Short Papers

A Linear Time Algorithm for Computing Exact Euclidean Distance Transforms of Binary Images in Arbitrary Dimensions

Calvin R. Maurer, Jr., *Member*, *IEEE*, Rensheng Qi, and Vijay Raghavan, *Senior Member*, *IEEE*

Abstract—A sequential algorithm is presented for computing the exact Euclidean distance transform (DT) of a *k*-dimensional binary image in time linear in the total number of voxels N. The algorithm, which is based on dimensionality reduction and partial Voronoi diagram construction, can be used for computing the DT for a wide class of distance functions, including the L_p and chamfer metrics. At each dimension level, the DT is computed by constructing the intersection of the Voronoi diagram whose sites are the feature voxels with each row of the image. This construction is performed efficiently by using the DT in the next lower dimension. The correctness and linear time complexity are demonstrated analytically and verified experimentally. The algorithm may be of practical value since it is relatively simple and easy to implement and it is relatively fast (not only does it run in $O(N)$ time but the time constant is small). A simple modification of the algorithm computes the weighted Euclidean DT, which is useful for images with anisotropic voxel dimensions. A parallel version of the algorithm runs in $O(N/p)$ time with p processors.

Index Terms—Euclidean distance transform, closest feature transform, Voronoi diagram.

1 Introduction

A k-dimensional (k-D) binary image is a function I from the elements of an $n_1 \times \ldots \times n_k$ array to $\{0, 1\}$. The elements are called pixels when $k = 2$ and voxels when $k \geq 3$. Voxels of value 0 and 1 are called background voxels and foreground or feature voxels (FVs), respectively. For a given distance metric, the *distance transform* (DT) of an image I is an assignment to each voxel $\mathbf{x}$ of the distance between $\mathbf{x}$ and the closest feature voxel (CFV) in I. The *closest feature transform* (FT), also known as the *nearest feature transform* and the *nearest neighbor transform*, of an image I is an assignment to each voxel $\mathbf{x}$ of the identity of the CFV in I. It is clear that a DT can be computed from a FT in time linear in the total number of voxels $N = n_1 \times \ldots \times n_k$.

DTs are useful for a variety of image processing and computer vision applications [25]. Examples include nearest-neighbor interpolation, planar tesselation, morphological image processing (e.g., thinning, thickening, and skeletonization), pattern matching (e.g., object detection and stereo feature matching), robot collision avoidance, and path finding. DTs are widely used in medical image processing. For example, in surface-based image registration, the DT of a binary image in which the FVs represent a surface provides a convenient and efficient method for precomputing and storing point-to-surface distance [12], [14], [19], [39], [40]. DTs have also been used in nonrigid image registration [22], [27], morphological image segmentation [15], volume visualization (distance-based acceleration techniques for ray-tracing) [37], [49], and shape-based interpolation [16], [32].

The Euclidean DT (EDT) is the DT for which the metric is Euclidean distance. Sometimes the EDT is used, but often, even when an exact EDT is desired, an approximation of the EDT such as the chamfer DT is used because it is substantially faster to compute. For some applications, an exact EDT is required. For example, various approximations of the EDT have been used to generate skeletons of binary objects [1], [24], but only the exact EDT can produce an accurate skeleton that is reversible (i.e., allows exact reconstruction of the object), rotation invariant, and minimal [13]. The 3D EDT has recently been used to generate skeletons of radiosurgical targets (e.g., brain tumors) for treatment planning and optimization in multiisocentric stereotactic radiosurgery [45], [46].

The exact EDT of a k-D image can obviously be computed by using a brute-force exhaustive search for CFVs, which is a procedure that inherently requires $O(N^2)$ time. Algorithms for computing the exact EDT of a 2D image have been reported that require $O(N^{3/2})$ time [25], [26], and $O(N \log N)$ time [20]. It is clear that the exact EDT of a k-D image can be computed in $O(N \log N)$ time by calculating the Voronoi diagram of the FVs [6], [29]. A relatively efficient, but not linear time, algorithm for computing the EDT of a k-D image was achieved using gray-scale morphology decomposition [18].

A substantial amount of effort has been devoted to algorithms that compute the DT by propagation. One type of algorithm propagates distance values from voxels to neighboring voxels. Algorithms that require $O(N)$ time for computing the DT of a k-D image for the L_1, L_∞, and chamfer (a weighted L_1 metric that provides an approximation to the L_2 metric) metrics with a two-pass sequential raster scan are well-known [4], [5], [25], [26], [34], [35], [48]. The weights in the kernel (propagation mask) can be chosen to optimize the approximation of a chamfer DT to the EDT [3], [5], [33], [38], [41], [44]. This type of approach has also been used to compute constrained DTs, which are DTs where distance is defined only within a specified constraint region [28], [42]. An alternative type of algorithm propagates the position of the CFV (a vector pointing to the CFV) rather than distance (a scalar). This idea was originally proposed as a sequential raster-scan algorithm for 2D images [9] and was later generalized to arbitrary dimensions [31] and images with anisotropic voxel dimensions [23]. These raster-scan algorithms require $O(N)$ time and produce an approximate EDT with small error (the maximum error in 2D is less than one pixel [9]). An alternative to raster-scan propagation is ordered propagation. In these algorithms, the information (distance [28], [42] or a vector pointing to the CFV [8], [10], [30], [43]) is propagated from each voxel to its neighbors, starting from the contours of the object and using a dynamic list to store the pixels in the propagation front. Standard propagation algorithms provide only approximate EDTs because the tiles of the Voronoi diagram on a discrete lattice are not necessarily connected sets (the tiles on a continuous plane are connected sets). Parallel algorithms can overcome this limitation and provide exact EDTs by allowing multiple propagation fronts to follow each other [11], [18], [47]. Ordered propagation can provide an exact EDT by emulating multiple-front propagation [10], [23], [30], but the multiple propagation mechanics common to these exact EDT algorithms require many unnecessary computations while successive propagation fronts reach the same voxel and update its value, and these algorithms require $O(N^{3/2})$ time for some 2D images [8]. One recent algorithm first computes an approximate EDT using an ordered propagation scheme and then produces an exact EDT by restoring the connectivity of the Voronoi tiles by considering larger neighborhoods on the tile boundaries [8] (a similar approach using raster-scan propagation has also been reported [7]). This is the only propagation algorithm we are aware of that produces an exact EDT in apparently $O(N)$ time, but the time complexity was verified only experimentally, and extension to multidimensional or anisotropic images has not been reported.

- *C.R. Maurer, Jr. is with Image Guidance Laboratories, Department of Neurosurgery, Stanford University, 300 Pasteur Drive, MC 5327, Room S-012, Stanford, CA 94305-5327. E-mail: calvin.maurer@igl.stanford.edu.*
- *R. Qi is with Imaging Diagnostic Systems, Inc., Plantation, FL 33313. E-mail: qi@imds.com.*
- *V. Raghavan is with the Department of Electrical Engineering and Computer Science, Vanderbilt University, Nashville, TN 37235. E-mail: raghavan@vuse.vanderbilt.edu.*

Manuscript received 20 Mar. 2001; revised 27 May 2002; accepted 23 July 2002. Recommended for acceptance by H. Christensen.
For information on obtaining reprints of this article, please send e-mail to: tpami@computer.org, and reference IEEECS Log Number 113833.

