# Efficient Euclidean Distance Transform Using Perpendicular Bisector Segmentation [*†]

Jun Wang and Ying Tan [‡]
Key Laboratory of Machine Perception (Ministry of Education), Peking University
Department of Machine Intelligence, School of EECS, Peking University, Beijing, P.R. China
bedouins@pku.edu.cn ytan@pku.edu.cn

## Abstract

*In this paper, we propose an efficient algorithm for computing the Euclidean distance transform of two-dimensional binary image, called PBEDT (Perpendicular Bisector Euclidean Distance Transform). PBEDT is a two-stage independent scan algorithm. In the first stage, PBEDT computes the distance from each point to its closest feature point in the same column using one time column-wise scan. In the second stage, PBEDT computes the distance transform for each point by row with intermediate results of the previous stage. By using the geometric properties of the perpendicular bisector, PBEDT directly computes the segmentation by feature points for each row and each segment corresponding to one feature point. Furthermore, by using integer arithmetic to avoid time consuming float operations, PBEDT still achieves exact results. All these methods reduce the computational complexity significantly. Consequently, an efficient and exact linear time Euclidean distance transform algorithm is implemented. Detailed comparison with state-of-the-art linear time Euclidean distance transform algorithms shows that PBEDT is the fastest on most cases, and also the most stable one with respect to image contents.*

## 1. Introduction

Given a binary image, whose elements have only the value 0 — background (feature) pixels — and 1 — feature (background) pixels, its distance transform computes the distance for each pixel between that pixel and the feature pixel closest to it [21]. Distance transform (DT) algorithms are excellent tools for a variety of applications, such as image processing, computer vision, pattern recognition, morphological filtering and robotics, etc. [9][19][21]. In practice, several distance metrics, such as the city-block ($L_1$), the chessboard ($L_\infty$), the octagonal and the Euclidean metric, are all used for different situations. One of the most natural and appropriate one of these is the Euclidean metric, which is radially symmetric and virtually invariant to rotation [1][9][16], and used in many applications. However, the $DT$ in exact Euclidean metric, called EDT, is time consuming.

Intuitionally, treated as a global operation, EDT can be computed by using an exhaustively brute-force searching algorithm: for each pixel of the image, calculate the distance between that pixel and each feature pixel. This requires $O(N^2)$ time ($N$ is the number of image pixels) [1][16]. Numerous algorithms have been proposed to restrict the searching for the closest feature pixel in order to realize a fast EDT computation. In terms of searching mode, all these algorithms can be roughly classified into three categories, ordered propagation[2][20], raster scan [5][20][21] and independent scan [1][6][12][13][15][16][22]. However, so far there is no algorithm which can compute the EDT with good efficiency and good precision [9]. Some extensive surveys are introduced in [2] and [9]. They indicate that an efficient EDT should be based on obtaining feature pixels information from limited region, and should avoid global searching. Although many of these algorithms are of liner time complexity, some are not stable when the image content is changed or have a large constant term [2][9].

In this paper, we propose an efficient algorithm for computing the Euclidean distance transform of two-dimensional binary image, called PBEDT (Perpendicular Bisector Euclidean Distance Transform). PBEDT is a two-stage independent scan algorithm. In the first stage, PBEDT computes the distance from each point to its closest feature point in the same column using one time column-wise scan. In the second stage, PBEDT computes the distance transform for

*This work is supported by National Natural Science Foundation of China (NSFC), under grant number 60875080 and 60673020, and partly supported by the National High Technology Research and Development Program of China (863 Program), with grant number 2007AA01Z453.

†The source code of PBEDT is available on http://cil.pku.edu.cn/algorithm/distancetransform/

‡Ying Tan is the corresponding author.

each point by row with intermediate results of the previous stage. By using the geometric properties of the perpendicular bisector, PBEDT directly computes the segmentation by feature points for each row and each segment corresponding to one feature point. Furthermore, by using integer arithmetic to avoid time consuming float operations, PBEDT still achieves exact results. The remainder of this paper is organized as follows. In Section 2, the detail of PBEDT is introduced. The refined implementation of the proposed algorithm is presented in Section 3. Experiments of comparison with state-of-the-art algorithms on variant feature objects and in-depth discussion are reported in Section 4. Finally, Section 5 gives the concluding remarks of this paper.

## 2. PBEDT by perpendicular bisector segmentation

### 2.1. Preliminary

As usual, this algorithm takes a n by m binary image as the input and outputs a distance transform, usually in squared distance. Let $I$ denote the point set of the image, $I_r$ denote the points in row $r$, and $F_I$ denote the set of feature pixels. (Generally, let uppercase letter denote point set and lower case letter denote point.) Let $f(u)$ denote the closest feature point of $u$, and $\| \cdot \|$ is the Euclidean metric:

$$f(u) = \underset{v \in F_I}{\arg\min} \|u - v\|, u \in I. \tag{1}$$

Thus, the distance transform of image $I$ can be computed by:

$$DT(u) = \|u - f(u)\|^2, u \in I \tag{2}$$

The feature points in the column $c$ are denoted by :

$$C_c = \{v | v.x = c, v \in F_I\}, 0 \le c < n \tag{3}$$

PBEDT is a two-stage independent scan algorithm. In the first stage, it computes the closest feature points for each row. Then in the second stage, it computes the distance of each point to its closest feature point by row [1][16].

