SIViP
DOI 10.1007/s11760-012-0419-9

ORIGINAL PAPER

A fast algorithm for computation of discrete Euclidean distance transform in three or more dimensions on vector processing architectures

Yuriy Mishchenko

Received: 11 July 2012 / Revised: 3 December 2012 / Accepted: 5 December 2012

Abstract In this note, we introduce a function for calculating Euclidean distance transform in large binary images of dimension three or higher in Matlab. This function uses transparent and fast line-scan algorithm that can be efficiently implemented on vector processing architectures such as Matlab and significantly outperforms the Matlab's standard distance transform function "bwdist" both in terms of the computation time and the possible data sizes. The described function also can be used to calculate the distance transform of the data with anisotropic voxel aspect ratios. These advantages make this function especially useful for high-performance scientific and engineering applications that require distance transform calculations for large multidimensional and/or anisotropic datasets in Matlab. The described function is publicly available from the Matlab Central website under the name "bwdistsc", "Euclidean Distance Transform for Variable Data Aspect Ratio".

Keywords Distance transform · Line-scan distance transform algorithm · Linear-time distance transform algorithm · Distance transform Matlab · Anisotropic aspect ratio

1 Introduction

Distance transforms play an important role in many image and data processing applications, and the development of efficient computational tools for calculating distance transforms is of great importance [1–6]. Two-dimensional distance transforms are broadly used in image processing and computer vision applications, and higher dimensional distance transforms play important role in pattern recognition. Several different distance metrics, such as cityblock, chessboard, and Euclidean metrics have been used in the context of such applications. Of these, the Euclidean metric is one of the most appropriate and the most natural choice for many applications given its invariance under rotations. However, the computation of the exact Euclidean distance transform (EDT) is generally time consuming: while efficient algorithms for non-Euclidean distance transforms have been reported since 1966, fast algorithms for EDT started to appear only in the 1990s [12,13].

Treated as a global operation, EDT can be computed in principle by an exhaustive brute-force search—by calculating for each pixel of the image the distance between that pixel and all other pixels, and computing the smallest distance. However, such an algorithm would require $O(N^2)$ operations (N is the number of pixels in the image) and would not be practically suitable. Among more efficient algorithms for calculating EDT are ordered propagation algorithms, raster scan algorithms, and dimension-induction algorithms.

The ordered propagation algorithms were among the first efficient algorithms proposed for EDT. In ordered propagation, the distance information is propagated away from the boundary of the object [14–18]. While such algorithms can be exceedingly fast for simpler metrics [19], in the case of EDT such algorithms are often affected by numerical errors and can be computationally expensive, while improving their performance can be difficult.

Raster scan algorithms are based on the conjecture that it should be possible to deduce the value of the EDT distance at a pixel from the value of such distances at pixel's neighbors. These algorithms perform computation of the distance transform by subsequently scanning the pixels in the image and updating the distance information at neighbor pixels. The raster scan methods are fast since the pixels are

Y. Mishchenko (✉)
School of Engineering, Toros University, Mersin, Turkey
e-mail: yuriy.mishchenko@gmail.com

Published online: 04 January 2013

processed in series using a concise update step. However, while the above conjecture is correct for simpler cityblock or chessboard metric, it is not strictly so for the Euclidean metric restricted to the locations on an integer grid. For this reason, the raster scan algorithms, while extremely successful for non-Euclidean metrics, can produce only an approximate result for EDT. The first raster scan algorithm for non-Euclidean metrics was proposed by Rosenfeld and Pfaltz [3], and Danielsson [20] subsequently proposed a method that obtains EDT distance information in such a raster scan from neighbor points. A method for calculating EDT using chamfer metrics was proposed by Borgefors [21]. Chamfer metrics approximate the value of EDT at a pixel via an appropriately chosen weighted average of the distances in a neighborhood of that pixel, thus making the calculations possible using the raster scans. Svensson and Borgefors introduced chamfer metrics method with $5 \times 5 \times 5$ neighborhoods for three-dimensional EDT [22], and Strand and Borgefors described using face-centered cubic and body-centered cubic neighborhoods to compute three-dimensional EDT with higher precision [23]. Cuisenaire and Macq proposed a method using a sequence of neighborhoods to approximate EDT with high precision [17].

Given that the central assumption of the raster scan algorithms is not strictly satisfied in EDT, all raster scan EDT algorithms are approximate algorithms. For this reason, in 1990s, a series of efficient algorithms for exact EDT had appeared. These algorithms can be divided generally into three broad categories: wavefront propagation algorithms, Voronoi diagram algorithms, and dimension-induction algorithms. The wavefront propagation algorithms calculate EDT by propagating a wavefront at a constant speed in the radial direction away from the image's boundary, for example, using Fast Marching Method [24]. While such algorithms are $O(N)$ algorithms, they can be susceptible to numerical errors and can be overall computationally expensive. Voronoi diagram EDT algorithms build upon fast $O(N)$ algorithms for constructing Delaunay tessellations and Voronoi diagrams [25–28]. Voronoi diagram for a set of points or sites $\{p_1, p_2, \ldots, p_n\}$ is a partition of space into a set of so-called Voronoi cells consisting of every point whose distance to p_k is less than or equal to its distance to any other site. If the Voronoi diagram for a set of points had been constructed, then the distance transform can be trivially calculated in $O(N)$ time by calculating the distances for the points in every Voronoi cell to their respective site p_k [7–10,29]. The dimension-induction algorithms construct EDT inductively, by first constructing the one-dimensional distance transform for each row (or column) of the image independently, and then using this intermediate result to construct the full two-dimensional distance transform [1,30–32]. The first dimension-induction algorithm had been proposed by Rosenfeld and Pfaltz [3] and subsequently developed by Saito and Toriwaki [33] as well as others [11,34,35]. In these algorithms, the speed up during the second phase is achieved by calculating the lower envelop of the parabolas to avoid exhaustive search for the nearest image point in each column. Dimension-induction algorithms are $O(N)$ algorithms and can be applied in any number of dimensions. Representatives of this group are also among the fastest exact EDT algorithms currently available [12].