Saito and Toriwaki [36] presented an algorithm for computing the exact EDT of a 2D or 3D image that is based on dimensionality reduction, that is, at each dimension level, the EDT is determined by using the EDT in the next lower dimension. They used some geometrical bounds to reduce search intervals. The computational cost of their algorithm is image dependent and requires $O(N^{3/2})$ time for some 2D images; this time complexity has been observed experimentally by us (unpublished) and others [7], [8]. Breu et al. [6] presented an algorithm which also uses the idea of dimensionality reduction for computing the Euclidean FT of a 2D image in $O(N)$ time by constructing the intersection of the Voronoi diagram whose sites are the FVs with each row of the image. Then, the EDT is computed from the FT. Guan and Ma [17] improved the computational performance of the Breu approach by taking advantage of the fact that neighboring voxels tend to have the same CFVs. They propagate CFV information in the form of segment lists rather than individual voxels.

In this paper, we first present an algorithm for computing the FT of a k-D binary image in $O(N)$ time. This algorithm can be used for a wide class of distance functions, including Euclidean distance as well as the L_p and chamfer metrics. We then present an algorithm for computing, directly, the EDT in arbitrary dimensions that runs in $O(N)$ time. This algorithm is in a sense a generalization of the Breu algorithm to arbitrary dimensions. Like Guan and Ma, we use properties of the L_2 metric to improve computational performance with respect to Breu. Unlike both Breu and Guan and Ma, we compute the EDT directly rather than first compute the FT; as a result, our algorithm is an order of magnitude faster than their algorithms. We believe this is the first algorithm that computes the exact EDT of a k-D image in $O(N)$ time without first computing a FT and for which the correctness and time complexity are formally verified. This algorithm may be of practical value since it is relatively simple and easy to implement and it is relatively fast (not only does it run in $O(N)$ time but the time constant is small). A simple modification of the algorithm computes the weighted Euclidean DT, which is useful for images with anisotropic voxel dimensions. A parallel version of the algorithm runs in $O(N/p)$ time with p processors, $1 \leq p \leq \min(n_1, \ldots, n_k)$.

2 Distance Functions and Properties

We are interested in the L_p distance metric

$$\Delta(\mathbf{x}, \mathbf{y}) = \left(\sum_{i=1}^{k} |x_i - y_i|^p \right)^{1/p},$$

where $\mathbf{x}$ and $\mathbf{y}$ are k-tuples, x_i and y_i are the ith coordinates of $\mathbf{x}$ and $\mathbf{y}$, and $1 \leq p \leq \infty$. The L_1, L_2, and L_∞ metrics are known as the Manhattan or city-block, Euclidean, and chessboard distances, respectively. We are also interested in the weighted L_p distance metric

$$\Delta(\mathbf{x}, \mathbf{y}) = \left(\sum_{i=1}^{k} |w_i(x_i - y_i)|^p \right)^{1/p}, \quad (1)$$

where w_i is the weight of the ith coordinates of $\mathbf{x}$ and $\mathbf{y}$. This is useful for images with anisotropic voxel dimensions. For example, medical 3D image volumes are frequently stacks of 2D image slices where the slice thickness (w_3) is different from the in-slice pixel size ($w_1 = w_2$). We are interested specifically in Euclidean and weighted Euclidean distance. We are interested more generally in distance metrics $\Delta : \mathcal{R}^k \times \mathcal{R}^k \rightarrow \mathcal{R}$ that satisfy the following properties:

Property 1. *Positive definiteness.* $\Delta(\mathbf{x}, \mathbf{y}) = 0$ *iff* $\mathbf{x} = \mathbf{y}$.

Property 2. *Symmetry.* $\Delta(\mathbf{x}, \mathbf{y}) = \Delta(\mathbf{y}, \mathbf{x})$ *for any* $\mathbf{x}$ *and* $\mathbf{y}$.

Property 3. *Triangle inequality.* $\Delta(\mathbf{x}, \mathbf{z}) \leq \Delta(\mathbf{x}, \mathbf{y}) + \Delta(\mathbf{y}, \mathbf{z})$ *for any* $\mathbf{x}$, $\mathbf{y}$, *and* $\mathbf{z}$.

Property 4. *Monotonicity. Let* $\mathbf{x}$ *and* $\mathbf{y}$ *be two* k-tuples *that differ only in the values of the* d*th coordinates (i.e.,* $x_i = y_i$, $i \neq d$*). For concreteness, assume that* $x_d < y_d$*. For any* $\mathbf{u}$ *and* $\mathbf{v}$ *such that either 1)* $\Delta(\mathbf{x}, \mathbf{u}) \leq \Delta(\mathbf{x}, \mathbf{v})$ *and* $\Delta(\mathbf{y}, \mathbf{v}) < \Delta(\mathbf{y}, \mathbf{u})$ *or 2)* $\Delta(\mathbf{x}, \mathbf{u}) < \Delta(\mathbf{x}, \mathbf{v})$ *and* $\Delta(\mathbf{y}, \mathbf{v}) \leq \Delta(\mathbf{y}, \mathbf{u})$ *holds,* $u_d < v_d$.

Property 5. *Let* $\mathbf{x}$ *and* $\mathbf{y}$ *be two* k-tuples *that differ only in the values of the* d*th coordinates (i.e.,* $x_i = y_i$, $i \neq d$*). Let* $\mathbf{u}$ *and* $\mathbf{v}$ *be two* k-tuples *with identical values of the* d*th coordinates (i.e.,* $u_d = v_d$*). If* $\Delta(\mathbf{x}, \mathbf{u}) \leq \Delta(\mathbf{x}, \mathbf{v})$*, then* $\Delta(\mathbf{y}, \mathbf{u}) \leq \Delta(\mathbf{y}, \mathbf{v})$.

It is well known that the L_p, weighted L_p, and chamfer metrics satisfy Properties 1-4. Property 5 follows from the contrapositive of Property 4. Thus, any distance function that satisfies Property 4 also satisfies Property 5. Property 5 is not an independent property, but it is convenient to include for later reference.

In the next section, we give an algorithm for computing the FT of a k-D binary image for any distance metric that satisfies Properties 1, 2, 3, 4, and 5. This algorithm is not appropriate for all distance functions. For example, consider the distance function $\Delta(\mathbf{x}, \mathbf{y}) = |x_1 - y_1||x_2 - y_2|$, which is given as an example in [26]. This function satisfies Properties 2 and 5 but not 1, 3, and 4.