### 2.2. First stage

For row $r$, given two feature points $u, v \in C_c$, if $\|u.y - r\| < \|v.y - r\|$, then any point in row $r$ is closer to $u$ than to v. If $\exists u \; \forall v \in C_c$, $\|u.y - r\| \le \|v.y - r\|$, $u$ is called r's closest feature point in column c. All closest feature points in columns of row $r$ are denoted by $S_g^r$, and $|S_g^r| \le n$. Therefore, $f(u)$ is rewriten as:

$$f(u) = \underset{v \in S_g^r}{\arg\min} \|u - v\|, u \in I_r. \tag{4}$$

### 2.3. Second stage

In this stage, we give an effective method to compute (4). Let $S_f^r$ denote the closest feature points of row $r$, and $S_f^r \subseteq S_g^r$.

$$S_f^r = \bigcup_{u \in I_r} f(u) \tag{5}$$

Let points in $S_g^r$ and $S_f^r$ are increasingly ordered by x coordinate. Let $P_t^r$ denote the set of points in row $r$ whose closest feature point is $t$, called $t$'s region of influence in row $r$.

$$P_t^r = \{v | f(v) = t, v \in I_r\}, t \in S_g^r \tag{6}$$

If $t \in S_f^r$, then $P_t^r \neq \emptyset$; otherwise, $P_t^r = \emptyset$. Hence, $S_f^r = S_g^r - \{u | P_u^r = \emptyset, u \in S_g^r\}$.

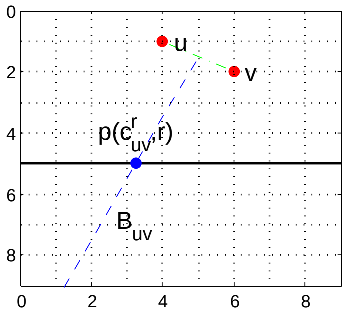


Figure 1. The perpendicular bisector of $u$ and $v$ intersects row r.

We take advantage of the geometrical property of perpendicular bisector to verify the points whose region of influence is empty. Let $B_{uv}$ denote the perpendicular bisector of $u$ and $v$ (Fig.1). $B_{uv}$ is the set of points which have equal distance to $u$ and $v$. Consequently, the points located at the $u$ side of $B_{uv}$ are closer to $u$ than to $v$. Let $c_{uv}^r$ denote the x coordinate of the intersection of $B_{uv}$ with row $r$. Hence, point $p(c_{uv}^r, r)$ (Fig.1) is the separation point of $P_u^r$ and $P_v^r$ ($P_u^r \neq \emptyset, P_v^r \neq \emptyset, u.x < v.x$), and we assign $p \in P_u^r, p \notin P_v^r$. Given $u.x < v.x$, the points in $I_r$ at the left of $c_{uv}^r$ are closer to $u$ than to $v$. The situations in Fig.2(a)(b) clearly indicate a point's region of influence in row $r$ is empty.

If $c_{uv}^r > c_{vw}^r (u, v, w \in S_g^r)$, then $P_v^r = \emptyset$ (Fig.2(d)). Next, $u$ and $w$ will be compared with other points in $S_g^r$. If $c_{uv}^r < c_{vw}^r (u, v, w \in S_g^r)$, then $P_v^r \neq \emptyset$ (Fig.2(c)). However, point $v$ will not be compared with all the points in $S_g^r$, since the region of influence has some special properties:

**Property 1.** *The points in set $P_t^r$ are continuous in x-coordinate, or, $\nexists v\{v | u.x \le v.x \le w.x, u, w \in P_t^r, v \notin P_t^r\}$. (If $v$ exists, then $s = f(v)$. Thus, $B_{st}$ intersects row $r$ between u and v, and between v and w, too. This violates the fact that two lines only intersects at most once.)*

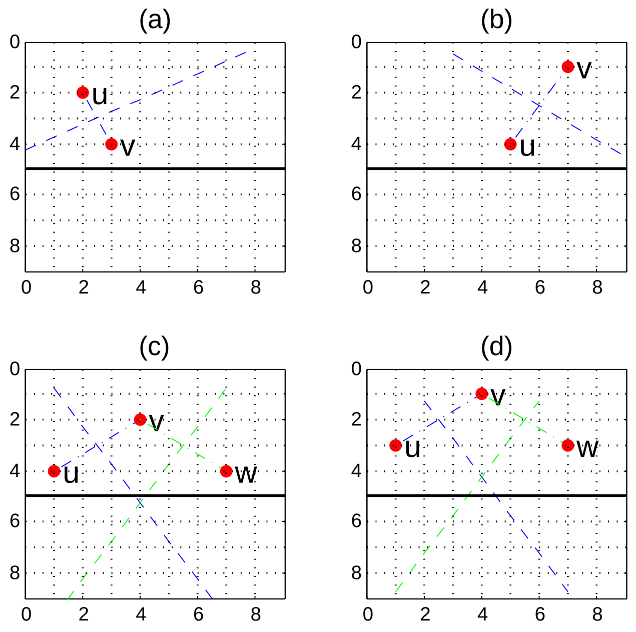


Figure 2. Four situations when a point is added in $S_1$. (a) $P^r_u = \emptyset$; (b) $P^r_v = \emptyset$; (c) $P^r_u \neq \emptyset$; (d) $P^r_v = \emptyset$;

