Pergamon

Pattern Recognition, Vol. 27, No. 11, pp. 1551–1565, 1994
Elsevier Science Ltd

0031-3203/94 $7.00 + .00

0031-3203(94)00057-3

NEW ALGORITHMS FOR EUCLIDEAN DISTANCE TRANSFORMATION OF AN n-DIMENSIONAL DIGITIZED PICTURE WITH APPLICATIONS

TOYOFUMI SAITO and JUN-ICHIRO TORIWAKI
Department of Information Engineering, Faculty of Engineering, Nagoya University, Furo-cho, Chikusa-ku, Nagoya-shi, 46401 Japan

(*Received* 1 *April* 1993; *in revised form* 25 *April* 1994; *received for publication* 7 *May* 1994)

Abstract—In this paper, we propose a new method to obtain the Euclidean distance transformation and the Voronoi diagram based on the exact Euclidean metric for an n-dimensional picture. We present four algorithms to perform the transformation which are constructed by the serial composition of n-dimensional filters. When performed by a general purpose computer, they are faster than the method by H. Yamada for a two-dimensional picture. Those algorithms require only one n-dimensional array for storing input/output pictures and a single one-dimensional array for a work area, if an input picture needs not be preserved.

Image processing Distance transformation Euclidean distance Three-dimension Voronoi diagram

1. INTRODUCTION

The distance transformation (DT) of binary pictures is one of basic tools of shape analysis in digital picture processing.[1] DT of a digitized binary picture was defined by Rosenfeld *et al.* using the 4- and the 8-neighbor distance functions.[2] They showed both the sequential and the parallel algorithms to calculate DT by computer. One important disadvantage of DT using the 4- and the 8-neighbor distance metrics is that the distance values are often far too different from the Euclidean distance values. The absolute difference between such distance value and the Euclidean distance may become arbitrarily large and cannot be bounded by any type of upper limit.[3]

A number of improved algorithms have been developed to make calculated distance values closer to the Euclidean distance values. Major approaches for improvement are classified into two categories. Algorithms classified into the first category use scalar values as the elements of templates or propagated information. Other algorithms use multivalued vectors.

All algorithms in the first category make use of templates or weight matrices in the local operations to propagate distance values. Those algorithms can be divided into three subcategories according to the type of templates. The first subcategory of algorithms use a single template to calculate DT. DTs of this subcategory are called the chamfer DT. The Chamfer 3-4 DT, the Chamfer 5-7-11 DT[4–8] and the quasi-Euclidean DT[9] which uses arbitrary real numbers approximating irrational numbers such as $\sqrt{2}$ and $\sqrt{3}$ as elements of a weight matrix are included in this subcategory. The second one employs two or more kinds of distance metrics alternatively of specific orders such as the variable neighborhood transformation as represented in detail in references (10–13). The octagonal distance transformation should be included in the first subcategory since it is implemented using a single 5×5 mask. However, it should also be included in the second one because it is executed by using the 4-neighbor operation and the 8-neighbor operation alternatively. The third subcategory recently proposed in reference (14) is characterized by the use of a grey-scale morphology operation. This method can be implemented by a kind of local parallel operation with a 3×3 neighborhood and changes the weight of the template every step of propagation executed.

The second category of algorithms utilizes a vector with multivalued elements to propagate distance values. Danielsson proposed the sequential Euclidean DT algorithm[15,16] of this category which generates the Euclidean distance map with no significant errors but not totally error-free according to the expression in Ragnemalm.[17,18] Yamada and Ragnemalm developed completely error-free algorithms.[17,19] Yamada's algorithm is of a parallel type and employs a 3×3 neighborhood operator, while Ragnemalm's uses contour scan.

The important feature of Yamada's algorithm is to use two two-dimensional arrays as work spaces which are of the same size as an input picture and store coordinate values of the 0-pixel that is closest to each 1-pixel. A fixed-neighborhood operator of 3×3 pixels is employed to propagate these coordinate values. A modification of this algorithm was published in reference (20) which decomposes the necessary operations to serial execution of two one-dimensional operators

in the row direction and the column direction. This leads to an efficient implementation of the algorithm by the hardware. Although the coordinate value arrays in the above method sometimes provide useful information, its execution by general purpose computer only to obtain DT is often time-consuming and requires too large a memory. This becomes a serious drawback in the case of processing three- or higher-dimensional pictures.

Both of the algorithms in references (14, 19) are of parallel type in the sense that the output value should be calculated using only the values of an input picture. Hence, two different two-dimensional arrays are required to store the input and the output picture separately. Both algorithms are of an iterative type. That is, they consist of local operations applied to the whole picture iteratively. The iteration finishes once no pixel value changes by the operation. Therefore the number of iterations strongly depends on an input picture and is not bounded by a significant upper limit beforehand. Mask parameters must be adjusted in the method of reference (14) in accordance with the number of iterations.

Extendability of algorithms to higher-dimensional pictures is also an important factor because pictures to be processed have extended recently from two-dimensional to three-dimensional pictures such as X-ray computed tomography (CT) images of human body. DT with the 4- and the 8-neighborhood distances is extended straightforwardly to a three-dimensional digitized picture only if a suitable distance metric is selected.

Three metrics called the 6-, the 18-, and the 26-neighbor distances are reported with the corresponding sequential algorithms of DT.[21-23] They also suffer from the same problem of sifnigicant bias from the Euclidean distance in the three-dimensional space.

Extension of the chamfer distance to the three or higher-dimensional picture is described in reference (4) with several weight matrices and their upper limits of error. Mullikin extended the Euclidean DT of Dannielson's type to the three-dimensional space and improved the accuracy of distance values.[24] Ragnemalm presented a 4-scan algorithm for three-dimensional pictures and the smallest possible number of scans algorithm for the arbitrary-dimensional pictures.[25] These algorithms are not the error-free EDT, however.

In this paper, we present new algorithms to calculate exactly the DT based upon the Euclidean metric (Euclidean DT) for an arbitrary binary picture of arbitrary dimensionality. The presented algorithms for n-dimensional (n-D) pictures consist of n one-dimensional local operations executed serially, each of which corresponds to the direction of each coordinate axis. We show in the paper two types of algorithms of DT, the basic one and the faster version. These algorithms are not classified into either of two categories described above. They do not use the vector propagation, nor the fixed template and still they always give exact Euclidean distance.

Features of the proposed algorithms are summarized as follows.

(1) They always give the exact Euclidean DT for an arbitrary binary picture.

(2) They are applicable to general n-dimensional binary pictures without any change.

(3) They are also applicable with slight modification to a digitized picture sampled with the different sampling interval in each coordinate axis. In the case of three-dimensional CT images of human body the interval between slices are usually larger than the pixel size in a slice. The algorithms proposed here can be applied to such types of pictures without any difficulty

(4) Memory requirements is minimum. Only one n-dimensional array of the same size as an input picture and a single one-dimensional array for work space are needed to execute the n-dimensional Euclidean DT. In the n-dimensional array an input picture is stored first and the resultant DT picture is stored after finishing the calculation. The size of the one-dimensional work space array is equal to the maximum length of the sides of an input picture. This point is critically important in practical applications treating three- or higher-dimensional pictures such as medical X-ray CT images.

(5) The algorithms are iterative. The number of times of global scan is $2n$ for processing an n-dimensional picture by the faster version of the algorithm. Local operations involved are always one-dimensional.

(6) Computation time is reasonably small. Roughly speaking the computation time of the proposed algorithms is proportional to both the average radius of a figure and the total number of voxels in an input picture. According to experimental results they are significantly faster than the direct implementation of the algorithm in reference (19). Sometimes its computation time is almost the same as that of the 8-neighbor DT (for two-dimensional pictures) and of the 26-neighbor DT (for three-dimensional pictures).

(7) They are suitable for execution by ordinary general purpose computer, although not necessarily the best for implementation by special purpose hardware with a local operation function of the fixed neighborhood.

In this paper, after describing the Euclidean DT algorithm, we present its application to calculation of the Voronoi division. The Voronoi division was first defined as a kind of division of the space into a set of cells based upon a given set of isolated points. It was later extended to a set of connected components on a digitized space and called the extended digital Voronoi division (EDVD).[26] The Voronoi division of the two-dimensional space is understood well by drawing the division result as a two-dimensional picture. Such pictorial representation is called the Voronoi diagram and the extended digital Voronoi diagram. The Voronoi diagram and the extended digital Voronoi diagram are closely related to the DT because the division of the space is performed by using distance values from an arbitrary point of the background to the closest point (or the closest connected component) in a given point

(component) set. The proposed algorithm of DT can also be utilized to derive the EDVD based upon the Euclidean distance.

