Linear time algorithms for exact distance transform

Krzysztof Chris Ciesielski,[a,b,*] Xinjian Chen,[b]
Jayaram K. Udupa,[b,†] and George J. Grevera[c,b]

[a]Department of Mathematics, West Virginia University,
Morgantown, WV 26506-6310

[b]Dept. of Radiology, MIPG, Univ. of Pennsylvania, Blockley Hall – 4th Floor,
423 Guardian Dr., Philadelphia, PA 19104-6021

[c]Mathematics and Computer Science Department, Saint Joseph's University,
5600 City Avenue, Philadelphia, PA 19131

Abstract

In 2003, Maurer *at al.* [10] published a paper describing an algorithm that computes the exact distance transform in linear time (with respect to image size) for the rectangular binary images in the k-dimensional space $\mathbb{R}^k$ and distance measured with respect to L_p-metric for $1 \leq p \leq \infty$, which includes Euclidean distance L_2. In this paper we discuss this algorithm from theoretical and practical points of view. On the practical side, we concentrate on its Euclidean distance version, discuss the possible ways of implementing it as signed distance transform, and experimentally compare implemented algorithms. We also describe the parallelization of these algorithms and discuss the computational time savings associated with them. All these implementations will be made available as a part of the CAVASS software system developed and maintained in our group [7]. On the theoretical side, we prove that our version of the signed distance transform algorithm, *GBDT*, returns the *exact* value of the distance from the

*E-mail: KCies@math.wvu.edu; web page: `http://www.math.wvu.edu/~kcies`

†J.K. Udupa was partially supported by NIH grant R01-EB004395.

geometrically defined object boundary. We provide a *complete proof* (which was not given in [10]) that all these algorithms work correctly for L_p-metric with $1 < p < \infty$. We also point out that the precise form of the algorithm from [10] is not well defined for L_1 and L_∞ metrics. In addition, we show that the algorithm can be used to find, in linear time, the exact value of the *diameter of an object*, that is, the largest possible distance between any two of its elements.

1 Introduction

For a metric space X with a distance Δ and its non-empty subset $B \subset X$, a *distance transform* DT is a mapping from X such that $DT(x) = \Delta(x, B)$ for every x in X, where $\Delta(x, B) \stackrel{\text{def}}{=} \inf_{b \in B} \Delta(x, b)$. In other words, for every x the value of $DT(x)$ is a result of minimization:

$$DT(x) = \inf_{b \in B} \Delta(x, b). \tag{1}$$

A *feature transform* FT is a related argument minimization:

$$FT(x) = \arg \inf_{b \in B} \Delta(x, b). \tag{2}$$

In particular, if $FT(x) \in B$ exists (which is always the case for non-empty finite B), then $DT(x) = \Delta(x, FT(x))$. However, the value of $FT(x)$ need not be unique, see Figure 1. In this paper we will consider only the situation when X is either the k-dimensional Euclidean space $\mathbb{R}^k$ or its finite subset C, treated as a digital scene.

1.1 Background

Distance transform in digital spaces is an important tool in image processing [1, 2, 4–6, 8, 12, 15, 16]. (See also [25–30].) It finds widespread use in a variety of image operations such as filtering, interpolation, segmentation, registration, shape analysis, shape modeling, image compression, skeletonization or medial axis transform, and morphological operations. Some examples of these operations are as follows; most are applicable in $\mathbb{R}^k$. A binary image can be interpolated guided by the shape of the object it represents by first applying a distance transform to the binary image, then interpolating

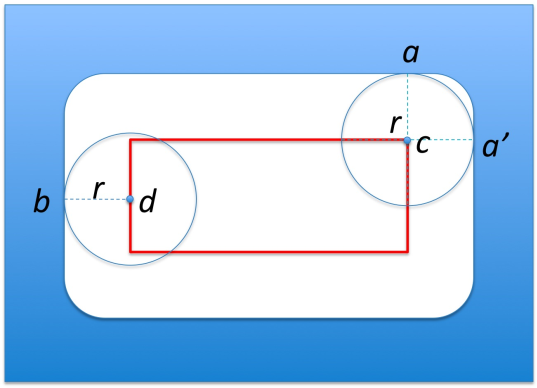


Figure 1. The inside rectangle (red) represents the set of all points of distance r from the image background (outside, in blue) and can be viewed as its boundary propagation at time $t = r/v$. We have $DT(c) = DT(d) = r$. $FT(d)$ is uniquely defined as b; $FT(c)$ can be either a or a'.

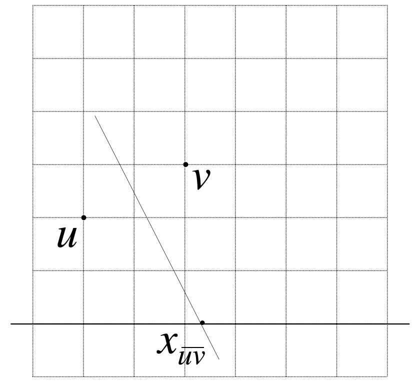


Figure 2. Representation in $\mathbb{R}^2$ of the point $x_{\overline{uv}}$ from (P2) on a horizontal line for Euclidean distance. Points on the slanted line are equidistant from u and v.

the distance map, and finally connecting the interpolated distance map back to a binary image [12]. This principle can also be applied to a gray level image [5, 6] by representing an n-dimensional gray image as a surface shape (binary image) in an $(n + 1)$-dimensional space where image intensity forms the height of the surface in $(n + 1)$th dimension. Medial axis representation of a shape [11] is a powerful concept that has numerous applications. One of its manifestations in the digital space is in the form of algorithms to "skeletonize" binary images. The distance transforms (and the feature transforms) find extensive use in robust "skeletonization" operations [1, 2, 17, 18]. Shape model-based techniques are funding extensive use in medical image segmentation, object motion tracking, shape analysis, and change detection. In constructing a shape model from the shape samples given for a shape family, distance transforms are used in ways analogous to their employment in interpolation. For example, the given shape samples are first distance transformed, and subsequently, the distance maps are averaged to estimate the mean of the given sample shapes [19, 20]. Distance transforms are also useful in image segmentation both in binary and gray level images [21, 22]. The distance transform is commonly used in image segmentation algorithms that utilize front (e.g., object boundary) propagation. (For more on this, in a non-digital setting, see also comment (D2) on page 8.) When a front is propagated from an initial surface S with a constant speed v, the front/surface position at time t is precisely represented as the set of points c at which the value of the distance transform (i.e., the distance from c to S) is equal to vt.

The distance transform can be used as a basic tool in constructing other analysis tools. The algorithm LTdiam we present in Section 3 for finding the exact object diameter is one such. Roughly, the diameter of an object is the (length of the) longest line segment connecting any two of its points. This tool is useful, for example, in creating 3D rendered images of a given object. In scaling the object properly for creating its projection in the rendered image, information about the radius of the smallest sphere that just encloses the object is very useful. Similarly, in Radiology, a standard measure, as per the RECIST criterion, used to define the size of a lesion is its diameter [23]. This is what the algorithm LTdiam output estimates automatically, unlike in the RECIST guidelines wherein the measurement is manual.

1.2 Preliminaries

The focal point of the discussion presented in this paper is the linear time distance transform $LTDT$ algorithm, which constitutes our version of the algorithm of Maurer *et al.* [10]. It returns both a distance transform DT and a feature transform FT. We implemented $LTDT$ for the Euclidean distance Δ, but it also works for any metric Δ satisfying the property (P) described in Definition 1.1. This generality requires very little additional effort. Nevertheless, for most readers it may be natural to assume for the rest of the paper that Δ stands simply for the Euclidean distance.

The algorithm $LTDT$ works on binary images defined on the rectangular grid $C = \{x^1_0, \ldots, x^1_{n_1-1}\} \times \cdots \times \{x^k_0, \ldots, x^k_{n_k-1}\} \subset \mathbb{R}^k$ of any dimension $k \geq 1$. For a *non-trivial binary image* I on C (i.e., a mapping from C *onto* $\{0, 1\}$; that is, with non-empty foreground and background), $LTDT$ returns a distance transform function DT from C into $\mathbb{R}$ defined, for $c \in C$, as $DT(c) = \Delta(c, B_I)$, where $B_I = \{d \in C\colon I(d) = 0\}$ is the *image background.* It also calculates an associated feature transform map FT. The algorithm $LTDT$ runs in time $O(n)$, where $n = n_1 \cdot \ldots \cdot n_k$ is the size of the image domain C.

Definition 1.1 Let C be as described above. We say that a metric Δ on $\mathbb{R}^k$ satisfies the property (P) provided the following holds.

(P) For every $d = 1, \ldots, k$, line R in $\mathbb{R}^k$ parallel to the d-axis, and $u = (u_i)$ and $v = (v_i)$ from $C \subset \mathbb{R}^k$:

(P1) If $u_d = v_d$, $y \in R$ is such that $y_d = u_d$, and $\Delta(u, y) \leq \Delta(v, y)$, then $\Delta(u, x) \leq \Delta(v, x)$ for every $x \in R$.

(P2) If $u_d < v_d$, then there is an $x_{\overline{uv}} \in \mathbb{R}$ (computable in $O(1)$ time) such that for every $x = (x_i)$ from R: if $x_d < x_{\overline{uv}}$, then $\Delta(u, x) < \Delta(v, x)$; and if $x_d \geq x_{\overline{uv}}$, then $\Delta(u, x) \geq \Delta(v, x)$. (See Figure 2.)

Intuitively, conditions (P1) and (P2) both address what happens with the property "x is closer to u than to v" (expressible as $\Delta(u, x) \leq \Delta(v, x)$ or $\Delta(u, x) < \Delta(v, x)$) when u and v are fixed and x moves along a fixed line R parallel to one of the axis, labeled as d-axis. (P1) addresses the case when either $u = v$ or the line through u and v is perpendicular to the d-axis; it tells that the property "x is closer to u than to v" remains unchanged independently of the position of x on R. (P2) addresses the case when the line through u and v is not perpendicular to the d-axis. It expresses the fact that, in this case, there is a point $x_{\overline{uv}}$ on line R which is Δ-equidistant from u and v and that the validity of "x is closer to u than to v" depends only on which side of $x_{\overline{uv}}$ on R point x lies. See Figure 2.

Recall that, for $1 \leq p < \infty$, the L_p metric on $\mathbb{R}^k$ is defined by the formula $\Delta(x, y) = \left(\sum_{i=1}^{k} |x_i - y_i|^p\right)^{1/p}$. In particular, the L_2 metric is the standard Euclidean distance. It is easy to see that, for $p > 1$, the L_p metric satisfies property (P). (See Proposition 4.4.) In what follows, we always assume that $x_0^d < \cdots < x_{n_d-1}^d$ for every $d = 1, \ldots, k$, although we need not assume that the images are isotropic, that is, the numbers $x_{i+1}^d - x_i^d$ can be different for different indices i and/or d. Nevertheless, all our figures are presented for isotropic images and our implementations are tested for this case. The elements of the grid C will be referred to as *spels*, short for space elements. Notice that, in this theoretical setting, the spels are represented as the sequences $\langle x_{i_d}^d \rangle_{d=1}^k$ of the actual coordinate values (indicating the distances in real distance units, like mm), rather than as their indices, $\langle i_d \rangle_{d=1}^k$, which is a common representation in image processing. For the isotropic images and L_p distances, this distinction actually makes no difference for the algorithms presented in this article. However, the distinction is important for anisotropic images, as discussed in Remark 2.3.