Moreover,

**Property 2.** *If* $u, v \in S^r_f$, $u.x < v.x$, *then* $s.x \leq t.x$ , $\forall s \in P^r_u$ *and* $\forall t \in P^r_v$.

Therefore, if the points in $S^r_g$ at the left of point $v$ are all verified, then none of them is closer to the right of $c^r_{uv}$ than $v$ (Property.1, Property.2).

```
Compute$S_1$
    $S_1$ and $S_c$ are stacks. Initially, $S_1$ and $S_c$ are both empty.
    while $S_0$ is not empty
        withdraw the lowest point $w$ from $S_0$;
        if $S_1$ is empty
            push($S_1$, w); push($S_c$, 0);
        else
            $v$ is top of $S_1$ and $c^r_{uv}$ is top of $S_c$;
            calculate $c^r_{vw}$ with $v$ and w;
            $\begin{cases} pop(S_1),\ pop(S_c), add\ w\ back\ to\ the\\ front\ of\ S_0,\ \ when\ c^r_{vw} < c^r_{uv};\\ \\ pop(S_1),\ pop(S_c), push(S_1, w),\\ push(S_c, c^r_{vw}),\ \ when\ c^r_{vw} = c^r_{uv};\\ \\ push(S_1, w),\ push(S_c, c^r_{vw}),\\ when\ c^r_{vw} > c^r_{uv}\ and\ c^r_{vw} < cols; \end{cases}$
    return $S_1$;
```

Consequently, we give an algorithm COMPUTE$S_1$ to obtain $S^r_f$ from $S^r_g$. Let $S_1$ and $S_c$ be stack structures. Initially, let $S_0 = S^r_g$, and $S_1 = \emptyset$. Points are moved from $S_0$ to $S_1$ one by one. If an added point results in point $p$ of $S_1$ whose region of influence in row $r$ is empty, then $p$ should be removed from $S_1$. When $S_1$ is empty, we push 0 into $S_c$ as the first intersection point. In the next loop, we use the newly added point $w$ and the stack top of $S_1$ point $v$ to compute $c^r_{vw}$, and compare $c^r_{vw}$ to the stack top of $S_c$. The 0 at the bottom of stack labels the edge of the processing row.

Next, we prove $S_1 = S^r_f$.

**Theorem 1.** $S_1 = S^r_f$

*Proof:* This proof has two steps:

1. $P^r_v = \emptyset, \forall v \in S^r_g - S_1$;

   In the algorithm COMPUTE$S_1$, the points of $S_0$ are moved to $S_1$ one by one, and only the point whose region of influence on $r$ is empty is removed from $S_1$. Therefore, $P^r_v = \emptyset, \forall v \in S^r_g - S_1$.

2. $P^r_v \neq \emptyset, \forall v \in S_1$;

   Following algorithm COMPUTE$S_1$, each point $v$ (except the endpoints) and its adjacent points $u$, $w$ $(u, v, x \in S_1\ and\ u.x < v.x < w.x)$ have two separation position $c^r_{uv}$ and $c^r_{vw}$. $c^r_{uv} < c^r_{vw}$, thus $c^r_{uv} < P^r_v.x \leq c^r_{vw}$.

   The left endpoint $v$ and its adjacent point $w$ have a separation position $c^r_{vw}$. $c^r_{vw} > 0$, thus $0 < P^r_v.x \leq c^r_{vw}$. (The right end point is in a similar way.) Therefore, $P^r_v \neq \emptyset, \forall v \in S_1$.

Therefore, $S_1 = S^r_f$.
□

Thus, each point in $I_r$ is compared with two adjacent points in $S^r_f$ at most to get its closest feature point (Property.1, Property.2). By processing each row with COMPUTE$S_1$, we get the closest feature point for each point of I. Thus, we calculate the EDT by (2).

## 3. Implementation of PBEDT

In this section, we introduce the implementation of PBEDT.

### 3.1. First stage algorithm

Different from former independent scan algorithms [1][16], which compute the squared distance in 1-D by twice scan, in the first stage, we compute the relative squared distance between the closest feature points in column and the left end point of each row by a forward scan with back propagation. $INFTY$ used in SQUARE-DISTANCE-1D$(I)$ is an integer bigger than the squared diagonal distance of the image.

```
SQUARE-DISTANCE-1D($I$)
for $c = 0 \to n\text{-}1$
    $mid \Leftarrow -1$
    for $r = 0 \to m\text{-}1$
        if ($I(r,c) = 1$)
            if ($mid > -1$)
                $mid \Leftarrow (r + mid)/2$;
            for $\overline{r} = mid + 1 \to r$
                $I(\overline{r}, c) \Leftarrow (\overline{r} - r)^2 + c \times c$;
            $mid \Leftarrow x$;
        else
            if ($mid = -1$)
                $I(r,c) \Leftarrow INFTY$;
            else
                $I(r,c) \Leftarrow (r - mid)^2 + c \times c$;
```

## 3.2. Second stage algorithm

We implement the second stage algorithm introduced by COMPUTE$S_1$ in an effective way.