Finally we present an example of applications to practical pictures. The algorithms were successfully applied to analysis of three-dimensional microscope images obtained from successive tissue sections of pathological samples.

2. NOTATIONS AND DEFINITIONS

In this paper we consider only digitized pictures sampled at rectangular ordered arrays.

Let us consider a three-dimensional digitized picture represented as $F = \{f_{ijk}\}$, where f_{ijk} is the value of the voxel (i, j, k) at the ith row and jth column of the kth plane (Fig. 1). For the sake of simplicity in explanation we consider processing of three-dimensional pictures devotedly in the paper. Note that results can be extended to pictures of arbitrary dimensionality immediately. We assume that a picture has L rows, M columns and N planes. A picture which is composed of only two kinds of voxels having the value 0 or 1 is called a binary picture, and each voxel is called a 0-voxel or a 1-voxel according to its value. A set of 0-voxels is called the background, and a set of 1-voxels is called a figure.

Let us denote a distance between two voxels (i, j, k) and (p, q, r) by $d_t(i, j, k), (p, q, r)$. The Euclidean metric is used unless explicitly declared otherwise.

Definition 1. Let $D = \{d_{ijk}\}$ and $S = \{s_{ijk}\}$ be the Euclidean distance transformation (EDT) of a binary picture $F = \{f_{ijk}\}$ and the squared EDT, respectively. Then the value d_{ijk} is defined as the minimum distance value from the voxel (i, j, k) to the closest 0-voxel in the input picture F, that is,

$$\begin{aligned} s_{ijk} &= \min_{(p,q,r)} \{d_t((i,j,k),(p,q,r))^2; \\ &\qquad f_{pqr} = 0, 1 \le p \le L, 1 \le q \le M, 1 \le r \le N\} \\ &= \min_{(p,q,r)} \{(i-p)^2 + (j-q)^2 + (k-r)^2; \\ &\qquad f_{pqr} = 0, 1 \le p \le L, 1 \le q \le M, 1 \le r \le N\}, \quad (1) \\ d_{ijk} &= \min_{(p,q,r)} \{d_t((i,j,k),(p,q,r)); \\ &\qquad f_{pqr} = 0, 1 \le p \le L, 1 \le q \le M, 1 \le r \le N\}. \\ &= \sqrt{s_{ijk}}. \end{aligned}$$

Note that the distance value s_{ijk} at any of 0-voxels (background voxels) is 0 because the closest 0-voxel is itself.

We use the term distance transformation (DT) to represent both the transformation to calculate the picture D from the input picture F and the distance picture D itself.

Metrics different from the Euclidean metric may be employed in equation (1) in the above definition. Several examples widely used in digital picture processing are shown in Table 1. Other examples are presented in references (12, 22, 23, 26, 27). Each of the DTs are often called by the names which represent metrics used there such as the 8-neighbor DT and the 26-neighbor DT.

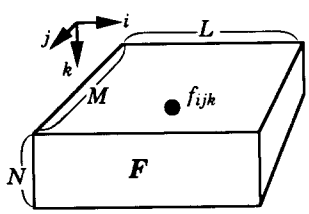


Fig. 1. Three-dimensional digitized picture $F = \{f_{ijk}\}$.

3. ALGORITHM OF THREE-DIMENSIONAL EUCLIDEAN DISTANCE TRANSFORMATION

3.1. *Basic algorithm*

Before proceeding to the detailed description of the EDT algorithm, we will explain briefly the basic idea of the proposed method. The point is summarized in the following two items.

(1) To minimize the square of the Euclidean distance instead of the exact distance in the process of transformations.

(2) To implement the transformation by decomposing the procedure into serial execution of the three one-dimensional transformation.

Table 1. Examples of distance metrics for a digitized picture

	$d((i,j),(p,q))$ or $d((i,j,k),(p,q,r))$	
	Name	Definition
Two-dimensional	4-neighbor distance	$\lvert i-p\rvert + \lvert j-q\rvert$
	8-neighbor distance	$\max\{\lvert i-p\rvert, \lvert j-q\rvert\}$
Three-dimensional	6-neighbor distance	$\lvert i-p\rvert + \lvert j-q\rvert + \lvert k-r\rvert$
	18-neighbor distance	$\max\{\max(\lvert i-p\rvert, \lvert j-q\rvert, \lvert k-r\rvert), \operatorname{int}((\lvert i-p\rvert + \lvert j-q\rvert + \lvert k-r\rvert + 1)/2)\}$
	26-neighbor distance	$\max\{\lvert i-p\rvert, \lvert j-q\rvert, \lvert k-r\rvert\}$

The following Algorithm 1 shows the essential part of the proposed algorithms in the form of picture to picture transformations instead of strict description of the practically implemented algorithm.

Algorithm 1. EDT—expression as a parallel operation. Input picture: $F = \{f_{ijk}\}(1 \le i \le L,\ 1 \le j \le M,\ 1 \le k \le N)$.

Transformation 1. Derive from F a picture $G = \{g_{ijk}\}$ defined as follows—(transformation in the i-axis direction) (Fig. 2)

$$g_{ijk} = \min_x \{(i-x)^2;\ f_{xjk} = 0, 1 \le x \le L\}. \quad (2)$$

Transformation 2. Derive from the above picture G a picture $H = \{h_{ijk}\}$ given by the following equation—(*transformation in the* j-axis direction) (Fig. 3)

$$h_{ijk} = \min_y \{g_{iyk} + (j-y)^2;\ 1 \le y \le M\}. \quad (3)$$

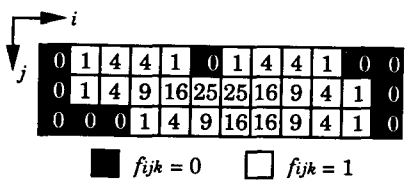


Fig. 2. Example of Transformation 1 of Algorithm 1.

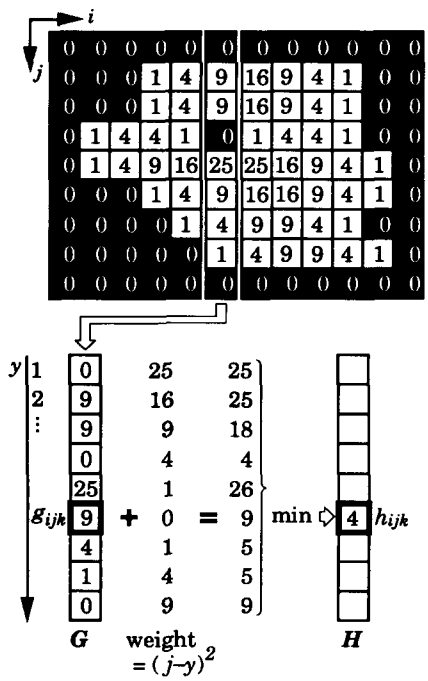


Fig. 3. Example of Transformation 2 of Algorithm 1. To calculate the distance value h_{ijk} of a certain pixel (i, j, k), (1) consider a column including g_{ijk} corresponding to the pixel (i, j, k), (2) add the weight $(j-y)^2$ to each value in the column, and (3) search the minimum value in the results of additions. Then we get the value of h_{ijk} as the minimum value found in the above process.

Transformation 3. Obtain from the above picture H a picture $S = \{s_{ijk}\}$ defined by the following equation—(transformation in the k-axis direction)

$$s_{ijk} = \min_z \{h_{ijz} + (k-z)^2;\ 1 \le z \le N\}. \quad (4)$$

Then, the following property is proved.

Property 1. The picture $S = \{s_{ijk}\}$ is the squared EDT of a picture $F = \{f_{ijk}\}$. That is, a voxel (i, j, k) in the picture $S = \{s_{ijk}\}$ has a value equal to the square of the Euclidean distance from the voxel (i, j, k) to the closest 0-voxel.