The paper is organized as follows. In Section 2, we describe and discuss different versions of signed distance transform algorithms that can be derived from the basic algorithm $LTDT$. The material presented there is self-contained, except that it relies on Theorem 1.2.

In a short Section 3, we note that a small modification $LTDT^{\max}$ of the $LTDT$ algorithm returns, for a binary image I, an exact maximal feature transform $MFT\colon C \to B_I$, that is such that, $MDT(x) = \sup_{b \in B_I} \Delta(x, b)$. We also use $LTDT^{\max}$ to describe an algorithm which returns, in linear time with respect to the size n of C, a pair $s, t \in B_I$ for which $\Delta(s,t)$ is equal to the diameter of the set B_I, which is such that, $\Delta(s,t) = \sup\{\Delta(c,d)\colon c, d \in B_I\}$.

In Section 4, we describe in detail the algorithm $LTDT$ and provide a complete proof of the following theorem, in the statement of which we use the terminology and notation described above.

Theorem 1.2 *If C is a rectangular scene in $\mathbb{R}^k$ and Δ is a metric on $\mathbb{R}^k$ satisfying property (P), then for any non-trivial binary image I on C, the algorithm $LTDT$ returns the exact distance transform $DT(c) = \Delta(c, B_I)$ and a related feature transform FT. It runs in a linear time with respect to the size n of C.*

In particular, $LTDT$ works correctly for L_p metrics for $1 < p < \infty$, which includes the Euclidean metric.

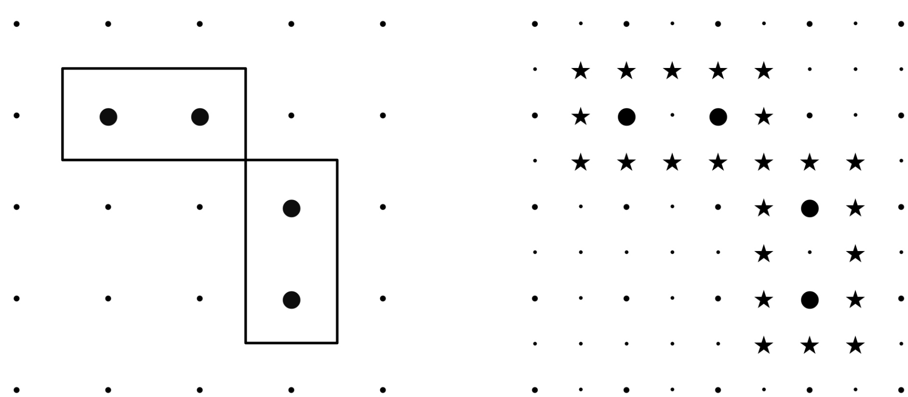
In Section 5, we report on the experimental results of applying some of the discussed algorithms on real medical image data for testing the algorithms discussed in the earlier sections. This section also presents an experimental comparison of different forms of the algorithm and a description of their parallelization. The paper is completed with some concluding remarks in Section 6.

2 Signed Distance Transform algorithms

Let I be a k-dimensional non-trivial binary image, that is, a function from $\Omega \subset \mathbb{R}^k$ onto $\{0,1\}$. It can be either digital (i.e., with Ω in the form of a digital grid C) or *geometrical* (i.e., with Ω equal to $\mathbb{R}^k$.) A *Signed Distance Transform for I* is usually defined on Ω as

$$SDT_I(x) = (-1)^{I(x)} \Delta(x, Bd_I),$$

where Bd_I is a boundary between the foreground $F_I = \{x \in \Omega\colon I(x) = 1\}$ of the image I and its background $B_I = \{x \in \Omega\colon I(x) = 0\}$. The main variability in this formula is caused by the use of different definitions of the boundary Bd_I. More precisely, for geometrical images, the boundary is always defined as the topological (geometrical) boundary, which can be expressed as $Bd_I^g = \{x \in \mathbb{R}^k \colon \Delta(x, F_I) = \Delta(x, B_I)\}$. However, for digital images. the set Bd_I^g is always disjoint from the grid C (see Figure 3(a)), so alternative definitions of the digital boundary are often used, although definitions conforming to Bd_I^g have also been pursued [15]. For example, the digital boundary Bd_I^{dig} for I is often defined as the set of all spels c in $B_I \subset C$ that are adjacent to some foreground spels. In fact, the ITK implementation of the Maurer's algorithm [14], called the exact distance transform EDT, is in the SDT_I form implemented in 3D and uses Bd_I^{dig} defined with 18-adjacency (i.e., c, d are adjacent when $||c - d|| < \sqrt{3}$). We also implemented this version of the algorithm, as $LTSDT$ (linear time signed distance transform), using our version of $LTDT$ and compared it with EDT. Nevertheless, the following arguments (D1)-(D4) listing the desired properties of distance transform algorithms show that, for most image processing tasks, the SDT_I^g — the SDT_I used with Bd_I^g — should be favored over all possible different versions of SDT_I.



(a) (b)

Figure 3. (a) Geometric boundary Bd_I^g of a binary image I on a 5×5 rectangular grid C with four foreground spels marked by large dots [15]. (b) The same binary image on the 9×9 double resolution grid C', where the smallest dots represent added spels. The digital boundary Bd_I' on C', marked by stars, consists of the intersection of Bd_I^g with C'.

(D1) Exact linear time implementation. The exact value of SDT_I^g can be calculated in linear time with respect to the size of C of a binary image — see algorithm Geometric Boundary Distance Transform $GBDT$ described on page 11.

(D2) Agreement with the geometric version. The fact that precisely the same formula for SDT can be used for discrete and geometric images is of particular importance for the energy optimization image segmentation technics (like level sets or active contours) that find the energy minimizing surface (object boundary) via its evolution according to the Euler-Lagrange equations. The evolution requires analytic representation of the current position of the object boundary, which is usually done implicitly as a level set of some function Ψ from $\mathbb{R}^k$ (for a k-dimensional image) into $\mathbb{R}$, that is, $Bd = \{x \in \mathbb{R}^k \colon \Psi(x) = 0\}$. The usual initialization of Ψ is as SDT_I, which in the continuous case is always taken as SDT_I^g, and it makes sense only to use the same formula for its digital version, used in the numerical approximation. Here, the $GBDT$ implementation (see toward the end of this section) of SDT_I^g in linear time is of great importance, since during the boundary (front) evolution, the evolving function Ψ is often reinitialized to SDT of the new position of the front. Since the algorithm for calculating SDT is invoked multiple times, this may introduce and accumulate errors if not done correctly and consistently. An example of a front evolution via DT is shown schematically in Figure 1.

(D3) Agreement with hyper cube interpretation of spels. It is common practice to identify each spel c in the isotropic rectangular digital image with the unit side k-dimensional cube centered at c, and the boundary as a union of the faces of all such cubes shared by foreground and background points [15]. (See Figure 3.) There are many advantages of such definitions of a digital boundary (see [16]) in visualization, processing, and image analysis. For example, when distance transforms are used in interpolating object shape [12], it has been shown that distances determined with respect to boundaries so defined lead to more accurate results [6, 8]. The point here is that SDT_I^g is equal to the boundary obtained with a cube-based interpretation of spels.

(D4) Symmetry with respect to background and foreground. The SDT_I^g, and any other version of SDT_I used with the boundary notion for which the boundary of the background is equal to the boundary of the foreground, have the following *reversibility property*, where $1 - I$ is the reversed image of I (i.e. the foreground of I is the background of $1 - I$, and vice versa):

(r) $SDT_I(x) = -SDT_{1-I}(x)$ for every x in the domain of image I.

Clearly, any SDT with this property leads to a more consistent distance map when distances from boundary are needed in an application for both foreground and background points. The problem with EDT implemented in ITK (or $LTSDT$), is that it fails to have property (r). In fact, no definition of boundary as a subset of B_I satisfies (r), as shown by the following result.

Theorem 2.1 *If SDT is defined via formula $SDT_J(x) = (-1)^{J(x)}\Delta(x, Bd_J)$ and the property (r) is satisfied by a non-trivial digital image $I : C \to \{0, 1\}$, then $Bd_I \cap C = Bd_{1-I} \cap C$. In particular, any spel from $Bd_I \cap C = Bd_{1-I} \cap C$ belongs to the background of one of the images $I, 1-I$, and to the foreground of the other.*

PROOF. If $x \in Bd_I \cap C$, then $\Delta(x, Bd_{1-I}) = |-SDT_{1-I}(x)| = |SDT_I(x)| = \Delta(x, Bd_I) = 0$, so $x \in Bd_{1-I}$. This proves $Bd_I \cap C \subset Bd_{1-I} \cap C$. The other inclusion is proved analogously. The additional comment holds for any spel from C. ∎

Of course, if a boundary Bd_J of an image $J : C \to \{0, 1\}$ is defined, for example, as the set of *all* $c \in C$ for which there is an adjacent $d \in C$ with $J(c) \neq J(d)$, then the property (r) holds for SDT_I. However, this creates a "thick" boundary, and some crucial information on the distances close to the geometrical boundary of the object is lost.

Next, we describe the algorithm $GBDT$, Geometric Boundary Distance Transform, mentioned above. It works for the L_p distances with $1 < p < \infty$. For a grid $C = \{x_0^1, \ldots, x_{n_1-1}^1\} \times \cdots \times \{x_0^k, \ldots, x_{n_k-1}^k\}$, define grid $C' = \{y_0^1, \ldots, y_{2n_1-2}^1\} \times \cdots \times \{y_0^k, \ldots, y_{2n_k-2}^k\}$, where, for all appropriate d and i, $y_{2i}^d = x_i^d$ and y_{2i+1}^d is the mid point between y_{2i}^d and y_{2i+2}^d. In other words, we double the resolution of the image grid in each direction. Let $Bd'_I = Bd_I^g \cap C'$ — see Figure 3. The basis for calculating the exact values of $SDT_I(c) = (-1)^{I(c)}\Delta(c, Bd_I^g)$, $c \in C$, in $O(n)$ time, and the rationale for $GBDT$, are provided by the following result.

Theorem 2.2 *$\Delta(c, Bd_I^g) = \Delta(c, Bd_I')$ for every $c \in C$.*

Proof. Clearly $\Delta(c, Bd_I^g) \le \Delta(c, Bd_I')$, since $Bd_I' \subset Bd_I^g$. To see the other inequality, let $c \in C$ and $d \in Bd_I^g$ be such that $\Delta(c, d) = \Delta(c, Bd_I^g)$. It is enough to show that $d \in C'$. This can be justified by a simple geometric argument sketched below.

Let $F \subset Bd_I^g$ be a face of a k-dimensional cube centered at c, such that F contains d. Let p be the orthogonal projection of c onto the $(k-1)$-dimensional hyperplane containing F. Note that $p \in C'$, as it has $k-1$ coordinates identical to those of c and one that identifies F; that is, the mid point between some y_{2i}^d and y_{2i+2}^d. If p belongs to F, then $d = p$ (this is obvious for Euclidean distance L_2, but holds also for other L_p distances) and so $d \in C'$. Otherwise, d must belong to one of the $(k-2)$-dimensional hyperplanes forming the boundary of F, and the argument may be repeated for this hyperplane. (Formally, the induction on the dimension of a hyperplane should be used.) ∎

In the algorithm *GBDT*, and all other algorithms throughout the paper, we identify the coordinate numbers x_m^d with their subscripts m; that is, the grid $C = \{x_0^1, \ldots, x_{n_1-1}^1\} \times \cdots \times \{x_0^k, \ldots, x_{n_k-1}^k\}$ is identified with the coordinate set $C^* = \{0, \ldots, n_1 - 1\} \times \cdots \times \{0, \ldots, n_k - 1\}$. Similar identification will be done for the grid C'.

