inform. Theory* 32 (1), 2–8.
- Paglieroni, D.W. (1992). A unified distance transform algorithm and architecture. *Machine Vision Appl.* 5, 47–55.
- Painter, J. and K. Sloane (1989). Antialiased ray-tracing by adaptive progressive refinement. *Computer Graphics (SIGGRAPH)* 23 (3), 281–288.
- Perez, A., S. Kamata, and E. Kawaguchi (1992). Peano scanning of arbitrary size images. *Proc. Internat. Conf. on Pattern Recognition*. IEEE Press, New York.
- Ragnemalm, I. (1993). The Euclidean distance transform in arbitrary dimensions. *Pattern Recognition Lett.* 14, 883–888.
- Serra, J. (1982). *Image Analysis and Mathematical Morphology*. Academic Press, London.
- Shamos, M.I. and F.P. Preparata (1989). *Computational Geometry*. Springer, Berlin.
- Stevens, R.J., A.F. Lehar, and F.H. Preston (1983). Manipulation and presentation of multidimensional image data using the Peano scan. *IEEE Trans. Pattern Anal. Machine Intell.* 5 (5), 520–526.
- Vincent, L. (1991). Exact Euclidean distance function by chain propagations. *Proc. Internat. Conf. on Computer Vision and Pattern Recognition*. IEEE Press, New York, 520–525.
- Yamada, H. (1984). Complete Euclidean distance transformation by parallel operations. *Proc. Internat. Conf. on Pattern Recognition*, 69–71.
- Zhang, Y., and R.E. Webber (1993). Space diffusion: An improved parallel halftoning technique using space-filling curves. *Computer Graphics Proc., Ann. Conf. Series*. ACM, New York, 305–312.