In this paper, we introduce an original fast dimension-induction algorithm and a respective Matlab implementation for calculating exact EDT of large binary images of dimension three or higher.

Matlab is a very popular computational system widely employed for complex engineering and scientific calculations. Thanks to its accessibility and rich computational libraries, Matlab is widely used around the world for fast prototyping as well as, in many cases, analysis of actual scientific and engineering data. Good EDT algorithms for this system, therefore, are of significant importance. Yet, while in two dimensions, a very good implementation for EDT is available in Matlab [7], for dimension three or higher, the situation is much less satisfactory. In this work, we describe and publicly share via Matlab Central website a Matlab function that calculates high-dimensional EDTs significantly faster than the Matlab's standard distance transform function "bwdist", can do so for significantly larger images, and also for the images with anisotropic voxel aspect ratio, such as produced in confocal microscopy, computed tomography, or magnetic resonance imaging. This last feature, in particular, makes it possible to calculate EDT for such data directly, without resampling or any other additional pre-processing—an important advantage for the mentioned applications.

These advances are of particular interest to high-performance scientific and engineering applications requiring calculations of large multidimensional distance transforms in Matlab. For example, the described Matlab function had been used by the author in the context of the study of neuropil organization using large-scale serial electron microscopy reconstructions of neural tissue in [36], where this function had been used for calculating the distance transforms in anisotropic 3D electron microscopy imaging datasets over 1 Gvoxel in size. The described function also had found numerous applications in the research and engineering projects of other researchers, ranging from the analysis of transport properties of porous materials [37] to segmentation of functional magnetic resonance imaging (fMRI) medical data [38].

The EDT algorithm we introduce here builds upon the linear-time dimension-induction algorithm of [11] but modifies it significantly to make it more suitable for vector processing architectures such as Matlab. In particular, the algorithm in [11] uses a list comprising parabolas to represent and update the lower envelop of the parabolas in the second-phase scan. This makes vectorization of that algorithm

difficult and forces the second-phase scan to be performed over all pixels in the image essentially serially, one-by-one. The present algorithm formulates the update step in a form that is more suitable for vectorization. This allows our algorithm, implemented as a plain script in Matlab, to outperform the standard compiled Matlab's distance transform function "bwdist" both in terms of the computation time and the largest possible data size. The described algorithm can be also advantageous for other high performance computing platforms that use vector processing, such as High Performance Fortran or C/C++ MPI super-computing applications.

The paper is organized as follows: In Methods, we describe the dimension-induction algorithm used as the basis for the described Matlab function here. In results, we survey several examples of this function's applications and perform experiments to demonstrate that it achieves superior performance relative to the Matlab's standard distance transform function "bwdist" or any other state-of-the-art EDT algorithms' implementations in Matlab. We give concluding remarks in summary.

2 Methods

Discrete Euclidean distance transform of a binary image $\mathcal{I}$, defined on a set of pixels $\mathcal{X}$, is defined as a function that assigns to every point in $\mathcal{X}$ the distance from that point to the closest "on" pixel in $\mathcal{I}$. More specifically,

$$D(\mathbf{x}) = \min_{\mathbf{y}|\mathcal{I}(\mathbf{y})=1} \|\mathbf{x}-\mathbf{y}\|_2. \tag{1}$$

Among many approaches for calculating this distance transform, the central idea of the dimension-induction algorithms is in constructing the distance transform iteratively, by building a family of lower k-dimensional transforms for "slices" of $\mathcal{I}$ independently, and then advancing k inductively by performing scans along certain lines in the family of the lower dimensional distance transforms.

Consider, for example, the calculation of the distance transform of a binary image in three dimensions. Suppose that we have pre-computed all two-dimensional distance transforms for all "z-slices" of the image, $D(x, y; z)$. In order to construct the 3D distance transform $D(x, y, z)$ from a family of such transforms $D(x, y; z)$, let us find for each point (x^*, y^*, z^*) the slice z along z-axis such that $D(x^*, y^*; z)^2 + (z - z^*)^2$ is the smallest. It can be shown then that the value of $D(x^*, y^*, z^*)$ can be obtained directly from such z as $D(x^*, y^*, z^*)^2 = D(x^*, y^*; z)^2 + (z^* - z)^2$ for such a z.[1]

It is easy to see that the above process can be straightforwardly generalized to higher dimensions. Specifically, let $D(x_1, \ldots, x_k; x_{k+1}, \ldots, x_d)$ be the distance transform for $\mathcal{I}$ in the k-dimensional slice defined by the last $d-k$ coordinates $(x_{k+1}, \ldots, x_d)$; then,

$$\begin{aligned} &D(x_1, \ldots, x_k, x_{k+1}; x_{k+2} \ldots, x_d)^2 \\ &\quad = \min_y [|x_{k+1} - y|^2 + D(x_1, \ldots, x_k; y, x_{k+2} \ldots, x_d))^2]. \end{aligned} \tag{2}$$

Each iteration in k, therefore, reduces to constructing a lower envelop for a family of one-dimensional parabolas,

$$F_y(x_{k+1}) = |x_{k+1} - y|^2 + D(x_1, \ldots, x_k; y, x_{k+2} \ldots, x_d)^2. \tag{3}$$

To construct such an envelop here, we make use of the following lemma:

Lemma 1 *Let $\mathcal{G}(x)$ be a lower envelop of a finite family of one dimensional parabolas $\mathcal{F} = \{F_y(x) = |x - y|^2 + g(y)\}$ and let f be a parabola $f(x) = x^2 + b$. Then, there is at most a single interval (x^1, x^2) where $f(x) < \mathcal{G}(x)$.*

Intuitively, this statement follows from the simple observation that, if two parabolas with the same leading coefficient intersect, they can do so at most at a single point, and on either side of such an intersection one of the parabolas lies always above the other. Since $\mathcal{G}(x)$ is piece-wise composed from the parabolas in $\mathcal{F}$, once $\mathcal{G}(x)$ is crossed by $f(x)$ at some point x^1, that is, by crossing a corresponding parabola in $\mathcal{F}$, $f(x)$ will not be able to cross $\mathcal{G}(x)$ again for the entire interval $x > x^1$ or $x < x^1$ because $f(x)$ would lie entirely above the respective parabola there and, therefore, also the lower envelop $\mathcal{G}(x)$. Existence of at most two crossing points and uniqueness of the interval (x^1, x^2) then immediately follows. (For a formal proof of this lemma, see "Appendix").