### 3.2.1 Calculate the intersection points

$B_{uv}$ intersects row $r$ at $(c^r_{uv}, r)$, thus $\|u - (c^r_{uv}, r)\| = \|v - (c^r_{uv}, r)\|$ (Fig.1), and we obtain $c^r_{uv}$:

$$c^r_{uv} = \{(v.x)^2 - (u.x)^2 - 2 \times r \times (v.y - u.y) + (v.y - r)^2 - (u.y - r)^2\}/\{2 \times (v.x - u.x)\}$$

We use the relative distance to replace the y coordinate, $\bar{y} = y - r$, then $c^r_{uv}$ can be simplified as:

$$c^r_{uv} = \frac{((v.x)^2 + (v.\bar{y})^2) - ((u.x)^2 + (u.\bar{y})^2)}{2(v.x - u.x)};$$

Moreover, let

$$du = (u.x)^2 + (u.\bar{y})^2 \qquad (7)$$

, then

$$c^r_{uv} = \frac{dv - du}{2(v.x - u.x)}. \qquad (8)$$

This calculation can be organized as a function (Intersection), which only has 1 division , 1 multiplication, and 2 substraction.

```
Intersection($ux, vx, du, dv$)
    return $(dv - du)/(2 * (vx - ux))$;
```

From (7), $du$ is the squared distance of $u$ to the left most point of row $r$ which can be computed in the first stage (SQUARE-DISTANCE-1D).

### 3.2.2 Integer calculation

We use integer division to replace the float division in (8). The integer division abandons the decimal and keep the integer, which is faster but not always accurate. In our algorithm, we use the efficiency of integer arithmetic and keep the accuracy meanwhile. Fig.3 shows two different situations, but $c^r_{uv} = c^r_{vw} = 4$ in every situation by integer division. In both of situations, we assign point p(4,5) is close to the leftmost point of this triple — $u$, since the point coordinates of $I$ are all integers. Even if $B_{uv}$ and $B_{vw}$ all intersect at point p(4,5), the assignment is valid. Therefore, an equivalent integer division will be more efficient than float division.

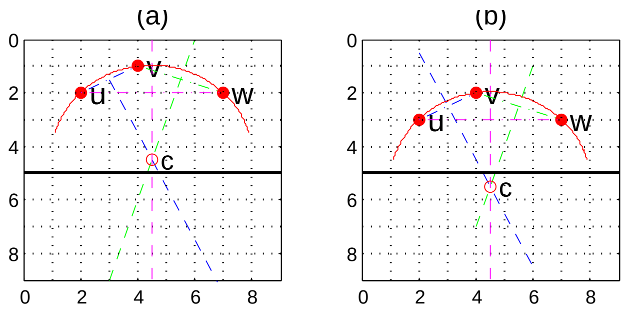


Figure 3. Different situations of assigning same closer feature point: (a) $c^r_{uv} > c^r_{vw}$; (b) $c^r_{uv} < c^r_{vw}$

Negative values between two integers will be rounded to the lager integer (-0.9 will be rounded to 0), 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 while we use 0 in COMPUTE$S_1$. Moreover, we prejudge the sign of $c^r_{uv}$ before calculating it, which promotes the execution speed. In (8), $vx < ux$ is known, thus the sign of $c^r_{uv}$ is relative to $(dv - du)$.

Therefore, Intersection is rewritten as:

```
Intersection-INT($ux, vx, du, dv$)
    if ($dv > du$)
        return $(dv - du)/(2 * (vx - ux))$;
    else
        return -2;
```

### 3.2.3 Euclidean distance computation

We also use $du$ which comes from (7) in the distance computation. Given $u$ is a point on row $r$ ($u.y = r$) and $v$ is the closest feature point of $u$, then

$$\begin{aligned}\|u - v\|^2 &= (u.x - v.x)^2 + (u.y - v.y)^2 \\ &= (u.x)^2 - 2(u.x)(v.x) + (v.x)^2 + (v.\bar{y})^2 \\ &= (u.x)(u.x - 2 \times v.x) + dv\end{aligned}$$

This can be organized as a function, which only has one addition, one substraction and two multiplications.

```
Distance($ux, vx, dv$)
    return $ux * (ux - vx * 2) + dv$;
```