Proof. From equation (2),

$$g_{ijk} = \min_x \{(i-x)^2;\ f_{xjk} = 0, 1 \le x \le L\} \quad (5)$$
$$= \text{the squared distance to the closest 0-voxel in the same row as } (i, j, k).$$

By substituting equation (5) with equation (3), we obtain

$$h_{ijk} = \min_y \left\{ \min_x \{(i-x)^2;\ f_{xyk} = 0, 1 \le x \le L\} + (j-y)^2;\ 1 \le y \le M \right\}$$
$$= \min_y \left\{ \min_x \{(i-x)^2 + (j-y)^2;\ f_{xyk} = 0, 1 \le x \le L\};\ 1 \le y \le M \right\}$$
$$= \min_{(x,y)} \{(i-x)^2 + (j-y)^2;\ f_{xyk} = 0, 1 \le x \le L, 1 \le y \le M\} \quad (6)$$
$$= \text{the squared distance to the closest 0-voxel in the same plane as } (i, j, k).$$

By substituting the result to equation (4),

$$s_{ijk} = \min_z \left\{ \min_{(x,y)} \{(i-x)^2 + (j-y)^2;\ f_{xyz} = 0, 1 \le x \le L, 1 \le y \le M\} + (k-z)^2;\ 1 \le z \le N \right\}$$
$$= \min_z \left\{ \min_{(x,y)} \{(i-x)^2 + (j-y)^2 + (k-z)^2;\ f_{xyz} = 0, 1 \le x \le L, 1 \le y \le M\};\ 1 \le z \le N \right\}$$
$$= \min_{(x,y,z)} \left\{ (i-x)^2 + (j-y)^2 + (k-z)^2;\ f_{xyz} = 0, 1 \le x \le L, 1 \le y \le M, 1 \le z \le N \right\}. \quad (7)$$

Thus it is shown that the picture S is the squared EDT of the picture F. [Q.E.D.]

3.2. *Extension to the case of cuboid voxel*

Algorithm 1 is extended to the form applicable to a picture digitized with sampling intervals different in

three axes each other by slightly modifying Transformations 2 and 3 as shown in the following Transformations 2′ and 3′. Let us assume that the ratio among sampling intervals in three axes be $1:\alpha:\beta$. Other notation is as in Section 3.1.

Algorithm 2. EDT for cuboid voxel pictures—parallel operation expression.

Transformation 2′. Replace equation (3) of the Transformation 2 by the following equation (3′). Note that the weight α is multiplied to the second term of the equation inside the braces

$$h_{ijk} = \min_{y} \{g_{iyk} + (\alpha(j-y))^2;\ 1 \le y \le M\}. \quad (3')$$

Transformation 3′. Use equation (4′) below instead of equation (4) in Transformation 3

$$s_{ijk} = \min_{z} \{h_{ijz} + (\beta(k-z))^2;\ 1 \le z \le N\}. \quad (4')$$

It is proved in the same way as Property 1 that the picture S is the squared EDT of a picture F.

3.3. *Algorithms on the digitized picture space*

The above algorithms represent the proposed EDT algorithms in the form of picture to picture transformations. Although they suggest the outline of possible algorithms, they do not provide fixed algorithms nor a way of implementation by practical computers. Here we show that algorithms presented above are implemented in a remarkably efficient way by ordinary general purpose computer (or a serial machine with a single processor) by considering that the distance between adjacent voxels is of unit length and by employing operations of the sequential type. Concrete descriptions of algorithms are given below (we assume that $\alpha = \beta = 1$ for simplicity of description).

Algorithm 3. EDT (basic type).

Step 1 (*Transformation* 1).

(1.1) Let an input binary picture be $F = \{f_{ijk}\}$, and the output picture be $G' = \{g'_{ijk}\}$ which is initialized as $g'_{ijk} = L(\forall_j, \forall_k)$ and $g'_{ijk} = 0$, otherwise. Perform for each values of j and k

$$\begin{cases} g'_{ijk} \leftarrow (\sqrt{g'_{(i-1)jk}} + 1)^2, & \text{if } f_{ijk} \neq 0, \\ g'_{ijk} \leftarrow 0, & \text{if } f_{ijk} = 0, \end{cases} \quad i = 2, 3, \ldots, L. \quad (8)$$

The suffix i should be changed from 2 to L [left to right in each row of each picture plane (= horizontal cross section of a three-dimensional picture)] sequentially (forward scan).

(1.2) Let an input picture be $G' = \{g'_{ijk}\}$ [= the output of Step (1.1)], and the output picture be $G = \{g_{ijk}\}$ which is initialized as $g_{Ljk} = L$ $(\forall_j, \forall_k)$ and $g_{ijk} = 0$, otherwise. Perform for each values of j and k

$$g_{ijk} \leftarrow \min\{(\sqrt{g_{(i+1)jk}} + 1)^2, g'_{ijk}\}. \quad i = L-1, L-2, \ldots, 1. \quad (9)$$

The suffix i should be changed from $L-1$ to 1 (right to left in each row of each picture plane) sequentially (backward scan).

Step 2 (*Transformation* 2). Let the input picture be $G = \{g_{ijk}\}$ [= the output of Step (1–2)], and the output picture be $H = \{h_{ijk}\}$.

Execute the following procedure at each voxel.

$$h_{ijk} \leftarrow \min_{-r \le n \le r} \{g_{i(j+n)k} + n^2\}, \text{ where } r = \sqrt{g_{ijk}}. \quad (10)$$

Here, the minimum value of the right-hand side is calculated by searching all values of the term inside the braces for n such that $(-r \le n \le r)$ and $(1 \le j+n \le M)$ (Fig. 4). Only the second suffix j is changed for each value of i and k.

Step 3 (*Transformation* 3). Let the input picture be $H = \{H_{ijk}\}$ (= the output of Step 2), and the output picture be $S = \{s_{ijk}\}$.

Execute the following procedure at each voxel.

$$s_{ijk} \leftarrow \min_{-r \le n \le r} \{h_{ij(k+n)} + n^2\}, \text{ where } r = \sqrt{h_{ijk}}. \quad (11)$$

The method to search the minimum value is the same as in Step 2. Only the third suffix k is changed for each value of i and j.

It is important from the viewpoint of computational cost that the search for minimization is always one-dimensional in either of the i-, j-, and k-axis directions. Bounds of the searching interval $\pm r$ [equations (10) and (11)] are obtained by denoting that at least one 0-voxel exists at the distance $\sqrt{g_{ijk}}$ and $\sqrt{h_{ijk}}$ from the current voxel (i, j, k), respectively. Therefore we need not use exact values of $\sqrt{g_{ijk}}$ if we do not like to treat non-integer values. The square root calculation in equations (8) and (9) is also excluded by considering the square of the amounts appeared in those equations. The programs we used in practice are given in the Appendix.

3.4. *Fast algorithm*

Computation time of the algorithms will be discussed in Section 4 in detail. Here we present another implementation of the above transformations by which

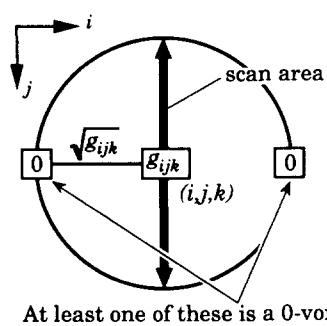


Fig. 4. Illustration of the scan area in Transformation 2 of Algorithm 3.

the computation time may be significantly reduced for some types of input pictures. Major improvements are achieved by limiting search areas for minimization in Steps 2 and 3 and *avoiding calculations of square-root operations.*

The notations are same as in Algorithm 3.

Algorithm 4. EDT (fast type).

Step 1 (*Transformation* 1). Same as Step 1 of Algorithm 3.

Step 2 (*Transformation* 2) (Fig. 5).

(2.1) Input picture: $G = \{g_{ijk}\}$, output picture: $H' = \{h'_{ijk}\}$. Perform the following procedure at each column (for each values of i and k) (Fig. 6).

The value of the suffix j is increased one by one from 2 to M (from top to bottom of each plane (= vertical cross-section) of the picture) (forward scan).

(a) If g_{ijk} is larger than $(g_{i(j-1)k} + 1)$,
then do the following procedure for the values of n such that $0 \le n \le (g_{ijk} - g_{i(j-1)k} - 1)/2$ (if $(j + (g_{ijk} - g_{i(j-1)k} - 1)/2)$ is greater than M, then $0 \le n \le M - j$).
(Note here that $n = (g_{ijk} - g_{i(j-1)k} - 1)/2$ is the intersection of the curves $f_1(n) = g_{ijk} + n^2$ and $f_2(n) = g_{i(j-1)k} + (n+1)^2$ in the region $n > j$.)
(i) if $g_{i(j-1)k} + (n+1)^2$ is larger than or equal to $g_{i(j+n)k}$,
then go to the next j,
(ii) else
substitute $g_{i(j-1)k} + (n+1)^2$ to $h'_{i(j+n)k}$.

(b) else
substitute g_{ijk} to h'_{ijk}.

(2.2) Input picture: $H' = \{h'_{ijk}\}$ [= the output of Step (2.1)].

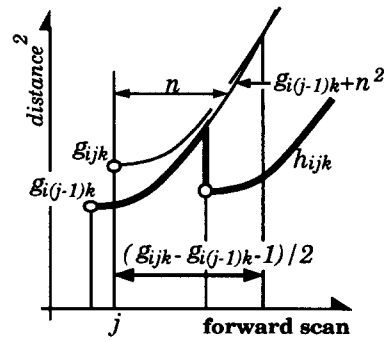


Fig. 6. Illustration of forward scan at Algorithm 4, Step 2.