More precisely, for $c = \langle c_i \rangle_{i=1}^k$ and $d = \langle d_i \rangle_{i=1}^k$ from C^*, let $\Delta^*(c, d)$ be defined as $\Delta(\langle x_{c_i}^i \rangle_i, \langle x_{d_i}^i \rangle_i)$, where Δ is the (Euclidean or L_p) distance satisfying (P) for which *DT* is calculated. Formally, in all the algorithms presented in this paper, we should use symbols Δ^* and C^* in place of Δ and C. However, to avoid additional burden, we will skip the *-superscript in the algorithm descriptions. This is additionally justified by the following fact.

Remark 2.3 Let Δ be an L_p metric, with $1 < p < \infty$. If the scene C is isotropic, with all numbers $N_j^i = x_{j+1}^i - x_j^i$ $(i = 1, \ldots, k,\ j = 0, \ldots, n_d - 2)$ equal to a fixed number θ (physical units), then $\Delta(c, d) = \Delta^*(c, d)\theta$ for all $c, d \in C^*$. Therefore, for isotropic images, the identification of C^* with C does not require any change in the definition of the distance function, except a multiplication at the end to express distance in physical units. However, for anisotropic images, the distance must be recovered from the numbers N_i^d, which need to be provided together with the image:

$$\Delta^*(\langle c_i \rangle, \langle d_i \rangle) = \left(\sum_{i=1}^k |x_{c_i}^i - x_{d_i}^i|^p \right)^{1/p},$$

where $|x^i_{c_i} - x^i_{d_i}| = \sum_{\min\{c_i,d_i\}<j\le\max\{c_i,d_i\}} N^j_i$.

In other words, for anisotropic images, the distance function Δ used in the algorithms (i.e., Δ^*) should be treated as a subroutine, given by the above formula.

Algorithm *GBDT*

Input: Dimension k (≥ 2) of the image; $n_1, \ldots, n_k$ — the size of the grid; a non-trivial binary image $I\colon C \to \{0,1\}$.

Output: A signed distance transform $SDT_I\colon C \to \mathbb{R}$ defined as $SDT_I(c) = (-1)^{I(c)}\Delta(c, Bd^g_I)$.

Auxiliary Data Structures: A grid $C' = \{0,\ldots,2n_1-2\}\times\cdots\times\{0,\ldots,2n_k-2\}$ having double resolution with respect to C, where we identify I with its copy $\hat{I}$ defined on $\hat{C} = \{0,2,\ldots,2n_1-2\}\times\cdots\times\{0,2,\ldots,2n_k-2\} \subset C'$ by $\hat{I}(2x) = I(x)$, where $x = (x_1,\ldots,x_k) \in C$ is arbitrary and $2x = (2x_1,\ldots,2x_k)$. A binary image I' on C' indicating points of Bd'_I of $\hat{I}$ (upon such identification) as the 0-value points.

begin

1. set $I'(c) = 1$ for all $c \in C'$;
2. *for* all $x \in C$ and $1 \le d \le k$ *do*
3.   *for* $i = 1$ *to* k *do*
4.   *if* $i \ne d$ *then* $y_i = x_i$, *else* $y_i = x_i + 1$;
5.   *endfor*;
6.   *if* $y \in C$ and $I(x) \ne I(y)$ *then*
7.   set $I'(c) = 0$ for each of the 3^{k-1}-many spels $c \in C'$ on the boundary face between x and y;
8.   *endif*;
9. *endfor*;
10. invoke *LTDT* with I' and appropriate Δ returning *DT* defined on C';
11. *for* every $x \in C$ set $SDT_I(x) = (-1)^{I(x)} \cdot DT(2x)$;
12. return SDT_I;

end

Theorem 2.4 *Algorithm GBDT invoked with the L_p distance, $1 < p < \infty$, on any non-trivial binary rectangular digital image I returns the signed distance to the geometric boundary Bd^g_I between foreground and background. Moreover, GBDT runs in $O(n)$ time.*

Proof. In this argument, we assume that Theorem 1.2 holds true.

The execution time of line 1 is of order $O(2^k n) = O(n)$. Each execution of lines 3-8 requires $O(k) + O(2^k) = O(1)$ operations. Since this loop is entered

$kO(n)$ times, execution of lines $1-9$ is done with $O(n)$ operations. Since $LTDT$ applied to I' runs in $O(2^k n) = O(n)$ time, and execution of line 11 requires n operations, $GBDT$ indeed runs in $O(n)$ time.

Next, note that after the execution of lines 1-9, map I' is as desired: $I'(c) = 0$ when $c \in Bd'_I$, and $I'(c) = 1$ for all remaining $c \in C'$. Indeed, after the initiation, in line 1 of I' with value 1 for all $c \in C'$, we examine (see lines 2-5) all pairs $x, y \in C$ of coordinate distance 1 (i.e., sharing a face of associated cubes), one from foreground, another from background. Then, in lines 6-8, we insure that, for all points $c \in C'$ on the common face between $2x$ and $2y$, the value $I'(c)$ is adjusted to 0.

After the execution of line 10, for every spel $c \in C'$ we have $FT(c) = \Delta(c, Bd'_I) = \Delta(c, Bd^g_I)$, where the second equation comes from Theorem 2.2. To finish the proof, it is enough to note that, in line 11, the factor $(-1)^{I(x)}$ fixes correctly the sign for the signed distance transform. ■

The pseudocode of the algorithm $LTSDT$, defined only in three dimensions, is just a simpler version of $GBDT$, with the boundary defined as a subset of the background defined with 18-adjacency:

$$Bd^{dig}_I = \{c \in C \colon I(c) = 0 \ \& \ ||c - d|| < \sqrt{3} \text{ for some } d \in C \text{ with } I(d) = 1\}.$$

Algorithm $LTSDT$

Input: n_1, n_2, n_3 —the size of the grid; a non-trivial binary image $I \colon C \to \{0, 1\}$.

Output: A signed distance transform $SDT_I \colon C \to \mathbb{R}$ defined as $SDT_I(c) = (-1)^{I(c)} \Delta(c, Bd^{dig}_I)$.

Auxiliary Data: A binary image I' on C indicating points of Bd^{dig}_I of I as the 0-valued points.

begin
set $I'(c) = 1$ for all $c \in C$;
for all $c \in C$ with $I(c) = 0$ *do*
    for $d \in C$ and $||c - d|| < \sqrt{3}$ *do*
        if $I(d) = 1$ *then* $I'(c) = 0$;
    endfor;
endfor;
invoke $LTDT$ with I' returning DT defined on C;
for every $x \in C$ set $SDT_I(x) = (-1)^{I(x)} \cdot DT(x)$;
return SDT_I;
end

3 Maximal Distance Transform and the diameter of an object

For a metric space X with a distance Δ and its non-empty subset $B \subset X$, let a *maximal distance transform* MDT be a mapping from X given by a formula $MDT(x) = \sup_{b\in B} \Delta(x,b)$ and let a *maximal feature transform* MFT map be a related argument maximization: $MFT(x) = \arg\sup_{b\in B} \Delta(x,b)$. Thus, for a non-empty finite set B, $MFT(x)$ exists for all $x \in X$ and it belongs to B. Clearly, the notions of maximal distance transform and maximal feature transform are dual, in a sense of interchanging minima and maxima, with the notions of distance transform and feature transform. It is therefore not surprising that a simple modification of the algorithm $LTDT$ gives us the following result.

Theorem 3.1 *If C is a rectangular scene in $\mathbb{R}^k$ and Δ is a metric on $\mathbb{R}^k$ satisfying property (P), then there exists an algorithm $LTDT^{\max}$ which for any non-trivial binary image I on C returns the exact maximal distance transform $MDT(c) = \sup_{b\in B} \Delta(c,b)$ and a related maximal feature transform MFT. It runs in a linear time with respect to the size n of C.*

In particular, $LTDT^{\max}$ works correctly for L_p metrics for $1 < p < \infty$, which includes the Euclidean metric.

Sketch of Proof. Let $\preceq$ be the reverse inequality on $\mathbb{R}$, that is, defined as

$$x \preceq y \quad \text{if and only if} \quad x \geq y.$$

Let $LTDT^{\max}$ be a modified $LTDT$ algorithm obtained by replacing the order relation $\leq$ with $\preceq$ in its code in every instance it is applied to Δ. In the pseudo-codes presented in Section 4, it means only one change, in line 7 in DimUp routine. (Notice that the change does not apply to the order of coordinates of points in the scene.)

The key result is that such created algorithm $LTDT^{\max}$ still returns distance and feature transforms with respect to this modified order $\preceq$. The proof of this fact is identical to that presented in Section 4, although it requires some care in noticing that L_p metrics satisfy modified condition (P) in which, once again, the order relation $\leq$ is replaced by $\preceq$ in every instance it is applied to Δ.

Clearly, distance and feature transforms for $\preceq$ are precisely maximal distance and feature transforms for the standard order $\leq$. ∎

With this result, we have the following algorithm, which, for a non-empty object S in C, returns the pair s, t from S for which $\Delta(s,t)$ is exactly equal to the *diameter* $\operatorname{diam}(S) = \sup\{\Delta(c,d) \colon c, d \in S\}$ of S. It is easy to see that it runs in linear time with respect to the size of C.

Algorithm LTdiam
Input: An object $S \neq \emptyset$ in a k-dimensional rectangular scene $C \subset \mathbb{R}^k$ represented as a background of a binary image $I \colon C \to \{0,1\}$; the L_p-metric Δ for some $1 < p < \infty$.
Output: A pair $s, t \in S$ for which $\Delta(s,t) = \max\{\Delta(c,d) \colon c, d \in S\}$.

begin

1. invoke $LTDT^{\max}$ for I and Δ returning MDT and MFT;
2. find an $s \in S$ with $MDT(s) = \max\{MDT(c) \colon c \in S\}$;
3. return s and $t = MFT(s)$;

end

Note that by performing the modifications for the GBDT version of DT, we can get geometric diameter for object S (i.e., the diameter for the object, in which each spel is replaced by appropriate rectangle/cube).

4 $LTDT$ and its parallelization

The $LTDT$ algorithm, described in this section, is only a minor modification of the Maurer *at al.* algorithm from [10]. We describe it here in detail, formally prove its correctness (i.e., Theorem 1.2), and describe its parallel version.

The material is presented in a "general to detailed" format, in which different routines used in $LTDT$ are introduced and discussed in the order from the most general (last to be used) routine to the most particular one. Although such presentation has its challenges, it is our belief that it gives the reader better overview of how the algorithm really works, emphasizing its general structure (general routines) and only successively exposing the reader to its deeper, more technical aspects. Thus, even without going through all details presented in this section, the reader will have a better chance to recognize the ideas that lie behind $LTDT$.

Actually, $LTDT$ computes a feature transform FT for I, see (2), and DT is calculated from FT only at the output stage by calling the function $DT(c) = \Delta(c, FT(c))$. The computation of FT is done recursively on the

DimUp input: Row $\mathbb{R}_d(x)$ indicators: $x \in C$ and $1 \le d \le k$; a function $F\colon C \to B_I \cup \{\emptyset\}$ which is a $(d-1)$-dimensional approximation of FT at every $c \in \mathbb{R}_d(x) \cap C$.