3 The Feature Transform Algorithm

The Voronoi diagram $\mathcal{V}_S$ of a set of Voronoi sites $S = \{\mathbf{f}_i\}$ for $i = 1, \ldots, n_S$ consists of a set of disjoint Voronoi cells $\mathcal{V}_S = \{\mathcal{C}_{\mathbf{f}_i}\}$ for $i = 1, \ldots, n_S$ [2], [29]. The Voronoi cell $\mathcal{C}_\mathbf{f}$ is the set of all points whose closest point is $\mathbf{f}$ together with the cell boundary formed by points equidistant from $\mathbf{f}$ and one or more other Voronoi sites. The Voronoi site $\mathbf{f}$ is also known as the Voronoi center of $\mathcal{C}_\mathbf{f}$. The FT of a binary image can be thought of as a discretized version of the Voronoi diagram whose Voronoi sites are the FVs of the image. If the complete Voronoi diagram is constructed, the FT can be computed easily by querying the Voronoi diagram, i.e., by recording for each voxel the Voronoi site of the Voronoi cell in which the voxel lies. In this algorithm, we do not construct the complete Voronoi diagram. Instead, our approach is based on the idea of dimensionality reduction and partial Voronoi diagram construction. At each dimension level, the FT is determined by constructing directly the intersection of the Voronoi diagram whose Voronoi sites are the FVs with each row of the image. This construction is performed efficiently by using the FT in the next lower dimension.

The algorithm takes as input the k-D binary image $I : [1..n_1] \times \ldots \times [1..n_k] \rightarrow \{0, 1\}$ and outputs the FT

$$F : [1..n_1] \times \ldots \times [1..n_k] \rightarrow [1..n_1] \times \ldots \times [1..n_k].$$

For each voxel $\mathbf{x}$ in I, $F(\mathbf{x})$ is the CFV in I. If two or more FVs are equidistant from $\mathbf{x}$, then one of them is chosen arbitrarily to be $F(\mathbf{x})$. The DT can be computed easily from F. Let $I_{d,\mathbf{x}}$ denote the d-dimensional subimage that is the restriction of I to the subspace whose last $k - d$ coordinates are identical to the corresponding coordinates of $\mathbf{x}$. Let F_d denote the FT at the dth dimension level, where for each voxel $\mathbf{x}$ in I, $F_d(\mathbf{x})$ is the CFV in $I_{d,\mathbf{x}}$. If two or more FVs are equidistant from $\mathbf{x}$, then one of them is chosen arbitrarily to be $F_d(\mathbf{x})$. Obviously $I_{k,\mathbf{x}} = I$ and $F_k = F$. We define $F_0(\mathbf{x}) = \mathbf{x}$ if $I(\mathbf{x}) = 1$, $\emptyset$ otherwise, where $\emptyset$ denotes an undefined CFV. Consider, for example, a 3D binary image I. For each voxel $\mathbf{x}$ in I, $F_1(\mathbf{x})$ is the CFV in $I_{1,\mathbf{x}}$, which is the image row containing $\mathbf{x}$; $F_2(\mathbf{x})$ is the CFV in $I_{2,\mathbf{x}}$, which is the image plane containing $\mathbf{x}$; and $F_3(\mathbf{x}) = F(\mathbf{x})$ is the CFV in the image $I_{3,\mathbf{x}} = I$.

Let $X_d = \{\mathbf{x}_i\}$ for $i = 1, \ldots, n_d$ denote the set of n_d voxels in I formed by varying the dth coordinate from 1 to n_d and fixing all other coordinates. Let $\mathcal{R}_d$ denote the "row" (the continuous line) running through the set of voxels X_d. There are $n_1 \times \ldots \times n_{d-1} \times n_{d+1} \times \ldots \times n_k = N/n_d$ such rows. Let S_d denote the set of FVs in the binary subimage $I_{d,\mathbf{x}_i}$ (all voxels $\mathbf{x}_i$ in the set X_d belong to the same subimage). Let $\mathcal{V}_d^* = \mathcal{V}_{S_d} \cap \mathcal{R}_d$ denote the intersection of the Voronoi diagram $\mathcal{V}_{S_d}$ whose Voronoi sites are the set of FVs S_d with the row $\mathcal{R}_d$. Let $S'_d = \{F_{d-1}(\mathbf{x}_i)\}$ denote the set of CFVs in the next lower dimension for the set of voxels $X_d = \{\mathbf{x}_i\}$ on the row $\mathcal{R}_d$. As stated above, if two or more FVs are equidistant from $\mathbf{x}$, then one of them is chosen arbitrarily to be $F_{d-1}(\mathbf{x})$. Thus, S'_d has at most one FV for each voxel $\mathbf{x}_i$. Clearly, $S'_d \subseteq S_d$.

