Pattern Recognition 46 (2013) 230–242

Contents lists available at SciVerse ScienceDirect

Pattern Recognition

journal homepage: www.elsevier.com/locate/pr

Efficient Euclidean distance transform algorithm of binary images in arbitrary dimensions ☆

Jun Wang [a,b,c], Ying Tan [a,b,*]

[a] Key Laboratory of Machine Perception (MOE), Peking University, China
[b] Department of Machine Intelligence, School of Electronics Engineering and Computer Science, Peking University, Beijing 100871, China
[c] Shandong Provincial Office, State Administration of Taxation, Jinan, China

ARTICLE INFO

Article history:
Received 8 August 2011
Received in revised form
10 April 2012
Accepted 29 July 2012
Available online 10 August 2012

Keywords:
Euclidean distance transform
Arbitrary dimensions
Independent scan
Linear time algorithm
Binary image

ABSTRACT

In this paper, we propose an efficient algorithm, i.e., PBEDT, for short, to compute the exact Euclidean distance transform (EDT) of a binary image in arbitrary dimensions. The PBEDT is based on independent scan and implemented in a recursive way, i.e., the EDT of a d-dimensional image is able to be computed from the EDTs of its $(d-1)$-dimensional sub-images. In each recursion, all of the rows in the current dimensional direction are processed one by one. The points in the current processing row and their closest feature points in $(d-1)$-dimensional sub-images can be shown in a Euclidean plane. By using the geometric properties of the perpendicular bisector, the closest feature points of $(d-1)$-dimensional sub-images are easily verified so as to lead to the EDT of a d-dimensional image after eliminating the invalid points. The time complexity of the PBEDT algorithm is linear in the amount of both image points and dimensions with a small coefficient. Compared with the state-of-the-art EDT algorithms, the PBEDT algorithm is much faster and more stable in most cases.

© 2012 Elsevier Ltd. All rights reserved.

1. Introduction

A d-dimensional binary image I is a function: $[1\ldots n_1]\times\cdots\times[1\ldots n_d]\rightarrow\{0,1\}$, each element of which has a value 0 or 1, referred to as background or feature (foreground) point, respectively. The distance transform (DT) of a binary image computes the distance between each image pixel and its closest feature point [1]. Distance transform is related to many other important entities, such as medial axes, Voronoi diagrams, shortest path computation, and image segmentation [2]. Distance transform algorithms are excellent tools for a variety of applications, such as image processing, computer vision, pattern recognition, shape analysis and computational geometry [3–9]. In practice, several distance metrics, such as the city-block (L_1), the chessboard (L_∞), the octagonal, and the Euclidean distance, are all used for different situations. One of the most natural and appropriate metrics is the Euclidean metric, which is the most adequate model to numerous geometrical facts of the human-scale world and used in many applications, since it is radially symmetric, virtually invariant to rotation [2,10,11]. However, the computation of exact Euclidean distance transform, named EDT, is time consuming.

Intuitionally, treated as a global operation, EDT can be computed by an exhaustively brute-force searching algorithm: for each pixel of the image, compute the distance between it and each feature pixel, which requires $O(N^2)$ time (N is the number of image pixels) [10,11]. It is generally agreed that an efficient EDT should be based on obtaining feature pixel information from a limited region to avoid global searching. Efficient non-Euclidean distance transform algorithms have been reported since 1966, while fast algorithms for EDT started to appear only in the 1990s. Some extensive surveys are introduced in [2,12]. These algorithms can be roughly classified into three approaches according to how to search for the nearest feature pixel:

1. *Ordered propagation*. The smallest distance information is computed starting from the feature pixels and iteratively propagating via contour pixels in order of increasing distance [13–17]. Piper and Granum propose a FIFO (first-in-first-out) strategy only to process the contour of propagation [18]. Verwer et al. use bucket sorting to store the contour pixels [13]. Eggers restricts the propagation only along the shortest Euclidean distance [15]. The main problem of this approach is that some pixels can be updated more than once [2,12]. The performance of the ordered propagation EDT algorithms is difficult to promote.
2. *Raster scan*. Rosenfeld and Pfaltz propose the first sequential distance transform algorithm by raster scan with non-Euclidean

☆This work was supported by the National Natural Science Foundation of China under grants No. 61170057 and 60875080. This work is also in part supported by the National High Technology Research and Development Program of China (863 Program), with grant number 2007AA01Z453.

*Corresponding author at: Department of Machine Intelligence, School of Electronics Engineering and Computer Science, Peking University, Beijing 100871, China. Tel./fax: +86 10 62767611.
E-mail addresses: bedouins@pku.edu.cn.(J. Wang)ytan@pku.edu.cn (Y. Tan).

0031-3203/$ - see front matter © 2012 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.patcog.2012.07.030

metric [1]. The raster scan method is technically fast, since the scan method processes pixels one by one and the process in each step is concise. The algorithm of Rosenfeld and Pfaltz obtains the distance to the closest feature point from neighbor points [1]. Danielsson proposes a method that obtains the position information of the closest feature point from neighbor points [19]. Borgefors uses a chamfer metric to define neighbor points [20]. Most algorithms of this approach are non-exact Euclidean distance transform. It is hard to obtain EDT by this approach since the algorithms based on the raster scan have an inherent error: as a discrete lattice image, the front wave of propagating is tiled, which results in the position/distance information of feature pixels not being propagated precisely [19]. Some researchers use many techniques to achieve exact EDT by this approach, but the complex data structures and the complicated operations decrease the performance [21–23]. Recent progresses in this approach are Cuisenaire and Macq propose a method by using multi-neighborhoods: a fast but approximate distance transform is first computed by using a coarse neighborhood; a sequence of larger neighborhoods is then used to refine this approximation [16]. They state that their algorithm apparently remains $O(N)$ even in the worst case, based on experiments in [12,16]. However, argued by Fabbri et al., Cuisenaire et al.'s conjecture has not been proved and the pre-computation of distances error boundary is time-consuming even for moderate distances [2]. Shih and Wu [22,23] also propose an algorithm which uses two raster-scans by using a 3×3 neighborhood with dynamically adjusted neighborhood size.

3. *Independent scan*. This approach computes the EDT by dimensional reduction, which is also devised by Rosenfeld and Pfaltz [1]. First, the one-dimensional distance transform is constructed for each row (or column) independently; next, the intermediate result is used in a second phase to construct the full two-dimensional distance transform [24–27]. As a derivative of the raster scan, independent scan methods inherit the succinct processing structure with a modified inner procedure. The first stage methods of all independent scan distance transform algorithms are similar. Specific properties of the Euclidean metric are exploited during the second phase in order to restrict the amount of computations. There are three variants of this approach:
   - The first variant uses the simulation of parabola intersection. Saito and Toriwaki propose an EDT algorithm which speeds up the second phase scan by calculating the lower envelope of parabolas to avoid exhausted searching for the nearest feature points in columns [28]. Similar methods are used in [29–31], which all take $O(N)$ time.
   - The second variant uses a mathematical morphological operation. Shih and Mitchell use gray scale morphological erosion and dilation to compute EDT [32]. Huang and Mitchell use a dilation with 3×3 structuring element to compute EDT [33]. Lotufo and Zampirolli propose a method of further decomposing the structuring element into one-dimensional structure element, which takes $O(N)$ time [34,35].
   - The third variant is based on Voronoi diagram intersection. This approach is explicitly based on fast computation of Voronoi diagram intersections with image lines, which adopts the principle that EDT can be computed by using general Voronoi diagram constructing algorithms [36–39]. As the image points have integer coordinates, Breu et al. propose an EDT algorithm by using the construction of the intersection of the partial Voronoi diagram of sampled feature pixels with each row [10,11,40,41]. Their algorithm takes $O(N)$ time. Gavrilova proposes a method to compute EDT by computing the intersection directly [41], which is time consuming. Guan and Ma use line segment to replace a single point in the calculation, since adjacent points in a row incline to have the same closest feature points [40]. Maurere et al. adapt the ideas of Breu et al. and Guan et al. to propose a more efficient algorithm, which also takes $O(N)$ time [11].

There is plenteous research on EDT of two-dimensional binary images. Although many of these algorithms are of liner time complexity, some are not stable when the image content is changed, and some have a large constant term [2,12]. Many parallel EDT algorithms are also proposed which are extensions of their sequential version [42,43]. Especially the high parallel processing capability of GPU provides more speed-ups of EDT [44,45]. After all, a fast EDT algorithm is decisive.

Two-dimensional EDT algorithms are broadly used in applications of image processing and computer vision, while higher dimensional EDT algorithms are more suited to pattern recognition [5,8,25,46,47]. The research on three or more dimensional binary image which appeared at 1980s is insufficient. Because of the heavy computational load of high dimensional EDT, many researchers propose fast but non-exact Euclidean distance transform algorithms by using the raster scan. Borgefors proposes a method which uses chamfer metric for multi-dimensional Euclidean distance transform [20]. Ragnemalm [21] gives the method to obtain the minimum scan times for multi-dimensional Euclidean distance transform. Borgefors proposes a new definition of chamfer metric for three-dimensional Euclidean distance transform [48]. Svensson and Borgefors [49] propose a $5 \times 5 \times 5$ neighbor to compute three-dimensional Euclidean distance transform. Borgefors proposes a definition of $3 \times 3 \times 3 \times 3$ neighbors to compute four-dimensional Euclidean distance transform [50]. Strand and Borgefors use face-centered cubic and body-centered cubic to replace the previous cubic in order to compute three-dimensional Euclidean distance transform [47], which has less error to the exact EDT. EDT algorithms for multi-dimensional binary images are all based on independent scan [11,28,46], in which all have the same principles as their two-dimensional version, respectively.

This paper is the extension of our preliminary work presented at the International Conference on Computer Vision and Pattern Recognition (CVPR 2011) [51]. In previous work, we proposed an efficient method to compute EDT for two-dimensional binary images. It used the geometric properties of the perpendicular bisector to compute the segmentation by the closest feature points for each row. The main contribution of this work is that it presents a thoroughly theoretical treatment of EDT by means of independent scan. In addition, we propose an effective unified EDT algorithm for binary images in arbitrary dimensions. The proposed algorithm is based on independent scan and implemented in a recursive way, i.e., the EDT of an d-dimensional image is able to be computed from the EDTs of its $(d-1)$-dimensional sub-images. In each recursion, all of the rows in the current dimensional direction are processed one by one. The points in the current processing row and their closest feature points in $(d-1)$-dimensional sub-images can be shown in a Euclidean plane. Thus, a common procedure is adapted to compute the EDT of every dimension. Furthermore, we refine the previous method in the distance calculation part: the distance between each point in the currently processed row and its closest feature point is computed by one distance calculation. By contrast, in this part, Wang and Tan [51] compare adjacent verified points to determine which one is the closest point of the operational point, which needs a two-distance calculation in each step of the loop.