### 3.2.4 Second stage algorithm

Based on the technical details above, we give the second stage algorithm (SQUARE-DISTANCE-2D).

```
SQUARE-DISTANCE-2D($I$)
for $r = 0$ to $m - 1$
    $stack\_c \Leftarrow \emptyset; stack\_cx \Leftarrow \emptyset$;
    $stack\_g \Leftarrow \emptyset; p \Leftarrow -1$
    for $c = 0$ to $n - 1$
        if ($I(r, c) < INFTY$)
            while (TRUE)
                if ($p \geq 0$)
                    $cx \Leftarrow$ Intersection-INT($stack\_c[p]$,
                        c,$stack\_g[p]$, abs($I(r, c)$))
                    if ($cx = stack\_cx[p]$)
                        $p \Leftarrow p - 1$;
                    else if ($cx < stack\_cx[p]$)
                        $p \Leftarrow p - 1$;
                        continue;
                    else if ($cx \geq (n - 1)$)
                        $break$;
                else
                    $cx \Leftarrow -1$;
                $p \Leftarrow p + 1$;
                $stack\_c[p] \Leftarrow c$;
                $stack\_cx[p] \Leftarrow cx$;
                $stack\_g[p] \Leftarrow I(r, c)$;
                break;
    if ($p < 0$)
        $return$ FALSE;
    $c \Leftarrow 0$;
    for $k = 0$ to $p$
        if ($k = p$)
            $cx \Leftarrow n - 1$;
        else
            $cx \Leftarrow stack\_cx[k + 1]$;
        while ($c \leq cx$)
            $I(r, c) \Leftarrow$ Distance($c, stack\_c[k]$,
                $stack\_g[k]$);
            $c \Leftarrow c + 1$;
```

## 3.3. Computational complexity

We discuss the computational complexity of PBEDT:

- The time complexity of PBEDT is $O(N)$ times.

  1. In the first stage, SQUARE-DISTANCE-1D takes O(N) time, since it scans forward once and propagates backward at most 0.5N. Thus, each element is accessed no more than twice, leading to an average number of 1.5 access times per element (computational complexity of $O(N)$).
  2. In the second stage, SQUARE-DISTANCE-2D processes m rows one by one. There are two processes for each row, one computes $stack_c x$ and the other computes squared distance.

     In process one, $|S_g^r| \leq n$, and each point in $S_g^r$ can only be added to $stack_c$ once. Each point in $stack_c$ can only be removed once, and $|stack_c| \leq n$. Hence, these adding and removing operations are executed at most 2n times. Thus, process one takes $O(n)$ time.

     Process two computes the distance between each point $w$ in row $r$ and w's closest feature point compute once, and takes $O(n)$ time.

     Hence, in the second stage, SQUARE-DISTANCE-2D takes $O(N)$ ($O(m * n)$) times.

- The space requirement of PBEDT is very low.

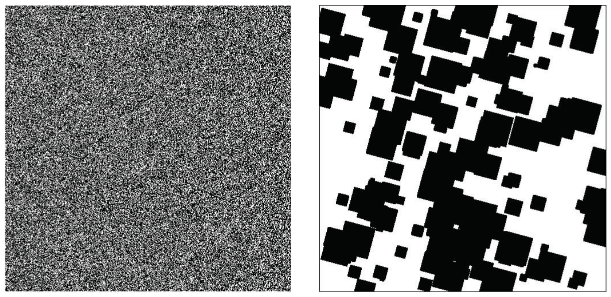
  In each stage, PBEDT recycles the memory of input image. In the second stage, PBEDT need a temporary memory whose size is 3n to store $stack_c$, $stack_{cx}$, $stack_g$.

## 4. Experiments and discussion

To evaluate its performance, PBEDT was compared with state-of-the-art EDT algorithms, such as Maurer et al.'s [16], Saito and Toriwaki's [22], Cuisenaire and Macq's [4], Lotufo and Zampirolli's [6], Meijster's [17] and Felzenszwalb et al.'s [10]. In order to improve readability, these algorithms are abbreviated as MAURER2003, SAITO1994, CUISENAIRE, LOTUFOZAMPIROLLI, MEIJSTER, FELZENSZWALB, respectively. Felzenszwalb et al. give the implementation of their algorithm [11], and other algorithms are implemented by Fabbri et al. [8].

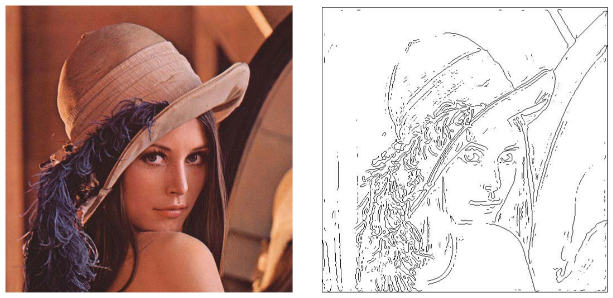
The tests were performed on a computer with an Intel Core2 Duo 2.53GHz processor, 2GB RAM, Ubuntu Linux OS with kernel v2.6.31. All algorithms are implemented in ANSI C/C++, and built by GCC v4.1.1. The performance of PBEDT was measured with images over a wide range of sizes and contents, as Fabbri et al. recommends in [9].

Comparing the outputs of PBEDT to those of other algorithms, no difference has been observed from all these tests. Additional tests with thousands of randomly generated images varying in width and number of feature points also indicate the correctness of our algorithm.