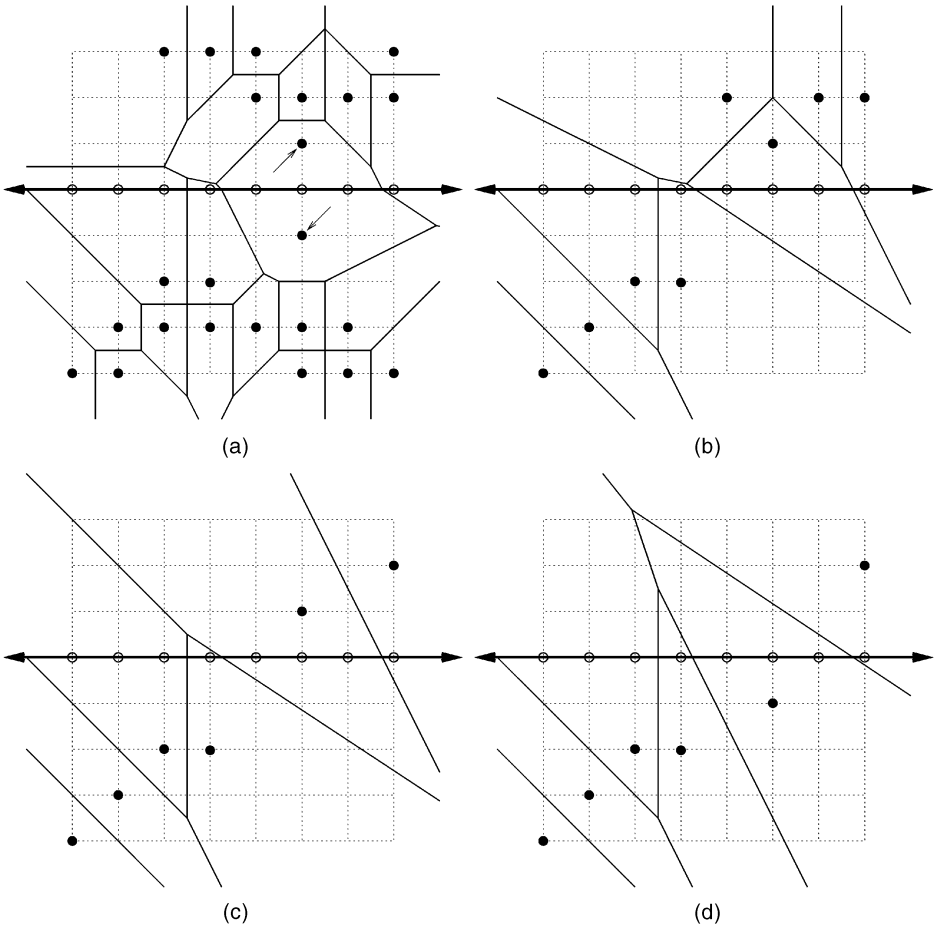

Fig. 1. Illustrative example of the relationship among S_d, S'_d, and S^*_d. In all panels, the dotted lines represent the grid of an 8×8 binary image, the open circles represent $X_d = \{\mathbf{x}_i\}$, the thick line running through the open circles represents the row $\mathcal{R}_d$, the filled circles represent FVs, the thin lines represent the Voronoi cell boundaries of the Voronoi diagram whose Voronoi sites are the FVs, and $d = k = 2$. (a) The FVs shown are the set S_d. (b) The FVs shown are the set $S'_d = \{F_{d-1}(\mathbf{x}_i)\}$, i.e., the set of CFVs in the next lower dimension for the set of voxels $\{\mathbf{x}_i\}$ on the row $\mathcal{R}_d$. In this example, $F_{d-1}(\mathbf{x}_i)$ is the CFV in the same column as $\mathbf{x}_i$. The two FVs indicated by arrows in the top left panel are equidistant from $\mathcal{R}_d$. The top one was chosen arbitrarily. (c) The FVs shown are the set S^*_d. (d) The FVs shown are the set S^*_d obtained by choosing the bottom of the two equidistant FVs in the top left panel. Note that $S^*_d \subseteq S'_d \subseteq S_d$ and $\mathcal{V}^*_d = \mathcal{V}_{S_d} \cap \mathcal{R}_d = \mathcal{V}_{S'_d} \cap \mathcal{R}_d = \mathcal{V}_{S^*_d} \cap \mathcal{R}_d$. This is true for both choices of the two equidistant FVs.

Remark 1. Let $\mathbf{f} = F_{d-1}(\mathbf{x})$, where $\mathbf{x}$ is a voxel on the row S_d. Clearly, the FV $\mathbf{f}$ belongs to the set S_d. Let $\mathbf{g}$ be any other FV belonging to the set S_d such that $\mathbf{f}$ and $\mathbf{g}$ have identical values of the dth coordinate (i.e., $f_d = g_d$). From the definition of F_d, $\Delta(\mathbf{x}, \mathbf{f}) \leq \Delta(\mathbf{x}, \mathbf{g})$. Let $\mathbf{y}$ be any other point on the row $\mathcal{R}_d$. By Property 5, $\Delta(\mathbf{y}, \mathbf{f}) \leq \Delta(\mathbf{y}, \mathbf{g})$, i.e., all points on the row $\mathcal{R}_d$ are at least as close to $\mathbf{f}$ as they are to $\mathbf{g}$. This means that either the Voronoi cell for site $\mathbf{g}$ does not intersect $\mathcal{R}_d$ ($\Delta(\mathbf{x}, \mathbf{f}) < \Delta(\mathbf{x}, \mathbf{g})$), or that the Voronoi cells for sites $\mathbf{f}$ and $\mathbf{g}$ both intersect $\mathcal{R}_d$ along a common boundary ($\Delta(\mathbf{x}, \mathbf{f}) = \Delta(\mathbf{x}, \mathbf{g})$) and the choice of $\mathbf{f}$ or $\mathbf{g}$ is arbitrary. Since all FVs in the set S_d are either in the set S'_d or have the same dth coordinate as a FV in the set S'_d, $\mathcal{V}^*_d = \mathcal{V}_{S_d} \cap \mathcal{R}_d = \mathcal{V}_{S'_d} \cap \mathcal{R}_d$. Thus, to construct $\mathcal{V}^*_d$, it is sufficient to consider the set S'_d (rather than the larger set S_d).

Let S^*_d denote the subset of S_d that are the Voronoi sites (FVs) of Voronoi cells in $\mathcal{V}^*_d$, i.e., that are the Voronoi sites of Voronoi cells in $\mathcal{V}_{S_d}$ that intersect $\mathcal{R}_d$. Clearly, $S^*_d \subseteq S'_d \subseteq S_d$. Fig. 1 shows an illustrative example of the relationship among S_d, S'_d, and S^*_d.

Remark 2. Let $\mathbf{f}$ and $\mathbf{g}$ be FVs belonging to the set S^*_d. Let $\mathbf{x}$ and $\mathbf{y}$ be voxels on the row $\mathcal{R}_d$ that lie in the Voronoi cells $\mathcal{C}_\mathbf{f}$ and $\mathcal{C}_\mathbf{g}$, respectively. By Property 4, if $x_d < y_d$, then $f_d < g_d$. Also, if $f_d < g_d$, then $x_d < y_d$. Thus, $\mathcal{V}^*_d$ is a set of disjoint line segments $\mathcal{V}^*_d = \{\mathcal{C}^*_{\mathbf{f}_i}\}$. If the set of Voronoi sites (FVs) S^*_d are sorted by the dth coordinate, the associated Voronoi cells are similarly ordered. That is, as the row $\mathcal{R}_d$ is traversed from low values of the dth coordinate to high values, $\mathcal{C}^*_\mathbf{f}$ is visited before $\mathcal{C}^*_\mathbf{g}$ iff $\mathbf{f}$ precedes $\mathbf{g}$ in the ordered set S^*_d. To compute F_d for each voxel on the row $\mathcal{R}_d$, it is not necessary to actually construct $\mathcal{V}^*_d = \{\mathcal{C}^*_{\mathbf{f}_i}\}$. It is sufficient to determine the ordered set S^*_d and visit each voxel by traversing the row in dth coordinate order.

Remark 3. Let $\mathbf{u}$, $\mathbf{v}$, and $\mathbf{w}$ be three FVs belonging to the set S'_d such that $u_d < v_d < w_d$. Let $\mathbf{x}_{\overline{\mathbf{uv}}}$ denote the point on the line $\mathcal{R}_d$ that is equidistant from $\mathbf{u}$ and $\mathbf{v}$, i.e., $\Delta(\mathbf{u}, \mathbf{x}_{\overline{\mathbf{uv}}}) = \Delta(\mathbf{v}, \mathbf{x}_{\overline{\mathbf{uv}}})$, and let $(\mathbf{x}_{\overline{\mathbf{uv}}})_d$ denote the dth coordinate of this point. Let $\mathbf{x}_{\overline{\mathbf{vw}}}$ denote another point defined analogously. By Property 4 and Remark 2, $\mathcal{C}_\mathbf{v}$ does not intersect $\mathcal{R}_d$ if $(\mathbf{x}_{\overline{\mathbf{uv}}})_d > (\mathbf{x}_{\overline{\mathbf{vw}}})_d$.