2. Background on independent scan

In this section, we introduce the principle of independent scan and explain why the d-dimensional EDT can be computed from EDT of all $(d-1)$-dimensional sub-images. As mentioned before, independent scan, which is also called dimensional reduction, is the one of the most common choices in EDT computation. The essential part of independent scan is that each dimension is processed separately [11,31]. As usual, the EDT algorithm takes a d-dimensional binary image I_d as input—$[1\dots n_1]\times\cdots\times[1\dots n_d]\to 0,1$, and outputs the distance transform, usually in squared distance. The feature points of I_d are denoted by F_d. Let $f(u)$ denote the closet feature point of u ($\|\cdot\|$ denote the Euclidean metric)

$$f(u)=\arg\min_{v\in F_d}\|u-v\|^2,\quad u\in I_d \tag{1}$$

(if more than one closest feature points, choose one arbitrarily).

The method of computing the closest feature point of each point is called the closest feature point transform (CFT) [11]. The EDT of image I_d can be computed from its CFT, which is of linear time complexity (each $\|\cdot\|$ calculation takes $O(1)$ time)

$$EDT(u)=\|u-f(u)\|^2,\quad u\in I_d. \tag{2}$$

I_d can be decomposed from two aspects:

1. Let $I^{d-1}_{i_d}$ denote a $(d-1)$-dimensional sub-image of I_d whose dth coordinate is at i_d. Thus, there are n_d $(d-1)$-dimensional sub-images, and
$$I_d=\bigcup_{i_d\in[1,n_d]} I^{d-1}_{i_d}. \tag{3}$$
The feature points in sub-image $I^{d-1}_{i_d}$ are denoted by $F^{d-1}_{i_d}$.
2. Let $R^d_{(i_1,\dots,i_{d-1})}$ denote a row in the dth dimensional direction of I_d. Thus, there are $n_1\times\dots\times n_{d-1}$ rows in this dimension, and
$$I_d=\bigcup_{i_k\in[1,n_k],k\in[1,d-1]} R^d_{(i_1,\dots,i_{d-1})}. \tag{4}$$

Given a row $R^d_{(i_1,\dots,i_{d-1})}$, each point in $R^d_{(i_1,\dots,i_{d-1})}$ can be located only by its dth coordinate, denoted by $R^d_{(i_1,\dots,i_{d-1})}(x)$ $(1\le x\le n_d)$. For a sub-image I^{d-1}_x, $R^d_{(i_1,\dots,i_{d-1})}$ is orthogonal to I^{d-1}_x, and the point $I(i_1,\dots,i_{d-1},x)$ is the cross-point of $R^d_{(i_1,\dots,i_{d-1})}$ and I^{d-1}_x. Therefore, the closest feature point of $I(i_1,\dots,i_{d-1},x)$ in F^{d-1}_x is closer to $I(i_1,\dots,i_{d-1},x)$ than other feature points, denoted by $N^{d-1}_{(i_1,\dots,i_{d-1})}(x)$. Besides, $N^{d-1}_{(i_1,\dots,i_{d-1})}(x)$ is closer to every point in $R^d_{(i_1,\dots,i_{d-1})}$ than other feature points in F^{d-1}_x ($R^d_{(i_1,\dots,i_{d-1})}$ is orthogonal to I^{d-1}_x).

Let $N^{d-1}_{(i_1,\dots,i_{d-1})}$ denote the $N^{d-1}_{(i_1,\dots,i_{d-1})}(x)$ of all points in $R^{d-1}_{(i_1,\dots,i_{d-1})}$. Thus

$$N^{d-1}_{(i_1,\dots,i_{d-1})}=\bigcup_{x\in[1,n_d]} N^{d-1}_{(i_1,\dots,i_{d-1})}(x) \tag{5}$$

Let $\Delta^{d-1}_{(i_1,\dots,i_{d-1})}(x)$ denote the squared distance between a point $R^d_{(i_1,\dots,i_{d-1})}(x)$ and it closest feature point in the $d-1$ sub-image

$$\Delta^{d-1}_{(i_1,\dots,i_{d-1})}(x)=\|R^d_{(i_1,\dots,i_{d-1})}(x)-N^{d-1}_{(i_1,\dots,i_{d-1})}(x)\|^2 \tag{6}$$

For a point $R^d_{(i_1,\dots,i_{d-1})}(x)$, if there does not exist a feature point in sub-image I^{d-1}_x, then its closest feature point in sub-image I^{d-1}_x also does not exist. In this situation, $N^{d-1}_{(i_1,\dots,i_{d-1})}(x)$ is assigned value *null*, and $\Delta^{d-1}_{(i_1,\dots,i_{d-1})}(x)$ is assigned value ∞ (a very large integer which greater than the maximum squared distance of the EDT of this image). Thus, Eq. (1) can be rewritten as

$$f(u)=\arg\min_{v\in N^{d-1}_{(i_1,\dots,i_{d-1})}}\|u-v\|^2,\quad u\in R^d_{(i_1,\dots,i_{d-1})}. \tag{7}$$

The calculation of CFT is simplified: by (1), the closest feature point of u is searched in F_d, while is only searched in $N^{d-1}_{(i_1,\dots,i_{d-1})}$ by (7). Furthermore, Eq. (7) can be rewritten as

$$\begin{aligned} f(u)&=N^{d-1}_{(i_1,\dots,i_{d-1})}\left(\arg\min_{x\in[1,n_d]}\|u-N^{d-1}_{(i_1,\dots,i_{d-1})}(x)\|^2\right)\\ &=N^{d-1}_{(i_1,\dots,i_{d-1})}\left(\arg\min_{x\in[1,n_d]}\|(u\cdot x-x)^2+\Delta^{d-1}_{(i_1,\dots,i_{d-1})}(x)\|\right),\quad u\in R^d_{(i_1,\dots,i_{d-1})}\end{aligned} \tag{8}$$

By (8), two conclusions are concluded:

- The CFT of I_d can be computed from the CFT of all the $(d-1)$-dimensional sub-images. In the same way, the CFT of the $(d-1)$-dimensional sub-image can be computed from the CFT of all the $(d-2)$-dimensional sub-images, and so on. Therefore, the CFT of d-dimensional binary image can be computed by a recursive procedure: the CFT of k-dimensional sub-image is computed from the CFT of all the $(k-1)$-dimensional sub-images, $1<k\le d$.
- In the computation of the CFT of a d-dimensional image, for a given row $R^d_{(i_1,\dots,i_{d-1})}$, only the points in $R^d_{(i_1,\dots,i_{d-1})}$ and the closest feature points in sub-images ($N^{d-1}_{(i_1,\dots,i_{d-1})}$) of points in $R^d_{(i_1,\dots,i_{d-1})}$ are concerned. The points in $R^d_{(i_1,\dots,i_{d-1})}$ and $N^{d-1}_{(i_1,\dots,i_{d-1})}$ can be mapped into a Euclidean plane. Let $R^d_{(i_1,\dots,i_{d-1})}$ be the x-coordinate and let the orthogonal direction be the y-coordinate, the coordinate of the points in $N^{d-1}_{(i_1,\dots,i_{d-1})}$ is $(x,\Delta^{d-1}_{(i_1,\dots,i_{d-1})}(x))$.

Accordingly, the CFT of I_d can be computed by a recursive procedure.

3. Principle of PBEDT

In this section, we describe the main ideas of the proposed algorithm—PBEDT (Perpendicular Bisector Euclidean Distance Transform). For illustration, we use the CFT to explain the principle of PBEDT. In the computation of d-dimensional CFT, each row is processed separately. The geometrical properties of the perpendicular bisector are used to verify candidate points and the d-dimensional CFT of currently processed row is computed, which PBEDT is named after. Processing each row with this method, the CFT of the d-dimensional image is computed. Accordingly, the EDT of the d-dimensional image is also computed.

3.1. Overview

Assuming that the CFT of all $(d-1)$-dimensional sub-images has been computed, let us focus on the procedure to compute the CFT of the d-dimensional image. There are $n_1\times\cdots\times n_{d-1}$ rows in the dth-dimensional direction and each row is processed separately. Without loss of generality, given a row $R^d_{(i_1,\dots,i_{d-1})}$, its closest feature points in all sub-images—$N^{d-1}_{(i_1,\dots,i_{d-1})}$ are computed by the CFT of all $(d-1)$-dimensional sub-images. $R^d_{(i_1,\dots,i_{d-1})}$ and $N^{d-1}_{(i_1,\dots,i_{d-1})}$ are transformed into a Euclidean plane as R' and N', respectively. Thus, Eq. (6) can also be expressed by a simpler form:

$$\Delta'(x)=\|R'(x)-N'(x)\|^2 \tag{9}$$

Let R' be the x-coordinate and let the orthogonal direction be the y-coordinate, so the coordinate of the points in N' is $(x,\Delta'(x))$.

Let $f^d(u)$ denote the closest feature point of $u \in R'$ in the d-dimensional image. Let Γ denote the list of the closest feature points of points in R' in the d-dimensional image

$$\Gamma = \{v | v = f^d(u),\ u \in R',\ v \in N'\} \tag{10}$$

Hence, $\Gamma \subseteq N'$. Let points in N' and Γ be increasingly ordered by the x-coordinate. Therefore, the objective of computing the CFT of the d-dimensional image from the CFT of all $(d-1)$-dimensional sub-images is to compute Γ from N'. Let $\Lambda(t)$ denote the set of points in R' whose closet feature point in the d-dimensional image is t, called t's region of influence in row R' of the d-dimensional image

$$\Lambda(t) = \{v | f^d(v) = t,\ v \in R'\}, \quad t \in N' \tag{11}$$

If $t \in \Gamma$, then $\Lambda(t) \neq \emptyset$; otherwise, $\Lambda(t) = \emptyset$. Therefore, Γ can be obtained from N' via

$$\Gamma = N' - \{u | \Lambda(u) = \emptyset,\ u \in N'\} \tag{12}$$

3.2. Verify points by using perpendicular bisector

Next, we take advantage of the geometrical property of the perpendicular bisector to verify the points in N' whose region of influence in row R' is empty. Given two feature points $u, v \in N'$, let B_{uv} denote the perpendicular bisector of u and v (Fig. 1). B_{uv} is the set of points whose distances to u and to v are equal. Consequently, the points located at the u side of B_{uv} are closer to u than to v or vice versa. Let c_{uv} denote the intersection point of B_{uv} with row R'. Given $u \cdot x < v \cdot x$, the points in R' at the left of $c_{uv} \cdot x$ are closer to u than to v, and we assign as c_{uv} belonging to the u side of B_{uv}. In Fig. 1, $c_{uv} \cdot x$ can be computed from $\|u - c_{uv}\| = \|v - c_{uv}\|$.

We defined the points in N' as increasingly ordered by the x-coordinate, and the points in which are verified from left to right.