(a) (b)

Figure 4. Randomly generated sample images. (a) Random points, feature pixel proportion: 40%, size: 500×500; (b) Random squares, feature pixel proportion: 15%, square angle: 15°, size: 500×500



(a) (b)

Figure 5. Test image Lenna. (a) Original image; (b) Edge image of Lenna

The following images are chosen for precisely analysis:

1. Random points. The image size varies from $100 \times 100$, $500 \times 500$, ..., to $4000 \times 4000$ with randomly generated feature points where the number of feature points comprises 1%, 10%, ..., 90%, and 99% of the image. One sample is shown in Fig. 4(a). This test provides an idea of the performance of the algorithms relative to the number of feature pixels [9][16].

2. Random squares. These images are generated by randomly choosing the centers and sizes of black squares rotated by $\theta \in [0, 90]$. The squares are filled and plotted into the image until the black pixels add up to a percentage value p. One sample is shown in Fig. 4(b). This test is based on a synthetic image having more similarity to real images with some orientation [7][9].

3. Special feature contents:

   - A feature square located at the corner of an image. In this case, EDT produces the largest and smallest possible distances for a given image size: diagonal and 1, respectively [9][16].
   - Binary images of real objects. The edge image from the Lenna image obtained by thresholding the response of an edge detector is used, as shown in Fig. 5(b). Lenna is chosen since it has been universally used as an impartial benchmark for image analysis algorithms [9].
   - A white disk inscribed in the image. 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, specially near the center of the disk [3][9][22].
   - Half-filled image. This is the worst case of the brute-force algorithm, and Maurer2003 also suffers a setback in this test [9].

As shown in Fig. 6 and Fig. 7, PBEDT is more stable than other algorithms with different proportions of feature pixels, and it is faster than MAURER2003, MEIJSTER and FELZENSZWALB at all parameter settings used, faster than SAITO1994 and LOTUFOZAMPIROLLI at small proportion feature pixels, slower than CUSENAIRE only at very large proportion of feature points. Besides, PBEDT is more stable than others with different slant angle of random squares. SAITO1994 and LOTUFOZAMPIROLLI are faster than PBEDT when slant angle is 0 or 90, but slower when other angles are used. Only when the proportion of feature pixels is higher than 70%, SAITO1994, CUSENAIRE and LOTUFOZAMPIROLLI are faster than PBEDT at every slant angel. Fig. 8(a) shows that PBEDT is the fastest with the test of edge image of Lenna at all sizes. Fig. 8(b) shows that PBEDT is the fastest in all feature contents.

In all these tests, PBEDT shows excellent performance. It is the fastest in most cases, and the most stable in terms of image contents. Neither the number of feature pixels nor the orientation of content objects can affect its performance. It is faster than MAURER2003, MEIJSTER and FELZENSZWALB in all cases, and faster than SAITO1994, LOTUFOZAMPIROLLI and CUISENAIRE in most cases. SAITO1994 and LOTUFOZAMPIROLLI are seriously affected by the orientation of feature objects, which show poor performance in most of angles. Moreover, their performance varies with different proportion of feature pixels. Table.1 shows the mean execution time on randomly generated squares with various feature point proportions and slant angels at size $3000 \times 3000$. Obviously, of all algorithms, PBEDT is the fastest one.

## 5. Conclusion

We have introduced a two-stage independent scan algorithm PBEDT, which is simpler than previous works: use partial Voronoi diagram [1][16] or lower-envelop of parabolas [10][14][17][22]. The time complexity of PBEDT is linear in the number of image points with a small constant

