Linear time algorithms for exact distance transform: elaboration on Maurer *et al.* algorithm

Krzysztof Chris Ciesielski,[a,b,*] Jayaram K. Udupa,[b,†] Xinjian Chen,[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.* [7] published a paper describing an algorithm that computes the exact distance transform in a 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 $1 \le p \le \infty$, 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 the computation time savings associated with such an implementation. The discussed implementations will be made available as a part of the CAVASS software system developed and maintained in our group [5]. On the theoretical side, we prove that our version of the signed distance transform algorithm, *GBDT*, returns, in a linear time, the *exact* value of the distance from the *geometrically defined object boundary.* We notice that, actually, the precise form of the algorithm from [7] is not well defined for L_1 and L_∞ metrics and point to our complete proof (not given in [7]) that all these algorithms work correctly for the L_p-metric with $1 < p < \infty$.

1. INTRODUCTION AND PRELIMINARIES

Distance transform in digital spaces is an important tool in image processing [2–4,6,8,10,11]. It finds widespread use in a variety of operations such as filtering, interpolation, segmentation, registration, shape analysis, and image compression. For example, the set of points of a fixed distance r from a surface S, treated as a front propagated with a constant speed, represents the front position at time $t = r$. Distance transform is most typically implemented in a signed form, discussed in the next section. (See e.g. [12].)

The key algorithm that we discuss here, the linear time distance transform *LTDT*, is our version of the algorithm of Maurer *et al.* [7]. This algorithm works in any dimension $k \ge 1$, on binary images defined on the rectangular grids $C = \{x^1_0, \dots, x^1_{n_1-1}\} \times \cdots \times \{x^k_0, \dots, x^k_{n_k-1}\} \subset \mathbb{R}^k$ and for the class of distances Δ, which includes the Euclidean distance and, more generally, any L_p metric for $1 < p < \infty$. Recall that, for $1 \le p < \infty$, 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.

In what follows we always assume that $x^d_0 < \cdots < x^d_{n_d-1}$ for every $d = 1, \dots, k$, although we need not assume that the images are isotropic, that is, that all numbers $x^d_{i+1} - x^d_i$ can be different. Nevertheless, all our figures are presented for isotropic images and our implementations are tested in this case. The elements of a grid C will be referred to as *spels*, short for space elements.

We start the paper with a discussion of the ways that the *LTDT* algorithm can be used to create different versions of distance transform algorithms. The theoretical discussion of *LTDT* is postponed to the second part of this article. The paper is finished with an experimental comparison of different forms of the algorithm and with a description of their parallelization.

[*]K.C. Ciesielski was partially supported by NSF grant DMS-0623906. 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.

Medical Imaging 2009: Image Processing, edited by Josien P. W. Pluim, Benoit M. Dawant, Proc. of SPIE Vol. 7259, 72592T · © 2009 SPIE
CCC code: 1605-7422/09/$18 · doi: 10.1117/12.811010