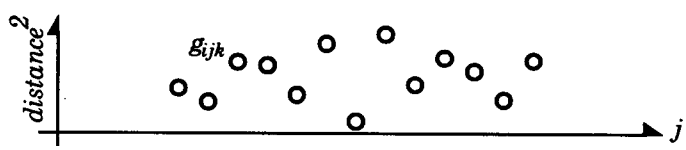
Output Picture: $H = \{h_{ijk}\}$.

Perform the following procedure at each column (for each value of i and k).

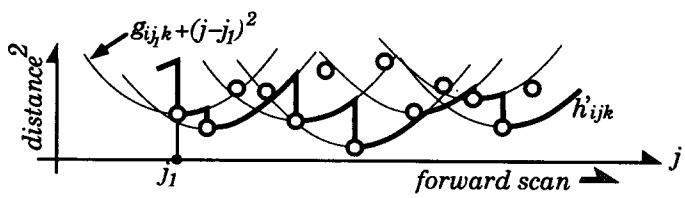
The value of the suffix j is decreased one by one from $M - 1$ to 1 [from bottom to top of each plane (= vertical cross-section) of the picture] (backward scan).

(a) If h'_{ijk} is larger than $(h'_{i(j+1)k} + 1)$,
then execute the following procedure for the values of n such that $0 \le n \le (h'_{ijk} - h'_{i(j+1)k} - 1)/2$ (if $(j - (h'_{ijk} - h'_{i(j+1)k} - 1)/2$ is less than 1, then $0 \le n \le j - 1$).
(i) if $h'_{i(j+1)k} + (n+1)^2$ is larger than or equal to $h'_{i(j-n)k}$
then go to the next j,
(ii) else
substitute $h'_{i(j+1)k} + (n+1)^2$ to $h_{i(j-n)k}$.

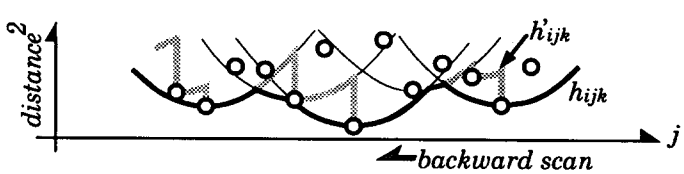
(b) else
substitute h'_{ijk} to h_{ijk}.



(a) A picture after *Step 1* is performed.



(b) Illustration of the forward scan.



(c) Illustration of the backward scan.

Fig. 5. Illustration of Step 2 in Algorithm 4 (see also Fig. 4).

Step 3 (*Transformation* 3). Same as Step 2 above except that G and H are replaced by H and S, respectively, and the third suffix k is changed instead of the second suffix j.

An example of the program we used is given in the Appendix.

Concerning memory requirement, two three-dimensional arrays of the same size are used to store an input picture F and a resultant squared DT picture, respectively. However, if the input picture F is not needed to be preserved as a binary picture after the execution of the DT, all pictures F, G', G, H', H, S' and S in the above algorithm can be assigned to physically the same address in computer memory (an input picture is still restored easily by replacing all positive values by 1 in the obtained squared DT picture). Additionally the single one-dimensional array with the size $\max(M,N)$ is required as a work area. Therefore the minimum memory requirement for performing the above EDT is a single three-dimensional array and a single one-dimensional array for the case of a three-dimensional picture.

3.5. *Extension to an* n*-dimensional picture*

Extension of the above algorithms to pictures of lower- and higher-dimensions than three-dimension is straightforward. The two-dimensional case is obtained by excluding Step 3 of the algorithms. In the case of n-dimensional pictures for $n > 3$, the same type of the procedure as Step 3 should be iterated $n-1$ times (including Steps 2 and 3 of the presented algorithms). Estimation of the amount of computation in Section 5 and memory requirement presented in the previous section are also valid for a picture of arbitrary dimensionality. It should be noted that only one n-dimensional array and a single one-dimensional array are enough to perform the n-dimensional EDT.

One disadvantage of these algorithms is the use of square root operation in the post processing to obtain the exact EDT values from the squared EDT obtained by the algorithm. However, this is not so serious as far as a general purpose computer is used as we see in the experimental results in Section 5.

4. DIGITAL VORONOI DIAGRAM

4.1. *Definitions*

Given a digitized binary picture $F = \{f_{ijk}\}$ including more than two figures (connected components), the modified digital Voronoi division (MDVD) of a picture F is defined as follows.[6,26]

Definition 2. Assuming that a binary picture F contains n connected components $C_1, C_2, \ldots, C_n$, a set of voxels T_r defined below is called a tile of the connected component C_r.

$$T_r = \{(i,j,k);\ d((i,j,k),C_r) < d((i,j,k),C_k), \forall_{k \neq r}\}, \tag{12}$$

where

$$d((i,j,k),C_r) = \text{the distance between a voxel } (i,j,k) \text{ and a connected component } C_r = \min\{d((i,j,k),(p,q,r)); (p,q,r) \in C_r\}. \tag{13}$$

A set of all tiles $[T_r; r = 1,2,3,\ldots,n]$ and the division of the space into such tiles are both called the modified digital Voronoi division (MDVD) of a picture F.

4.2. *Algorithm to obtain the MDVD*

Although any distance metric can be adopted for equation (13), in principle, only the 4- and the 8-neighbor distances were used in the case of digital binary pictures in reference (26). Very few reports have been published concerning practical applications of MDVD in the three-dimensional space.[28] Use of the Euclidean distance was time-consuming for a set of connected components in the higher-dimensional space, although a divide-and-conquer algorithm[29] works well for a set of isolated points.

By modifying the algorithms presented in Section 3, the MDVD employing the Euclidean distance can be obtained effectively. The basic idea is to propagate labels assigned to connected components beforehand synchronously with the process of calculating the EDT. An algorithm is derived from Algorithm 3 as follows. We simplify the description by referring to Algorithm 3.

Algorithm 5. Modified digital Voronoi diagram.

Input picture (label picture): $F = \{f_{ijk}\}$ $(1 \le i \le L,\ 1 \le j \le M,\ 1 \le k \le N)$. We assign labels such that $f_{ijk} = 0$ if (i,j,k) is a background voxel, and otherwise, f_{ijk} = positive integer showing a label of a connected component which the voxel (i,j,k) belongs to.

Step 1 (*Transformation* 1). The order to visit voxels (scanning mode) is the same as in Algorithm 3.

(1.1) Let an input picture be $F = \{f_{ijk}\}$ (label picture), and the output pictures $G' = \{g'_{ijk}\}$ and $V' = \{v'_{ijk}\}$. Perform for each values of j and k

$$\begin{cases} g'_{ijk} \leftarrow (\sqrt{g'_{(i-1)jk}} + 1)^2 \\ v'_{ijk} \leftarrow v'_{(i-1(jk} \end{cases} \quad \text{if } f_{ijk} = 0, \tag{14}$$

$$\begin{cases} g'_{ijk} \leftarrow 0 \\ v'_{ijk} \leftarrow 0 \end{cases} \quad \text{if } f_{ijk} > 0. \tag{15}$$

(1.2) Let input pictures be $G' = \{g'_{ijk}\}$ and $V' = \{v'_{ijk}\}$ [= outputs of the Step (1.1)], and the output pictures be $G = \{g_{ijk}\}$ and $V = \{v_{ijk}\}$. Perform for each values of j and k.

$$\begin{cases} g_{ijk} \leftarrow (\sqrt{g_{(i+1)jk}} + 1)^2 \\ v_{ijk} \leftarrow v_{(i+1)jk} \end{cases} \quad \text{if } (\sqrt{g_{(i+1)jk}} + 1)^2 < g'_{ijk}, \tag{16}$$

$$\begin{cases} g_{ijk} \leftarrow g'_{ijk} \\ v_{ijk} \leftarrow v'_{ijk} \end{cases} \quad \text{if } (\sqrt{g_{(i+1)jk}} + 1)^2 \ge g'_{ijk}. \tag{17}$$

Step 2 (*Transformation* 2). Perform the following procedure at each background voxel (i,j,k).