DimUp output: A modified $F\colon C \to B_I \cup \{\emptyset\}$ which is a d-dimensional approximation of FT at every $c \in \mathbb{R}_d(x) \cap C$. The values of F at points $c \notin \mathbb{R}_d(x)$ remain unchanged.

DimUp running time cost: $O(n_d)$, where number n_d is the size of the row $\mathbb{R}_d(x) \cap C$.

Figure 4. Properties of DimUp routine, used in $LTDT$, discussed in detail in Section 4.2.

dimension of the image. To express it precisely, we will need the following notation, in addition to that already introduced earlier. For every number $0 \le d \le k$ and $x \in C$, let $H_d(x) = \{c \in C\colon c_i = x_i \text{ for all } d < i \le k\}$ be the d-dimensional hyperplane containing x that results from fixing the terminal $k-d$ coordinates, that is, the coordinates with indices greater than d. Also, if $1 \le d \le k$, then $\mathbb{R}_d(x)$ will denote a one-dimensional row in $\mathbb{R}^k$ parallel to the d-th axis, that is, $\mathbb{R}_d(x) = \{c \in \mathbb{R}^k\colon c_i = x_i \text{ for all } i \ne d\}$. We say that a function $F\colon C \to B_I \cup \{\emptyset\}$ is a *d-dimensional approximation of FT at $x \in C$* provided $F(x)$ constitutes a value of the feature transform for $I \restriction H_d(x)$, the image I restricted to $H_d(x)$. We included the empty set $\emptyset$ in the range of F, since $B_{I\restriction H_d(x)} = B_I \cap H_d(x)$ can be empty, even when B_I is not; in such case we put $F(x) = \emptyset$, while in all other cases we require that $\Delta(x, F(x)) = \Delta(x, B_I \cap H_d(x))$. Such an F is a *d-dimensional approximation of FT* provided it is a d-dimensional approximation of FT at every $x \in C$; that is, when, for every $x \in C$, its restriction $F \restriction H_d(x)$ to $H_d(x)$ is a feature transform for $I \restriction H_d(x)$. Notice that the k-dimensional approximation of FT (for a k-dimensional image) is its true FT, while the 0-dimensional approximation F of FT has the property that $F(x)$ is equal to x for $x \in B_I$, and is equal to $\emptyset$ otherwise.

4.1 The algorithm outline: dimension step-up

In this subsection, we will construct the $LTDT$ algorithm using a subroutine DimUp described in Figure 4, which is a variant of VoronoiFV routine from [10]. We will also prove Theorem 1.2, assuming the properties of DimUp listed in Figure 4, that is, that $LTDT$ indeed returns the distance transform and that it runs in time $O(n)$. The detailed description of DimUp and the proof of the properties listed in Figure 4 are included in the latter part of this section.

For $1 \le d \le k$, let $C_d = \{x \in C \colon x_d = 1\}$ be the hyperplane passing through $(1, \ldots, 1)$ and perpendicular to $R_d(x)$. Note that C_d has size n/n_d.

Algorithm *LTDT*
Input: Dimension k (≥ 2) of the image; $n_1, \ldots, n_k$ — the size of the grid; a non-trivial binary image $I \colon C \to \{0, 1\}$.
Output: A distance transform $DT \colon C \to \mathbb{R}$ for the image I.
Auxiliary Data: A feature transform $F \colon C \to C \cup \{\emptyset\}$. A queue Q of points from C. Dimension counter d.

begin
1. *for* all $x \in C$ *do*
2.   *if* $I(x) = 0$ *then* $F(x) = x$ *else* $F(x) = \emptyset$;
3. *endfor*;
4. *for* $d = 1$ *to* k *do*
5.   push all points from C_d to Q;
6.   *while* Q is not empty *do*
7.   remove a point x from Q;
8.   invoke DimUp with x, d, and current F;
9.   *endwhile*;
10. *endfor*;
11. *for* all $x \in C$ *do*
12. $DT(x) = \Delta(x, F(x))$;
13. *endfor*;

end

Lines 1-10 of this algorithm represent procedure ComputeFT from [10]. In lines 1-3 we define F as 0-dimensional approximation of FT. Our main contribution here is the proof of the following lemma.

Lemma 4.1 *If algorithm DimUp works correctly, then for every non-trivial binary image I defined on a grid $C = \{0, \ldots, n_1 - 1\} \times \cdots \times \{0, \ldots, n_k - 1\}$, algorithm LTDT returns the exact distance transform DT for the image I. It does it in time $O(n)$, where n is the size of C.*

Proof. After execution of lines 1-3, the map F represents the 0-dimensional approximation of FT for I, as $H_0(x) = \{x\}$. This part runs in $O(n)$ time.

TRIM input: Spel $x \in C$ and number $1 \le d \le k$ indicating row $R = \mathbb{R}_d(x) \cap C$; a function $F\colon C \to B_I \cup \{\emptyset\}$ which is a $(d-1)$-dimensional approximation of FT at every $c \in R$.

TRIM output: A list $(q_1, \ldots, q_m)$, $0 \le m \le n_d$, of points from $G = \{F(x)\colon x \in R\}$ such that

(i) $\Delta(x, \{q_j\colon 1 \le j \le m\}) = \Delta(x, G)$ for every $x \in R$;

(ii) $(q_j)_d < (q_{j+1})_d$ for every $1 \le j < m$, and $x_{\overline{q_{i-1}q_i}} \le x_{\overline{q_i q_{i+1}}}$ for every $1 < i < m$.

TRIM running time cost: $O(n_d)$, where n_d is the size of the row R.

Figure 5. Properties of TRIM routine, used in DimUp, discussed in Section 4.3.

Next notice that for every $d = 1, \ldots, k$, when $LTDT$ enters lines 5-9, F is a $(d-1)$-dimensional approximation of FT for I; when it exits lines 5-9, F is a d-dimensional approximation of FT for I.

This statement is proved by mathematical induction on d. For $d = 1$, the entry requirement is guaranteed by lines 1-3. For $d > 1$, this is ensured by the inductive assumption. To finish the argument it is enough to show that the execution of lines 5-9 transforms $(d-1)$-dimensional approximation F of FT for I to the d-dimensional approximation of FT. This is guaranteed by the assumptions on DimUp: when executing lines 5-9, each row $R_d(x)$ of C is considered precisely once, and running DimUp for this row changes the values of F on this (and only this) row from $(d-1)$-dimensional approximation of FT to d-dimensional approximation of FT.

Next note that, for each d, the while loop from lines 6–9 is executed precisely n/n_d many times (the size of C_d), and each time the execution cost of DimUp is of order $O(n_d)$. Thus, each execution of lines 5-9 runs in time of order $(n/n_d)O(n_d) = O(n)$. Thus, the total time of running lines 1-10 is of order $O(n) + kO(n) = O(n)$.

Finally, note that, after the execution of the loop 4-10, F represents k-dimensional approximation of FT for I, which is the true FT for I. The execution of the loop 11-13 is still of order $O(n)$ (we assume that calculation of $\Delta(x, y)$ is $O(1)$) and the resulting DT is indeed an exact distance transform for I. ∎

4.2 DimUp procedure: further reduction

The goal of this subsection is to provide a detailed description of the DimUp routine, using a subroutine TRIM described in Figure 5, and prove, in Lemma 4.3, that it has the desired properties. The detailed description of TRIM and the proof of its properties listed in Figure 5 are included in the latter part of this section.

The main theoretical feature responsible for the correctness of the TRIM routine is the following result. In its statement it is possible that $B_I \cap H_d(z)$ is empty, in which case $\Delta(x, B_I \cap H_d(z)) = \Delta(x, \emptyset)$ is interpreted as ∞. In particular, the lemma says that $B_I \cap H_d(z)$ is empty if and only if G is.

Lemma 4.2 *Let $C = \{0, \dots, n_1 - 1\} \times \cdots \times \{0, \dots, n_k - 1\}$, I be a binary image on C, $R = \mathbb{R}_d(z) \cap C$ be a row in C, and $F\colon C \to B_I \cup \{\emptyset\}$ be a $(d-1)$-dimensional approximation of FT at every $x \in R$. If metric Δ has property (P1) and $G = \{F(x) \in B_I\colon x \in R\}$, then $G \subset B_I \cap H_d(z)$ and $\Delta(x, G) = \Delta(x, B_I \cap H_d(z))$ for every $x \in R$. In particular, for every $x \in R$, the value of a d-dimensional approximation of FT at x can be chosen from $G \cup \{\emptyset\}$.*

Proof. To see that $G \subset H_d(z)$, pick a $y \in G$ and let $x \in R$ be such that $y = F(x) \in H_{d-1}(x)$. Then $y_i = x_i$ for all $i \geq d$. Since $x \in R \subset \mathbb{R}_d(z)$ implies that $x_j = z_j$ for every $j \neq d$, we have $y_\ell = z_\ell$ for all $\ell > d$. So, $y \in H_d(z)$.

Inclusion $G \subset B_I \cap H_d(z)$ clearly implies $\Delta(x, G) \geq \Delta(x, B_I \cap H_d(z))$. To show the other inequality, choose an arbitrary $u \in B_I \cap H_d(z)$. We need to find a $v \in G$ such that $\Delta(x, v) \leq \Delta(x, u)$. Let $y \in R \subset H_d(z)$ be such that $y_d = u_d$. Since also, for all $\ell > d$, $y_\ell = z_\ell = u_\ell$, as $y, u \in H_d(z)$, we conclude that $u \in H_{d-1}(y)$. In particular, $B_I \cap H_{d-1}(y) \ni u$ is non-empty, so $v = F(y) \in G$ belongs to $B_I \cap H_{d-1}(y)$ and has a property $\Delta(y, v) = \Delta(y, B_I \cap H_{d-1}(y)) \leq \Delta(y, u)$, since F is a $(d-1)$-dimensional approximation of FT at $y \in R$. So, by (P1), $\Delta(x, u) \geq \Delta(x, v)$. ∎

Lemma 4.2 tells us that if we like to upgrade F from being a $(d-1)$-dimensional approximation of FT on $R = \mathbb{R}_d(z) \cap C$ to being a d-dimensional approximation of FT on R, the values of this new F can be chosen from the values of old F on R, that is, from $G = \{F(x)\colon x \in R\}$. In our upgrade procedure, we will need first to further restrict our choice of the values of new F on R to a subset of $G \cup \{\emptyset\}$. This will be done with the TRIM procedure with properties listed in Figure 5.

Using TRIM it is easy to describe the DimUp algorithm. This is actually a part (lines 15-24) of the VoronoiFT procedure from [10].

Algorithm DimUp
Input: A $(d-1)$-dimensional approximation F of FT on $R = \mathbb{R}_d(c) \cap C$.
Output: A d-dimensional approximation F of FT on R.
Auxiliary Data: A queue Q of points from C. A counter ℓ.

```
begin
  1. invoke TRIM for R and F to get list (q_1, ..., q_m);
  2. if m > 0 then
  3.    push all points from R to Q in the increasing order
                         (i.e., with x_d = 1 for the first removed point);
  4.    initialize ℓ = 1;
  5.    while Q is not empty do
  6.       remove a point x from Q;
  7.       while ℓ < m and Δ(x, q_ℓ) ≥ Δ(x, q_{ℓ+1}) do
  8.          ℓ = ℓ + 1;
  9.       endwhile;
 10.       F(x) = q_ℓ;
 11.    endwhile;
 12. endif;
end
```

Lemma 4.3 *Assume that Δ satisfies (P2). If algorithm TRIM works correctly and the input function F for DimUp is a $(d-1)$-dimensional approximation F of FT on $R = \mathbb{R}_d(c) \cap C$, then the output version of F for DimUp is a d-dimensional approximation of FT on R. Moreover, DimUp runs in $O(n_d)$ time, where n_d is the size of the row R.*

PROOF. By our assumptions on TRIM, the execution time of line 1 is of order $O(n_d)$. The total number of times lines 7-9 can be executed during the entire program run is bounded by $m \le n_d$. Since Q has a size n_d, this means that lines 5-11 are executed with $O(n_d)$ operations. So, DimUp requires only $O(n_d)$ operations.