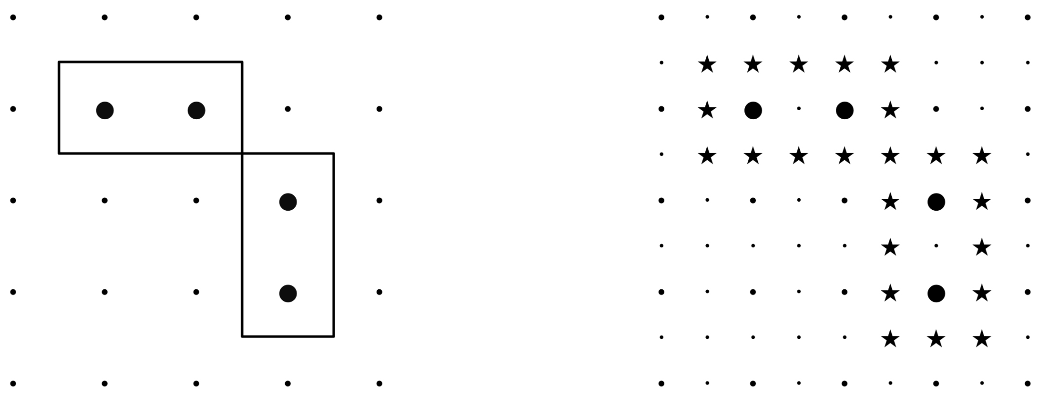

Fig. 1. Left: Geometric boundary Bd_I^g of a binary image I on a 5×5 rectangular grid C with four foreground spels marked by large dots. Right: The same binary image on the 9×9 double resolution grid C', where the smallest dots represent added spels. The digital boundary Bd'_I on C', marked by stars, consists of the intersection of Bd_I^g with C'.

2. SIGNED DISTANCE TRANSFORM ALGORITHMS

Let I be a k-dimensional binary image, that is, a function from $\Omega \subset \mathbb{R}^k$ into $\{0,1\}$. 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 image foreground $F_I = \{x \in \Omega \colon I(x) = 1\}$ and its background $B_I = \{x \in \Omega \colon I(x) = 0\}$. The main variability in this formula is caused by use of different definitions of the boundary Bd_I. More precisely, for geometrical scenes, 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 Fig. 1), so alternative definitions of the digital boundary are often used. 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 [9], 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 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.

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 $GBDT$ described below.

Agreement with 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 contour) 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 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 only sense to use the same formula for its digital version, used in the numerical approximation. Here, the $GBDT$ implementation 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, so the algorithm for calculating SDT is invoked multiple times.

Agreement with cube interpretation of spels. It is a 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 [10]. (See Figure 1.) There are many advantages of such definitions of a digital boundary (see [11]) in visualization, processing, analysis, and finding of such surfaces. For example, when distance transforms are used in interpolating object shape [8], it has been shown that distances determined with respect to boundaries so defined lead to more accurate results [4,6]. The point here is that SDT_I^g is equal to the boundary obtained with a cube-based interpretation of spels.

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 STD is defined via formula $SDT_J(x) = (-1)^{J(x)}\Delta(x, Bd_J)$ and the property (r) is satisfied by a digital image $I\colon C \to \{0,1\}$, 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 $\Delta(x, Bd_{1-I}) = |-STD_{1-I}(x)| = |STD_I(x)| = \Delta(x, Bd_I) = 0$, 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 $J\colon C \to \{0,1\}$ is defined, for example, as the set of all $c \in C$ for which there is an adjacent $d \in C$ in Bd_J 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 directed 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. Let $Bd'_I = Bd_I^g \cap C'$ — see Figure 1. The key fact 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 $GBDT$, is the following result.

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

PROOF. Clearly $\Delta(c, Bd_I^g) \leq \Delta(c, Bd'_I)$, 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 with 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 $GBDT$, 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 $\{0, \ldots, n_1 - 1\} \times \cdots \times \{0, \ldots, n_k - 1\}$. Similar identification will be done throughout the paper, including also grid C' below.

Algorithm $GBDT$

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

Output: A signed distance transform $SDT_I\colon C \to [0,\infty]$, $SDT_I(c) = (-1)^{I(c)}\Delta(c, Bd_I^g)$.

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 $1 \leq d \leq k$ do
 3.      for $i = 1$ to $k$ do
 4.          if $i \neq d$ then $y_i = x_i$ else $y_i = x_i + 1$;
 5.      endfor;
 6.      if $y \in C$ and $I(x) \neq I(y)$ then
 7.          set $I'(c) = 0$ for every $c \in C'$ on the boundary face between $x$ and $y$;
 8.      endif;
 9.  endfor;
10.  invoke $LTDT$ with $I'$ and appropriate $\Delta$ 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 2.3. *Algorithm GBDT invoked with the L_p distance, $1 < p < \infty$ on binary rectangular digital image I returns the signed distance to the geometric boundary Bd_I^g between foreground and background. Moreover, GBDT runs in $O(n)$ time.*

PROOF. 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 $c \in C'$ we have $FT(c) = \Delta(c, Bd'_I) = \Delta(c, Bd_I^g)$, 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. ■

3. *LTDT* AND ITS PARALLELIZATION

The $LTDT$ algorithm represents our implementation, with minor modifications, of the Maurer *at al.* algorithm from [7]. Therefore, we describe here only its parts necessary for describing its parallelization. The full description of $LTDT$ and a complete proof of its correctness will be presented in a full journal version of this paper. (But see also [1].)