Let input pictures be $G = \{g_{ijk}\}$ and $V = \{v_{ijk}\}$ [= outputs of Step (1.2)], and output pictures be $H = \{h_{ijk}\}$ and $V' = \{v'_{ijk}\}$.
Assuming that $\{g_{i(j+n)k} + n^2;\ -r \le n \le r$, where $r = \sqrt{g_{ijk}}\}$ is minimized with respect to n at $n = n^*$,

$$\begin{cases} h_{ijk} \leftarrow g_{i(j+n^*)k} + (n^*)^2 \\ v'_{ijk} \leftarrow v_{i(j+n^*)k}. \end{cases} \tag{18}$$

Step 3 (*Transformation* 3). Perform the following procedure at each background voxel (i, j, k).
Let input pictures be $H = \{h_{ijk}\}$ and $V' = \{v'_{ijk}\}$ (= outputs of Step 2), and output pictures be $S = \{s_{ijk}\}$ and $V = \{v_{ijk}\}$.
Assuming that $\{h_{ij(k+n)} + n^2);\ -r \le n \le r$, where $r = \sqrt{h_{ijk}}\}$ is minimized with respect to n at $n = n^*$,

$$\begin{cases} s_{ijk} \leftarrow h_{ij(k+n^*)} + (n^*)^2 \\ v_{ijk} \leftarrow v'_{ij(k+n^*)} \end{cases} \tag{19}$$

After finishing the procedure, the squared EDT is stored in S and the MDVD is obtained in the picture $V = \{v_{ijk}\}$. That is, a value v_{ijk} at each voxel (i, j, k) in V gives the label of the connected component closest to that voxel. Memory requirement is also the same as in Algorithm 3. If the input label picture F needs not be preserved after the procedure is terminated, only two three-dimensional arrays are necessary, one for the pictures F, V, and V', and the other for G', G, H and S. Additionally, two one-dimensional arrays are required as work areas for keeping $\{g_{ijk};\ 1 \le j \le M\}$ and $\{v_{ijk};\ 1 \le j \le M\}$, respectively, in Step 2. They are also used in Step 3.

Fast version of the EDT, Algorithm 4 can be employed in the similar way, but details are omitted here.

5. COST OF COMPUTATION

5.1. *Theoretical estimation of the amount of computation*

The computation time of the EDT by the algorithms presented above depends on the size of an input picture, the shape and the size of a figure in an input picture, and the implementation method. Here we derive estimates of the amount of computation, assuming that the transformation is implemented on a general purpose computer (serial machine) with single processor.

(1) Algorithm 3. Step 1 of Algorithm 3 is performed by scanning the whole of an input picture twice, by the forward raster scan followed by the backward raster scan once for each. In Steps 2 and 3 we need not scan any of the whole column and the whole vertical column. Instead, the number of voxels to be searched at each current voxel (i, j, k) is equal to twice of the square root of g_{ijk} or h_{ijk} in equation (1). Thus, computation time (or the amount of computation) of each transformation is estimated as follows.

Step 1: O(Num),
Step 2: $O(Av(\sqrt{g_{ijk}}) \times Num)$,
Step 3: $O(Av(\sqrt{h_{ijk}}) \times Num)$,

where Num is the number of voxels in the input picture and Av(x) represents the average of the value x over the corresponding picture.

Hence, by approximating the total amount of computation by the sum of the above estimation for each step, and approximating $Av(\sqrt{g_{ijk}})$ and $Av(\sqrt{h_{ijk}})$ by $Av(\alpha \times \sqrt{s_{ijk}})$ or the mean of distance values multiplied by a suitable constant α, we obtain

Total amount of computation:
O(mean value of DT × Num) = O(Av(d_{ijk}) × Num).

Thus we find that the computation time of Algorithm 3 is approximately proportional to both the mean value of the DT and the picture size.

(2) Algorithm 4. The computation time of Algorithm 4 is more sensitive to the shapes of figures in an input picture F than Algorithm 3. We consider here the case that the input picture includes only one solid sphere. Let us assume that the computation time of Step 2 is approximated by twice of the amount of the procedure in the forward scan, and that of Step 3 is also the same order as Step 2.

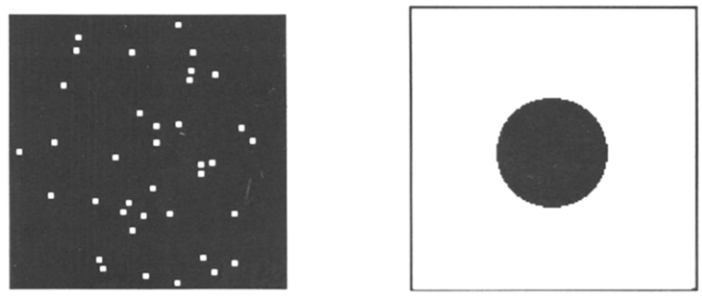
At an each voxel (i, j, k), if g_{ijk} is greater than $(g_{i(j-1)k} + 1)$ in (2.1)(a) of Step 2, $(g_{ijk} - g_{i(j-1)k} - 1)/2$ times of operations will be executed in the following iteration in (a). Otherwise, substitution will be executed only once in (b) at the voxel (i, j, k). Then, for a sphere with the radius R, computation time of the forward scan in Step 2 is approximately proportional to $(\alpha R^3 + \text{Num})$ where α is a suitable constant. Total computation time is estimated as four times of it.

5.2. *Experimental evaluation*

Because there are no other reports of concrete algorithms to calculate the EDT for three-dimensional pictures, we compared computation time of several algorithms developed for transforming two-dimensional pictures. Algorithms studied in the experiments are Algorithms 3 and 4 in this paper, the EDT of the method in reference (19) (implemented as the FORTRAN subroutine DTEU in SPIDER-II™), and the 8-neighbor DT. Algorithms 3 and 4 in the text were modified for a two-dimensional picture (Step 3 was omitted). Further, we compared our algorithms with the 26-neighbor DT for three-dimensional pictures.[21,22]

Two kinds of pictures were prepared for being processed by the algorithms above. One of them was generated by scattering a certain number of 0-pixels of which locations were selected by using uniform random number generator on the background filled with 1-pixels. The other consists of a circle or a sphere with the radius R locating at the center of the input picture (Fig. 7). Picture sizes are 500 × 500 pixels (two-dimensional pictures) and 120 × 120 × 120 voxels (three-dimensional pictures). All programs were written in FORTRAN and executed by the general purpose computer FACOM M-1800 in Nagoya University Computation Center.

Figure 8(a) and (b) show that computation times of Algorithm 3 and DTEU are approximately proportional to the square root of the mean value of S (= mean



(a) (b)

Fig. 7. Samples of input pictures for evaluation of computation time (white: 0-pixel, black: 1-pixel). (a) A sample of a random dots picture. (b) A sample of a circle picture.

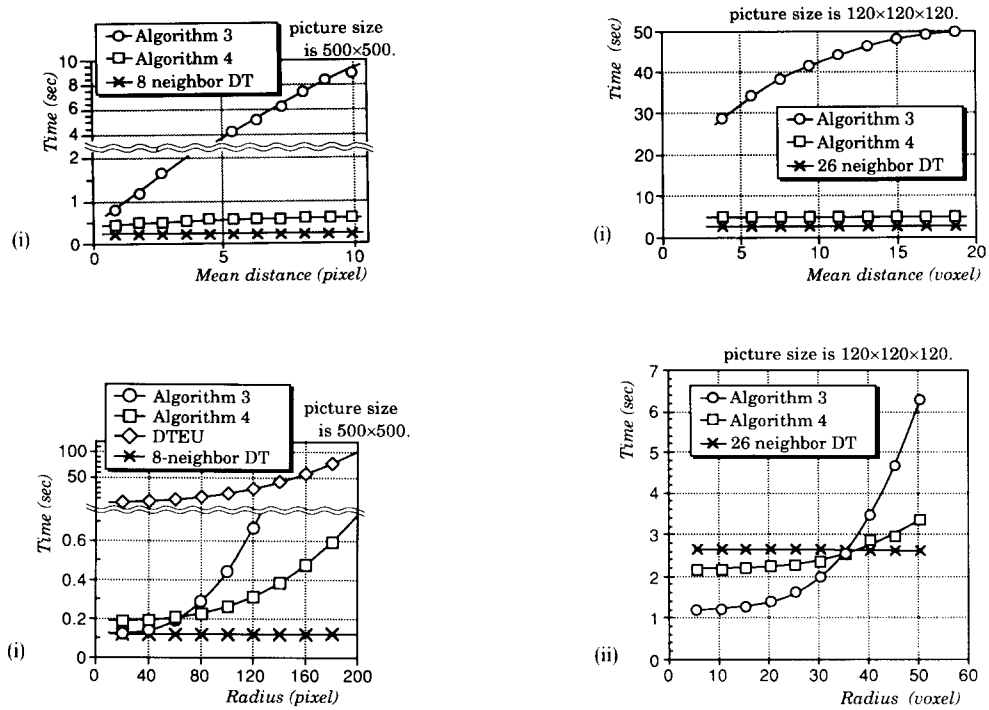


Fig. 8. Experimental evaluation of computation time. (a) Comparison among the proposed algorithms, DTEU and the 8-neighbor DT for two-dimensional pictures. (i) Random dots picture. (ii) Circle picture of radius R. (b) Comparison among the proposed algorithms and the 26-neighbor DT for three-dimensional pictures. (i) Random dots picture. (ii) Sphere picture of radius R.

square distance), while those of Algorithm 4 and the 8-neighbor DT are almost constant. For a circle picture, Algorithm 3 is a little faster than Algorithm 4 for smaller values of radius, and the time of Algorithm 4 is more constant for the change in the radius.