Now, $m = 0$ precisely when $F(x) = \emptyset$ for all $x \in R$, in which case $B \cap H_d(c) = \emptyset$, and the algorithm correctly leaves all these values unchanged. So, assume that $m > 0$, that is, that the set $H = \{q_1, \ldots, q_m\}$ is non-empty. We enter the loop from lines 5-11 precisely n_d times, and on its ith execution, we have $x_d = i$ for the removed x from the queue Q. Let ℓ_i be the value of the counter ℓ upon leaving the ith execution of the loop. Notice, that upon entering the loop for its ith execution the value of the counter ℓ is equal to ℓ_{i-1}, where $\ell_0 = 1$ by line 4 of the code. We will show, by induction on $i = 1, \ldots, n_d$, that upon leaving the ith execution of the loop, the following inductive condition holds.

(C_i) $F(x) = q_{\ell_i}$ and for every $1 \leq j < \ell_i < n \leq m$

$$\Delta(x, q_j) \geq \Delta(x, q_{\ell_i}) \ \&\ \Delta(x, q_{\ell_i}) < \Delta(x, q_n), \tag{3}$$

where $x \in R$ is such that $x_d = i$.

Notice that (3) implies that $\Delta(x, F(x)) = \Delta(x, q_{\ell_i}) \leq \Delta(x, \{q_1, \ldots, q_m\})$, while $\Delta(x, \{q_1, \ldots, q_m\}) = \Delta(x, G) = \Delta(x, B \cap H_d(x))$ is a consequence of Lemma 4.2 and the property (i) of TRIM output. Therefore, (3) implies that $\Delta(x, F(x)) \leq \Delta(x, B \cap H_d(x))$, that is, F becomes a d-dimensional approximation of FT at x upon leaving the ith execution of the loop from lines 5-11; remains so, since the value of F at x does not change any more during the further execution of TRIM. Consequently, the proof of (C_i) will complete the proof of the lemma.

To prove (C_i) fix an $i = 1, \ldots, n_d$ and assume that (C_{i-1}) holds provided that $i > 1$. First we will argue for the first inequality. So let $1 \leq j < \ell_i$. If $j < \ell_{i-1}$, then $i > 1$, since otherwise we would have $1 \leq j < \ell_0 = 1$, a contradiction. Let $\bar{x} \in R$ be such that $\bar{x}_d = i - 1$. So, by the inductive assumption (C_{i-1}), $\Delta(\bar{x}, q_j) \geq \Delta(\bar{x}, q_{\ell_{i-1}})$.

Since $(q_j)_d < (q_{\ell_{i-1}})_d$, property (P2) implies that $\bar{x}_d \geq x_{\overline{q_j q_{\ell_{i-1}}}}$. Thus, as $x_d = i > i - 1 = \bar{x}_d$, we have $x_d > x_{\overline{q_j q_{\ell_{i-1}}}}$ and, by property (P2), $\Delta(x, q_j) \geq \Delta(x, q_{\ell_{i-1}})$. Moreover, execution of the loop from lines 7-9 insures that $\Delta(x, q_{\ell_{i-1}}) \geq \Delta(x, q_t) \geq \Delta(x, q_{\ell_i})$ for every $\ell_0 \leq t \leq \ell_1$. This implies that $\Delta(x, q_j) \geq \Delta(x, q_{\ell_1})$ for every $1 \leq j < \ell_1$.

To show the second inequality, take $\ell_i < n \leq m$. Then $\ell_i + 1 \leq n \leq m$ and the fact that loop 7-9 stopped means that $\Delta(x, q_{\ell_i}) < \Delta(x, q_{\ell_i+1})$. For $n = \ell_i + 1$ this finishes the proof. Therefore, assume that we have $s = n - (\ell_i + 1) > 0$. Since $(q_{\ell_i})_d < (q_{\ell_i+1})_d$, property (P2) implies that $x_d < x_{\overline{q_{\ell_i} q_{\ell_i+1}}}$. Then, $\{x_{\overline{q_{\ell_i+j} q_{\ell_i+j+1}}} \colon j = 0, \ldots, s\}$ and $\{(q_{\ell_i+j})_d \colon j = 0, \ldots, s\}$ are increasing by the property (ii) of TRIM output, so, property (P2) (used s-times with the inequalities $x_d < x_{\overline{q_{\ell_i+j} q_{\ell_i+j+1}}}$) implies that also the sequence $\{\Delta(x, q_{\ell_i+j+1}) \colon j = 1, \ldots, s\}$ is strictly increasing. In particular, $\Delta(x, q_{\ell_i}) < \Delta(x, q_{\ell_i+1}) < \Delta(x, q_{\ell_i+1+s}) = \Delta(x, q_n)$, finishing the proof. ∎

4.3 The TRIM procedure

In this subsection we will provide a detailed description of the TRIM routine and prove, in Lemma 4.6, that it has the desired properties claimed in Figure 5. This, together with Proposition 4.4 and Lemmas 4.3 and 4.1, will complete the proof of Theorem 1.2.

We will start with proving, in Proposition 4.4, that the L_p metrics satisfy the property (P). Although the actual proof of Lemma 4.6 does not require this result, the TRIM routine uses the values of $x_{\overline{uv}}$ from the property (P), so it may be easier to follow TRIM description having already determined actual procedures for finding $x_{\overline{uv}}$ in the practical cases we emphasize.

Proposition 4.4 *For $1 < p < \infty$, if Δ is the L_p metric on $\mathbb{R}^k$, then it satisfies the property (P) defined in Definition 1.1.*

Proof. We will use the notation from (P). To see (P1), note that $\Delta(u,y)^p = \sum_{i\neq d}|u_i - y_i|^p$ and similarly for $\Delta(v,y)^p$, since $u_d = y_d = v_d$. Also, since x and y belong to the same line parallel to d-axis, $x_i = y_i$ for all $i \neq d$. So, $\Delta(u,x)^p = |u_d - x_d|^p + \sum_{i\neq d}|u_i - y_i|^p = |v_d - x_d|^p + \Delta(u,y)^p$. Similarly, $\Delta(v,x)^p = |v_d - x_d|^p + \Delta(v,y)^p$. So, since function $g(x) = x^p$ is strictly increasing on $[0,\infty)$, $\Delta(u,y) \le \Delta(v,y)$ implies $\Delta(u,y)^p \le \Delta(v,y)^p$, thus $\Delta(u,x)^p = |v_d - x_d|^p + \Delta(u,y)^p \le |v_d - x_d|^p + \Delta(v,y)^p = \Delta(v,x)^p$, and also $\Delta(u,x) \le \Delta(v,x)$.

To see (P2), notice that every point x on R parallel to the d-axis is uniquely determined by its dth coordinate x_d. Let $h(x_d) = \Delta(x,u)^p - \Delta(x,v)^p$. We will show that h is strictly increasing and that $x_{\overline{uv}}$ is a zero point of h. Indeed, it is easy to see that $\Delta(u,x) < \Delta(v,x)$ precisely when $h(x_d) < 0$. Thus, a zero of h must satisfy (P2).

Clearly function $h(x_d) = \sum_{i=1}^k |x_i - u_i|^p - \sum_{i=1}^k |x_i - v_i|^p$ is continuous. To prove that it is strictly increasing, it is enough to show that it has positive derivative at all points except possibly for $x_d = u_d$ and $x_d = v_d$. Since $h'(x_d) = \frac{d}{dx_d}\left(|x_d - u_d|^p - |x_d - v_d|^p\right)$, for $u_d < v_d < x_d$, we have $h'(x_d) = p\left((x_d - u_d)^{p-1} - (x_d - v_d)^{p-1}\right) > 0$ as $x_d - u_d > x_d - v_d > 0$ and function x^{p-1} is strictly increasing on $(0,\infty)$. Similarly, for $x_d < u_d < v_d$, we have $h'(x_d) = p\left(-(u_d - x_d)^{p-1} + (v_d - x_d)^{p-1}\right) > 0$ as $v_d - x_d > u_d - x_d > 0$. Finally, for the remaining case $u_d < x_d < v_d$, we have $h'(x_d) = p\left((x_d - u_d)^{p-1} + (v_d - x_d)^{p-1}\right) > 0$.

The existence of a zero point for h follows from the Intermediate Value Theorem and the fact that h attains both positive and negative values. To see this last fact, we note that $\lim_{u_d\to\pm\infty} h(u_d) = \pm\infty$. The argument for the limit requires some algebraic work, but it follows from a simple estimate [13, Lemma 3, page 121], proven with calculus tools, that $(a+b)^p \geq a^p + pba^{p-1}$ for non-negative a and b. ∎

Remark 4.5 As the above arguments show, for L_p metrics with $1 < p < \infty$ the number $x_{\overline{uv}}$ can be defined as the dth coordinate of the unique point on the line R equidistant from u and v. In fact, this is the way $x_{\overline{uv}}$ is defined in [10, Remark 3]. However, for the L_1 metric, such a point need not exist. On the plane $\mathbb{R}^2$ and for line R being the x-axis, this is justified by points $u = (0,0)$ and $v = (2,1)$, for which $\Delta(x,u)-\Delta(x,v) = |x_1|-|x_1-1|-2 \leq -1$ for any $x \in R$. For L_∞ metric (defined as $\Delta_\infty(u,v) = \max\{|u_i - v_i|\colon i = 1,\ldots,k\}$) number $x_{\overline{uv}}$ always exists, but it need not be unique. On the plane and the same line R, this is justified by points $u = (0,1)$ and $v = (1,1)$ since then any point $(a,0) \in R$ with $a \in [0,1]$ is equidistant from u and v.

Although this means that the precise recipe from [10] does not work for these two metrics, a simple modification of the algorithm (for $LTDT$ the change needs to be made in the definition of the CHECK subroutine) can still produce a correct version of the algorithm.

Assume that $1 < p < \infty$ and a row $R = \mathbb{R}_d(c) \cap C$ is fixed, where $c \in C$ and $d = 1,\ldots,k$. Let $u, v \in C$ be such that $u_d < v_d$. Then, according to Proposition 4.4, $x_{\overline{uv}}$ (for the row R) is the number x_d for which the function $h(x_d) = \Delta(x,u)^p - \Delta(x,v)^p = \sum_{i=1}^k |u_i - x_i|^p - \sum_{i=1}^k |v_i - x_i|^p$ is equal to 0. For a general value of p, this can be found by a simple numerical approximation. However, for $p = 2$ — the most important case of the Euclidean distance, the one which we actually implemented — the equation takes the form $(x_d - u_d)^2 + \sum_{i\neq d}(x_i - u_i)^2 = (x_d - v_d)^2 + \sum_{i\neq d}(x_i - v_i)^2$, or, equivalently, $(v_d - u_d)(2x_d - u_d - v_d) = \sum_{i\neq d}(x_i - v_i)^2 - \sum_{i\neq d}(x_i - u_i)^2$. Thus, $x_{\overline{uv}} = x_d$ is a solution of this linear equation.[1]

In what follows, we will use a boolean valued subroutine CHECK(u,v,w), which depends on a row R, is applied to $u, v, w \in C$ with $u_d < v_d < w_d$, and is true when there is no integer $i = 0,\ldots,n_d-1$ for which $x_{\overline{uv}} \leq x_j^d \leq x_{\overline{vw}}$. Note that CHECK is a refinement of procedure REMOVEFT from [10] defined as $x_{\overline{uv}} > x_{\overline{vw}}$. TRIM works correctly with either version of these procedures.