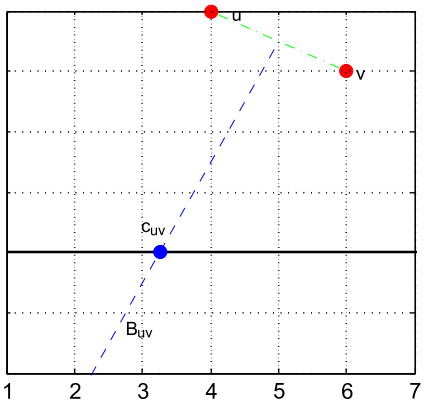

Fig. 1. The perpendicular bisector of u and v intersects row R'.

Assume the current verifying point is w, the points in N' whose x-coordinate is less than w's are all verified. Let v denote the last verified point and $\Lambda(v) \neq \emptyset$, and u be the second to last point. Thus, there are three situations of B_{vw} intersecting with row R':

1. B_{vw} intersects with the left extension of row R' ($c_{vw} \cdot x < 0$), thus $\Lambda(v) = \emptyset$ (Fig. 2(a)).
2. B_{vw} intersects with the right extension of row R' ($c_{vw} \cdot x > n_m$), thus $\Lambda(w) = \emptyset$ (Fig. 2(b)).
3. B_{vw} intersects with row R' ($0 < c_{vw} \cdot x \leq n_m$), which has two situations:
   - If $c_{uv} \cdot x < c_{vw} \cdot x (u, v, w \in N')$, then $\Lambda(v) \neq \emptyset$ and $\Lambda(w) \neq \emptyset$ (Fig. 2(c)).
   - If $c_{uv} \cdot x > c_{vw} \cdot x (u, v, w \in N')$, then $\Lambda(v) = \emptyset$ (Fig. 2(d)).

Therefore, the relation of $c_{uv} \cdot x$ and $c_{vw} \cdot x$ expresses whose region of influence is empty. It is obvious that if $\Lambda(w) = \emptyset$, then $w \notin \Gamma$ (Fig. 2(b)). Next, if $\Lambda(v) = \emptyset$, then w should be verified with the point prior to v (Fig. 2(a) and (d)). However, while $\Lambda(v) \neq \emptyset$ and $\Lambda(w) \neq \emptyset$, whether w should be verified with the point prior to v in N'? To answer this question, by the perpendicular bisector intersects with R', we observe some properties of the region of influence:

Property 1. *The points in set $\Lambda(t)$ are continuous with the x coordinate, or, $\nexists v\{v | u \cdot x \leq v \cdot x \leq w \cdot x, u \in \Lambda(t), w \in \Lambda(t), v \notin \{\Lambda(t)\}\}$.*

Proof. If $\exists v\{v | u \cdot x \leq v \cdot x \leq w \cdot x, u, w \in \Lambda(t), v \notin \{\Lambda(t)\}\}$, then exists $v \in \Lambda(s)$. Thus, B_{st} intersects with row R' between u and v, and between v and w, as well. A line cannot cross a hyperspace twice in a Euclidean space. Therefore, the hypothesis is false. □

Moreover

Property 2. *If $u, v \in \Gamma$, $u \cdot x < v \cdot x$, then $s \cdot x \leq t \cdot x$, $\forall s \in \Lambda(u)$ and $\forall t \in \Lambda(v)$.*

Accordingly, if the points in N' which is located at the left of w are all verified, then none of them is closer to the points of R' which is located at the right of $c_{vw} \cdot x$ than w (Properties 1 and 2). Therefore, when $\Lambda(v) \neq \emptyset$ and $\Lambda(w) \neq \emptyset$ (Fig. 2(c)), w need not be verified with the point prior to v in N'.

3.3. CFT computation for a row

Consequently, we use CFT_ROW() to show the idea of computing Γ from N', i.e., computing the CFT for a row. For current row R', Γ is the list of d-dimensional closest feature points to the points in R', and N' is the list of $(d-1)$-dimensional closest feature points to the points in R'. Processing each row with CFT_ROW(), the CFT of d-dimensional image is computed. Accordingly, the EDT of the d-dimensional image is also computed. *Stack* is used to store verified feature points, which is a last-in-first-out data

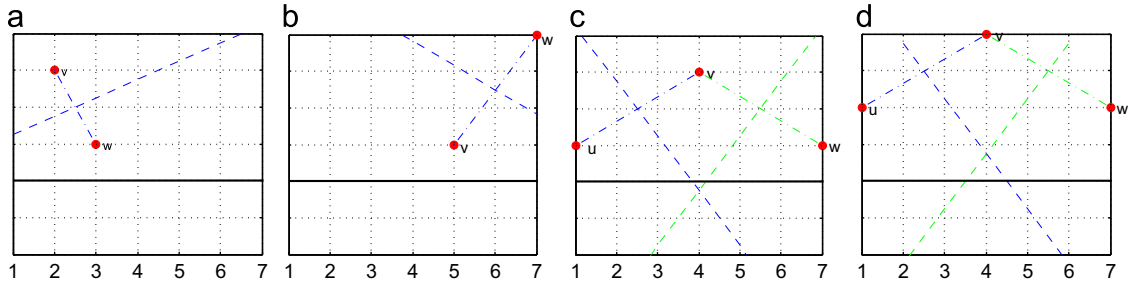

Fig. 2. Four situations when a point is verified. (a) $\Lambda(v) = \emptyset$; (b) $\Lambda(w) = \emptyset$; (c) $\Lambda(v) \neq \emptyset$ and $\Lambda(w) \neq \emptyset$; (d) $\Lambda(v) = \emptyset$.

structure (LIFO) and has three operations: add element to the top of *stack*, PUSH; delete the top element of *stack*, POP; access the top element of *stack*.

```
CFT_ROW()
1  stack and S_c are empty stacks.
2  while N' is not empty
3    withdraw the lowest point w from N';
4    if stack is empty
5      PUSH(stack, w)
6    else
7      v is top of stack, calculate c_vw · x;
8      u is the point in stack next to v;
9      if (u is not exist)
10       c_uv · x = 0;
11     else
12       calculate c_uv · x;
13   { POP(stack), add w back to the front of N', when c_vw · x < c_uv · x;
     { POP(stack), PUSH(stack, w), when c_vw · x = c_uv · x;
     { PUSH(stack, w), when c_vw · x > c_uv · x and c_vw · x < cols;
```

Initially, $stack=\emptyset$. Points are moved from N' to *stack* one by one. If a newly added point results in point p of *stack* whose region of influence on R' being empty, then p should be removed from *stack*. When *stack* is empty, the newly added point w is added to *stack*. When *stack* only has one element, let $c_{uv}\cdot x=0$. In the next loop, the newly added point w and the top point of *stack*—v are used to compute $c_{vw}\cdot x$, which is compared with $c_{uv}\cdot x$. Thus, the bisectors of every adjacent two points in Γ segment R', and each segmentation corresponds one point in Γ—its closest feature point (Properties 1 and 2).

3.4. "0"-dimensional CFT

In the recursive procedure, the CFT of the k-dimensional image is computed from the CFT of all $(k-1)$-dimensional sub-images, $1<k\le d$. Let us focus on the CFT of the one-dimensional image. As the two-dimensional image is a plane and the one-dimensional image is a row, thus, the '0'-dimensional image is a point. The CFT of the '0'-dimensional image can be defined as: if this point is a feature point, then its closest feature point is itself; otherwise, its closest feature point is $\emptyset$ (empty). Similarly, the EDT of the '0'-dimensional image can also be defined as if this point is a feature point, then the distance between this point and its closest feature point is 0; otherwise, the distance is ∞ (a very large integer which is greater than the maximum squared distance of the EDT of this image). These two definitions do not conflict with the common definitions of EDT and CFT. In the CFT of the one-dimensional image, the elements of N' are exactly the feature points in R'. Thus, $N'\in R'$ and $N'=\Gamma$. The bisector of each pair of adjacent points in N' intersects R' at the middle of these two points. Therefore, the one-dimensional CFT can also be computed by CFT_ROW() with the CFT of the '0'-dimensional sub-images.

4. Implementation of PBEDT

In this section, we implement PBEDT by principles introduced before, which computes the d-dimensional EDT from EDT of all $(d-1)$-dimensional sub-images by using independent scan without computing its CFT. In this implementation, we use several optimization specifics to avoid repeated calculations, which reduce the total calculation load.

4.1. The PBEDT algorithm

PBEDT is a recursive procedure and each recursion processes one dimension. Let us explain the inputs of PBEDT(I, d, k). I is the preprocessed original image. For each point p of the original image, if p is a feature point, then p is assigned a value with its squared first dimensional coordinate. Otherwise, it is assigned with ∞. This pretreatment is to make the input image conform to the recursive body. The reason will be explained in the following section. d denotes the amount of the dimensions of this image. k denotes the currently processed dimension.

A LIFO data structure—*stack* is used in PBEDT to store information about verified points. Each element of *stack* consists of three components [x, cx, sd]:

1. The first component is the x-coordinate of the intersection point of the bisector and the current row ($c_{uv}\cdot x$), denoted by $stack\cdot cx$.
2. The second component stores the x-coordinate of the closest feature points of the current dimension, since the point in N' can be located only by the x-coordinate, denoted by $stack\cdot x$.
3. The third component stores the relative squared distance between $N'(x)$ and the left most point of row R', denoted by $stack\cdot sd$. When accessing the top element of *stack*, e.g., $stack\cdot x$ means the second component of the top element of *stack*.

```
PBEDT(I, d, k)
1  if k > 1
2    PBEDT(I, d, k−1)
3  for i_d = 1 → n_d
4    …
5    for i_{k+1} = 1 → n_{k+1}
6      for i_{k−1} = 1 → n_{k−1}
7        …
8          for i_1 = 1 → n_1
9            Initiate stack to empty;
10           for i_k = 1 → n_k
11             sd ⇐ I(i_1,..,i_d);
12             if (sd < ∞)
                 LABEL_A:
13               if (stack not empty)
14                cx ⇐ Intersection-INT(stack.x,i_k,stack.sd,sd);
15                     if (cx = stack.cx)
16                       POP(stack); PUSH(stack, [cx, i_k, sd]);
17                     elseif (cx < stack.cx)
18                       POP(stack); GOTO LABEL_A;
19                     elseif (cx ≤ n_d)
20                       PUSH(stack, [cx, i_k, sd]);
21                   else
22                     PUSH(stack, [−1, i_k, sd]);
23           if (stack is empty)
24             continue;
25           for i_k = n_k → 1
26             if (i_k = stack.cx)
27               POP(stack);
28           I(i_1,..,i_d) ⇐ Distance(i_k,stack.x,stack.sd);
29           if (k < d)
30             I(i_1,..,i_d) ⇐ i_{k+1} × i_{k+1} + I(i_1,..,i_d);
```

The beginning part of PBEDT (line 1 and line 2) controls the recursive process and conducts the program to execute from one-dimensional EDT to d-dimensional EDT. The outer loops of PBEDT (lines 3–8) process each row of the current dimensional direction. When kth dimension is currently processed, there are $n_1\times\cdots\times n_{k-1}\times n_{k+1}\times\cdots\times n_d$ rows in the kth dimensional direction. The inner loop processes the current row (lines 10–22) by using the method introduced in CFT_ROW(). The $(k-1)$-dimensional closest

feature points of the points in the current row are verified along the kth coordinate from low to high (lines 10–22). Invalid points are discarded and the information about valid feature points is stored in *stack*. The intersection points stored in *stack* segment this row, and each segment corresponds to one element of *stack*. The other two components of the *stack* element contain information about computing the distance between the points of this segment and its closest feature point. Therefore, the distance between each point in the current row and its closest feature point can be directly computed by one distance calculation (lines 25–30). In contrast, in this part, Wang and Tan [51] compare adjacent verified points to determine which one is the closest point of current point in the current row, which needs two distance calculations in each step of the loop.