For a three-dimensional picture, computation times of Algorithms 3 and 4 show the tendency similar to the above, and are faster than even the 26-neighbor DT for the radius of a sphere being small. All of algorithms above include only calculation of integer values. The computation times of Algorithms 3 and 4 mean those of calculating the squared EDT. If the exact Euclidean distance values are required we need to calculate the square root of a resultant picture. The computation time for it was about 0.2 ms and 1.2 s for the above two-dimensional pictures and three-dimensional pictures, respectively. It is enough to know relative relation between the distance values in many applications such as extracting skeletons and feature points. In such cases we need not calculate the square root.

6. APPLICATIONS

The presented algorithm was used to analyse microscope images of successive tissue sections. An input picture is a digitized manual trace of border lines of portal and hepatic veins in microscope images of successive tissue sections obtained from the human liver. Sizes of input pictures and sizes of voxels are shown in Table 2. An example of slices and three-dimensional reconstructions of hepatic and portal veins are shown in Fig. 9(a)–(c). The results of the three-dimensional

Table 2. Size of input pictures and intervals between voxels

		Data 1	Data 2
Size of picture (pixel)	1 slice	420 × 594	420 × 594
	Number of slices	29	48
Sampling interval	In slice	10	10
Between voxels (μm)	Between slices	28	24

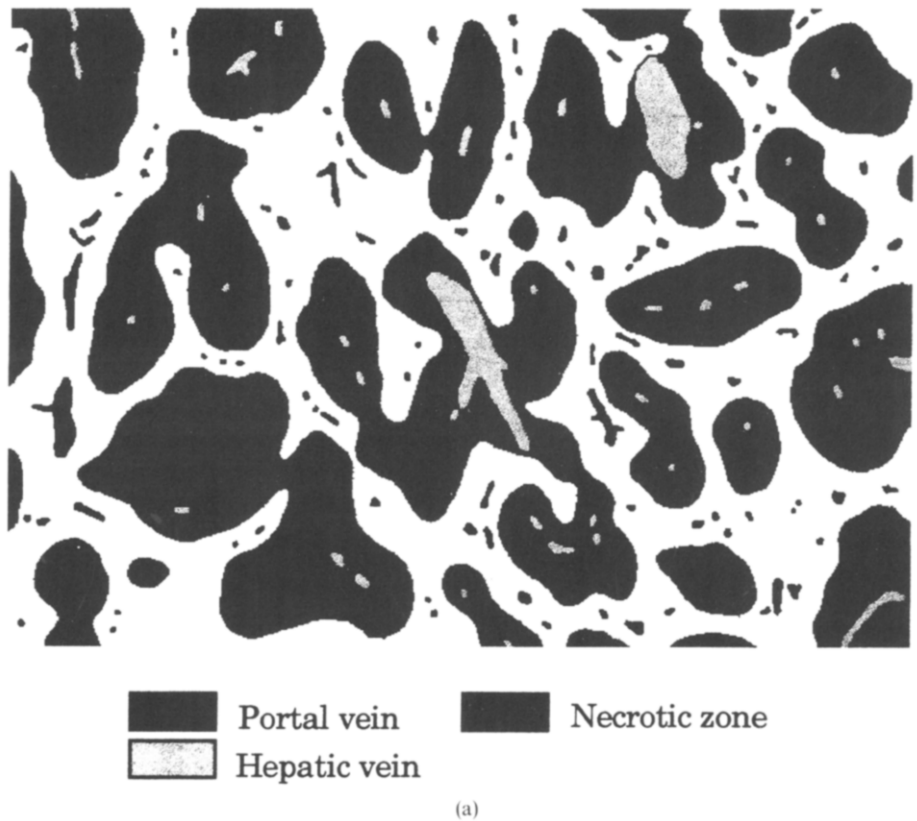


Fig. 9. Input picture. (a) Source picture (Data 1, slice No. 15). (b) Three-dimensional reconstruction of hepatic veins. (c) Three-dimensional reconstruction of portal veins.

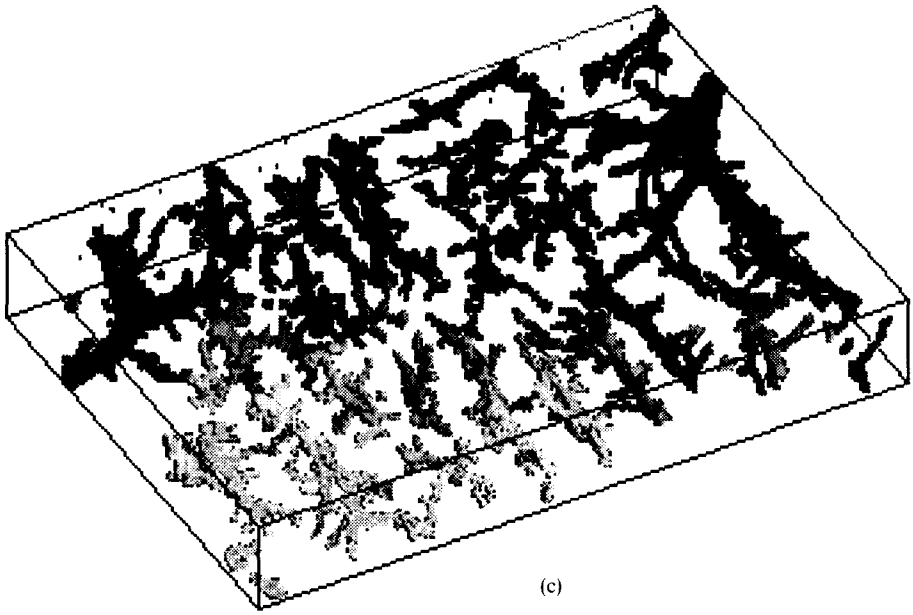


Fig. 9. (*Continued*)

EDT are shown in Fig. 10. Figure 11 shows a cross-section of the Voronoi diagram derived from a set of hepatic veins. For details of this application, see reference (30).

It was speculated, concerning the microstructure of the liver tissue, that for an arbitrary point in the tissue the length of the shortest path from the portal vein to the hepatic vein passing that point is almost constant everywhere. This speculation will be confirmed by examining the distribution of the sum of two EDT values at each point (EDT from the portal veins and that from the hepatic veins), and by finding that the portal veins lie on or near the borders of the Voronoi division derived from the hepatic veins (Fig. 11).

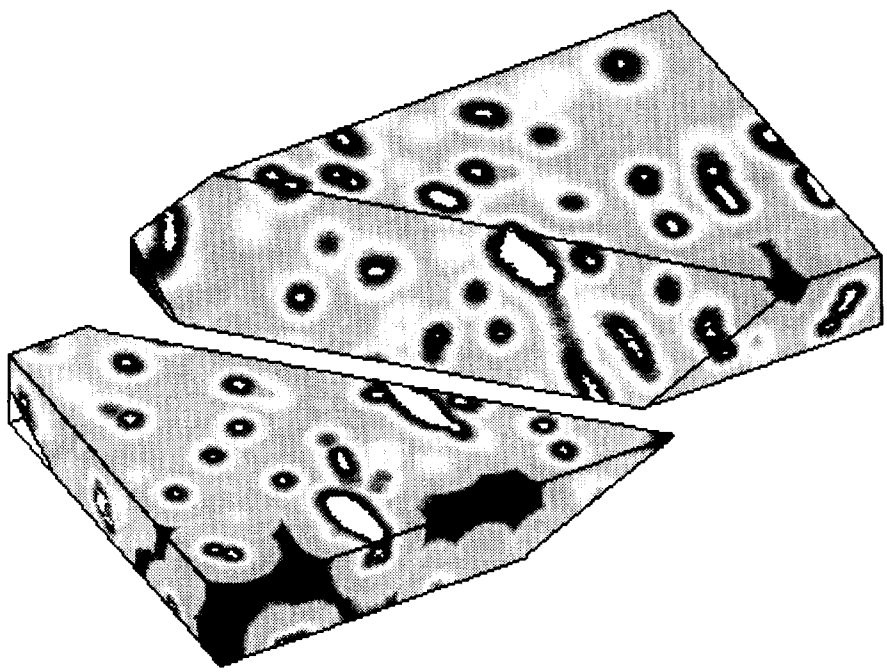


Fig. 10. Distance transformation pictures. (a) Distance transformation from hepatic veins. (b) Distance transformation from portal veins.