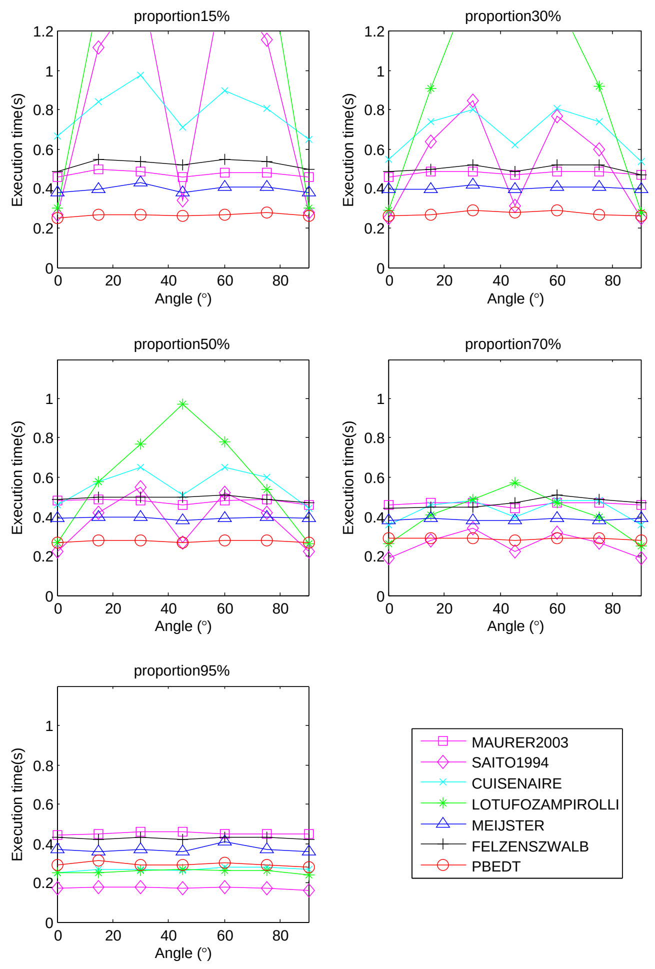


Figure 6. Execution time for random squares images with varying feature point proportion, size 3000 × 3000, slant angle varied from 0 to 90.

| Algorithms | Average time(s) | Comparison with PBEDT (%) |
|---|---|---|
| MAURER2003 | 0.469 | 168.2% |
| SAITO1994 | 0.441 | 158.0% |
| CUISENAIRE | 0.547 | 196.0% |
| LOTUFOZAMPIROLLI | 0.802 | 287.3% |
| MEIJSTER | 0.391 | 139.9% |
| FELZENSZWALB | 0.483 | 173.2% |
| PBEDT | 0.279 | 100% |

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

term and the memory requirement is very low. Compared with other state-of-the-art algorithms, PBEDT is the fastest in most cases, and also the most stable one with respect to image contents. The main innovations of this algorithm are: the geometric property of perpendicular bisector is used to compute the segmentation of rows directly; the integer arithmetic is used to avoid time consuming float operations and still keeps exactness. All these innovations reduce the computational complexity significantly. A parallel version of PBEDT can be easily implemented [18][21]. This algorithm can also be extended to three or higher dimensional binary images. Currently, we are evaluating the n-D version of this algorithm. Results will be published in a more elaborate paper which is currently in preparation.

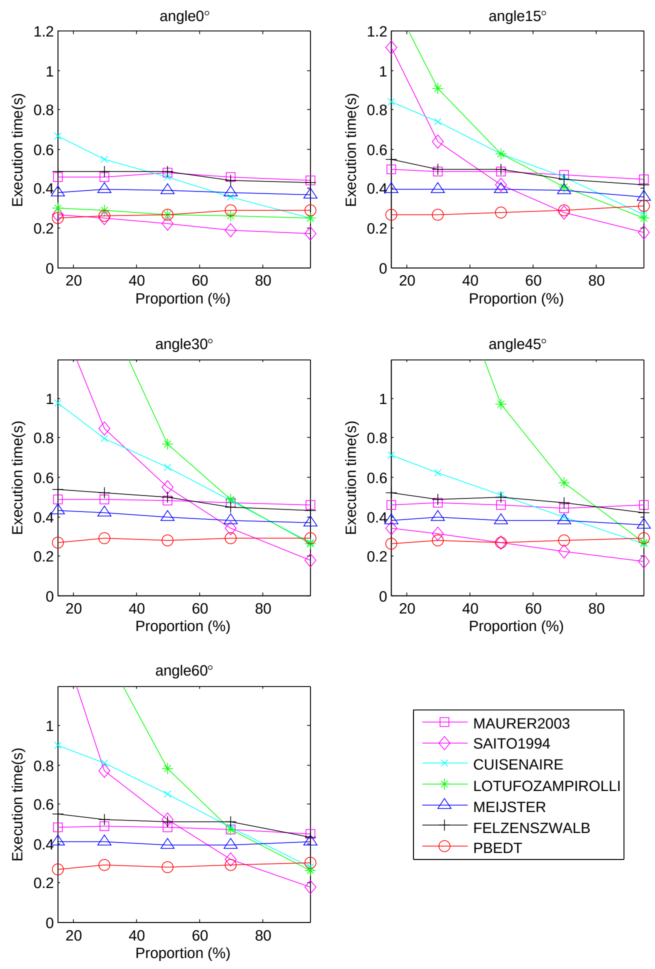
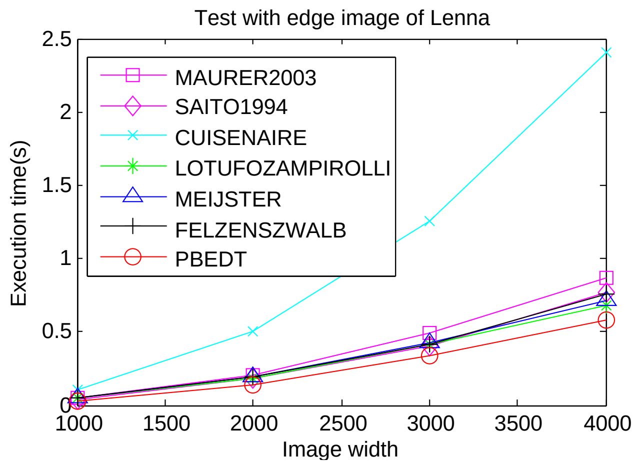


Figure 7. Execution time for random squares images with varying slant angle, size 3000 × 3000, feature points proportion ranged from 15% to 95%.