4.2. Details of optimization

The main computational load of PBEDT is coming from Intersection-INT() and Distance(). Next, we will explain the optimizations used in these methods in detail.

4.2.1. Intersection point calculation

In line 14, the x-coordinate of the intersection point of the bisector and the currently processed row is computed by a function *Intersection−INT*(). In Fig. 1, $c_{uv}\cdot x$ can be computed from $\|u-c_{uv}\|=\|v-c_{uv}\|$ (x-coordinate is the row direction, and $\|\cdot\|$ is the Euclidean distance in k-dimensional sub-image).

$$\begin{aligned}
&\|u-c_{uv}\|^2=\|v-c_{uv}\|^2\\
&\Rightarrow \sum_{m\in[1,k]}(v\cdot i_m-c_{uv}\cdot i_m)^2=\sum_{m\in[1,k]}(u\cdot i_m-c_{uv}\cdot i_m)^2\\
&\Rightarrow (v\cdot x-c_{uv}\cdot x)^2+\sum_{m\in[1,k-1]}(v\cdot i_m-c_{uv}\cdot i_m)^2=(u\cdot x-c_{uv}\cdot x)^2\\
&\quad+\sum_{m\in[1,k-1]}(u\cdot i_m-c_{uv}\cdot i_m)^2\\
&\Rightarrow (v\cdot x-c_{uv}\cdot x)^2+\Delta'(v\cdot x)=(u\cdot x-c_{uv}\cdot x)^2+\Delta'(u\cdot x)\\
&\Rightarrow c_{uv}\cdot x=\frac{((v\cdot x)^2+\Delta'(v\cdot x))-((u\cdot x)^2+\Delta'(u\cdot x))}{2(v\cdot x-u\cdot x)}
\end{aligned}\tag{13}$$

Let

$$du=(u\cdot x)^2+\Delta'(u\cdot x).\tag{14}$$

Thus, Eq. (13) can be simplified as

$$c_{uv}\cdot x=\frac{dv-du}{2(v\cdot x-u\cdot x)}.\tag{15}$$

This calculation only uses four arithmetic operations while Maurer et al.'s method uses 11 arithmetic operations to verify one point [2,11,52]. The definition of du in (14) is moving the calculation of x^2 from line 14 to line 30 of previous dimensional computation. Therefore, each x^2 can only be computed one time in line 30 while multiple times in line 14, which will reduce the total amount of computation. This is also why the original image should be preprocessed before it is used by PBEDT.

4.2.2. Integer arithmetic operation

Furthermore, the integer division is used to replace the float division in (15). The integer division abandons the decimal and keeps the integer, which is faster but not always accurate. Therefore, we propose some constraints to keep both the efficiency of integer arithmetic and the accuracy. Fig. 3 shows two different situations, but $c_{uv}\cdot x=c_{vw}\cdot x=4$ in every situation by integer division. In both situations, we assign point $p(4,5)$ close to the leftmost point of this triple—u, since the point coordinates of the image are all integers. Negative values in between two integers will be rounded to the lager integer (-1.9 will be rounded to -1), while positive values in between two integers will be rounded to the smaller integer (5.9 will be rounded to 5). Therefore, we use -1 to label the left edge of the processing row (line 22). Moreover, we prejudge the sign of $c_{uv}\cdot x$ before calculating it, which promote the execution speed. We use -2 as the return value when $dv\le du$, which leaves $c_{uv}\cdot x$ at the left the current row. Therefore, (15) is organized as a Function Intersection-INT(), where $u\cdot x$, $v\cdot x$, du and dv correspond to $stack\cdot x$, i_k, $stack\cdot sd$ and sd in PBEDT(), respectively. Intersection-INT() only uses four integer calculations—one division, one multiplication, and two substractions.

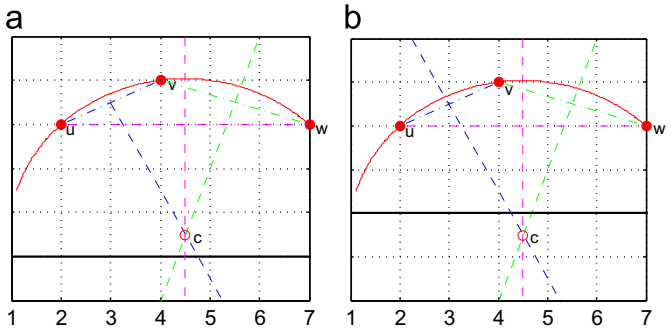

Fig. 3. Different situations of same integer result: (a) $c_{uv}\cdot x>c_{vw}\cdot x$; (b) $c_{uv}\cdot x<c_{vw}\cdot x$.

```
Intersection-INT(u·x, v·x, du, dv)
  If (dv > du)
    return (dv−du)/(2(v·x−u·x));
  else
    return −2;
```

4.2.3. Distance calculation

In line 28, the distance between a point in the currently processed row and its closest feature point of the currently processed dimension is computed by a function Distance(). Eq. (14) can also be used in the distance computation. Given p is a point in the currently processed row and v is the closest feature point of p in the current dimension processed, $\|p-v\|^2$ can be computed by (x-coordinate is the row direction and $\|\cdot\|$ is the Euclidean distance in k-dimensional sub-image):

$$\begin{aligned}
\|p-v\|^2&=\sum_{m\in[1,k]}(p\cdot i_m-v\cdot i_m)^2\\
&=(p\cdot x-v\cdot x)^2+\sum_{m\in[1,k-1]}(p\cdot i_m-v\cdot i_m)^2\\
&=p^2\cdot x-2(p\cdot x)(v\cdot x)+v^2\cdot x+\Delta'(v\cdot x)\\
&=(p\cdot x)(p\cdot x-2v\cdot x)+dv
\end{aligned}\tag{16}$$

This equation can be organized as a function, where $p\cdot x$, $v\cdot x$ and dv correspond to i_k, $stack\cdot x$ and sd in PBEDT(), respectively:

```
Distance (p·x, v·x, dv)
return (p·x)(p·x − 2v·x) +dv;
```

In Distance(), it should be noticed that only the current dimensional coordinates factor into calculation and the distance from lower dimensions need not be computed. This avoids repeated computation and reduces the total computational load. Eqs. (13) and (16) also manifest that the transformation the EDT of k-dimensional image into a Euclidean plane will not affect the distance between points and their closest feature points.

4.3. Computational complexity

The time complexity of PBEDT is $O(dN)$ ($N=n_1\times\cdots\times n_d$), where N is the amount of points in the input image and d is the

amount of dimensions of the input image). The space complexity of PBEDT is $O(N)$, which recycles the memory of the input image in each recursion. The original image is preprocessed before it is used by PBEDT: for each point p of the original image, if p is a feature point, then p is assigned a value with its squared first dimensional coordinate. Otherwise, it is assigned with ∞. The pretreatment scans the image and accesses each point only once, which is very fast. In the strictest sense, the time complexity of these parts is $O(C_1N+C_2dN)$, C_1 for '0'-dimensional EDT and C_2 for PBEDT ($C_1 \ll C_2$, since memory access is much faster than arithmetic operations). Next, we will prove that the time complexity of PBEDT is $O(dN)$.

Theorem 4.1. *The time complexity of PBEDT is $O(dN)$.*

Proof.

1. For a d-dimensional image, PBEDT recursively executes d times.
2. In each recursion to compute k-dimensional EDT from $(k-1)$-dimensional EDT ($1 \le k \le d$), $n_1 \times \cdots \times n_{k-1} \times n_{k+1} \times \cdots \times n_d$ rows are processed one by one. Given the currently processing row R', $|R'| \le n_k$. There are two processes for each row R'.
   The first process (lines 10–22) is a loop, which executes for n_k times. Although there is an inner loop (lines 13–18), the computational complexity of this part is $O(n_k)$. For each point p in R', the distance information between p and its $(k-1)$-dimensional closest feature points has been computed by previous recursions. Known $1 \le i_k \le n_k$, each i_k can only be pushed into *stack* once, and each element in *stack* can be popped no more than once. Thus, neither the PUSH nor the POP operation is executed more than n_k times. Besides, each intersection point calculation is accompanied with no more than twice stack operations (inner loop, lines 13–18). More precisely, let us focus on line 18: the POP operation leads to an additional intersection point calculation. As the POP operation is executed no more than n_k times, therefore, there are no more than $2 \times n_k$ intersection point calculations in the execution of first process. In addition, only the intersection point calculation is concerned with the computational complexity. Accordingly, the first process takes $O(n_k)$ time.
   The second process (lines 25–30) is also a loop and executes for n_k times. In each loop, the distance calculation is executed once. Thus, the second process takes $O(n_k)$ time.
   Therefore, each row of k-dimensional EDT takes $O(n_k)$ time. The total k-dimensional EDT takes $O(n_1 \times \cdots \times n_{k-1} \times n_{k+1} \times \cdots \times n_d \times n_k)$ time, i.e., $O(N)$ time.

Finally, the time complexity of PBEDT is $O(dN)$ time. □

5. Experiments

We conducted two groups of experiments to evaluate the proposed algorithm: on two-dimensional binary images and on three-dimensional binary images. The tests were performed on a computer with an Intel Core2 Duo 2.53 GHz processor, 2 GB RAM, Ubuntu Linux OS with kernel v2.6.31. All algorithms are implemented in ANSI C/C++, and built by GCC v4.1.1.

PBEDT was compared with state-of-the-art EDT algorithms, such as Maurer et al.'s [11], Saito and Toriwaki's [28], Cuisenaire and Macq's [16], Lotufo and Zampirolli's [34], Meijster's [30] and Felzenszwalb et al.'s [31]. In order to improve readability, these algorithms are abbreviated as MAURER2003, SAITO1994, CUISENAIRE, LOTUFOZAMPIROLLI, MEIJSTER, FELZENSZWALB, respectively. The two-dimensional EDT algorithms are implemented by Felzenszwalb et al. (FELZENSZWALB) [53] and Fabbri et al. (other algorithms) [52]. The test images used in two-dimensional experiments are also the same as Fabbri et al. used in [2,52].

Similar to the two-dimensional experiments, we conducted the three-dimensional experiments by principle of Fabbri et al.'s recommendation. By using the principle introduced in the previous section, all two-dimensional independent scan EDT algorithms can be implemented to their multi-dimensional version. Therefore, we implemented the three-dimensional EDT algorithms based on the implementation of [52,53]. We also chose 3-D test images by the principle of Fabbri et al.'s recommendation. Cuisenaire and Macq's algorithm requires three times the propagation in the two-dimensional implementation, which is difficult to be extend to the three-dimensional version. Thus, it was ignored in the three-dimensional binary images experiment.