Actually, $LTDT$ calculates a *feature transform* FT for I, that is, a function $FT\colon C \to B_I \cup \{\emptyset\} \subset C \cup \{\emptyset\}$ with $FT(c) = \emptyset$ whenever $DT(c) = \infty$, while otherwise, $FT(c) \in B_I$ is such that $\Delta(c, FT(c)) = \Delta(c, B_I) = DT(c)$. Only at the output stage DT is calculated from FT by calling the function $DT(c) = \Delta(c, FT(c))$.

The calculation of FT is done recursively on the dimension of the image. To express it precisely, we will need the following notation, in addition to that already introduced earlier. For $0 \leq d \leq k$ and $x \in C$, let $H_d(x) = \{c \in C\colon c_i = x_i \text{ for all } d < i \leq k\}$ 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 one-dimensional row in $\mathbb{R}^k$ parallel to the d-th axis, that is, $\mathbb{R}_d(x) = \{c \in \mathbb{R}^k \colon c_i = x_i \text{ for all } i \neq d\}$. 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 $F(x)$ is the correct value of a closest feature transform for $I \restriction H_d(x)$, the image I restricted to $H_d(x)$. That is, $F(x) = \emptyset$ when $B_I \cap H_d(x) = \emptyset$; otherwise, $F(x) \in B_I \cap H_d(x)$ and $\Delta(x, F(x)) = \Delta(x, B_I \cap H_d(x))$. 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 true 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 ∞ otherwise.

The key recursively used subroutine of $LTDT$, called DimUp (a variant of VoronoiFV from [7]), has the following properties.

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 $\mathbb{R}_d(x) \cap C$.

The full description of DimUp and the proof that it has the above mentioned properties can be found in the preprint [1]. It will be also published in the full journal version of this paper.