The algorithm for computing the FT F from the binary image I is performed with the initial invocation COMPUTEFT(k). The algorithm variables $I, F, n_1, \ldots, n_k$ are global variables. The procedure COMPUTEFT (Fig. 2) implements dimensionality reduction using recursion. The procedure VORONOIFT (Fig. 3) constructs and queries the partial Voronoi diagram $\mathcal{V}^*_d = \mathcal{V}_{S_d} \cap \mathcal{R}_d = \mathcal{V}_{S^*_d} \cap \mathcal{R}_d$. The algorithm variable F contains successively $F_0, F_1, \ldots, F_{k-1}, F_k = F$. It contains F_{d-1} before the call to VORONOIFT and F_d upon return. As noted in Remark 2, the algorithm does not actually construct $\mathcal{V}^*_d$, but instead determines the ordered set S^*_d (VORONOIFT, lines 1-14) and queries the diagram (visits each voxel) by traversing the row in dth coordinate order (lines 18-24). The set $S^*_d = \{\mathbf{g}_l\}$ is constructed from the set $S'_d = \{\mathbf{f}_i\}$ by deleting those FVs in S'_d that are the Voronoi sites of Voronoi cells that do not intersect $\mathcal{R}_d$. As noted in Remark 1, it is sufficient to consider the set $S'_d = \{F_{d-1}(\mathbf{x}_i)\}$. This is the fundamental basis of the dimensionality reduction approach. The set S^*_d is constructed in lines 1-14. It is initialized with the first two FVs of S'_d. In the outer loop, additional FVs are added from S'_d one at a time. In the inner loop, FVs that are the Voronoi sites of

```
1.  if d = 1 then      /* Compute F_{d-1} */
2.    for i_1 ← 1 to n_1 do
3.      if I(i_1, j_2, ..., j_k) = 1 then
4.        F(i_1, j_2, ..., j_k) ← (i_1, j_2, ..., j_k)
5.      else
6.        F(i_1, j_2, ..., j_k) ← φ
7.      endif
8.    endfor
9.  else
10.   for i_d ← 1 to n_d do
11.     COMPUTEFT(d − 1, i_d, j_{d+1}, ..., j_k)
12.   endfor
13. endif
14. for i_1 ← 1 to n_1 do      /* Compute F_d */
15.   ...
16.     for i_{d-1} ← 1 to n_{d-1} do
17.       VORONOIFT(d, i_1, ..., i_{d-1}, j_{d+1}, ..., j_k)
18.     endfor
19.   ...
20. endfor
21. return
```

Fig. 2. **Procedure** ComputeFT $(d, j_{d+1}, \ldots, j_k)$.

Voronoi cells that do not intersect $\mathcal{R}_d$ are removed. This is accomplished with the procedure REMOVEFT$(\mathbf{u}, \mathbf{v}, \mathbf{w}, \mathcal{R}_d)$, which returns TRUE if $(\mathbf{x}_{\overline{\mathbf{uv}}})_d > (\mathbf{x}_{\overline{\mathbf{vw}}})_d$, FALSE otherwise (see Remark 3). Let $S'^*_d = \{\mathbf{g}_1, \ldots, \mathbf{g}_l, \mathbf{f}_{i+1}, \ldots, \mathbf{f}_{n_d}\}$ denote an intermediate set of FVs during construction. Before entering the outer loop, $S'^*_d = \{\mathbf{f}_1, \ldots, \mathbf{f}_{n_d}\} = S'_d$. It is easy to verify that at the end of the inner loop, $\mathcal{V}_{S'_d} \cap \mathcal{R}_d = \mathcal{V}_{S'^*_d} \cap \mathcal{R}_d$. It is also easy to verify that at the end of the inner loop, all Voronoi cells in $\mathcal{V}_{\{\mathbf{g}_1, \ldots, \mathbf{g}_l\}}$ intersect $\mathcal{R}_d$. Thus, after exiting the outer loop, $S'^*_d = \{\mathbf{g}_1, \ldots, \mathbf{g}_{n_S}\} = S^*_d$. In summary,

$$\mathcal{V}^*_d = \mathcal{V}_{S_d} \cap \mathcal{R}_d = \mathcal{V}_{S'_d} \cap \mathcal{R}_d = \mathcal{V}_{S'^*_d} \cap \mathcal{R}_d = \mathcal{V}_{S^*_d} \cap \mathcal{R}_d.$$

There are two subtleties in VORONOIFT, line 20 worth noting. First, although an **if** statement may seem sufficient, in fact the **while** statement is necessary. The reason is that although the FT is discrete, the Voronoi diagram $\mathcal{V}_{S^*_d}$ and the row $\mathcal{R}_d$ are continuous. All Voronoi cells in $\mathcal{V}_{S^*_d}$ intersect $\mathcal{R}_d$, but the first several cells may, for example, intersect $\mathcal{R}_d$ only where the dth coordinate is negative (an example is illustrated in Fig. 1). Also, it is possible that a Voronoi cell in $\mathcal{V}_{S^*_d}$ intersects $\mathcal{R}_d$ only between two consecutive voxels $\mathbf{x}_i$ and $\mathbf{x}_{i+1}$ and, thus, the intersection does not contain any voxel on $\mathcal{R}_d$. Second, the "$>$" symbol in the distance comparison could be replaced with the "$\geq$" symbol. When two FVs are equidistant from the voxel $\mathbf{x}$, $F_d(\mathbf{x})$ is not unique and the choice is arbitrary.

Initialization of F_0 (COMPUTEFT, lines 2-8) takes $O(N)$ time. At each dimension d, the procedure VORONOIFT is executed for each of the N/n_d rows. For each row, construction of S^*_d takes $O(n_d)$ time, since there are n_d FVs in S'_d, and each FV is added to and removed from S^*_d at most once. This assumes that calculating $\mathbf{x}_{\overline{\mathbf{uv}}}$ requires $O(1)$ time. Querying (visiting each voxel by traversing the row) simply requires $O(n_d)$ time. Thus, at each dimension, the time complexity is $O(n_d \times N/n_d) = O(N)$, and the algorithm for computing the FT of I runs in $O(N)$ time. Finally, it is clear that the DT of I can be computed from the FT in $O(N)$ time.

4 The Euclidean Distance Transform Algorithm

If the distance metric is the L_2 metric (Euclidean distance), then the procedure REMOVEFT can be implemented simply using only integer arithmetic. The distance between $\mathbf{u}$ and $\mathbf{x}_{\overline{\mathbf{uv}}}$ can be computed as

```
1.  l ← 0      /* Construct partial Voronoi diagram */
2.  for i ← 1 to n_d do
3.    x_i ← (j_1, ..., j_{d-1}, i, j_{d+1}, ..., j_k)
4.    if (f_i ← F(x_i)) ≠ φ then
5.      if l < 2 then
6.        l ← l + 1, g_l ← f_i
7.      else
8.        while l ≥ 2 and REMOVEFT(g_{l-1}, g_l, f_i, R_d) do
9.          l ← l − 1
10.       endwhile
11.       l ← l + 1, g_l ← f_i
12.     endif
13.   endif
14. endfor
15. if (n_S ← l) = 0 then
16.   return
17. endif
18. l ← 1      /* Query partial Voronoi diagram */
19. for i ← 1 to n_d do
20.   while l < n_S and Δ(x_i, g_l) > Δ(x_i, g_{l+1}) do
21.     l ← l + 1
22.   endwhile
23.   F(x_i) ← g_l
24. endfor
25. return
```

Fig. 3. **Procedure** VoronoiFT$(d, j_1, \ldots, j_{d-1}, j_{d+1}, \ldots, j_k)$.