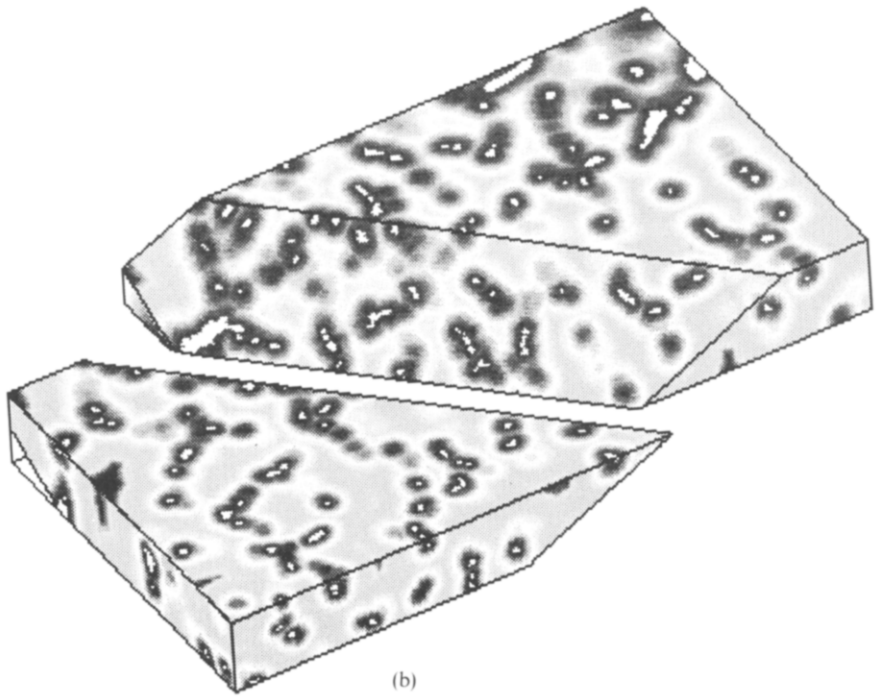


Fig. 10. (*Continued*)

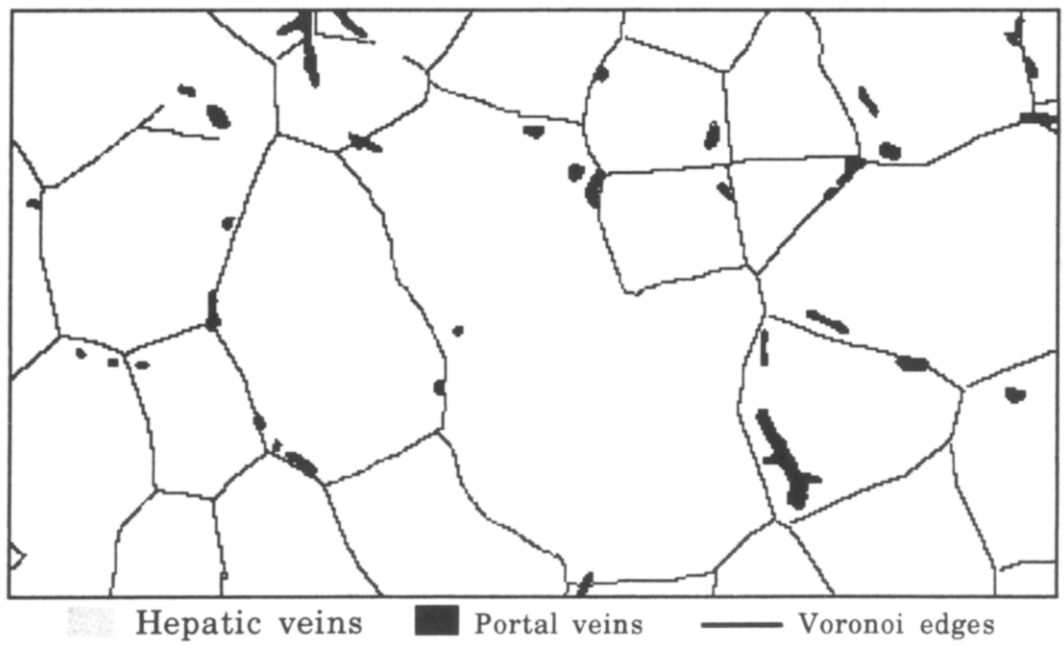


Fig. 11. Voronoi division derived from the hepatic veins.

Experimental results shown here will be utilized to confirm the speculation.

Computation times of EDT from the portal veins and that from the hepatic veins were 269 s and 313 s, respectively, by a general purpose computer FACOM M-1800 in Nagoya University Computation Center.

7. CONCLUSION

In this paper we presented new algorithms to calculate the exact Euclidean distance transformation and the Voronoi diagram for a digitized picture of arbitrary dimensionality and studied their performance. Impor-

tant features of the proposed algorithms are summarized below.

1. They calculate the exact Euclidean distance transformation (EDT) of an arbitrary digitized binary picture.

2. They are applicable to two-dimensional and three-dimensional pictures as shown in the paper, and extended to general n-dimensional binary pictures by slight modification.

3. They are faster than the widely known method of the EDT in reference (19) (the program is available as the FORTRAN subroutine DTEU in SPIDER-II) when implemented on a general purpose computer.

4. Memory requirement is minimum. That is, only one n-dimensional array of the same size as an input picture and a single one-dimensional array for work space are needed for performing the EDT.

5. Algorithms are iterative local operations. They are data-independent in the sense that the number of times of scanning the whole picture does not depend on an input picture. Local operations are always one-dimensional and their neighborhood size is variable.

Development of algorithms to extract skeleton from the Euclidean distance transformation and to perform the inverse distance transformation remains to be studied in the future.

Acknowledgements—Authors wish to thank Dr S. Yokoi and colleagues in their laboratory for helpful discussion. We owe helpful suggestions and microscope pictures in the experiment in 6 to Dr T. Takahashi (Tohoku University). We also thank Dr K. Abe (Shizuoka University) and Dr H. Yamada (Electro-Technical Laboratory) for helpful suggestion of references. Parts of the work were supported by the Grant-in-Aid for Scientific Research, Ministry of Education and the Grant-in Aid for Cancer Research, Ministry of Health and Welfare