[1] In the calculation of the number $x_{\overline{uv}}$, the distinction of the spel real coordinates $\langle x_{i_d}^d \rangle_{d=1}^k$ from their index representation $\langle i_d \rangle_{d=1}^k$ is of importance for the anisotropic images, see Remark 2.3.

However, our experiments show that the implementation with CHECK works slightly faster.

We implemented the algorithm for isotropic scenes and identified coordinates $x_0^d, \ldots, x_{n_d-1}^d$ with the indices $0, \ldots, n_d - 1$. In this case, we were able to implement CHECK(u,v,w) simply as $\lceil (x_{\overline{uv}})_d \rceil > \lfloor (x_{\overline{vw}})_d \rfloor$, where $\lceil r \rceil$ is the smallest integer greater than or equal to r, and $\lfloor r \rfloor$ is the greatest integer less than or equal to r.

Algorithm TRIM
Input: Row $R = \mathbb{R}_d(x) \cap C$ indicators: $x \in C$ and $1 \le d \le k$; a function $F\colon C \to B_I \cup \{\emptyset\}$ which is a $(d-1)$-dimensional approximation of FT at every $c \in R$.
Output: A list $Q = (q_1, \ldots, q_m)$ of points from $G = \{F(x) \in B_I \colon x \in R\}$ satisfying (i) and (ii).
Auxiliary Data: Counters i, m and point pointers u, v. Q is obtained by removing some points from the list G.

begin
set $m = 0$;
for $i = 1$ *to* n_d *do*
if $F(x_i) \neq \emptyset$ *then*
set $m = m + 1$;
set $q_m = F(x_i)$;
if $m > 1$ *then*
set $u = q_{m-1}$;
set $v = q_m$;
if $x_{\overline{uv}} \ge n_d$ *then*
set $m = m - 1$;
else
while $m > 2$ *and* CHECK(q_{m-2}, q_{m-1}, q_m) *do*
set $q_{m-1} = q_m$;
set $m = m - 1$;
endwhile;
if $m = 2$ *then*
set $u = q_1$;
set $v = q_2$;
if $x_{\overline{uv}} < 0$ *then*
set $q_1 = q_2$;
set $m = 1$;

```
22.                 endif;
23.             endif;
24.         endif;
25.     endif;
26.   endif;
27. endfor;
28. return sequence (q_1, ..., q_m) for the current value of m;
end
```

Lemma 4.6 *Assume that Δ satisfies (P). Then TRIM works correctly and runs in $O(n_d)$ time.*

Proof. The TRIM procedure should be viewed as starting with G as a first approximation of the queue Q and removing some of its terms to ensure the second part of condition (ii) from TRIM output requirement (property (C_i) below), while preserving (i) (condition (B_i)). The first part of (i) (property (A_i)) is preserved by any pruning, since G has already this property.

To see that TRIM runs in $O(n_d)$ time, note that it enters the loop from lines 2-27 precisely n_d times. The i-th run time of this loop is of order $O(1)+2P_i$, where P_i is the number of runs of the loop from lines 12-15. Since each time this loop is run, one value from the set $\{F(x_i)\colon i=1,\ldots,n_d\}$ is removed, we have $P_1+\cdots+P_{n_d}\leq n_d$. Therefore, TRIM indeed runs in time $\sum_{i=1}^{n_d}(O(1)+2P_i)=O(n_d)$.

To prove that the output of TRIM satisfies (i) and (ii), we will show, by induction on $i=1,\ldots,n_d$, that, after completing the i-th run of a loop from lines 2-27, the following holds, where m_i stands for the value of m at this point of program execution, and $G_i=\{F(x_j)\in B\colon 1\leq j\leq i\}$.

(A_i) $(q_j)_d<(q_{j+1})_d$ for every $1\leq j<m_i$, and all these q_j's belong to G_i.

(B_i) $\Delta(x,\{q_j\colon 1\leq j\leq m_i\})=\Delta(x,G_i)$ for every $x\in R$.

(C_i) $x_{\overline{q_{j-1}q_j}}\leq x_{\overline{q_jq_{j+1}}}$ for every $1<j<m$.

This will finish the proof, since then TRIM's output value of m is equal to m_{n_d}, the set G_{n_d} equals G from TRIM's output description, and so, the conditions (A_{n_d})-(C_{n_d}) are the restatement of (i) and (ii).

Assume that $m_0=0$. Then G_0 and the q-sequence are empty, so conditions (A_0)-(C_0) are satisfied. Thus, we just need to show that, for every

$i = 1, \ldots, n_d$, if conditions (A_{i-1})-(C_{i-1}) are satisfied upon entering the code lines 2-27, then (A_i)-(C_i) hold upon finishing their execution.

Note that, after each execution of lines 2-27, the q-sequence may have more than m_i elements. However, only the first m_i of its elements are of consequence, and these first m_i elements constitute the q-sequence (possibly empty) satisfying (A_i)-(C_i).

If $F(x_i) = \emptyset$, then $G_i = G_{i-1}$ and none of lines 4-25 is executed, so $m_i = m_{i-1}$ and the q-sequence remains unchanged. This clearly implies (A_i)-(C_i). So, for the rest of the proof, assume that $F(x_i) \neq \emptyset$.

The execution of lines 3-4 temporarily extends the q-sequence (by assigning to $m_i = m$ value $m_{i-1} + 1$) and puts $F(x_i)$ at its end. This initial assignment ensures (A_i) and (B_i). However, (C_i) may be false at this stage, and the sequence may need to be trimmed to ensure (C_i) while preserving (B_i). This is done in lines 6-25.

Clearly, by the inductive assumption (A_{i-1}), at this stage the sequence satisfies (A_i), since $(F(x_i))_d > x_d$ for every $x \in G_{i-1}$. To see that the execution of lines 6-25 preserves (A_i), it is enough to note that the only changes to this sequence in lines 6-25 are either through dropping the last sequence element (in line 10) or by replacing the second to the last of its elements by the last one and shortening the sequence by 1 (lines 13-14 or 20-21). These operations clearly preserve (A_i).

Now, if we enter line 6 with $m = m_i = 1$, then $m_{i-1} = 0$, and, by (B_{i-1}), we have $G_{i-1} = \emptyset$. Although, at this case, the condition in line 6 insures that no other lines are executed, this implies that $m_i = 1$ and $G_i = \{F(x_i)\} = \{q_1\}$, so (B_i) and (C_i) hold. So, assume that at line 6 we have $m_i = m > 1$, that is, that the q-sequence has at least two elements. Next we will decide whether its last element is in the proper position and, if not, modify the sequence.

Thus, when entering line 7 we know that our q-sequence has at least two elements. In lines 7-11 we check whether there is any reason to keep q_m in the sequence.[2] If not, we can simply remove it. More precisely, since $G_i = G_{i-1} \cup \{F(x_i)\}$, condition (B_{i-1}) implies that for every $x \in \mathbb{R}$ we have $\Delta(x, G_i) = \Delta(x, G_{i-1} \cup \{q_m\}) = \Delta(x, \{q_j \colon 1 \leq j \leq m\})$. Assume that the condition from line 9 is satisfied. Then, the only executed line in the rest of the loop is line 10, which discards the last element of the sequence. This

[2]Execution of lines 9-10 is not necessary for insuring correct output (i)-(ii) of TRIM. However, it may remove some unnecessary redundancy from the queue Q.

means that $m_i = m = m_{i-1}$. Now, to show that this sequence satisfies (B_i) and (C_i), note that $x_{\overline{uv}}$ is to the right of every $x \in R$. This means that for $m = m_{i-1} + 1$ we have $\Delta(x, q_{m-1}) = \Delta(x, u) \leq \Delta(x, v) = \Delta(x, q_m)$. In particular, $\Delta(x, G_i) = \Delta(x, \{q_j \colon 1 \leq j \leq m\}) = \Delta(x, \{q_j \colon 1 \leq j \leq m_{i-1}\})$ is equal to $\Delta(x, G_{i-1})$ for every $x \in R$. Therefore, in this case, (B_{i-1}) and (C_{i-1}) imply (B_i) and (C_i) for $m_i = m_{i-1}$.

Next, we assume that the condition from line 9 fails. Thus, we enter the key program loop of lines 12-15. We claim that upon exiting the loop, condition (B_i) is preserved, while (C_i) is already satisfied. Indeed, each time the lines 13-14 are executed, the second to the last element, q_{m-1}, is removed from the current queue $(q_1, \ldots, q_m)$ and the length indicator m is properly reduced. This operation does not influence the condition (B_i), since satisfaction of the predicate CHECK(q_{m-2}, q_{m-1}, q_m) means that for every $x \in R$ either $\Delta(x, q_{m-2}) < \Delta(x, q_{m-1})$ (when $x_d < x_{\overline{q_{m-2}q_{m-1}}}$) or $\Delta(x, q_m) \geq \Delta(x, q_{m-1})$ (when $x_d > x_{\overline{q_{m-1}q_m}}$), and therefore $\Delta(x, \{q_1, \ldots, q_{m-2}, q_m\}) = \Delta(x, \{q_1, \ldots, q_{m-2}, q_{m-1}, q_m\})$. Thus, property (B_i) survives execution of the loop. Also, upon leaving the loop, either $m = 2$, in which case (C_i) is satisfied in void, or $m > 2$ and CHECK(q_{m-2}, q_{m-1}, q_m) is false, which implies that $x_{\overline{q_{m-2}q_{m-1}}} \leq x_{\overline{q_{m-1}q_m}}$. This, together with the inductive assumption of (C_{i-1}) insures (C_i).

To finish the proof, it is enough to show that execution of lines 16-26 preserves conditions (B_i) and (C_i). This is obvious, when $m > 2$ after completing line 15. So, assume that we have $m = 2$ when entering line 16. Then, the situation is analogous to that from lines 9-10. The q-sequence consists of just two elements, q_1 and $q_2 = \hat{q}$. This sequence remains unchanged, unless $x_{\overline{q_1q_2}} < 0$, in which case we remove q_1 from the queue. This preserves (B_i), since $x_{\overline{q_1q_2}} < 0$ implies that $\Delta(x, q_1) > \Delta(x, q_2)$ for all $x \in R$. ∎

4.4 Algorithm parallelization

A parallel version of $LTDT$ is easy to create, since the task of finding FT by this algorithm is done recursively for each hyperplane H in $\mathbb{R}^k$ and the calculations are independent of each other for disjoint hyperplanes. Thus, the simplest way to parallelize algorithm $LTDT$ with m processors or threads of execution is to proceed with the following steps, where $LTDT^*$ returns FT instead of DT, that is, it is run with only the first 10 lines of $LTDT$.

(S1) Split n_k hyperplanes $H_{k-1}(x) \cap C$ perpendicular to the kth axis into m

disjoint families $\mathcal{H}_j$ of approximately equal size of n_k/m.

(S2) For each $j = 1, \ldots, m$ and each hyperplane H from $\mathcal{H}_j$ apply $LTDT^*$ on the j-th processor to calculate FT for the image $I \restriction H$. Each such part is calculated in time of $O(n/n_k)$. Since the multiprocessors are run simultaneously, all calculations will be completed in time $O(n_k/m)O(n/n_k) = O(n/m)$.