Using the above lemma, we suggest the following procedure for constructing the lower envelop $\mathcal{G}(x)$. Let $\mathbf{G}$ be a subfamily of $\mathcal{F}$ and let $\mathcal{G}_\mathbf{G}(x)$ be the lower envelop for $\mathbf{G}$. Let $F_y(x)$ be some new parabola from $\mathcal{F}$, and let $\mathbf{G}' = \mathbf{G} \bigcup \{F_y(x)\}$. Observe that the lower envelop for $\mathbf{G}'(x)$ can differ from $\mathcal{G}_\mathbf{G}(x)$ on at most a single interval (x^1, x^2), by the above lemma, where $F_y(x) < \mathcal{G}_\mathbf{G}(x)$. To find such an interval, let x^0 be any point such that $F_y(x^0) < \mathcal{G}_\mathbf{G}(x^0)$. Then, the interval (x^1, x^2) can be efficiently constructed, and $\mathcal{G}_\mathbf{G}(x)$ respectively updated, by solving for the two intersection points $F_y(x^{1,2}) = \mathcal{G}_\mathbf{G}(x^{1,2})$ on the left and on the right of x^0, respectively. By repeating the above procedure for all parabolas in $\mathcal{F}$, $\mathcal{G}(x)$ can be constructed.

[1] To see this latter point, assume that an "on" point $\mathbf{x} = (x, y, z)$ in $\mathcal{I}$ is the closest to $\mathbf{x}^* = (x^*, y^*, z^*)$. Clearly, then, $\|\mathbf{x}^* - \mathbf{x}\|_2^2 = (z^* - z)^2 + \|(x^*, y^*) - (x, y)\|_2^2$, and it is only necessary to make sure that (x, y) is the point closest to (x^*, y^*) in $\mathcal{I}$ in the respective slice z, so that $\|(x^*, y^*) - (x, y)\|_2^2 \equiv D(x^*, y^*; z)^2$. Indeed, if this was not the case and there was another point (x', y') in $\mathcal{I}$ in slice z that was yet closer to (x^*, y^*), then (x', y', z) would also have to be closer to (x^*, y^*, z^*) in 3D, and we have a contradiction.

Algorithm 1 Line-scan algorithm for calculating discrete Euclidean distance transform.

Set $k = 0$.
while $k \leq d$ **do**
  Assume $D(x_1, \ldots, x_k; x_{k+1}, \ldots, x_d)$ had been calculated for all $\mathbf{x}$ (for $k = 1$, assume $D(\emptyset; x_1, \ldots, x_d) = 0$).
  for all x do
    Set $F_y(x) = |x - y|^2 + D(x_1, \ldots, x_k; y, x_{k+2} \ldots, x_d)^2$.
    Set $\mathbf{F} = \{F_y(x)$ for all $y\}$.
    Set $\mathbf{G} = \emptyset$, $\mathcal{G}(x) = +\inf$, $Y(x) = null$.
    while $\mathbf{G} \neq \mathbf{F}$ **do**
      Select a parabola $F_y \in \mathbf{F} \setminus \mathbf{G}$.
      Select an initial point x^0 such that $F_y(x^0) \leq \mathcal{G}(x^0)$ (Algorithm 2).
      Identify $x^{1,2}$ such that $F_y(x^{1,2}) = \mathcal{G}(x^{1,2})$ using initial point x^0.
      Set $\mathcal{G}(x) = F_y(x)$ and $Y(x) = y$ on (x^1, x^2).
      Set $\mathbf{G} = \mathbf{G} \bigcup \{F_y\}$.
    end while
    Set $D(x_1, \ldots, x_k, x_{k+1}; x_{k+2} \ldots, x_d)^2 = \mathcal{G}(x_{k+1})$.
  end for
  Set k=k+1.
end while

Algorithm 2 Select initial points for the line-scans in Algorithm 1.

For a parabola $F_y(x)$ and a current lower envelop $\mathcal{G}(x)$:
Set $n = 1$ and $x^1 = y$.
while $F_y(x^{(n)}) > \mathcal{G}(x^{(n)})$ and $x^{(n)}$ is inside the respective image domain **do**
  Solve $F_y(x) = F_{Y(x^{(n)})}(x)$.
  Set $x^{(n+1)}$ to the found solution.
  Set $n = n + 1$.
end while
If $x^{(n)}$ is outside of the image domain, then the initial point does not exist—$F_y(x) > \mathcal{G}(x)$ everywhere in the domain—otherwise $x^{(n)}$ is the initial point.

The starting point x_0 can be found, in general case, using Algorithm 2. Specifically, we maintain a parameter $Y(x)$ identifying at each point x the parabola from $\mathbf{G}$ that achieves the lower envelop at that point, that is, $\mathcal{G}(x) = F_{Y(x)}(x)$. The starting point, then, can found by solving the sequence of linear equation $F_y(x^k) = F_{Y(x^{k-1})}(x^k)$ starting at point $x^0 = y$ (Algorithm 2).