5.1. Experiments on two-dimensional images

At first, PBEDT was compared with these algorithms on two-dimensional binary images. The performance was measured with images over a wide range of sizes and contents, as Fabbri et al. recommend in [2,52]:

1. Random points (Fig. 4a): The image size is varying from 100×100, 500×500, 1000×1000, 2000×2000, 3000×3000 to 4000×4000 with randomly generated feature points where the number of feature points comprises 1%, 10%, 20%, 30%, 40%, 50%, 60%, 70%, 80%, 90%, and 99% of the image. This test provides an idea of the performance of the algorithms relative to the number of feature pixels [2,11].
2. Random squares (Fig. 4b): The image size is varying from 100×100, 500×500, 1000×1000, 2000×2000, 3000×3000, to 4000×4000. These images are generated by randomly choosing the centers and sizes of black squares rotated by θ at 0°, 15°, 30°, 45°, 60°, 75°, and 90°. The squares are filled and plotted into the image until the black pixels add up to a percentage 15%, 30%, 50%, 70%, and 95%. This test is based on a synthetic image having more similarity to real images with some orientation [2,15].
3. Binary image of real object: The edge image from the Lenna image obtained by thresholding the response of an edge detector is used, as shown in Fig. 4d. Lenna was chosen since it has been universally used as an impartial benchmark for image analysis algorithms [2].
4. Special feature contents:
   - A feature square located at the corner of an image (top-left and bottom-right) (Fig. 4e and f). In this case, the EDT produces the largest and smallest possible distances for a given image size: diagonal and 1, respectively [2,11].
   - A white disk inscribed in the image (Fig. 4g). It is a perfect test for exactness, since the Voronoi diagram of the pixels along a circle is very regular in the continuous plane, however, the discrete Voronoi regions in this case are irregular, especially near the center of the disk [2,28].
   - Half-filled image (Fig. 4h). This is the worst case of the brute force algorithm [2].

In all these tests, PBEDT is faster and more stable than else in most cases. As shown in Fig. 5, PBEDT is faster than MAURER2003, MEIJSTER and FELZENSZWALB at all parameter settings used, slower than CUSENAIRE, SAITO1994 and LOTUFOZAMPIROLLI only at a very large proportion of the feature points. Fig. 6a shows that PBEDT is faster than other algorithms in most cases with a different proportion of randomly generated feature points. Fig. 6b shows that in all the tests with special contents, PBEDT is almost the fastest one, just a little slower than SAITO1994 in the test of the half-filled image. Table 1 shows the average execution time on randomly generated squares with varying feature points

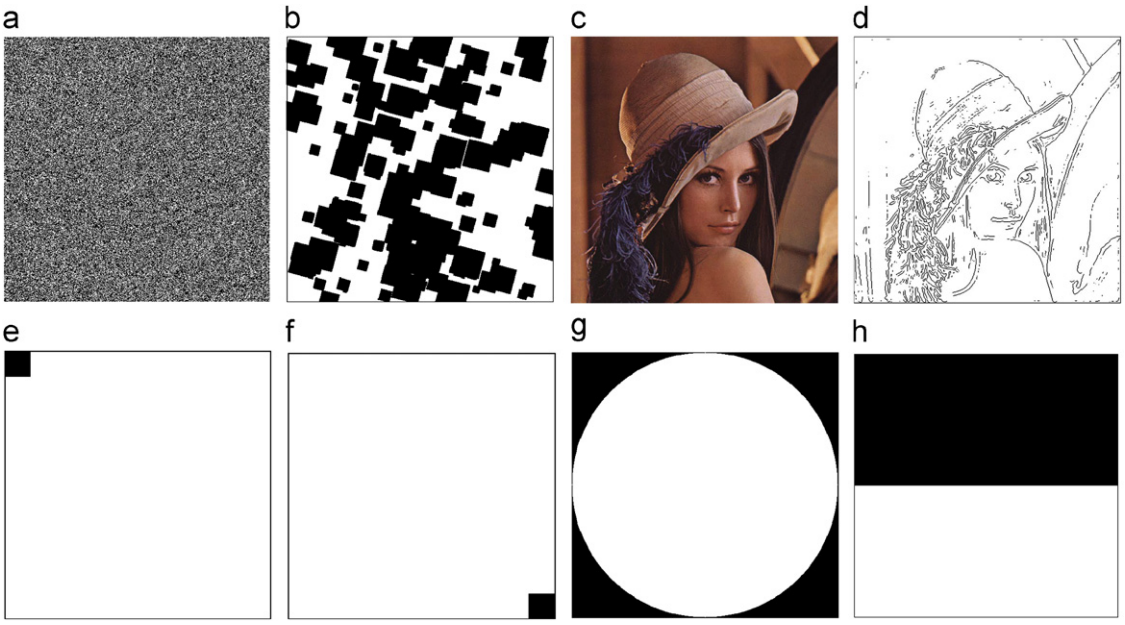

Fig. 4. Two-dimensional binary images used in the experiment, size: 500 × 500. (a) Randomly generated points, feature pixels proportion: 40%; (b) randomly generated squares, feature pixels proportion: 15%, square angle: 15°; (c) original image Lenna; (d) edge image of Lenna; (e) a feature square is located at the top-left corner of an image; (f) a feature square is located at the bottom-right corner of an image; (g) the feature points are located at the out of a disk; and (h) a image is half filled by feature points.

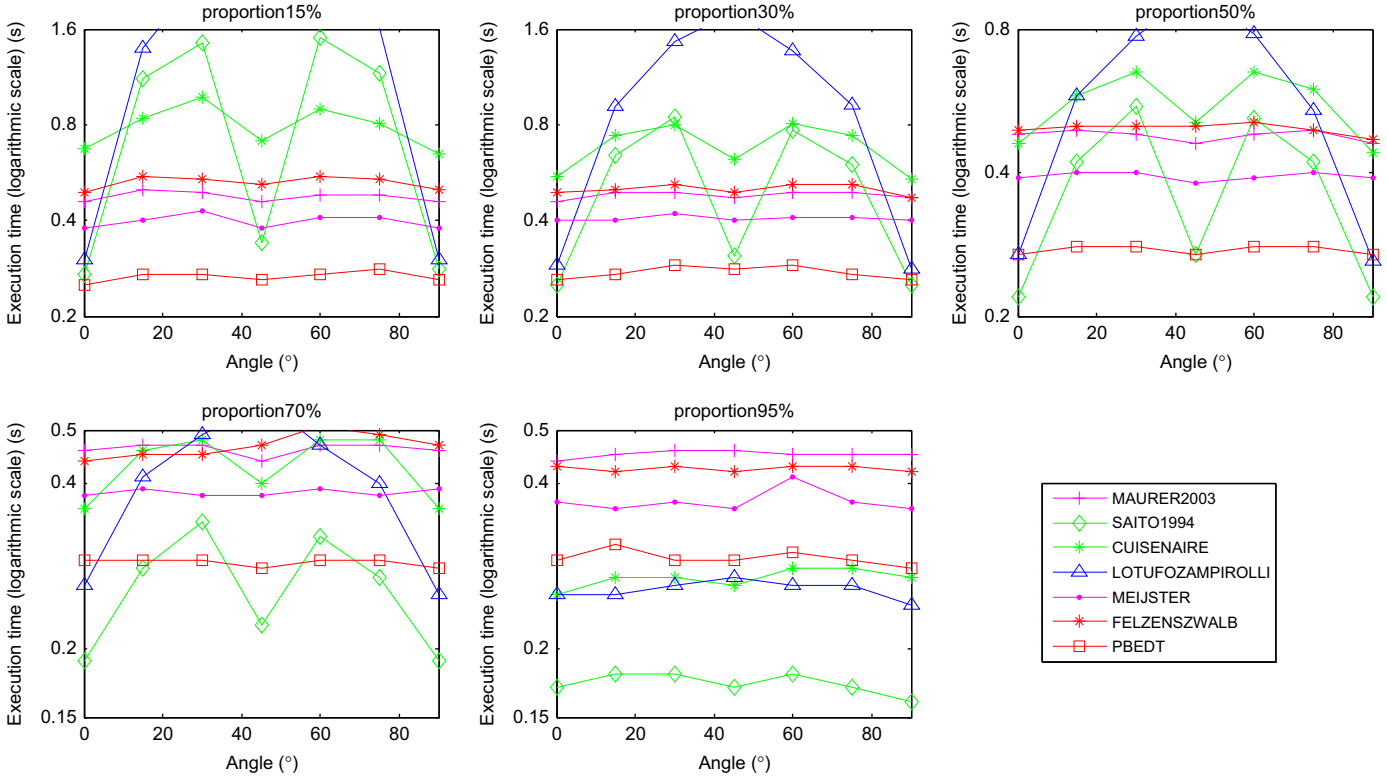

Fig. 5. Execution time for random squares images with varying feature point proportion, size 3000 × 3000, slant angle varied from 0 to 90.

proportion and slant angle at size 3000 × 3000. In Table 1, the advantage of PBEDT is more obvious.

5.2. Experiments on three-dimensional images

Next, PBEDT was compared with these algorithms on three-dimensional binary images. The performance was measured with images over a wide range of sizes and contents, similar to the two-dimensional test:

1. Random points (Fig. 7a): The image size is varying from $128 \times 128 \times 128$, $256 \times 256 \times 256$, to $512 \times 512 \times 512$ with randomly generated feature points where the number of feature points comprises 0.01%, 0.1%, 1%, 10%, 20%, 30%, 40%, 50%, 60%, 70%, 80%, 90%, and 99% of the image.