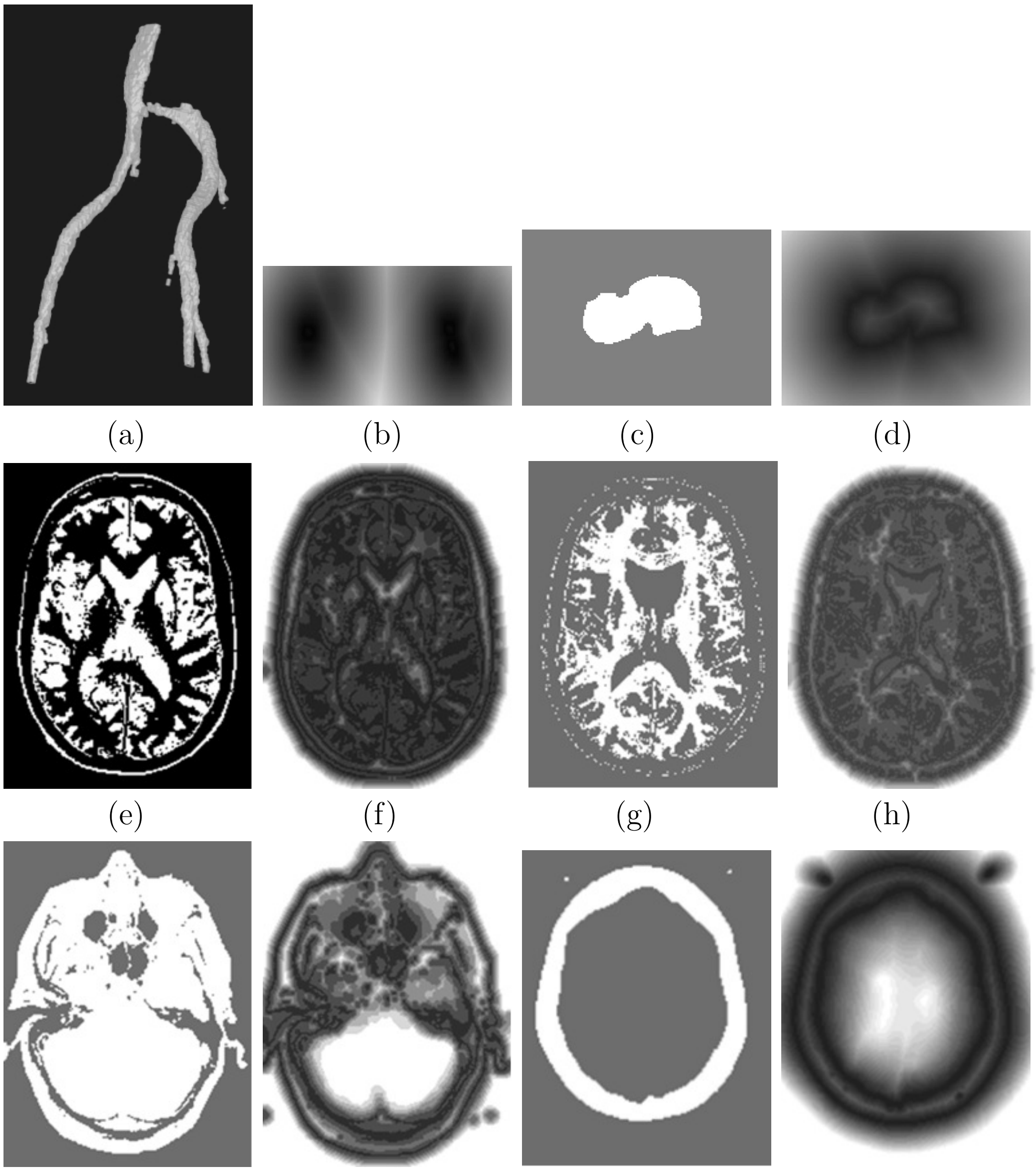
(S3) After step (S2) is finished, apply lines 5-9 of $LTDT$ with $d = k$. Then execute lines 11-13 to return DT.

This algorithm returns proper DT, and, assuming that $n_k \geq m$, runs in time $O(n/m)$. Moreover, if in any of the algorithms we replace $LTDT$ by its parallel version described above, the running time of the resulting algorithm will be reduced m-fold.

5 The experiments

In this section, we report the experimental results of applying some of the discussed algorithms on real medical image data for calculating the signed distance transform $SDT_I(x) = (-1)^{I(x)}\Delta(x, Bd_I)$ for two different definitions of the image boundary: Bd'_I (which is equivalent to using geometric boundary Bd^g_I) and Bd^{dig}_I.

All algorithms were implemented for the Euclidean distance and isotropic images. The programs were implemented on a cluster using the MPI/Open MPI standard. Each computer in the cluster is a Dell Optiplex GX620, which consists of a 3.6 GHz Intel Pentium D dual core [24] processor with 2 GB of RAM, running the Windows XP OS. These computers are connected by an inexpensive 1-gigabit switch (Dell Power-Connect 2608 8-port Ethernet switch). In our presentation of results, "Gold" denotes the gold standard method wherein distances are calculated via an exhaustive comparison. This method is not usable on large data sets and the symbol '$> n$ hr' means that we have terminated the execution of the program after n hours. In addition, for each tested image we compared the outputs of all tested algorithms, when appropriate, to experimentally confirm that their outputs actually agree, which should be the case for the exact DT algorithms. No discrepancies were detected.



(a) (b) (c) (d)

(e) (f) (g) (h)

(i) (j) (k) (l)

Figure 6. Slices from some of the 3D binary images used in the experiments and their respective distance transform images. (a,b): pelvic vessels; (a) shows a 3D rendition of the vessel tree and (b) shows the DT in a slice located near the bottom of the vessel tree. (c,d): talus bone of the foot. (e,f): gray matter. (g,h): white matter. (i,j): head soft tissue. (k,l): skull.

Image	Size	No. of spels	Object	Source
I_1	$256 \times 256 \times 256$	$16,777,216$	Pelvic vessels	MRI
I_2	$254 \times 214 \times 65$	$3,533,140$	Talus bone	MRI
I_3	$256 \times 256 \times 46$	$3,014,656$	Gray matter	MRI
I_4	$256 \times 256 \times 46$	$3,014,656$	White matter	MRI
I_5	$256 \times 256 \times 46$	$3,014,656$	Head soft tissue	MRI
I_6	$512 \times 512 \times 90$	$23,592,960$	Pelvic bone	CT
I_7	$512 \times 512 \times 90$	$23,592,960$	Pelvic soft tissue	CT
I_8	$512 \times 512 \times 256$	$67,108,864$	Pelvic bone	CT
I_9	$512 \times 512 \times 256$	$67,108,864$	Pelvic soft tissue	CT
I_{10}	$512 \times 512 \times 459$	$120,324,096$	Pelvic bone	CT
I_{11}	$512 \times 512 \times 459$	$120,324,096$	Pelvic soft tissue	CT
I_{12}	$1023 \times 1023 \times 128$	$133,955,712$	Head soft tissue	CT
I_{13}	$1023 \times 1023 \times 128$	$133,955,712$	Skull	CT

Table 1. A description of the binary images used in our experiments.

The binary images used in our experiments are obtained by thresholding patient MR and CT images of the head and pelvis from a variety of past/ongoing clinical research projects. For example, the MRI brain images pertain to Multiple Sclerosis patients, where our goal was to study the effectiveness of image-based markers in characterizing the disease. In the MRI pelvic images, our goal was to display the vessels free from clutter. In the pelvic and foot images, our goal was to create statistical models of the shape of the objects in these body regions for their automatic segmentation, delineation, and motion analysis. A description of these images, including the objects they represent and their sizes, is summarized in Table 1. Some binary image slices from these objects, together with their distance transforms obtained via LTSDT, are displayed in Figure 6.

Tables 2 and 3 summarize the experiments performed on eleven 3D binary images. Performances of both sequential and parallel algorithms are listed in these tables. The times reported in Table 2 constitute total computation time for the entire process — taking a binary image as input, doing all necessary operations, and producing a gray distance image as an output. The algorithm L2 (i.e., LTSDT with FT) computes the signed distance transform $SDT_I(x) = (-1)^{I(x)}\Delta(x, Bd_I^{dig})$, where $\Delta(x, Bd_I^{dig})$ is computed with $LTDT$, in which FT is recorded. The algorithm L1 (LTSDT with no FT) computes

No. of processors	Algo-rithm	Running time in seconds										
		I_3	I_4	I_5	I_6	I_7	I_8	I_9	I_{10}	I_{11}	I_{12}	I_{13}
1	E1	4	4	4	24	27	70	77	125	145	133	155
1	L1	3	3	2	20	26	56	77	105	141	125	145
1	L2	3	4	3	23	30	66	87	128	153	147	157
3	P-L2	3	3	3	24	30	67	90	127	170	155	161
7	P-L2	4	4	3	24	30	58	83	107	150	136	145
11	P-L2	4	4	4	24	32	55	78	101	145	131	138

Table 2. Comparison of running times of the SDT_I algorithms used with the non-symmetric boundary Bd_I^{dig}, see Sec. 2. Algorithms: E1 — EDT from ITK, output only distance transform, no feature transform; L1 — Our proposed LTSDT, output only distance transform, no feature transform; L2 — Our proposed LTSDT, output distance transform and feature transform; P-L2: Our proposed parallel LTSDT, output distance transform and feature transform.

the same values with $LTDT^*$, in which the value of the feature transform function F that $LTDT$ and DimUp return is replaced by the distance transform function F^*. This reduces memory use, and, slightly, the running time. The reduction works for the L_p distances, since in such settings the value $\Delta(x, q_\ell)$ can be easily calculated: $\Delta(x, q_\ell) = \left((x_d - y_d)^p + F^*(y)^p\right)^{1/p}$, where y is on the line parallel to the d-axis passing through x and $q_\ell = F(x)$.

Our motivation to parallelize distance transform algorithms was that, in several segmentation and registration methods, DT is called repeatedly (100s of times). Therefore, even if each (sequential) application were to take only a few minutes, the total time before the main application is completed could be prohibitive. Thus parallelization has the potential to save a considerable amount of time in such precesses. Although all algorithms presented here run in linear time with respect to the number of spels, this is not born out in Table 2. Surprisingly, this is mainly due to the fact that the actual distance computation part for the algorithm is a small fraction (4% - 25%) of the total time. A bulk of the time is taken up by the three house keeping operations — creating boundary image for the input binary image (30% - 50%), network transmission of image chunks between Master and Slaves (17% - 21%), combining results and producing output (25% - 40%). A break up of these factors is listed in Table 3 for the four largest images. It is clear that the actual DT computation time is inversely proportional to the

Image	No. of processors	Running time in sec				No. of processors times C2
		C1	C2	C3	C4	
I_{10}	3	35	33	19	40	99
	7	35	14	18	40	98
	11	35	9	17	40	99
I_{11}	3	70	30	22	48	90
	7	70	13	19	48	81
	11	70	8	19	48	88
I_{12}	3	62	25	33	35	75
	7	62	11	28	35	77
	11	62	7	27	35	77
I_{13}	3	70	22	31	38	66
	7	70	9	28	38	63
	11	70	6	24	38	66

Table 3. Break up of the four component times in the parallel implementation. C1 – boundary finding initial operation, C2 – actual calculation of DT, for two-dimensional (co-dimension one) hyperplanes, C3 – image data transfer between Master and Slaves, C4 – combination of the results, including finding DT for the last dimension. The last column represents the combined time the slaves use to calculate DT, which, as expected, is approximately the same for each image, independently of the number of processors used.

number of processors used and linear with respect to the image size. It is also clear that, since the actual DT valuation is very rapid, speed up in DT operations on binary images can be harnessed only by parallelizing some of the house-keeping, particularly the boundary finding, operations.