We further note that in the special case when the parabolas are added to the envelop in an orderly manner, such as in the order of increasing values of y, the Algorithm 2 is effectively trivialized. In this case observe that, in terms of Lemma 1, for a new parabola $f(x)$ and if all $F_{y'}(x)$ in $\mathbf{G}$ are such that $y' < 0$, then $f(x) < F_{y'}(x)$ can be only achieved on a right semi-interval (x_1, ∞). This can be seen immediately by explicitly writing out the above inequality as $x^2 + b < |x - y'|^2 + g(y') \Leftrightarrow b < -2xy' + y'^2 + g(y')$, $y' < 0$. Since the above is true for all $F_{y'}(x)$ in $\mathbf{G}$, it follows that $f(x) < \mathcal{G}_{\mathbf{G}}(x)$ is also achieved on some right semi-interval (x_*, ∞). Therefore, if a newly added parabola is below the lower envelop $\mathcal{G}(x)$ anywhere in the image domain, then it also has to be so at the rightmost point of that domain, and, thus, such rightmost point can be used as the starting point x_0 for the above scan. Organizing the scan in this manner effectively removes the need for x_0 search from the algorithm.

Note that in the above procedure we construct $\mathcal{G}(x)$ for all points on the x_{k+1}-axis simultaneously as the lower envelop over this entire axis, not one value of x_{k+1} at a time. Also, the equations $F_y(x) < \mathcal{G}_{\mathbf{G}}(x)$ can be uniformly formulated and solved for all $\mathbf{x}$ in a vector-based manner, which gives this algorithm significant advantage on vector processing architectures.

The described algorithm derives from the dimension-induction algorithm of [11] but differs from it in the type of the data structures used to represent the lower envelop during the scan and the way the lower envelop is thus updated. Specifically, in order to represent the lower envelop, [11] used a list of parabolas and intervals that comprised the lower envelop at every point of the scan. (Recall that the lower envelop at every step is a piece-wise composition of the parabolas in $\mathbf{G}$.) During the scan, for each newly added parabola, the algorithm performed a search in this list in order to find the parabolas in the current envelop that were above the new parabola and to reorganize the intervals accordingly. Such list-lookup process makes the algorithm essentially impossible to vectorize and forces implementations of this algorithm to perform updates in serial manner, one-by-one. (Although it should be noted that the updates still can be done in parallel in case of a multiple-processor system, they just cannot be represented in a vectorized form).

In the described algorithm, the lower envelop is represented directly by its values $\mathcal{G}_{\mathbf{G}}(x)$ at every step, and the update is performed vectorially based on the condition $F_y(x) < \mathcal{G}_{\mathbf{G}}(x)$. While algorithmically this modification is strictly speaking suboptimal: whereas the algorithm in [11] at every step only has to modify a single or a several elements in the list representation of $\mathcal{G}(x)$, here at each step, we may have to update a larger number of values in $\mathcal{G}(x)$—the possibility to perform this update in vector-based form and, in particular, to vectorize it over all points $\mathbf{x}$ results in dramatic improvement in the speed of this algorithm on vector processing architectures such as Matlab, as will be described below.

Like the other dimension-induction algorithms, the described algorithm is linear-time in the number of pixels in the image. To construct the three-dimensional distance transform $D(x_1, x_2, x_3)$ from the two-dimensional distance transforms $D(x_1, x_2; x_3)$ for an image with dimensions nmp, for example, our algorithm has to perform nm scans at every point (x_1, x_2) along x_3-axis, which involves p updates of the lower envelop for every parabola centered at a different pixel in the x_3-axis, thus, requiring $O(nmp)$ total time. Similarly, to construct the two-dimensional distance transform $D(x_1, x_2; x_3)$ from $D(x_1; x_2, x_3)$, our algorithm has to perform np scans at every point (x_1, x_3) along x_2-axis, which

themselves involve m lower envelop updates, again resulting in $O(nmp)$ time. Therefore, the algorithm constructs the EDT of a d-dimensional image with N pixels in $O(dN)$ time (see also Fig. 2).

The described line-scan algorithm allows a straightforward modification to allow calculating the EDT of the data with anisotropic voxel aspect ratios. Such data are routinely produced, for example, in confocal light microscopy, serial electron microscopy, computed tomography, fMRI, and many other applications. Currently, the calculation of EDT in such cases involves data re-sampling to create an intermediate isotropic dataset. This results in unnecessary computational overheads and can significantly degrade performance. In our algorithm, if the voxel's aspect ratio in k^{th} dimension is a_k, we simply modify Eq. (3) as,

$$F_y(x_{k+1}) = a_k^2|x_{k+1} - y|^2 + D(x_1, \ldots, x_k; y, x_{k+2} \ldots, x_d)^2.$$

3 Results

We implemented the above algorithm as a Matlab function "bwdistsc", available for download, including the source code, from the Matlab Central website under the name "3D Euclidean Distance Transform for Variable Data Aspect Ratio". Although a plain Matlab script, this function significantly outperforms the Matlab's standard distance transform function "bwdist" both in terms of the computation time and the largest possible data size, as well as many other EDT algorithms' implementations in Matlab.

We performed a series of numerical experiments in order to compare performance of bwdist and bwdistsc, as well as other state-of-the-art EDT algorithms. For these tests, we used a set of artificially created random 3D images, constructed as follows. Each image was defined on a cubic domain with the linear size ranging from 50 to approximately 350 pixels ($N \approx 10^5$ to $N \approx 4 \times 10^7$ total pixels). A set of 3D shapes such as spheres, cubes, tetrahedras, etc., were then placed at random locations in the images, each shape having linear dimensions of about 10 pixel. From 3 to 5 shapes were placed in an image on average. Figure 1 shows an example of one such random 3D configuration used in the tests. The distance transform was then calculated for each image using either bwdist or bwdistsc. A set of 25 random images was used to evaluate the average execution time of each algorithm on images of each size.