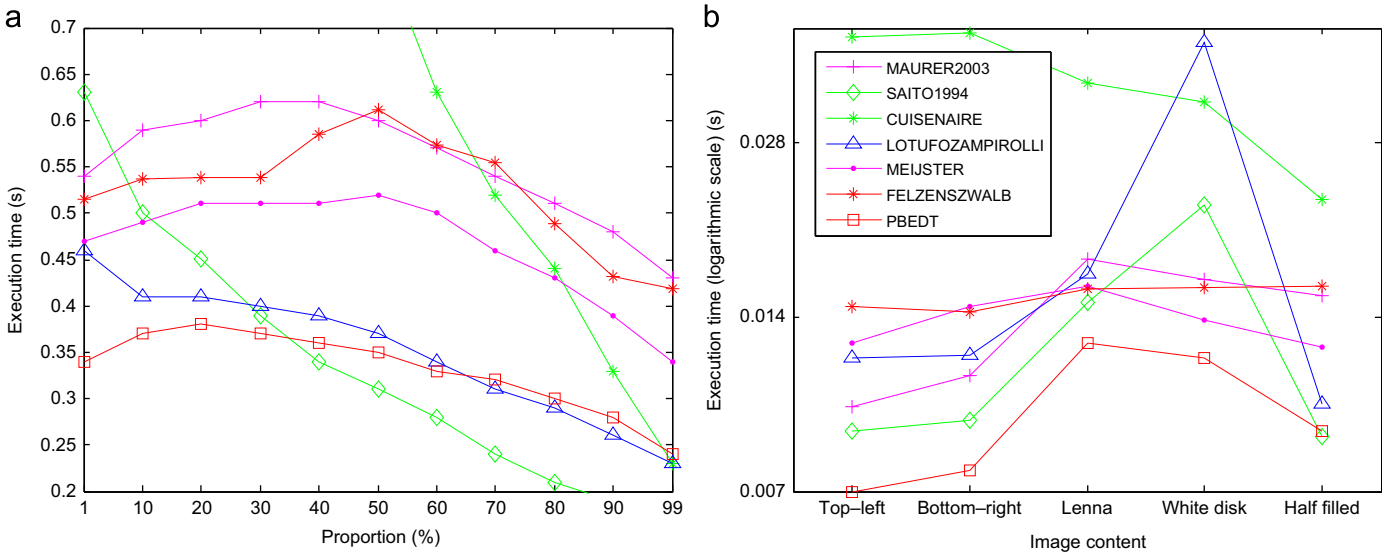

Fig. 6. Test results of two-dimensional images. (a) Random points images with feature points proportion ranged from 1 to 99 , size 3000×3000 and (b) execution time for images of special features, size 500×500.

Table 1
Comparison of average execution time for two-dimensional randomly generated squares images.

Algorithms	Average time(s)	Comparison with PBEDT (%)
MAURER2003	0.469	178.3
SAITO1994	0.441	167.7
CUISENAIRE	0.547	208.0
LOTUFOZAMPIROLLI	0.802	304.9
MEIJSTER	0.391	148.7
FELZENSZWALB	0.483	183.7
PBEDT	0.263	100

2. Random cubes (Fig. 7b): The image size is varying from $128 \times 128 \times 128$, $256 \times 256 \times 256$, to $512 \times 512 \times 512$, randomly choosing the centers and sizes of black squares rotated with every axis of coordinates by θ at 0°, 15°, 30°, 45°, 60°, 75°, and 90°. The cubes are filled and plotted into the image until the black pixels add up to 0.01%, 0.1%, 1%, 10%, 20%, 30%, 40%, 50%, 60%, 70%, 80%, 90%, and 99% of the image.
3. Special feature contents:
   - A real object sample. The scanned point cloud of the male lion status in front of the library of Peking University, as shown in Fig. 7c.
   - A spherical shell is located in a cubic Fig. 7d, which is used to test the exactness.
   - A feature cube located at the top-left-front corner of an image (top-left).

As shown in Fig. 8, PBEDT is faster than Maurer2003_3D, LOTUFOZAMPIROLLI_3D and FELZENSZWALB_3D at all parameter settings used, faster than SAITO1994_3D and MEIJSTER_3D at small proportion feature pixels. SAITO1994_3D is faster than PBEDT when the proportion of feature pixels is higher than 10% with a slant angle is of 0, 75, 90, and the proportion of feature pixels is higher than 30% with a slant angle is of 15, 30, 45, and 60. In the tests of three-dimensional randomly generated cubes, PBEDT is not affected by the orientation of the objects and is significantly faster than others when the proportion of feature pixels is less than 10%. This is meaningful, since in most practically three-dimensional binary images, the feature points are relatively sparse (Fig. 7c). Fig. 7a and b also shows that 1% feature pixels are crowded. Fig. 9 shows that PBEDT is faster than other algorithms in all cases with different proportions of randomly generated feature pixels and with special contents. Table 2 shows the average execution time on randomly generated point images with varying feature point proportion at size $512 \times 512 \times 512$. In Table 2, the advantage of PBEDT is more obvious.

5.3. Discussion

Although the computational complexities of these algorithms are all in linear with image size [2,11,16,28,30,31,34], they show different performances in experiments, since they are based on different fundamental principles. By the principle of the independent scan, all independent scan based multi-dimensional EDT algorithms have a similar program framework. The difference in performance is still reflected by computation in the recursive body, i.e., in how to eliminate invalid feature points in a row. Theoretically, these multi-dimensional EDT algorithms' performances should have shown a similar tendency to their two-dimensional versions'. However, the executional time of an algorithm is also affected by many factors, such as memory access, temporary memory occupancy, data structure, invoke procedure. Usually, these factors are not counted into computational complexity but essentially to the efficiency of algorithm. For example, with the increase in image size, the time for allocating and freeing temporary memory will increase. Besides, with the increase in image dimension, the number of times for allocating and freeing temporary memory will also increase.

Comparing linear time algorithms is extremely difficult, since there is a real danger that we are measuring the specifics of the machine on which the algorithm runs, and the quality of the implementation of the algorithm instead of the algorithm itself. Fabbri et al. provide a convincing baseline to compare state-of-the-art two-dimensional EDT algorithms [2]. They implemented every algorithm to be optimal [2,52], which reduced the effect of implementation. They used a wide variety of binary image contents to comprehensively measure the performance of these algorithms. Therefore, we followed Fabbri et al.'s experimental procedure and compared PBEDT with the same implementations of state-of-the-art EDT algorithms on the same test images.

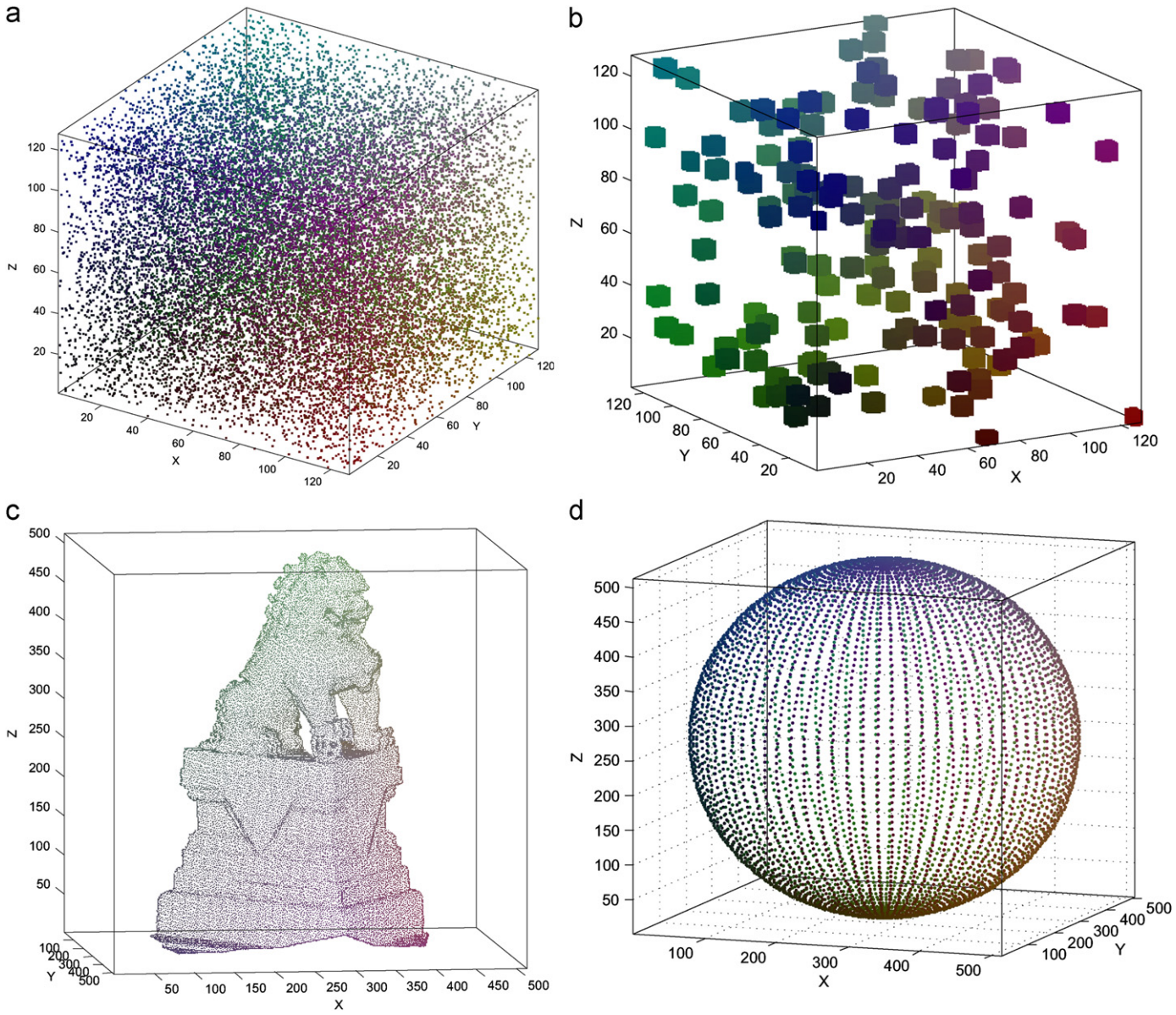

Fig. 7. Sample of three-dimensional test binary images: (a) random points, feature pixels proportion: 1%, size: $128 \times 128 \times 128$; (b) random cubes, feature pixels proportion: 1%, square angle: 0°, size: $128 \times 128 \times 128$; (c) point cloud of lion status, size: $512 \times 512 \times 512$; and (d) a spherical shell is located in a cubic, size: $512 \times 512 \times 512$.

5.3.1. PBEDT vs. MAURER2003

By Fabbri et al.'s analysis, MAURER2003 is the most stable and fastest one in most cases among these algorithms. Thus, we compare PBEDT with MAURER2003 in detail. These two algorithms are both based on the independent scan and have a similar program framework. They both use stack structure to store verified points. The only difference between them is the way in which they eliminate invalid feature points in a row:

- MAURER2003 is based on partial Voronoi diagram intersection, which uses three adjacent candidate feature points to determine the validity of the central point [11]. For example, in Fig. 2c and d, if the intersection point of B_{uv} and B_{vw} is under the currently processed row, then v is kept. Otherwise, v is abandoned. To avoid directly computing the intersection point, they use a determination:

$$c \cdot v \cdot \bar{y}^2 - b \cdot u \cdot \bar{y}^2 - a \cdot w \cdot \bar{y}^2 - a \cdot b \cdot c > 0, \tag{17}$$

where $a = v \cdot x - u \cdot x$, $b = w \cdot x - v \cdot x$ and $a = w \cdot x - u \cdot x = a + b \cdot u \cdot \bar{y}$, $v \cdot \bar{y}$, $w \cdot \bar{y}$ are distanced from u, v, w to the currently processed row, which are computed by the previous recursion. This determination takes 11 arithmetic operations [2,11,52].
- PBEDT computes the intersection point of B_{uv} (B_{vw}) and the currently processed row and compares the intersection points to determine whether v should be kept. Intersection-INT() only takes four arithmetic operations to compute the intersection point. It also takes advantage of integer arithmetic to avoid time consuming float operations. At the same time, we also indicate that the integer arithmetic has the same precision as float operations. Besides, because the intersection points are computed, the distance between each point in the currently processed row and its closest feature point can be directly computed by one distance calculation. In contrast, in this part, MAURER2003 compares adjacent verified points to determine which one is the closest point to the current point in the currently processed row, which needs two distance calculations in each step of the loop [11,52].