REFERENCES

1. J. Toriwaki, M. Okada and T. Saito, Distance transformation and skeletons for shape feature analysis, in C. Arcelli, L. P. Cordella and G. S. di Baja (Eds) *Visual Form: Analysis and Recognition (Proc. of the International Workshop on Visual Form*, 1991. 5.27–30), pp. 547–563. Plenum Press, New York (1992).
2. A. Rosenfeld and J. L. Pfaltz, Sequential operations in digital picture processing, *J. Ass. Comput. Mach.* **13**, 471–494 (1966).
3. M. Yamashita and T. Ibaraki, Distances defined by neighborhood sequences, *Pattern Recognition* **19**, 237–246 (1986).
4. G. Borgefors, Distance transformations in arbitrary dimensions, *CVGIP* **27**, 321–345 (1984).
5. G. Borgefors, A new distance transformation approximating the Euclidean distance, *Proc. 8th ICPR*, pp. 336–339 (1986).
6. G. Borgefors, Distance transformations in digital images, *CVGIP* **34**, 344–371 (1986).
7. E. Thiel and A. Montanvert, Chamfer masks: discrete distance functions, geometrical properties and optimization, *Proc. 11th ICPR*, Vol. 3, pp. 244–247 (1992).
8. F. Leymarie and M. D. Levine, Fast raster scan distance propagation on the discrete rectangular lattice, *CVGIP: Image Understanding* **55**, 84–94 (1992).
9. U. Montanari, A method for obtaining skeletons using a quasi-Euclidean distance, *J. Ass. Comput. Mach.* **15**, 600–624 (1968).
10. J. Toriwaki and S. Yokoi, Distance transformation and skeletons of digitized pictures with applications, in A. Rosenfeld and L. Kanal (eds.) *Progress in Pattern Recognition*, Vol. 1, pp. 187–264. North Holland, Amsterdam (1981).
11. S. Yokoi, J. Toriwaki and T. Fukumura, On generalized distance transformation of digitized pictures. *IEEE Trans. PAMI* **PAMI-3**, 424–443 (1981).
12. A Rosenfeld and J. L. Pfalts, Distance functions on digital pictures, *Pattern Recognition* **1**, 33–61 (1968).
13. S. Yokoi, J. Toriwaki and T. Fukumura, Distance transformation and fusion of digitized binary pictures using a variable neighborhoods sequence, *Trans. IECE*, **J63-D**, 386–393 (1980) (in Japanese).
14. C. T. Huang and O. R. Mitchell, Rapid Euclidean distance transformation using grey scale morphology decomposition, *Proc. CVPR'91*, pp. 695–697 (1991).
15. P. E. Danielsson, Euclidean distance mapping. *Comput. Graphics Image Process.* **14**, 227–248 (1980).
16. I. Ragnemalm, Generation of Euclidean distance maps, Linköping Studies in Science and Technology Thesis No. 206, Linköping, Sweden (1990).
17. I. Ragnemalm, Contour processing distance transforms, in Cantoni *et al.* (eds.), *Progress in Image Analysis and Processing*, pp. 202–212. World Scientific, Singapore (1990).
18. I. Ragnemalm, Neighborhoods for distance transformations using ordered propagation. *CVGIP: Image Understanding* **56**, 399–409 (1992).
19. H. Yamada, Complete Euclidean distance transformation by parallel operation, in *Proc. 7th Int. Conf. on Pattern Recognition Montreal, Canada*, pp. 69–71 (1984).
20. D. W. Paglieroni, Distance transforms: properties and machine vision applications, *Graph. Models Image Process.* **54**, 56–74 (1992).
21. A. Kuwabara, S. Yokoi, J. Toriwaki and T. Fukumura, Distance function and distance transformation on 3-D digital image data, *Trans. IECE*, J65-D, 8, pp. 967–974 (1982) (in Japanese).
22. N. Okabe, J. Toriwaki and T. Fukumura, Fundamental properties of distance functions on the three-dimensional digitized image data, *Trans. IECE*, J66-D, 3, pp. 259–266 (1983) (in Japanese).
23. N. Okabe, J. Toriwaki and T. Fukumura, Paths and distance function on three-dimensional digitized pictures, *Pattern Recognition Lett.* **1**, 205–212 (1993).
24. J. C. Mullikin, The vector distance transform in two and three dimensions, *CVGIP* **54**, 526–535 (1992).
25. I. Ragnemalm, The Euclidean distance transform in arbitrary dimensions, *Pattern Recognition Lett.* **14**, 883–888 (1993).
26. J. Toriwaki and S. Yokoi, Voronoi and related neighbors on digitized two-dimensional space with applications to texture analysis, in Toussaint G. T. (ed.) *Computational Morphology*, pp. 207–228. Elsevier Science, North Holland (1988).
27. F. Phodes, Discrete Euclidean metrics. *Pattern Recognition Lett.* **13**, 623–628 (1992).
28. E. Bertin and J. M. Chassery, 3-D Voronoi diagram: application to segmentation, *Proc. 11th ICPR*, **3**, pp. 197–200 (1990).
29. K. Sugihara, Y. Ooishi and T. Imai, Topology-oriented approach to robustness and its applications to several Voronoi-diagram algorithms, in J. Urrutia (ed.) *Proc. of the Seasonal Canadian Conference in Computational Geometry, Ottawa*, pp. 36–39 (1990).
30. T. Saito and J. Toriwaki, Algorithms of three-dimensional Euclidean distance transformation and extended digital Voronoi diagram, and analysis of human liver section images. *J. Inst. Image Electronics Engrs Japan* **21**, 468–474 (1992) (in Japanese).

APPENDIX

Euclidean distance transformation (basic type) by Algorithm 3 for three-dimensional pictures.

- Input picture: $F = \{f_{ijk}\}$, $f_{ijk} = 0$ or 1.
- Output picture (distance transformation): $F = \{f_{ijk}\}$.
- L, M, N: sizes of pictures (numbers of rows, columns and planes).
- buff(n): one-dimensional work array with the size n.
- int(x): function to convert the data type from the real type to the integer type.
- min(x, y): function to select smaller of x and y.
- sqrt(x): function to calculate the square root of x.

```
(Step 1)
————————forward scan————————
for (k = 1 → N) {
  for (j = 1 → M) {
    df = L;
    for (i = 1 → L) {
      if (f_ijk ≠ 0)  df = df + 1;
      else df = 0;
      f_ijk = df²;
    }
  }
}
————————backward scan————————
for (k = 1 → N) {
  for (j = 1 → M) {
    db = L;
    for (i = L → 1) {
      if (f_ijk ≠ 0)  db = db + 1;
      else db = 0;
      f_ijk = min(f_ijk, db²);
    }
  }
}
(Step 2)
for (k = 1 → N) {
  for (i = 1 → L) {
    for (j = 1 → M) {
      buff(j) = f_ijk;
    }
    for (j = 1 → M) {
      d = buff(j);
      if (d ≠ 0) {
        rMax = int(sqrt(d)) + 1;
        rStart = min(rMax, (j − 1));
        rEnd = min(rMax, (M − j));
        for (n = −rStart → rEnd) {
          w = buff(j + n) + n²;
          if (w < d) d = w;
        }
      }
      f_ijk = d;
    }
  }
}
```

(Step 3)

Same as Step 2 of Algorithm 3

(End)

Euclidean distance transformation (fast type) by Algorithm 4 for three-dimensional pictures.

Notations are same as those in Algorithm 3.

```
(Step 1)
Same as Step 1 of Algorithm 3.

(Step 2)
for (k = 1 → N) {
  for (i = 1 → L) {
    for (j = 1 → M) {
      buff(k) = f_ijk;
    }
————————forward scan————————
    a = 0
    for (j = 2 → M) {
      if (a > 0) a = a − 1
      if (buff(j) > buff(j − 1) + 1) then
        b = (buff(j) − buff(j − 1) − 1)/2
        if ((j + b) > M) b = M − j
        for (n = a → b)
          m = buff(j − 1) + (n + 1)²
          if (buff(j + n) ≤ m) goto L1:
          if (m < f_i(j+n)k f_i(j+n)k = m
        }
L1:     a = b
      else
        a = 0
      endif
    }

————————backward scan————————
    a = 0
    for (j = M − 1 → 1) {
      if (a > 0) a = a − 1
      if (buff(j) > buff(j + 1) then
        b = (buff(j) − buff(j + 1) − 1)/2
        if ((j − b) < 1) b = j − 1
        for (n = a → b)
          m = buff(j + 1) + (n + 1)²
          if (buff(j − n) ≤ m) goto L2:
          if (m < f_i(j−n)k) f_i(j−n)k = m
        }
L2:     a = b
      else
        a = 0
      endif
    }
  }
}

(Step 3)
  Same as Step 2 of Algorithm 4.
(End)
```

About the Author—TOYOFUMI SAITO received the B.S. degree in electronics engineering, and the M.S. degree and the Ph.D. degree in information engineering from Nagoya University, Japan, in 1986, 1988 and 1993, respectively. Since 1991, he has been a Research Associate at the Department of Information Engineering at Nagoya University.

His research interests are pictorial pattern recognition and image processing. He is a member of the Institute of Electronics, Information and Communication Engineers of Japan and the Japan Society of Medical Electronics and Biological Engineering.

About the Author—JUN-ICHIRO TORIWAKI received the B.S. and M.S. degrees in electronics engineering and the Ph.D. degree in electrical engineering from Nagoya University in 1962, 1964 and 1969. He was a Research Fellow during 1967–1969, a Lecturer in 1969–1970, and an Associate Professor from 1970–1974 at the Department of Electrical Engineering, Nagoya University. From 1974–1976, he was an Associate Professor at the Computation Center. From 1976–1980, he was an Associate Professor in the Department of Electrical Engineering, and the Department of Information Science. From 1980 until 1982, he served as a Professor in the Department of Information and Computer Sciences, Toyohashi University of Technology. Since 1983, he has been a Professor at the Department of Information Engineering at Nagoya University.

Dr Toriwaki specializes in the areas of pictorial pattern recognition, biomedical image processing, and computer graphics with applications. He has been devoted to the theoretical analysis of the algorithms for digital picture processing, and three-dimensional digital geometry, associated software development for image processors such as SLIP and SPIDER. Other areas of interest include computerized screening systems for chest and stomach X-ray images, and display of three-dimensional images, including CT and microscope. Most recently, he is involved in three-dimensional medical image analysis, the development of surgical simulation studies, and neural network fundamentals. He has published about 250 scientific papers. He received Niwa Takayanagi Award (the best author award) of the Institute of Television Engineers of Japan in 1991 and the Paper Award of the Institute of Image Electronics Engineers of Japan in 1992.

Dr Toriwaki is a member of the IEEE, the Institute of Electronics, Information and Communication Engineers of Japan, the Information Processing Society of Japan, and the Japan Society of Medical Electronics and Biological Engineering.