We also performed comparison of the described algorithm with Matlab implementations of the state-of-the-art EDT algorithms including wavefront propagation with Fast Marching Method (FMM) [14–16], Voronoi diagrams based EDT (VDEDT) [25–27], Meijster's algorithm (MEIJ) [11], and the algorithm of Felzenszwalb (FELDT) [35]. The last

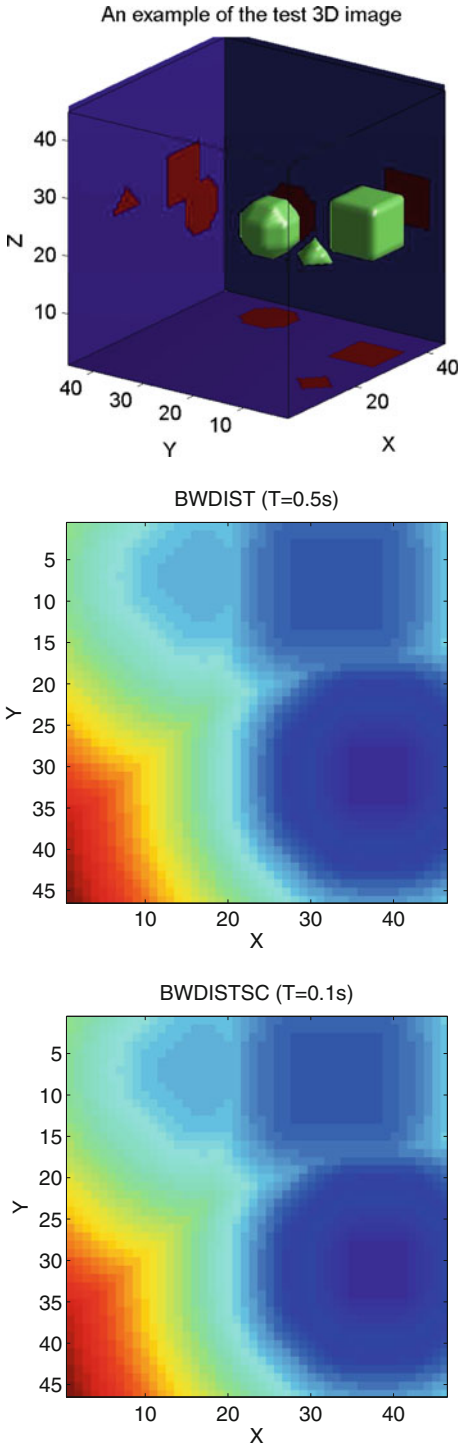

Fig. 1 An example of a 3D image used to evaluate performance of the Matlab's standard distance transform function "bwdist" and this paper's algorithm "bwdistsc". Two slices from the corresponding distance transforms produced by bwdist and bwdistsc and their respective execution times on an intel dual-core 2 GHz laptop are shown

two algorithms, in particular, are among the fastest EDT algorithms available today [12]. Existing or standard Matlab's packages were used to provide implementations for

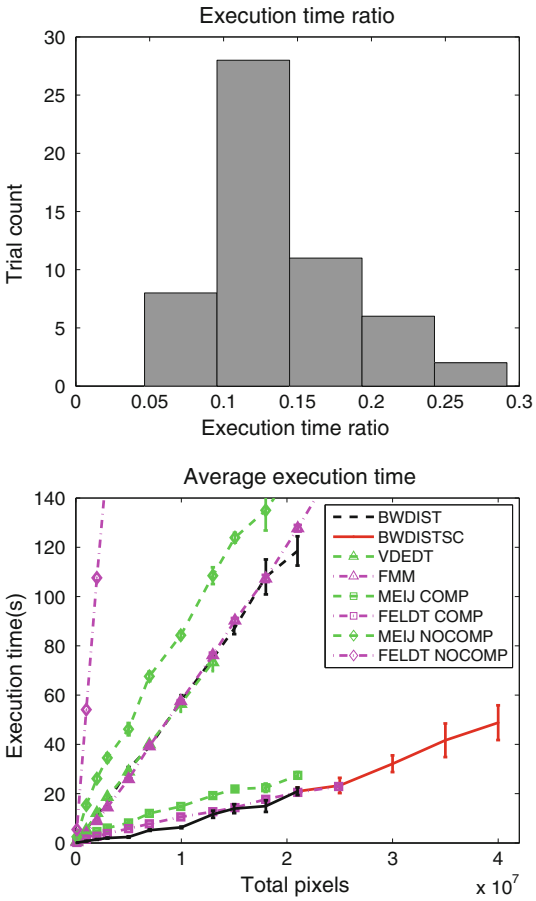

Fig. 2 The ratio of the execution times for bwdist and bwdistsc on a set of random 3D test images with sizes varying from from 10^5 to 4×10^7 pixels (*top*). Average execution times for different algorithms discussed in the paper, as a function of the total number of pixels in the image. All algorithms show linear-time performance but bwdistsc is on average five to seven times faster than bwdist and as fast or faster than compiled Matlab versions of Meijster and Felzenszwalb's algorithms. *Red line* indicates the region where bwdist produced "OUT OF MEMORY" error while bwdistsc was still able to obtain the result (*bottom*) (color figure online)

these algorithms, as appropriate. Specifically, for FMM, "Accurate Fast Marching" mex-library for the Fast Marching Method in Matlab was used, available from Matlab Central website [39]. Matlab's built-in functions for constructing and searching the Delaunay tessellations were used for the implementation of VDEDT. A very fast "DT"-implementation of FELDT algorithm, available from the Matlab Central [40], was used for this latter method, and a 3D-implementation of MEIJ was developed by ourselves following [11].