$$\Delta^2(\mathbf{u}, \mathbf{x}_{\overline{\mathbf{uv}}}) = \Delta^2(\mathbf{u}, \mathcal{R}_d) + (u_d - (\mathbf{x}_{\overline{\mathbf{uv}}})_d)^2,$$

where

$$\Delta^2(\mathbf{u}, \mathcal{R}_d) = \sum_{i \neq d} (u_i - r_i)^2$$

is the distance between $\mathbf{u}$ and the row $\mathcal{R}_d$. Since $\mathbf{x}_{\overline{\mathbf{uv}}}$ denotes the point on $\mathcal{R}_d$ that is equidistant from $\mathbf{u}$ and $\mathbf{v}$, $\Delta^2(\mathbf{u}, \mathbf{x}_{\overline{\mathbf{uv}}}) = \Delta^2(\mathbf{v}, \mathbf{x}_{\overline{\mathbf{uv}}})$, which can be rearranged to obtain

$$(\mathbf{x}_{\overline{\mathbf{uv}}})_d = \frac{\Delta^2(\mathbf{v}, \mathcal{R}_d) - \Delta^2(\mathbf{u}, \mathcal{R}_d) + v_d^2 - u_d^2}{2(v_d - u_d)}.$$

A similar expression can be found for $(\mathbf{x}_{\overline{\mathbf{vw}}})_d$, from which it is easy to verify that the inequality $(\mathbf{x}_{\overline{\mathbf{uv}}})_d > (\mathbf{x}_{\overline{\mathbf{vw}}})_d$ is equivalent to the inequality

$$c \cdot \Delta^2(\mathbf{v}, \mathcal{R}_d) - b \cdot \Delta^2(\mathbf{u}, \mathcal{R}_d) - a \cdot \Delta^2(\mathbf{w}, \mathcal{R}_d) - abc > 0, \qquad (2)$$

where $a = v_d - u_d$, $b = w_d - v_d$, $c = w_d - u_d = a + b$. This inequality requires only eleven integer arithmetic operations for its evaluation if the squared distances between the FVs $\mathbf{u}$, $\mathbf{v}$, and $\mathbf{w}$ and the row $\mathcal{R}_d$ are known (e.g., precomputed).

The algorithm in the previous section provides a method for computing the FT of the binary image I. The DT still needs to be computed from the FT. For the L_p metric, in general, and the L_2 metric, in particular, it is possible to compute the DT directly. Let us consider the squared EDT D. For each voxel $\mathbf{x}$ in I, $D(\mathbf{x}) = \Delta^2(\mathbf{x}, F(\mathbf{x}))$ is the squared Euclidean distance between $\mathbf{x}$ and the CFV in I. By analogy with the definition of F_d in the previous section, let $D_d(\mathbf{x}) = \Delta^2(\mathbf{x}, F_d(\mathbf{x}))$. We define $D_0(\mathbf{x}) = 0$ if $I(\mathbf{x}) = 1$, ∞ otherwise. We observe that if $\mathbf{u} = F_{d-1}(\mathbf{x})$, then

$$\Delta^2(\mathbf{u}, \mathcal{R}_d) = \Delta^2(\mathbf{x}, \mathbf{u}) = \Delta^2(\mathbf{x}, F_{d-1}(\mathbf{x})) = D_{d-1}(\mathbf{x}).$$

This important observation allows us to simply modify the FT algorithm procedures COMPUTEFT and VORONOIFT to obtain

```
1.  if d = 1 then      /* Compute D_{d-1} */
2.    for i_1 ← 1 to n_1 do
3.      if I(i_1, j_2, ..., j_k) = 1 then
4.        D(i_1, j_2, ..., j_k) ← 0
5.      else
6.        D(i_1, j_2, ..., j_k) ← ∞
7.      endif
8.    endfor
9.  else
10.   for i_d ← 1 to n_d do
11.     COMPUTEEDT(d − 1, i_d, j_{d+1}, ..., j_k)
12.   endfor
13. endif
14. for i_1 ← 1 to n_1 do      /* Compute D_d */
15.   ...
16.     for i_{d-1} ← 1 to n_{d-1} do
17.       VORONOIEDT(d, i_1, ..., i_{d-1}, j_{d+1}, ..., j_k)
18.     endfor
19.   ...
20. endfor
21. return
```

Fig. 4. **Procedure** ComputeEDT($d, j_{d+1}, \ldots, j_k$).

```
1.  l ← 0      /* Construct partial Voronoi diagram */
2.  for i ← 1 to n_d do
3.    x_i ← (j_1, ..., j_{d-1}, i, j_{d+1}, ..., j_k)
4.    if (f_i ← D(x_i)) ≠ ∞ then
5.      if l < 2 then
6.        l ← l + 1, g_l ← f_i, h_l ← i
7.      else
8.        while l ≥ 2 and REMOVEEDT(g_{l-1}, g_l, f_i, h_{l-1}, h_l, i) do
9.          l ← l − 1
10.       endwhile
11.       l ← l + 1, g_l ← f_i, h_l ← i
12.     endif
13.   endif
14. endfor
15. if (n_S ← l) = 0 then
16.   return
17. endif
18. l ← 1      /* Query partial Voronoi diagram */
19. for i ← 1 to n_d do
20.   while l < n_S and g_l + (h_l − i)^2 > g_{l+1} + (h_{l+1} − i)^2 do
21.     l ← l + 1
22.   endwhile
23.   D(x_i) ← g_l + (h_l − i)^2
24. endfor
25. return
```

Fig. 5. **Procedure** VoronoiEDT($d, j_1, \ldots, j_{d-1}, j_{d+1}, \ldots, j_k$).

the squared EDT algorithm procedures COMPUTEEDT (Fig. 4) and VORONOIEDT (Fig. 5).

The algorithm for computing the squared EDT D from the binary image I is performed with the initial invocation COMPUTEEDT(k). The algorithm variable D contains successively D_0, D_1, ..., D_{k-1}, $D_k = D$. It contains D_{d-1} before the call to VORONOIEDT and D_d upon return. In VORONOIEDT, the procedure variable $f_i = D_{d-1}(\mathbf{x}_i) = \Delta^2(\mathbf{f}_i, \mathcal{R}_d)$, $g_l = \Delta^2(\mathbf{g}_l, \mathcal{R}_d)$, and h_l is the dth coordinate of $\mathbf{g}_l$. The FV deletion procedure for the squared EDT algorithm is REMOVEEDT ($\Delta^2(\mathbf{u}, \mathcal{R}_d)$, $\Delta^2(\mathbf{v}, \mathcal{R}_d)$, $\Delta^2(\mathbf{w}, \mathcal{R}_d)$, u_d, v_d, w_d), which returns TRUE if the inequality in (2) holds, FALSE otherwise.

5 Discussion

The EDT algorithm as presented above produces the squared EDT for isotropic voxels of unit dimension. All computations can be implemented in integer arithmetic. The EDT can obviously be obtained simply from the algorithm output by taking the square root of each element. The output can be scaled to account for nonunit voxel dimension. And, the algorithm can be modified easily to accommodate the weighted EDT (see (1)), e.g., for medical 3D images with anisotropic voxel dimensions. The conversion requires simply replacing the six occurrences of the coordinate variable "i" (but not the four occurrences of the subscript "i," which is an array index) in VORONOIEDT, lines 6, 8, 11, 20, and 23, by "$w_d i$," where w_d is the dth coordinate weight (see (1)).