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:	A binary image $I \colon C \to \{0,1\}$; dimension k (≥ 2) of the image; $n_1, \ldots, n_k$ — the size of the grid.
Output:	A distance transform $DT \colon C \to [0, \infty]$ 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.     if $F(x) = \emptyset$ then $DT(x) = \infty$ else $DT(x) = \Delta(x, F(x))$;
  13. endfor;
end
```

Lines 1-10 of this algorithm represent procedure ComputeFT from [7]. In lines 1-3, we define F as 0-dimensional approximation of FT. Our main contribution here is the proof of the following Theorem. In its proof, we assume that the algorithm DimUp works correctly.

THEOREM 3.1. *For every binary image I on a rectangular grid $C = \{0, \ldots, n_1 - 1\} \times \cdots \times \{0, \ldots, n_k - 1\}$, the algorithm $LTDT$ returns the exact distance transform F 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.

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 the $(d-1)$-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 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 the 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 the calculation of $\Delta(x, y)$ is $O(1)$) and the resulting DT is indeed an exact distance transform for I. ■

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 $LTDT$.

(1) 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.

(2) 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 $O(n_k/m)O(n/n_k) = O(n/m)$.

(3) After step (2) 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 $LTDT$ with its parallel version described above, the running time of the resulting algorithm will be reduced m-fold.

4. THE EXPERIMENTS

In this section we report the experimental results of applying on real medical image data some of the discussed algorithms 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 run on an Intel Pentium IV PC with 3.4 GHZ DuoCore CPU, 2.0G RAM on Windows XP OS. Our experimental results are presented in the following tables. "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 to experimentally confirm that their outputs actually agree, which should be the case for the exact DT algorithms. No discrepancies were detected.

Table 1 reports the experiments performed on the 3D binary images I_1-I_6, created from actual medical images, of respective sizes: $256 \times 256 \times 46$, $512 \times 512 \times 203$, $512 \times 512 \times 459$, $1023 \times 1023 \times 128$, $1023 \times 1023 \times 224$, and $1023 \times 1023 \times 256$. Some slices of the original MR image I_1 of a human brain, its binary version with white matter segmented approximately by simple thresholding, and the resulting DT (distance from the object) are displayed in Fig. 2. Note that, for larger images, $LTSDT$ outperforms EDT.

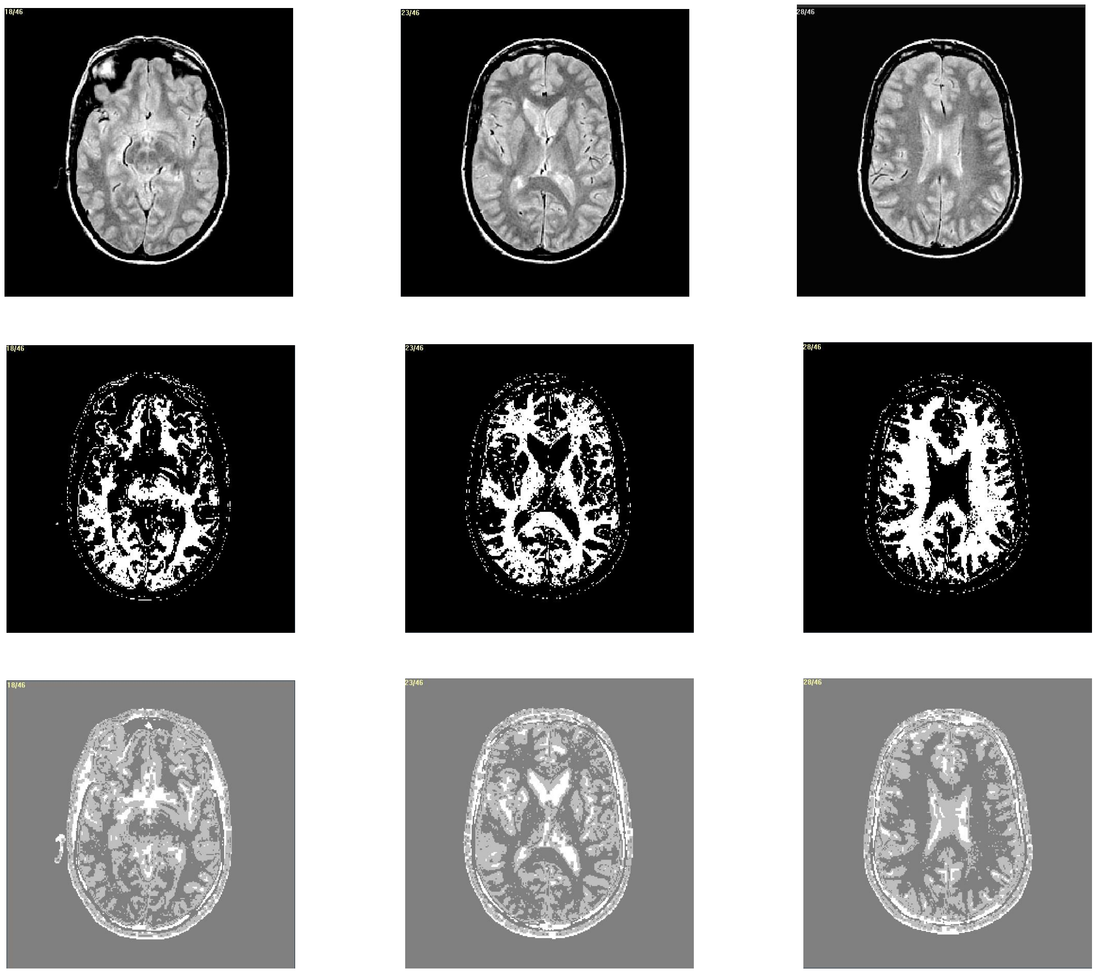