Our algorithm demonstrated superior performance to all of these methods, with on average fivefold improvement in speed relative to BWDIST, FMM, and VDEDT, and up to 2-fold improvement relative to MEJI. Compared to the fastest algorithms hereby tested, the "DT"-implementation of Felzenszwalb's algorithm [35], our algorithm demonstrated comparable or better performance (see Table 1; Fig. 2). For these latter two algorithms, it should be noted that the comparison results hereby stated are for the compilation-optimized MEIJ and FELDT implementations in Matlab. Without compilation optimization, these algorithms performed in Matlab at least 10-times slower that bwdistsc, due to their reliance on a large number of nonvectorizable list operations, as had been discussed earlier (MEIJ NOCOMP and FELDT NOCOMP in Fig. 2). All calculations were performed on an intel dual-core 2GHz laptop computer with 3GB physical RAM and Matlab version 7.7.

Table 1 Comparison of average execution times for bwdistsc and Matlab implementations of other state-of-the-art algorithms, on randomly generated $100 \times 100 \times 100$ images

Algorithm	Average time(s)	Comparison with "bwdistsc" (%)
BWDISTSC	1.14	100
FELDT	1.54	135.1
MEIJ	2.68	235.1
BWDIST	6.49	569.3
VDEDT	5.99	525.4
FMM	4.02	352.6

Our Matlab implementation of the discussed algorithm has also following two advantages. Taking advantage of the structure of the discussed algorithm, bwdistsc was implemented in Matlab using Matlab's cell arrays to store and manipulate its data. This made it possible for bwdistsc to process images of the size far exceeding that processable with bwdist. For example, the author used bwdistsc in his work to perform distance transform calculations in serial electron microscopy datasets over 1Gvoxel in size [36]—far beyond the size causing "out of memory" errors in bwdist. Among the algorithms hereby tested, only Fast Marching Method could calculate EDT for the datasets of comparable size (Fig. 2). Furthermore, bwdistsc allows calculating EDT for the data with anisotropic voxel aspect ratios, making it suitable for direct applications in confocal light microscopy, computed tomography, magnetic resonance imaging, serial electron microscopy, etc., which otherwise typically require data re-sampling leading to a significant performance loss.

These advantages: faster calculations, larger admissible data size, and possibility of anisotropic voxel aspect ratios—make the described Matlab function particularly interesting for high performance scientific and engineering applications that require calculating large-scale 3D Euclidean distance transforms in Matlab. For example, Fig. 3 illustrates a set of equidistant surfaces built around a dendritic structure in a 1Gvoxel serial electron microscopy reconstruction of a block of neural tissue in the brain, constructed using bwdistsc, and used to analyze relative distribution of different structures

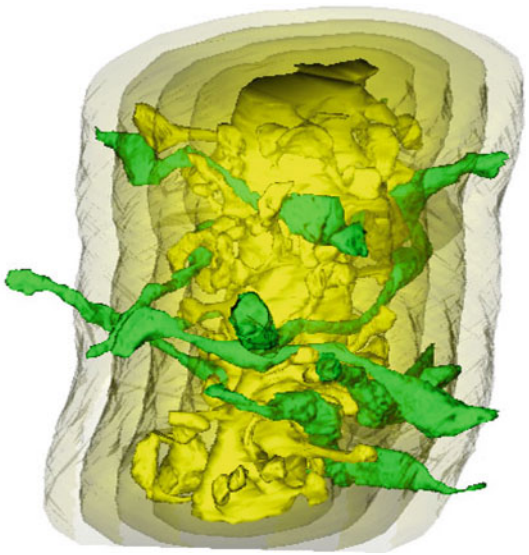

Fig. 3 A set of equidistant surfaces, constructed using bwdistsc, and several fragments of nearby axons for one dendritic process from a 1Gvoxel serial electron microscopy reconstruction of neural tissue in [36], used to study small scale organization of neural tissue in the brain

in the neural tissue in [36]. In (A. Scott et al., "Influence of voids on damage mechanisms in carbon/epoxy composites via a high resolution computed tomography", preprint) bwdistsc had been used for building tessellation maps in fiber-reinforced composite materials, used to study material micro-structures and damage mechanisms during fiber-reinforced material failures. In [37], our function had been used to calculate centered and shortest stream fluid paths through complex 3D porous structures; Fig. 4 illustrates such paths calculated in an instance of a complex pore space in [37]. Dima et al. [38] used bwdistsc in a variational segmentation approach for registration of the images of liver tissue in medical fMRIs, for malignant tumor diagnosis. Our Matlab function was also used in a number of other research projects that we do not have the possibility to mention here.

4 Summary

In this paper, we introduce a Matlab function for calculating Euclidean distance transform in large binary images of dimension three or higher. The function uses a fast line-scan algorithm that can be efficiently implemented on vector processing architectures. Implemented as a plain Matlab script, this function outperforms the Matlab's standard compiled distance transform function bwdist, as well as many other state-of-the-art algorithms, both in terms of the computation time and the largest possible dataset, and can also calculate the distance transforms with anisotropic voxel aspect ratios. These advantages make this function particularly advantageous for high-performance scientific and engineering applications requiring calculations of distance transforms in large and/or anisotropic datasets. To date, this function had found a number of applications in projects ranging from basic neuroscience [36] to porous material transport [37] and segmentation of medical data in Matlab [38]. The described algorithm can be also used in other high performance computing applications on platforms that leverage vectorized computing, such as High Performance Fortran or C/C++ MPI super-computing applications. The described function is available publicly for free download, including the source code, from the Matlab Central website under the name of "bwdistsc", "3D Euclidean Distance Transform for Variable Data Aspect Ratio".

5 Appendix: Proof of Lemma 1

Here, we present a formal proof of Lemma 1, namely that for a family of one-dimensional parabolas $\mathcal{F} = \{F_y(x) = |x-y|^2 + g(y)\}$ and an arbitrary parabola $f(x) = x^2 + b$ there can exist at most a single interval (x_1, x_2) such that $f(x) < \mathcal{G}(x)$, where $\mathcal{G}(x)$ is the lower envelop of the parabolas in the family $\mathcal{F}$.