The squared EDT algorithm has a small time constant. All of the arithmetic operations occur in VORONOIEDT, lines 8 and 20. In line 8, the procedure REMOVEEDT is essentially the evaluation of (2), which requires 11 integer arithmetic operations. In line 20, the distance comparison requires seven arithmetic operations. As explained earlier, both of these lines will execute a maximum of twice per voxel per dimension. Thus, the squared EDT algorithm requires a maximum of 36 integer arithmetic operations per voxel per dimension, not including index increment/decrement, indirection, and assignment operations. It is easier to implement COMPUTEEDT for a fixed number of dimensions (e.g., $k = 3$) by computing D_0, D_1, ..., D_{k-1}, and $D_k = D$ in consecutive separately coded loops than by recursion. We note that the computation of D_1 can be implemented more efficiently as a simple forward-and-reverse distance propagation than as a call to the procedure VORONOIEDT. This requires a maximum of eight integer arithmetic operations per voxel. Also, we note that the number of arithmetic operations in VORONOIEDT, line 20, can be reduced by taking advantage of similarities in successive computations instead of treating them as independent ones; the distance comparison requires only four arithmetic operations when l is incremented. Using these simple optimizations, the squared EDT algorithm requires a maximum of $8 + 33(k-1)$, which is 41 for $k = 2$ and 74 for $k = 3$, integer arithmetic operations per voxel, not including index increment/decrement, indirection, and assignment operations.

The squared EDT algorithm executes substantially faster than the FT algorithm because much of the distance computation necessary for the FV deletion procedure (see (2)) is inherently stored in D_{d-1}. We measured the execution time of straightforward (but not necessarily well optimized) implementations of the FT and squared EDT algorithms for 2D and 3D images over a wide range of sizes and content. Our tests included images with a single FV in a corner; images with randomly generated FVs where the number of FVs was set at 1, 2, 5, 10, 20, 50, 80, 90, 95, 98, or 99 percent; images with solid circles/spheres or squares/cubes where the number, size, position, and orientation of the objects were randomly generated; and medical images (CT and MR images) of heads, where FVs were obtained by using various edge operators. The fastest execution times occurred for the special cases of one FV in a corner or with a small number of randomly generated FVs. This is probably due to the fact that for images with sparse FVs, execution of line 8 in VORONOIEDT is frequently skipped. Except for these special cases, the execution times were remarkably constant and independent of image content, and there was little variation of execution time per voxel with the total number of image voxels N. The execution time of the squared EDT algorithm was approximately 0.23μs/voxel (4.3 Mvoxel/s) for 2D and 0.38μs/voxel (2.6 Mvoxel/s) for 3D on a Sun Microsystems SunBlade 2000 workstation with 1.05 GHz UltraSPARC III cpu, and approximately 0.21μs/voxel (4.8 Mvoxel/s) for 2D and 0.36μs/voxel (2.8 Mvoxel/s) for 3D on a PC with 2.2 GHz Pentium 4 cpu. The execution time of the FT algorithm was approximately four times longer in each case than that of the squared EDT algorithm. The memory requirements of the EDT algorithm are approximately half

that of the FT algorithm because no intermediate FT is computed. We also experimentally verified the correctness of the squared EDT algorithm by comparing the output with EDTs computed using two other algorithms [18], [36] and an implementation based on k-D binary search trees. We observed no differences between the outputs of the squared EDT algorithm and the outputs of the other algorithms for more than one million randomly generated tests.

Because both the FT and squared EDT algorithms work one row at a time, they can be easily parallelized. Since the N/n_d row computations at the dth dimension level are independent of each other, all computations at the dth dimension level can be performed in $O(n_d \times N/n_d \times 1/p) = O(N/p)$ time with p processors, $1 \leq p \leq n_d$. Thus, parallel versions of these algorithms run in $O(N/p)$ time with p processors, $1 \leq p \leq \min(n_1, \ldots, n_k)$. This approach is similar to the parallel method in [36]. It is fundamentally different from the one in [11], which is a divide-and-conquer algorithm that parallelizes a raster-scan propagation DT algorithm [21].

ACKNOWLEDGMENTS

Calvin Maurer was supported in part by funds from the Ronald L. Bittner Endowed Fund in Biomedical Research and by CBYON, Inc., Mountain View, CA. Rensheng Qi received partial support via a Special Opportunity Award from the Whitaker Foundation to the University of Rochester. Vijay Raghavan was supported in part by the US National Science Foundation under Grant No. 9820840. The authors thank Philippe Batchelor and Kensaku Mori for their helpful comments and suggestions.

REFERENCES

[1] C. Arcelli and G. Sanniti di Baja, "Finding Local Maxima in a Pseudo-Euclidean Distance Transform," *Computer Vision, Graphics, and Image Processing*, vol. 43, pp. 361-367, 1988.

[2] F. Aurenhammer, "Voronoi Diagrams—A Survey of a Fundamental Geometric Data Structure," *ACM Computer Surveys*, vol. 23, pp. 345-405, 1991.

[3] A.L.D. Beckers and A.W.M. Smeulders, "A Comment on 'A Note on Distance Transformations in Digital Images,'" *Computer Vision, Graphics, and Image Processing*, vol. 47, pp. 89-91, 1989.

[4] G. Borgefors, "Distance Transformations in Arbitrary Dimensions," *Computer Vision, Graphics, and Image Processing*, vol. 27, pp. 321-345, 1984.

[5] G. Borgefors, "Distance Transformations in Digital Images," *Computer Vision, Graphics, and Image Processing*, vol. 34, pp. 344-371, 1986.

[6] H. Breu, J. Gil, D. Kirkpatrick, and M. Werman, "Linear time Euclidean Distance Transform Algorithms," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 17, pp. 529-533, 1995.

[7] O. Cuisenaire and B. Macq, "Fast and Exact Signed Euclidean Distance Transformation with Linear Complexity," *Proc. IEEE Int'l Conf. Acoustics Speech Signal Processing*, pp. 3293-3296, 1999.

[8] O. Cuisenaire and B. Macq, "Fast Euclidean Distance Transformation by Propagation Using Multiple Neighborhoods," *Computer Vision and Image Understanding*, vol. 76, pp. 163-172, 1999.

[9] P.-E. Danielsson, "Euclidean Distance Mapping," *Computer Vision, Graphics, and Image Processing*, vol. 14, pp. 227-248, 1980.

[10] H. Eggers, "Two Fast Euclidean Distance Transformations in Z^2 Based on Sufficient Propagation," *Computer Vision and Image Understanding*, vol. 69, pp. 106-116, 1998.

[11] H. Embrechts and D. Roose, "A Parallel Euclidean Distance Trasnformation Algorithm," *Computer Vision and Image Understanding*, vol. 63, pp. 15-26, 1996.

[12] J.M. Fitzpatrick, D.L.G. Hill, and C.R. Maurer, Jr., "Image Registration," *Handbook of Medical Imaging, Volume 2: Medical Image Processing and Analysis*, SPIE Press, M. Sonka and J.M. Fitzpatrick, eds., pp. 447-513, 2000.

[13] Y. Ge and J.M. Fitzpatrick, "On the Generation of Skeletons from Discrete Euclidean Distance Maps," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 18, pp. 1055-1066, 1996.

[14] Y. Ge, C.R. Maurer, Jr., and J.M. Fitzpatrick, "Surface-Based 3-D Image Registration Using the Iterative Closest Point Algorithm with a Closest Point Transform," *Medical Imaging 1996: Image Processing*, vol. 2710, pp. 358-367, 1996.