Fig. 2. Top row: Three typical MR slices of the image I_1 of a human brain. Middle row: The same slices of the binary image obtained from the original via thresholding for highlighting of the white matter. Bottom row: The values of the distance transform for the entire 3D image displayed on the slices from the middle row. The values are presented in the form of pixel intensities: the lighter the pixel, the larger is its distance from the foreground (so, the white matter appears darker). For better clarity, we display the output of $LTDT$, rather than of a signed DT. Also, the area outside the head was left uniformly dark for better contrast and its intensity does not reflect the value of DT.

Algorithm	Running time in sec					
	I_1	I_2	I_3	I_4	I_5	I_6
Gold	2810	> 2hr	> 2hr	> 2hr	> 2hr	> 2hr
EDT (from ITK)	10	231	550	561	1135	1213
$LTSDT$	10	247	572	511	987	1156

Table 1. Comparison of running times of the SDT_I algorithms used with the non-symmetric boundary Bd_I^{dig}, see Sec. 2.

The fact that the maximal running times of EDT are $LTSDT$ 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 of our experiments, as functions of image size, in Figure 3. Indeed, for both algorithms the relation is approximately linear.

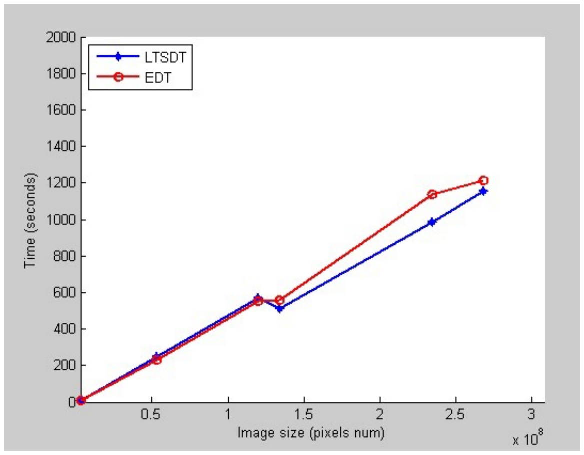

Fig. 3. The display of running time of $LTSDT$ and EDT with respect to image size. As expected, the relation is approximately linear.

Table 2 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 run the experiments on the 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_1 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 that expected 8 times. This perhaps has something to do with some peculiarity of our implementations.

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.

Algorithm	Running time in sec		
	J_1	J_2	J_3
Gold	24669	> 10hr	> 10hr
$GBDT$	17	130	1202

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

5. 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.* [7] 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 have shown that the ideas underlying [7] can be extended to this new design. Although it is computationally more expensive, the new algorithm $GBDT$ is a preferred method for accurate, true, and theoretically consistent distance transform.

REFERENCES

1. 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.
2. Cuisenaire, Olivier: Distance transformations: fast algorithms and applications to medical image processing, Dissertation, 1999.
3. 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.
4. 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.
5. 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.
6. Herman, G.T., Zheng, J., and Bucholtz, C.A.: Shape-based interpolation, *IEEE Computer Graphics and Applications* **12**(3) (1992), 69–79.
7. 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.
8. Raya, S.P. and Udupa, J.K.: Shape-based interpolation of multidimensional objects, *IEEE Transactions on Medical Imaging* **9**(1) (1990), 32–42.
9. 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).
10. Udupa, J.K.: Multidimensional digital boundaries, *Graphical Models Image Processing* **56**(4) (1994), 311–323.
11. Udupa, J.K. and Grevera, G.J.: Go digital, go fuzzy, *Pattern Recognition Letters* **23** (2002), 743–754.
12. Ibanez, L. and Schroder, W.: *The ITK Software Guide 2.4*, chapter 6, Kitware Inc., 2005.