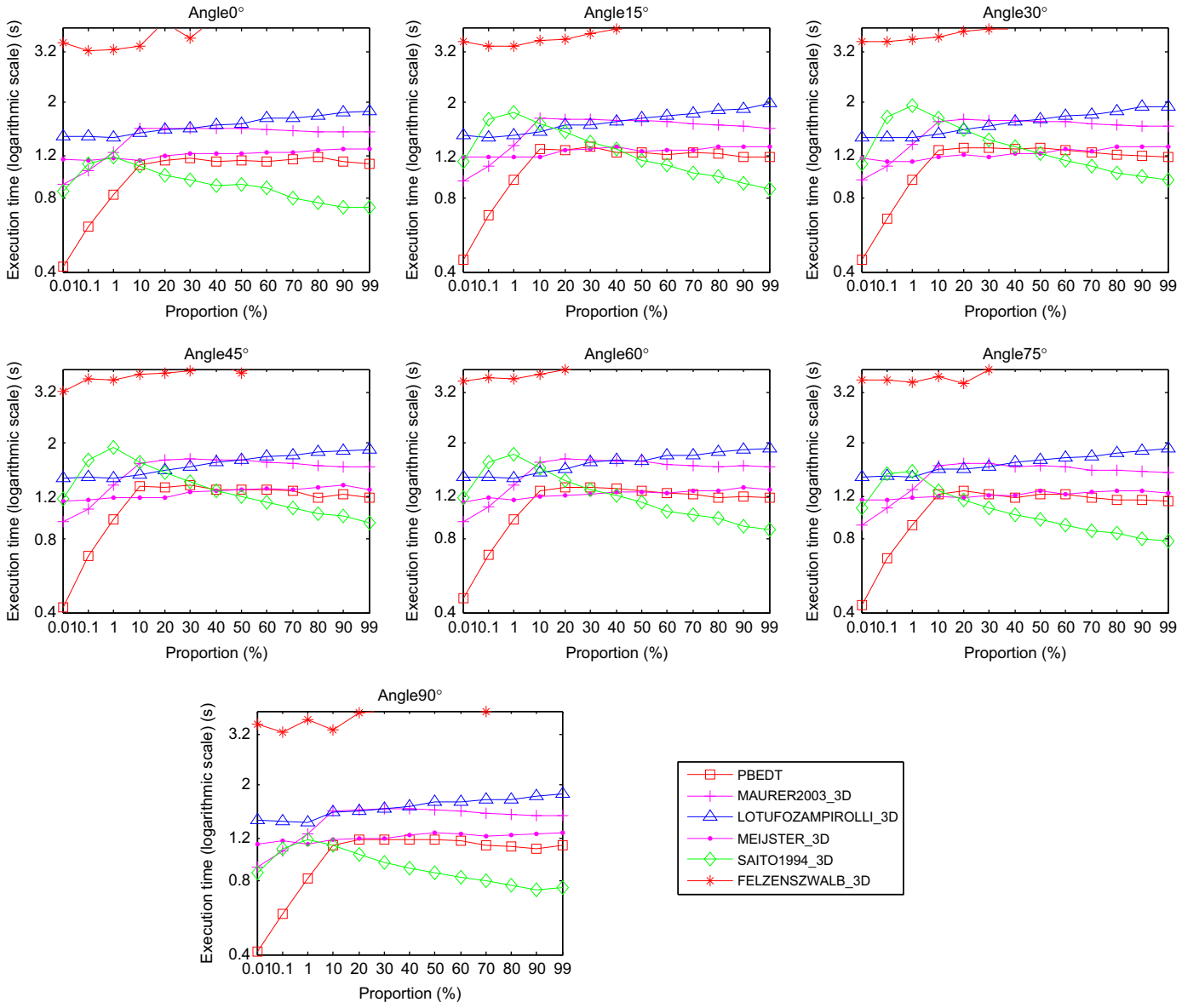

Fig. 8. Execution time for random cube images with varying feature point proportion, size: $256 \times 256 \times 256$, slant angle varied from 0 to 90.

Therefore, PBEDT uses fewer calculations in the kernel part than MAURER2003. The theoretical analysis and the experimental results all support that the computational complexity of PBEDT has a smaller coefficient than MAURER2003. However, because they have the common framework, the executional performance of PBEDT surpassing of MAURER2003 is not as significant as the kernel part.

5.3.2. *PBEDT vs. others*

Next, we briefly analyze other algorithms. CUSENAIRE is an ordered propagation method: a fast but approximate distance transform is first computed by using a coarse neighborhood; a sequence of larger neighborhoods is then used to refine these approximations [16]. Cuisenaire et al. state that their algorithm apparently remains $O(N)$ even in the worst case, based on experiments in [12,16]. However, argued by Fabbri et al., Cuisenaire et al.'s conjecture has not been proved and the pre-computation of distance error boundary is time-consuming even for moderate distances [2]. Our experiments also indicate that its performance is poor at lower feature point percentages while fairly good at higher feature point percentages, since the smaller neighborhood is used in dense circumstances while the bigger neighborhood is used in sparse circumstances.

Other algorithms are all independent scan methods. The independent scan algorithms have similar framework, and the difference among these algorithms is how to process in a row. LOTUFOZAMPIROLLI uses mathematical morphological approach [34]. It simulates the operation of morphological dilation, which recursively computes the correct distance to the nearest feature point for each point in the row. The main problem of this algorithm is that it cannot avoid repetitious computation of distance. SAITO1994, MEIJSTER and FELZENSZWALB all use the simulation of parabola intersection [28,30,31]. Let us take MEIJSTER as an example. The most complex part in the row-processing is the computation of the intersection point of adjacent parabolas, which takes eight arithmetic operations [2,52]. However, it also has the problem of repetitious computation of distance.

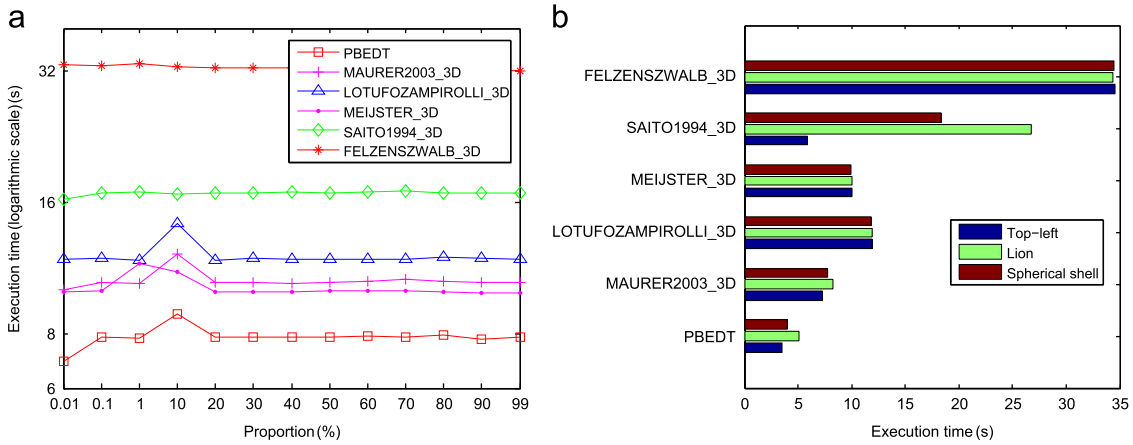

Fig. 9. 3-D images tests. (a) Execution time for random points images with feature points proportion ranged from 1 to 99, size: $512 \times 512 \times 512$ and (b) execution time with images of feature cube at top-left-front corner, point cloud of lion status and spherical shell, size: $512 \times 512 \times 512$.

Table 2
Comparison of average execution time for three-dimensional randomly generated points images.

Algorithms	Average time(s)	Comparison with PBEDT (%)
MAURER2003_3D	10.59	134.8
SAITO1994_3D	16.78	213.6
LOTUFOZAMPIROLLI_3D	12.05	153.4
MEIJSTER_3D	10.19	129.7
FELZENSZWALB_3D	32.53	370.8
PBEDT	7.85	100

6. Conclusion

In this paper, we proposed an efficient EDT algorithm of a binary image in arbitrary dimensions, named PBEDT. The time complexity of the PBEDT algorithm is linear with the amount of both image points and dimensions with a small coefficient ($O(dN)$). Compared with the state-of-the-art EDT algorithms, the PBEDT algorithm shows much better performance on all of the two- and three-dimensional images with variant image contents. The PBEDT is much faster and more stable than current algorithms in most cases. In addition, by comparing the output of PBEDT with that of other algorithms, no difference is observed in our tests. Additional tests for thousands of randomly generated images varying in image width and amount of feature points again demonstrated the correctness and efficiency of the PBEDT algorithm. As an independent scan method, the PBEDT can be easily implemented to parallel version for a fast computation.