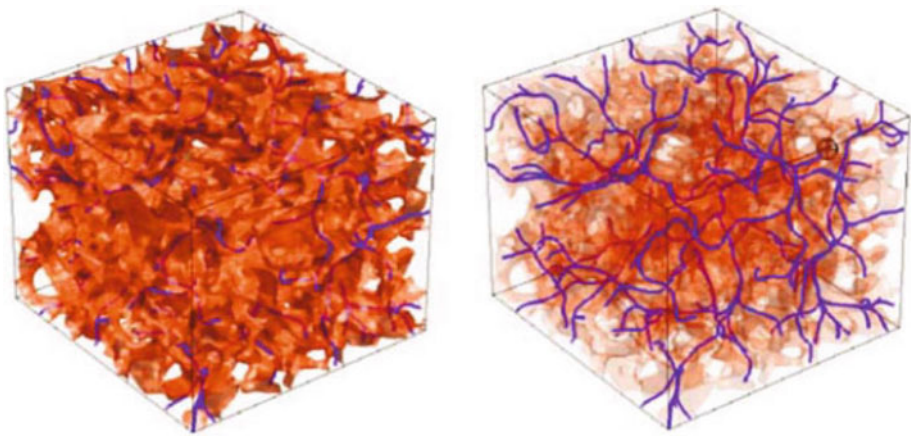

Fig. 4 A set of shortest stream fluid paths (*right*) through a pore space (*left*) constructed using the algorithm of [37] for centered and shortest stream-lines in pore space networks. Reproduced with the permission from G. Schena

We begin by noting that, in general, three possibilities can exist above: $f(x) > \mathcal{G}(x)$ everywhere on x; $f(x) < \mathcal{G}(x)$ everywhere on x; and $f(x) < \mathcal{G}(x)$ at some points and $f(x) > \mathcal{G}(x)$ at other points. The first two possibilities are obviously accommodated by assuming $(x_1, x_2) = \emptyset$ and $(x_1, x_2) = (-\infty, \infty)$, respectively.

For the third case, taking into account continuity of $\mathcal{G}(x)$ and $f(x)$, there should exist at least one crossing point such that $\mathcal{G}(x) = f(x)$. Let x_1 be such a crossing point, $\mathcal{G}(x_1) = f(x_1)$, and let $F_{y_1}(x)$ be the parabola in $\mathcal{F}$ such that $\mathcal{G}(x) = F_y(x)$ at $x = x_1$. We shall dismiss the trivial case $F_{y_1}(x) \equiv f(x)$, in which case $\mathcal{G}(x) = f(x)$ for the entire interval where $\mathcal{G}(x) = F_{y_1}(x)$ and, thus, x_1 cannot be a crossing point since $\mathcal{G}(x) > f(x)$ nowhere in that entire interval. For a real crossing point, therefore, $F_{y_1}(x) \not\equiv f(x)$ should hold. Then, by an obvious property of the intersection of the parabolas of the form given above, $f(x) > F_{y_1}(x)$ necessarily on a single semi-interval $(-\infty, x_1)$ or (x_1, ∞), and therefore $f(x) > F_{y_1}(x) \geq \mathcal{G}(x)$ on the same semi-interval. Without loss of generality we shall assume now that $f(x) > \mathcal{G}(x)$ on the semi-interval $(-\infty, x_1)$.

Let us now assume that x_2 is another crossing point $\mathcal{G}(x_2) = f(x_2)$, and $F_{y_2}(x)$ is the parabola in $\mathcal{F}$ such that $\mathcal{G}(x) = F_y(x)$ at $x = x_2$. In that case, by the same obvious property of the intersection of the parabolas in $\mathcal{F}$ and $f(x)$, $f(x) > \mathcal{G}(x)$ necessarily on a single semi-interval $(-\infty, x_2)$ or (x_2, ∞). Since we already have that $f(x) > \mathcal{G}(x)$ in $(-\infty, x_1)$ while $f(x_1) = \mathcal{G}(x_1)$, the only self-consistent possibility for x_2 to be a crossing point is $f(x) > \mathcal{G}(x)$ on (x_2, ∞) while $x_1 < x_2$.

Note that it is impossible to self-consistently add a third crossing point x_3, since that would imply that $f(x) > \mathcal{G}(x)$ on either $(-\infty, x_3)$ or (x_3, ∞), and that would necessarily violate either $f(x_1) = \mathcal{G}(x_1)$ or $f(x_2) = \mathcal{G}(x_2)$, or $f(x_3) = \mathcal{G}(x_3)$.

Therefore, there can be at most two crossing points for $f(x)$ and $\mathcal{G}(x)$, x_1 and x_2, and $f(x) < \mathcal{G}(x)$ can be achieved on at most a single interval (x_1, x_2), respectively.