The fact that the maximal running times of EDT and $LTSDT$ from Table 2 are of linear order of magnitude with respect to the image size suggests that the actual times should also be approximately linearly dependent on the image size. To test this hypothesis, we displayed the times estimated in our experiments, as functions of image size, in Figure 7 for the sequential implementation. Indeed, for all algorithms the relation is approximately linear.

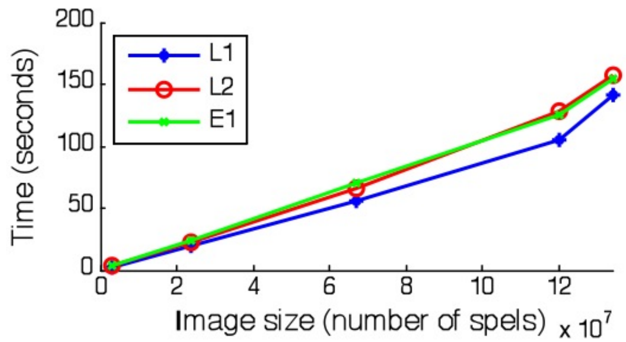


Figure 7. A plot of the running time of sequential algorithms: $LTSDT$ (both versions) and EDT with respect to image size for the sequential algorithms. As expected, the relation is approximately linear. (We display here the results for only images I_5, I_6, I_7, I_{10}, and I_{13}.)

Figure 7 and Tables 1-3 also show that both versions of LTSDT outperform EDT, the difference in performance being greater as the image size increases.

No. of processors	Algorithm	Running time in sec		
		J_1	J_2	J_3
1	Gold	5125	$>$ 10hr	$>$ 10hr
1	$GBDT$	3	18	161
3	parallel $GBDT$	3	19	175
7	parallel $GBDT$	3	18	153
11	parallel $GBDT$	3	18	149

Table 4. Comparison of running times of the SDT_I algorithms used with the geometric boundary Bd_I^g, implemented with Bd'_I, see Sec. 2.

Table 4 reports the experimental comparison of $GBDT$ and a version of "Gold" for this setting. The grid size is increased 8-fold (doubled in each dimension), so we ran the experiments on smaller 3D binary images J_1-J_3 of respective sizes: $128 \times 128 \times 24$, $256 \times 256 \times 46$, and $512 \times 512 \times 96$. Notice that the size of images J_2 and I_3 are the same, so the actual image on which $GBDT$ calculates DT is 8 times the size of that for $LTSDT$. The actual running time of $GBDT$ in that image is 13 times that of $LTSDT$, rather than the expected 8 times. This perhaps has something to do with some peculiarity of our implementations.

Object	diameter in mm of the	
	digital object	geometric object
I_1: pelvic vessels	298.6719	298.8380
I_2: talus bone	51.0434	51.3698
I_3: gray matter	211.8546	212.2860
I_4: white matter	213.0751	213.5265
I_5: head soft tissue	215.8026	216.2616
I_6: pelvic bone	344.0323	344.4512
I_{13}: skull	149.8486	150.2962

Table 5. Object diameter (in mm) calculated using LTdiam for the seven objects listed in Table 1. The object is defined as digital or geometric, as explained in the text.

Although the Gold completed calculation of DT only for the smallest image J_1, it should be stressed that its output fully agreed with that from *GBDT*. Note also that for *GBDT* the relation between image size and running time seems also to be linear in nature.

The diameters of 7 objects listed in Table 1 obtained by LTdiam algorithm are listed in Table 5. For each image $I\colon C \to \{0,1\}$ we identified its foreground in two different ways: as a digital object $F_I = \{c \in C\colon I(c) = 1\}$, and as a geometric version F_I^g of F_I, which is defined as a union of all unit cubes centered at spels c from F_I. Actually, the diameter of F_I^g is equal to the diameter of the object $F_I' = F_I^g \cap C'$, where the C' is the double resolution scene. (The argument for this is similar to that for Theorem 2.2.) Thus, to calculate its diameter we actually apply LTdiam to F_I'. Notice that the diameters of F_I^g are slightly larger than those for F_I, as can be expected.

6 Concluding remarks

Distance transform is a computationally expensive but ubiquitously needed operation in image processing. Given its extensive use, expense, the ever increasing spatial and temporal resolution of medical images, and the need to handle 2D, 3D, and 4D concepts for objects and boundaries in relation to DT, efficient, generalizable, and parallelizable schemas for DT are very crucial. The algorithm of Maurer *et al.* [10] was an important contribution from these considerations. In this paper, we have extended their method in two ways. First, we have constructed a full theoretical justification of those ideas. Second, we have designed a new DT definition with respect to the geometric boundary, which affords nicer theoretical properties and more refined distance values, and we have shown that the ideas underlying [10] can be extended to this new design. Although it is computationally more expensive, the new algorithm *GBDT* is a preferred method for an accurate, true, and a theoretically consistent distance transform. Note that this becomes especially important when measurements are made based on DT. Finally, since the actual DT operations in the family studied here are extremely rapid, parallelization for saving considerable amount of time on repeated use of DT (100s of times) on binary images should focus on housekeeping operations that support DT.

Acknowledgement The authors like to thank Thaw Htaik for assisting them in setting up the parallel environment and helping to run the experiments in this environment.

References

[1] Bai, X., and Latecki, L. J., and Liu, W.: Skeleton Pruning by Contour Partitioning with Discrete Curve Evolution, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **29**(3) (2007), 449–462.

[2] Beristain, A. and Grana, M.: Pruning algorithm for Voronoi skeletons, *Electronics Letters* **46**(1) (2010), 39–41.

[3] Ciesielski, K.C., Udupa, J.K., and Grevera, G.: *Linear time distance transform algorithm*, MIPG Technical Report #337, Medical Image Processing Group, Department of Radiology, University of Pennsylvania, Philadelphia, 2006.

[4] Cuisenaire, Olivier: Distance transformations: fast algorithms and applications to medical image processing, Dissertation, 1999.

[5] Grevera, G.J.: Chapter 2: Distance Transform, in *Parametric and Geometric Deformable Models: An application in Biomaterials and Medical Imagery*, Springer Publishers, Jasjit S. Suri and Aly Farag, editors.

[6] Grevera, G.J. and Udupa, J.K.: Shape-based interpolation of multidimensional grey-level images, *IEEE Transactions on Medical Imaging* **15**(6) (1996), 881–892.

[7] Grevera, G., Udupa, J., Odhner, D., Zhuge, Y., Souza, A., Iwanaga, T., and Mishra, S.: CAVASS: a Computer Assisted Visualization and Analysis Software System, *J. Digital Imaging* **20**(1) (2007), 101–118.

[8] Herman, G.T., Zheng, J., and Bucholtz, C.A.: Shape-based interpolation, *IEEE Computer Graphics and Applications* **12**(3) (1992), 69–79.

[9] Ibanez, L. and Schroder, W.: *The ITK Software Guide 2.4*, chapter 6, Kitware Inc., 2005.

[10] Maurer, C.R., Jr., Qi, R., and Raghavan, V.: A linear time algorithm for computing exact Euclidean distance transforms of binary images in arbitrary dimensions, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **25**(2) (2003), 265–270.

[11] Stephen M. Pizer, Guido Gerig, Sarang C. Joshi, Stephen R. Aylward: Multiscale medial shape-based analysis of image objects, *Proceedings of the IEEE* **91**(10):1670–1679, 2003.

[12] Raya, S.P. and Udupa, J.K.: Shape-based interpolation of multidimensional objects, *IEEE Transactions on Medical Imaging* **9**(1) (1990), 32–42.

[13] Royden, H.L.: *Real Analysis*, MacMillan Publishing Company, New York, 1988.

[14] Tustison, N.J., Siqueira, M., and Gee, J.C.: *N*-D Linear Time Exact Signed Euclidean Distance Transform, *Insight Journal*, January-June, 2006 (http://hdl.handle.net/1926/171).

[15] Udupa, J.K.: Multidimensional digital boundaries, *Graphical Models Image Processing* **56**(4) (1994), 311–323.

[16] Udupa, J.K. and Grevera, G.J.: Go digital, go fuzzy, *Pattern Recognition Letters* **23** (2002), 743–754.

[17] Y. Ge and J. M. Fitzpatrick: On the Generation of Skeletons from Discrete Euclidean Distance Maps, *IEEE Trans. PAMI* **18**(11):1055–1066, 1996.

[18] L. da Fontoura Costa: Robust Skeletonization through Exact Euclidean Distance Transform and its Application to Neuromorphometry, *Real-Time Imaging* **6**(6):415–431, 2000.

[19] A. Souza, and J.K. Udupa: Automatic Landmark Selection for Active Shape Models, *Proc. SPIE Medical Imaging* **5747**:1377–1383, 2005.

[20] A. Tsai, W. Well, C. Tempany, E. Grimson, and A. Willsky: Mutual information in coupled multi-shape model for medical image segmentation, *Medical Image Analysis* **8**(4):429–445, 2004.

[21] G.E. Marai, D.H. Laidlaw, and J.J. Crisco: Super-Resolution Registration Using Tissue-Classified Distance Fields, *IEEE Trans. Medical Imaging* **25**(2):177–187, 2006.

[22] Nyul, L.G., Udupa, J.K, Saha, P.K.: Incorporating a measure of local scale in voxel-based 3-D image registration, *IEEE Transactions on Medical Imaging* **22** (2003), 228–237.

[23] Theresse P., Arbuck, S.G., Eisenhauer, E.A., et al.: New guidelines to evaluate the response to treatment in solid tumors, European Organization for Research and Treatment of Cancer, National Cancer Institute of the United States, *Journal of National Cancer Institute* **92**: 205–216, 2000.

[24] Intel Pentium D 800 Processor 800 Sequence Datasheet (download.intel.com/support/processors/pentiumd/sb/307506.pdf), 2006.

[25] J.L. Díaz De León S. and J.H. Sossa-Azuela: Mathematical Morphology Based on Linear Combined Metric Spaces on Z^2 (Part I): Fast Distance Transforms, *Journal of Mathematical Imaging and Vision* **12**(2): 137–154, 2000.

[26] Andrew J.H. Mehnert and Paul T. Jackway: On Computing the Exact Euclidean Distance Transform on Rectangular and Hexagonal Grids, *Journal of Mathematical Imaging and Vision* **11**(3): 223–230, 1999.

[27] Ming Xu and David Pycock: A Scale-Space Medialness Transform Based on Boundary Concordance Voting, *Journal of Mathematical Imaging and Vision* **11**(3): 277–299, 1999.

[28] Ron Kimmel, Nahum Kiryati and Alfred M. Bruckstein: Sub-pixel distance maps and weighted distance transforms, *Journal of Mathematical Imaging and Vision* **6**(2-3): 223–233, 1996.

[29] Ralph Costa Teixeira: Medial Axes and Mean Curvature Motion II: Singularities, *Journal of Mathematical Imaging and Vision* **23**(1): 87–105, 2005.

[30] Sung Woo Choi and Hans-Peter Seidel: Linear One-Sided Stability of MAT for Weakly Injective Domain, *Journal of Mathematical Imaging and Vision* **17**(3): 237–247, 2002.