References

[1] A. Rosenfeld, J.L. Pfaltz, Sequential operations in digital picture processing, Journal of the ACM 13 (1966) 471–494.
[2] R. Fabbri, L. da Fontoura Costa, J.C. Torelli, O.M. Bruno, 2d Euclidean distance transform algorithms: a comparative survey, ACM Computing Surveys 40 (1) (2008).
[3] C. Arcelli, G. Sanniti di Baja, L. Serino, Distance-driven skeletonization in voxel images, IEEE Transactions on Pattern Analysis and Machine Intelligence 33 (4) (2011) 709–720.
[4] W.H. Hesselink, J.B.T.M. Roerdink, Euclidean skeletons of digital image and volume data in linear time by the integer medial axis transform, IEEE Transactions on Pattern Analysis and Machine Intelligence 30 (12) (2008) 2204–2217.
[5] S. Loncaric, A survey of shape analysis techniques, Pattern Recognition 31 (8) (1998) 983–1001.
[6] A. Peter, A. Rangarajan, Information geometry for landmark shape analysis: unifying shape representation and deformation, IEEE Transactions on Pattern Analysis and Machine Intelligence 31 (2) (2009) 337–350.
[7] L. Lam, S.-W. Lee, C. Suen, Thinning methodologies—a comprehensive survey, IEEE Transactions on Pattern Analysis and Machine Intelligence 14 (9) (1992) 869–885.
[8] M. Jones, J. Baerentzen, M. Sramek, 3d distance fields: a survey of techniques and applications, IEEE Transactions on Visualization and Computer Graphics 12 (4) (2006) 581–599.
[9] B. Zitov, J. Flusser, Image registration methods: a survey, Image and Vision Computing 21 (11) (2003) 977–1000.
[10] H. Breu, J. Gil, D. Kirkpatrick, M. Werman, Linear time Euclidean distance transform algorithms, IEEE Transactions on Pattern Analysis and Machine Intelligence 17 (5) (1995) 529–533.
[11] J. Maurer, R. Qi, V. Raghavan, A linear time algorithm for computing exact Euclidean distance transforms of binary images in arbitrary dimensions, IEEE Transactions on Pattern Analysis and Machine Intelligence 25 (2) (2003) 265–270.
[12] O. Cuisenaire, Distance Transformations: Fast Algorithms and Applications to Medical Image Processing, Ph.D. Dissertation, Louvain-la-Neuve, Belgium, 1999.
[13] B.J. Verwer, P.W. Verbeek, S.T. Dekker, An efficient uniform cost algorithm applied to distance transforms, IEEE Transactions on Pattern Analysis and Machine Intelligence 11 (4) (1989) 425–429.
[14] I. Ragnemalm, Neighborhoods for distance transformations using ordered propagation, CVGIP: Image Understanding 56 (3) (1992) 399–409.
[15] H. Eggers, Two fast Euclidean distance transformations in z^2 based on sufficient propagation, Computer Vision and Image Understanding 69 (1) (1998) 106–116.
[16] O. Cuisenaire, B.M. Macq, Fast Euclidean distance transformation by propagation using multiple neighborhoods, Computer Vision and Image Understanding 76 (2) (1999) 163–172.
[17] A.X. Falcao, J. Stolfi, S. de Alencar, The image foresting transform: intelligence theory, algorithms, and applications, IEEE Transactions on Pattern Analysis and Machine 26 (1) (2004) 19–29.
[18] J. Piper, E. Granum, Computing distance transformations in convex and non-convex domains, Pattern Recognition 20 (November) (1987) 599–615.
[19] P.-E. Danielsson, Euclidean distance mapping, Computer Vision, Graphics, and Image Processing 14 (1980) 227–248.
[20] G. Borgefors, Distance transformations in arbitrary dimensions, Computer Vision, Graphics, and Image Processing 27 (5) (1984) 321–345.
[21] I. Ragnemalm, The Euclidean distance transform in arbitrary dimensions, Pattern Recognition Letters 14 (11) (1993) 883–888.
[22] F.Y. Shih, Y.-T. Wu, Fast Euclidean distance transformation in two scans using a 3×3 neighborhood, Computer Vision and Image Understanding 93 (2) (2004) 195–205.
[23] F.Y.-C. Shih, Y.-T. Wu, The efficient algorithms for achieving Euclidean distance transformation, IEEE Transactions on Image Processing 13 (8) (2004) 1078–1091.
[24] D.W. Paglieroni, Distance transforms: properties and machine vision applications, CVGIP: Graphical Model and Image Processing 54 (1) (1992) 56–74.
[25] D.W. Paglieroni, A unified distance transform algorithm and architecture, Machine Vision and Applications 5 (1) (1992) 47–55.
[26] M.N. Kolountzakis, K.N. Kutulakos, Fast computation of the euclidian distance maps for binary images, Information Processing Letters 43 (4) (1992) 181–184.
[27] L. Chen, H.Y.H. Chuang, A fast algorithm for Euclidean distance maps of a 2-d binary image, Information Processing Letters 51 (1) (1994) 25–29.
[28] T. Saito, J. Ichiro Toriwaki, New algorithms for Euclidean distance transformation of an n-dimensional digitized picture with applications, Pattern Recognition 27 (11) (1994) 1551–1565.

[29] T. Hirata, A unified linear-time algorithm for computing distance maps, Information Processing Letters 58 (3) (1996) 129–133.
[30] A. Meijster, J.B.T.M. Roerdink, W.H. Hesselink, A general algorithm for computing distance transforms in linear time, Computational Imaging and Vision 18 (8) (2000) 331–340.
[31] P.F. Felzenszwalb, D.P. Huttenlocher, Distance Transforms of Sampled Functions, Cornell Computing and Information Science, Tech. Rep., September 2004.
[32] F.Y.-C. Shih, O.R. Mitchell, A mathematical morphology approach to Euclidean distance transformation, IEEE Transactions on Image Processing 1 (2) (1992) 197–204.
[33] C. Huang, O. Mitchell, A Euclidean distance transform using grayscale morphology decomposition, IEEE Transactions on Pattern Analysis and Machine Intelligence 16 (4) (1994) 443–448.
[34] R. de Alencar Lotufo, F.A. Zampirolli, Fast multidimensional parallel Euclidean distance transform based on mathematical morphology, in: SIBGRAPI, IEEE Computer Society, 2001, pp. 100–105.
[35] F. de Assis Zampirolli, R. de Alencar Lotufo, Classification of the distance transformation algorithms under the mathematical morphology approach, in: SIBGRAPI, IEEE Computer Society, 2000, pp. 292–299.
[36] D.F. Watson, Computing the *n*-dimensional Delaunay tessellation with application to Voronoi polytopes, Computer Journal 24 (2) (1981) 167–172.
[37] F. Aurenhammer, Voronoi diagrams—a survey of a fundamental geometric data structure, ACM Computing Surveys 23 (3) (1991) 345–405.
[38] S. Fortune, A sweepline algorithm for Voronoi diagrams, Algorithmica 2 (1987) 153–174.
[39] R.L. Ogniewicz, O. Kübler, Voronoi tessellation of points with integer coordinates: time-efficient implementation and online edge-list generation, Pattern Recognition 28 (12) (1995) 1839–1844.
[40] W. Guan, S. Ma, A list-processing approach to compute Voronoi diagrams and the Euclidean distance transform, IEEE Transactions on Pattern Analysis and Machine Intelligence 20 (7) (1998) 757–761.
[41] M.L. Gavrilova, M.H. Alsuwaiyel, Two algorithms for computing the Euclidean distance transform, International Journal of Image and Graphics (2001) 635–645.
[42] O. Bruno, L. Costa, A parallel implementation of exact Euclidean distance transform based on exact dilations, Microprocessors and Microsystems 28 (3) (2004) 107–113.
[43] J. Torelli, R. Fabbri, G. Travieso, O. Bruno, A high performance 3d exact Euclidean distance transform algorithm for distributed computing, International Journal of Pattern Recognition and Artificial Intelligence 24 (06) (2010) 897–915.
[44] C. Sigg, R. Peikert, M. Gross, Signed distance transform using graphics hardware, in: Visualization, 2003, VIS 2003, IEEE, October 2003, pp. 83–90.
[45] T. Cao, K. Tang, A. Mohamed, T. Tan, Parallel banding algorithm to compute exact distance transform with the GPU, in: Proceedings of the 2010 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games, ACM, 2010, pp. 83–90.
[46] F.Y. Shih, Y.-T. Wu, Three-dimensional Euclidean distance transformation and its application to shortest path planning, Pattern Recognition 37 (1) (2004) 79–92.
[47] R. Strand, G. Borgefors, Distance transforms for three-dimensional grids with non-cubic voxels, Computer Vision and Image Understanding 100 (3) (2005) 294–311.
[48] G. Borgefors, On digital distance transforms in three dimensions, Computer Vision and Image Understanding 64 (3) (1996) 368–376.
[49] S. Svensson, G. Borgefors, Distance transforms in 3d using four different weights, Pattern Recognition Letters 23 (12) (2002) 1407–1418.
[50] G. Borgefors, Weighted digital distance transforms in four dimensions, Discrete Applied Mathematics 125 (1) (2003) 161–176.
[51] J. Wang, Y. Tan, Efficient Euclidean distance transform using perpendicular bisector segmentation, in: IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2011, Proceedings CVPR '2011, June 2011.
[52] R. Fabbri, L. da Fontoura Costa, J.C. Torelli, O.M. Bruno, Complete Results of the Benchmark Between Exact EDT Algorithms, ⟨http://distance.sourceforge.net⟩, 2006.
[53] P.F. Felzenszwalb, D.P. Huttenlocher, Distance Transforms of Sampled Functions (program), ⟨http://people.cs.uchicago.edu/pff/dt/⟩, 2004.

Jun Wang received the B.S. degree in Mechanical Engineering from Yantai University, Yantai, China, in 1996, the M.S. degree in Computer Science from Shandong University, Jinan, China, in 2006, and the Ph.D. degree in Computer Science from the Key Laboratory of Machine Perception (Ministry of Education) and the Department of Machine Intelligence, School of Electronics Engineering and Computer Science, Peking University, Beijing, China, in 2012. He is currently a scientist of Shandong Provincial Office, State Administration of Taxation, Jinan, China. His current research interests include data mining, image processing, computer vision and genetic programming.

Ying Tan (M'98-SM'02) received the B.S. degree in Electronic Engineering from the Electronic Engineering Institute, Hefei, China, in 1985, the M.S. degree in Electronic Engineering from Xidian University, Xi'an, China, in 1988, and the Ph.D. degree in Signal and Information Processing from Southeast University, Nanjing, China, in 1997. In 1997, he became a Postdoctoral Research Fellow and then an Associate Professor with the Department of Electronic Engineering and Information Science, University of Science and Technology of China, Hefei. He was a Full Professor, an advisor of Ph.D. candidates, and the Director of the Institute of Intelligent Information Science, University of Science and Technology of China. He was with the Chinese University of Hong Kong, Shatin, Hong Kong, in 1999 and during 2004–2005. He was an electee of the 100-Talent Program of the Chinese Academy of Sciences, Beijing, China, in 2005. He is currently a Full Professor and an advisor of Ph.D. candidates with the Key Laboratory of Machine Perception (Ministry of Education) and the Department of Machine Intelligence, School of Electronics Engineering and Computer Science, Peking University, Beijing. He is also the Head of the Computational Intelligence Laboratory, Peking University. He has authored or coauthored more than 200 academic papers in refereed journals and conferences and several books and book chapters. His current research interests include computational intelligence, swarm intelligence, artificial immune systems, intelligent information processing, pattern recognition, bioinformatics, statistical learning theory, and their applications.

Tan was the Program Committee Chair for the 2008 International Symposium on Neural Networks and the General Chair for the 2010 International Conference on Swarm Intelligence (ICSI). He is the General Chair of ICSI 2011. He is an Associate Editor of IEEE Transactions on Systems, Man and Cybernetics, Part B: Cybernetics and an Associate Editor of the International Journal of Swarm Intelligence Research and the IES Journal B: Intelligent Devices and Systems. He is a member of the Advisory Board of the International Journal of Knowledge-Based and Intelligent Engineering Systems and of the Editorial Board of the Journal of Computer Science and Systems Biology and Applied Mathematical and Computational Sciences. He is also the Editor of Springer Lecture Notes on Computer Science, LNCS 5263, 5264, 6145, and 6146, and the Guest Editor of several referred journals, including Information Science, Softcomputing, Neurocomputing, and the International Journal of Artificial Intelligence. He was the recipient of a number of academic and research achievement awards from his country and universities due to his outstanding contributions and distinguished works, including the 2009 National Natural Science Prize of China.