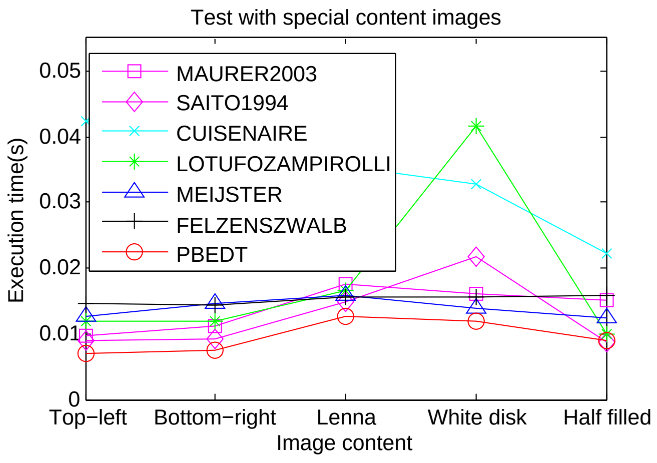
## References

[1] H. Breu, J. Gil, D. Kirkpatrick, and M. Werman. Linear time euclidean distance transform algorithms. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(5):529–533, 1995.

[2] O. Cuisenaire. *Distance transformations: fast algorithms*



(a)



(b)

Figure 8. Test results. (a) Results with Lenna edge images of ranged size; (b) Results with images of special features, size 500×500.

*and applications to medical image processing*. PhD thesis, Louvain-la-Neuve, Belgium, 1999.

[3] O. Cuisenaire and B. Macq. Fast and exact signed euclidean distance transformation with linear complexity. In *ICASSP '99: Proceedings of the Acoustics, Speech, and Signal Processing, 1999, IEEE International Conference*, pages 3293–3296, Washington, DC, USA, 1999. IEEE Computer Society.

[4] O. Cuisenaire and B. M. Macq. Fast euclidean distance transformation by propagation using multiple neighborhoods. *Computer Vision and Image Understanding*, 76(2):163–172, 1999.

[5] P.-E. Danielsson. Euclidean distance mapping. *Computer Vision, Graphics, and Image Processing*, 14:227–248, 1980.

[6] R. de Alencar Lotufo and F. A. Zampirolli. Fast multidimensional parallel euclidean distance transform based on mathematical morphology. In *SIBGRAPI*, pages 100–105. IEEE Computer Society, 2001.

[7] H. Eggers. Two fast euclidean distance transformations in z$^2$based on sufficient propagation. *Computer Vision and Image Understanding*, 69(1):106–116, 1998.

[8] R. Fabbri, L. da Fontoura Costa, J. C. Torelli, and O. M. Bruno. Complete results of the benchmark between exact edt algorithms. ”`http://distance.sourceforge.net`”, 2006.

[9] R. Fabbri, L. da Fontoura Costa, J. C. Torelli, and O. M. Bruno. 2d euclidean distance transform algorithms: A comparative survey. *ACM Comput. Surv.*, 40(1), 2008.

[10] P. F. Felzenszwalb and D. P. Huttenlocher. Distance transforms of sampled functions. Technical report, Cornell Computing and Information Science, September 2004.

[11] P. F. Felzenszwalb and D. P. Huttenlocher. Distance transforms of sampled functions (program). `http://people.cs.uchicago.edu/~pff/dt/`, 2004.

[12] M. L. Gavrilova and M. H. Alsuwaiyel. Two algorithms for computing the euclidean distance transform. *Int. J. Image Graphics*, pages 635–645, 2001.

[13] W. H. Hesselink and 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):2204–2217, 2008.

[14] T. Hirata. A unified linear-time algorithm for computing distance maps. *Inf. Process. Lett.*, 58(3):129–133, 1996.

[15] Y. Lucet. New sequential exact euclidean distance transform algorithms based on convex analysis. *Image Vision Comput.*, 27(1-2):37–44, 2009.

[16] J. Maurer, R. Qi, and 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):265–270, 2003.

[17] A. Meijster, J. B. T. M. Roerdink, and W. H. Hesselink. A general algorithm for computing distance transforms in linear time. *Computational Imaging and Vision*, 18(8):331–340, 2000.

[18] M. Miyazawa, P. Zeng, N. Iso, and T. Hirata. A systolic algorithm for euclidean distance transform. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(7):1127–1134, 2006.

[19] D. W. Paglieroni. Distance transforms: Properties and machine vision applications. *CVGIP: Graphical Model and Image Processing*, 54(1):56–74, 1992.

[20] I. Ragnemalm. Neighborhoods for distance transformations using ordered propagation. *CVGIP: Image Underst.*, 56(3):399–409, 1992.

[21] A. Rosenfeld and J. L. Pfaltz. Sequential operations in digital picture processing. *J. ACM*, 13:471–494, 1966.

[22] T. Saito and J. ichiro Toriwaki. New algorithms for euclidean distance transformation of an n-dimensional digitized picture with applications. *Pattern Recognition*, 27(11):1551–1565, 1994.