[15] J. Goutsias and S. Batman, "Morphological Methods for Biomedical Image Analysis," *Handbook of Medical Imaging, Vol. 2: Medical Image Processing and Analysis*, M. Sonka and J.M. Fitzpatrick, eds., SPIE Press, pp. 175-272, 2000.

[16] G.J. Grevera and J.K. Udupa, "Shape-Based Interpolation of Multidimensional Grey-Level Images," *IEEE Trans. Medical Imaging*, vol. 15, pp. 881-892, 1996.

[17] W. Guan and S. Ma, "A List-Processing Approach to Compute Voronoi Diagrams and the Euclidean Distance Transform," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 20, pp. 757-761, 1998.

[18] C.T. Huang and O.R. Mitchell, "A Euclidean Distance Transform Using Grayscale Morphology Decomposition," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 16, pp. 443-448, 1994.

[19] H. Jiang, R.A. Robb, and K.S. Holton, "A New Approach to 3D Registration of Multimodality Medical Images by Surface Matching," *Visualization in Biomedical Computing (VBC)*, vol. 1808, pp. 196-213, 1992.

[20] M.N. Kolountzakis and K.N. Kutulakos, "Fast Computation of the Euclidean Distance Maps for Binary Images," *Information Processing Letters*, vol. 43, pp. 181-184, 1992.

[21] F. Leymarie and M.D. Levine, "Fast Raster Scan Distance Propagation on the Discrete Rectangular Lattice," *CVGIP: Image Understanding*, vol. 55, pp. 84-94, 1992.

[22] J.A. Little, D.L.G. Hill, and D.J. Hawkes, "Deformations Incorporating Rigid Structures," *Computer Vision Image Understanding*, vol. 66, pp. 223-232, 1997.

[23] J.C. Mullikin, "The Vector Distance Transform in Two and Three Dimensions," *CVGIP: Graphical Models and Image Processing*, vol. 54, pp. 526-535, 1992.

[24] C.W. Niblack, P.B. Gibbons, and D.W. Capson, "Generating Skeletons and Centerlines from the Distance Transform," *CVGIP: Graphical Models and Image Processing*, vol. 54, pp. 420-437, 1992.

[25] D.W. Paglieroni, "Distance Transforms: Properties and Machine Vision Applications," *CVGIP: Graphical Models and Image Processing*, vol. 54, pp. 56-74, 1992.

[26] D.W. Paglieroni, "A Unified Distance Transform Algorithm and Architecture," *Machine Vision Applications*, vol. 5, pp. 47-55, 1992.

[27] G.P. Penney, J.A. Little, J. Weese, D.L.G. Hill, and D.J. Hawkes, "Deforming a Preoperative Volume to Represent the Intraoperative Scene," *Computer Aided Surgery*, vol. 7, pp. 63-73, 2002.

[28] J. Piper and E. Granum, "Computing Distance Transformations in Convex and Non-Convex Domains," *Pattern Recognition*, vol. 20, pp. 599-615, 1987.

[29] F.P. Preparata and M.I. Shamos, *Computational Geometry: An Introduction*. Springer Verlag, 1985.

[30] I. Ragnemalm, "Neighborhoods for Distance Transformations Using Ordered Propagation," *CVGIP: Image Understanding*, vol. 56, pp. 399-409, 1992.

[31] I. Ragnemalm, "The Euclidean Distance Transform in Arbitrary Dimensions," *Pattern Recognition Letters*, vol. 14, pp. 883-888, 1993.

[32] S.P. Raya and J.K. Udupa, "Shape-Based Interpolation of Multidimensional Objects," *IEEE Trans. Medical Imaging*, vol. 9, pp. 32-42, 1990.

[33] E. Remy and E. Thiel, "Optimizing 3D Chamfer Masks with Norm Constraints," *Proc. Int'l Workshop Combinatorial Image Analysis*, pp. 35-56, 2000.

[34] A. Rosenfeld and J.L. Pfaltz, "Sequential Operations in Digital Picture Processing," *J. ACM*, vol. 13, pp. 471-494, 1966.

[35] A. Rosenfeld and J.L. Pfaltz, "Distance Functions on Digital Pictures," *Pattern Recognition*, vol. 1, pp. 33-61, 1968.

[36] T. Saito and J.-I. Toriwaki, "New Algorithms for Euclidean Distance Transformation of an n-dimensional Digitized Picture with Applications," *Pattern Recognition*, vol. 27, pp. 1551-1565, 1994.

[37] M. Sramek and A. Kaufman, "Fast Ray-Tracing of Rectilinear Volume Data Using Distance Transforms," *IEEE Trans. Visualization Computer Graphics*, vol. 6, pp. 236-252, 2000.

[38] S. Svensson, "Representing and Analyzing 3D Digital Shape Using Distance Information," PhD thesis, Swedish Univ. of Agricultural Sciences, Uppsala, 2001.

[39] M. van Herk, "Image Registration Using Chamfer Matching," *Handbook of Medical Imaging Processing and Analysis*, San Diego: Academic Press, I.N. Bankman, ed., pp. 515-527, 2000.

[40] M. van Herk and H.M. Kooy, "Automated Three-Dimensional Correlation of CT-CT, CT-MRI, and CT-SPECT Using Chamfer Matching," *Medical Physics*, vol. 21, pp. 1163-1178, 1994.

[41] B.J.H. Verwer, "Local Distances for Distance Transformations in Two and Three Dimensions," *Pattern Recognition Letters*, vol. 12, pp. 671-682, 1991.

[42] B.J.H. Verwer and P.W. Verbeek, S.T. Dekker, "An Efficient Uniform Cost Algorithm Applied to Distance Transforms," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 11, pp. 425-429, 1989.

[43] L. Vincent, "Exact Euclidean Distance Function by Chain Propagations," *Proc. IEEE Int'l Conf. Acoustics Speech Signal Processing*, pp. 520-525, 1991.

[44] A.M. Vossepoel, "A Note on 'Distance Transformations in Digital Images,'" *Computer Vision and Graphics Image Processing*, vol. 43, pp. 88-97, 1988.

[45] Q.J. Wu and J.D. Bourland, "Morphology-Guided Radiosurgery Treatment Planning and Optimization for Multiple Isocenters," *Medical Physics*, vol. 26, pp. 2151-2160, 1999.

[46] Q.J. Wu and J.D. Bourland, "Three-Dimensional Skeletonization for Computer-Assisted Treatment Planning in Radiosurgery," *Computer Medical Imaging Graphics*, vol. 24, pp. 243-251, 2000.

[47] H. Yamada, "Complete Euclidean Distance Transformation by Parallel Operation," *Proc. Seventh Int'l Conf. Pattern Recognition*, pp. 69-71, 1984.

[48] M. Yamashita and T. Ibaraki, "Distances Defined by Neighborhood Sequences," *Pattern Recognition*, vol. 19, pp. 237-246, 1986.

[49] K.J. Zuiderveld, A.H.J. Koning, and M.A. Viergever, "Acceleration of Ray-Casting Using 3D Distance Transforms," *Visualization in Biomedical Computing (VBC)*, vol. 1808, pp. 324-335, 1992.