References

1. Paglieroni, D.: Distance transforms : properties and machine vision applications, graphical models and image processing. CVGIP Graph. Model Image Process. **54**, 56–74 (1992)
2. Baggett, D., Nakaya, M., McAuliffe, M., et al.: Whole cell segmentation in solid tissue sections. Cytometry Part A **67**, 137–143 (2005)
3. Rosenfeld, A., Pfaltz, J.: Sequential operations in digital picture processing. J ACM **13**, 471–494 (1966)
4. Rousson, M., Paragios, N., Deriche, R.: Implicit active shape models for 3D segmentation in MR imaging. MICCAI **3216**, 209–216 (2004)
5. Town, C., Sinclair, D.: Content based image retrieval using semantic visual categories. Technical report (2001)
6. Gavrila, D.M.: Pedestrian detection from a moving vehicle. In: Vernon, D. (ed.) Computer Vission—ECCV 2000, pp. 37–49. Springer, Berlin (2000)
7. Breu, H., Gil, J., Kirkpatrick, D., Werman, M.: Linear time Euclidean distance transform algorithms. IEEE Trans. Pattern Anal. **17**, 529–533 (1995)
8. Maurer, C., Qi, R., Raghavan, V.: A linear time algorithm for computing exact Euclidean distance transforms of binary images in arbitrary dimensions. IEEE Trans. Pattern Anal. **25**, 265–270 (2003)
9. Tustison, N.J., Siqueira, M., Gee, J.C.: N-D linear time exact signed euclidean distance transform. Insight J. (2006) http://hdl.handle.net/1926/171
10. Guan, W., Ma, S.: A list-processing approach to compute Voronoi diagrams and the Euclidean distance transform. IEEE Trans. Pattern Anal. **20**, 757–761 (1998)
11. Meijster, A., Roerdink, J.B.T.M., Hesselink, W.H.: A general algorithm for computing distance transforms in linear time. Comput. Imaging Vis. **18**, 331–340 (2000)
12. Fabbri, R., da Fontoura Costa, L., Torelli, J.C., Bruno, O.M.: 2d Euclidean distance transform algorithms: a comparative survey. ACM Comput. Surv. **40** (2008)
13. Cuisenaire, O.: Distance transforms: fast algorithms and applications to medical imaging. Ph.D. Dissertation, Louvain-la-Neuve, Belgium (1999)
14. Verwer, B.J., Verbeek, P.W., Dekker, S.T.: An efficient uniform cost algorithm applied to distance transforms. IEEE Trans. Pattern Anal. **11**, 425–429 (1989)
15. Ragnemalm, I.: Neighborhoods for distance transformations using ordered propagation. CVGIP Image Underst. **56**, 399–409 (1992)
16. Eggers, H.: Two fast Euclidean distance transformations in z^2 based on sufficient propagation. Comput. Vis. Image Underst. **69**, 106–116 (1998)
17. Cuisenaire, O., Macq, B.M.: Fast Euclidean distance transformation by propagation using multiple neighborhoods. Comput. Vis. Umage Underst. **76**, 163–172 (1999)
18. Falcao, A.X., Stolfi, J., de Alencar, S.: The image foresting transform intelligence theory, algorithms, and applications. IEEE Trans. Pattern Anal. **26**, 19–29 (2004)
19. Porikli, F., Kocak, T.: Fast distance transform computation using dual scan line propagation. In: SPIE Conference Real-Time Image Processing, vol. 6496, February (2007)
20. Danielsson, P.-E.: Euclidean distance mapping. Comput. Vis. Graph. **14**, 227–248 (1980)
21. Borgefors, G.: Distance transformations in arbitrary dimensions. Comput. Vis. Graph. **27**, 321–345 (1984)
22. Svensson, S., Borgefors, G.: Distance transforms in 3d using four different weights. Pattern Recogn. Lett. **23**, 1407–1418 (2002)
23. Strand, R., Borgefors, G.: Distance transforms for three-dimensional grids with non-cubic voxels. Comput. Vis. Image Underst. **100**, 294–311 (2005)
24. Sethian, J.A.: A fast marching level set method for monotonically advancing fronts. Proc. Natl. Acad. Sci. **93**, 1591–1595 (1996)
25. Watson, D.F.: Computing the n-dimensional Delaunay tessellation with application to Voronoi polytopes. Comput. J. **24**, 167–172 (1981)
26. Aurenhammer, F.: Voronoi diagrams—a survey of a fundamental geometric data structure. ACM Comput. Surv. **22**, 345–405 (1991)
27. Fortune, S.: A sweepline algorithm for Voronoi diagrams. Algorithmica **2**, 153–174
28. Ogniewicz, R.L., Kubler, O.: Voronoi tessellation of points with integer coordinates time efficient implementation and online edge-list generation. Pattern Recogn. **28**, 1839–1844 (1995)
29. Gavrilova, M.L., Alsuwaiyel, M.H.: Two algorithms for computing the Euclidean distance transform, IJIG, pp. 635–645 (2001)

30. Paglieroni, D.W.: A unified distance transform algorithm and architecture. Mach. Vis. Appl **5**, 47–55 (1992)
31. Kolountzakis, M.N., Kutulakos, K.N.: Fast computation of the euclidean distance maps for binary images. Inform. Process. Lett. **43**, 181–184 (1992)
32. Chen, L., Chuang, H.Y.H.: A fast algorithm for Euclidean distance maps of a 2-d binary image. Inform. Process. Lett. **51**, 25–29 (1994)
33. Saito, T., Ichiro, Toriwaki J.: New algorithms for Euclidean distance transformation of an n-dimensional digitized picture with applications. Pattern Recogn. **27**, 1551–1565 (1994)
34. Hirata, T.: A unified linear-time algorithm for computing distance maps. Inform. Process. Lett. **58**, 129–133 (1996)
35. Felzenszwalb, P.F., Huttenlocher, D.P.: Distance transforms of sampled functions. Cornell Computing and Information Science. Technical report, September (2004)
36. Mishchenko, Y., Hu, T., Spacek, J., et al.: Ultrastructural analysis of hippocampal neuropil from the connectomics perspective. Neuron **67**, 1009–1020 (2010)
37. Schena, G., Favretto, S.: Pore space network characterization with sub-voxel definition. Transp. Porous Med **70**, 181–190 (2007)
38. Dima, T., Domingo, J., Dura, E.: A local level set method for liver segmentation in functional MR imaging. In: Proceedings of Nuclear Science Symposium and Medical Imaging Conference (NSS/MIC), pp. 3158–3161 (2011)
39. Kroon, D.-J.: Accurate Fast Marching, Toolbox, Matlab Central website, 23 June 2009
40. Kennedy, R.: Generalized distance transform, Toolbox, Matlab Central website, 26 May 2011