Enhancing distance transform computation by leveraging the discrete nature of images

Guillaume Fuseiller, Romain Marie, Gilles Mourioux, Erick Duno, Ouiddad Labbani-Igbida

Noname manuscript No.
(will be inserted by the editor)

Enhancing distance transform computation by leveraging the discrete nature of images

Guillaume Fuseiller · Romain Marie · Gilles Mourioux · Erick Duno · Ouiddad Labbani-Igbida

Received: date / Accepted: date

Abstract This paper presents a major reformulation of a widely used solution for computing the exact Euclidean distance transform of n-dimensional discrete binary shapes. Initially proposed by Hirata, the original algorithm is linear in time, separable, and easy to implement. Furthermore, it accounts for the fastest existing solutions, leading to its widespread use in the state of the art, especially in real-time applications. In particular, we focus on the second step of this algorithm, where the lower envelope of a set of parabolas has to be computed. By leveraging the discrete nature of images, we show that some of those parabolas can be merged into line segments. It reduces the computational cost of the algorithm by about 20% in most practical cases, while maintaining its exactness. To evaluate the proposed improvement on different cases, two state-of-the art benchmarks are implemented and discussed.

Keywords Distance transform · Linear time algorithm · Dynamic programming

Mathematics Subject Classification (2020) MSC 68U10 · MSC 68U05 · 94A08

This work was partially supported by Valeo.

G. Fuseiller, R.Marie, G.Mourioux, O.Labbani-Igbida
XLim Institute, UMR CNRS 7252, University of Limoges, 16 rue Atlantis, 87068 Limoges Cedex, FRANCE
E-mail: firstname.lastname@xlim.fr

G.Fuseiller, E.Duno
VALEO Materiaux de Friction Limoges, FRANCE

R.Marie
3iL Groupe, 43 rue de Sainte-Anne, 87015 Limoges, FRANCE
E-mail: marie@3il.fr

1 Introduction

First introduced by Rosenfeld et al. [1], the Distance Transform (DT) of a binary shape associates to each point its distance to the nearest boundary point. When this distance is defined by the Euclidean metric (which is by far the most common case), the DT is then referred as Euclidean Distance Transform (EDT). This fundamental geometric operator finds applications in various fields, such as shape analysis, data compression, computer graphics or robotics. As such, it has been widely investigated over the years, leading to a large number of real-time solutions in arbitrary dimensions.

As pointed out by Fabbri et al. [2], existing EDT algorithms can generally be classified into three categories, depending on the order in which pixels are processed:

- *Ordered propagation* algorithms emulate the Eikonal equation: A wavefront is initiated at the boundaries of the considered binary shape, and propagates at constant speed towards its center, whilst assigning to each encountered pixel the (current) distance traveled by the wave.
- *Raster scan* algorithms process pixels by scanning each line (forward then backward) in a sequential order.
- *Dimensional reduction* algorithms first compute the 1D DT for each row (or column) independently. This intermediate result is then used in a second step where the 2D DT is obtained. When the considered shape is defined in higher dimensions, this process repeats iteratively along each direction until the n-dimensional distance transform is finally obtained (i.e. the actual DT).

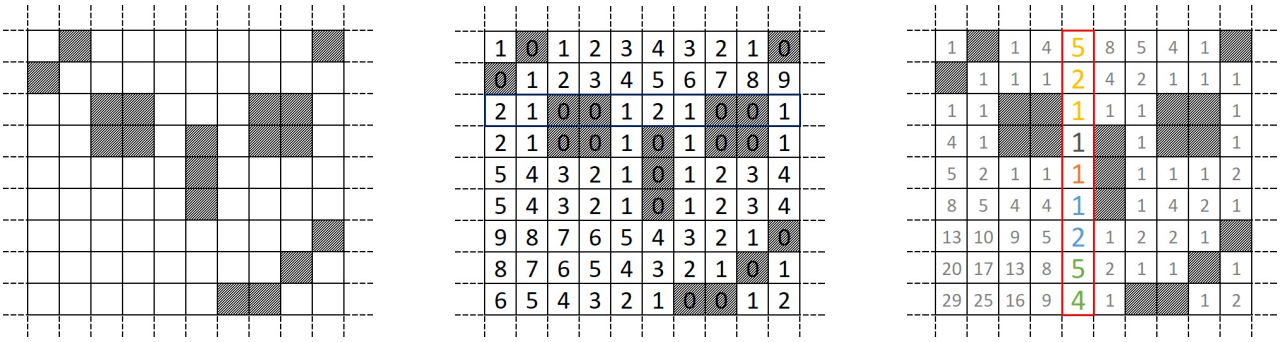


Fig. 1 (left) An example of a binary shape, where black cells represent obstacles. (center) The one-dimensional DT, where each cell's value corresponds to its distance to the nearest black cell on the same row. (right) The resulting squared EDT. The column in red is the one considered on Fig.2.

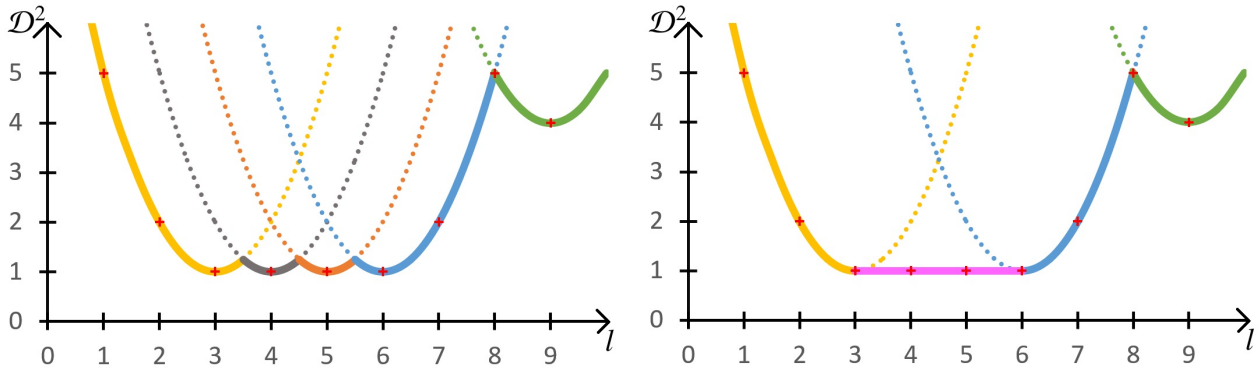


Fig. 2 Squared EDT computation for the red column in Fig.1 based on: (left) the lower envelope of a set of parabolas [3], and (right) the lower envelope of a set of line segments and parabolas. Notice how, at discrete points (red crosses), both representations generate the same values.

The work described hereafter relates to this last category. In such approaches, although the first step (1D DT) is straightforward, this is not the case for the following ones. Specific properties of the Euclidean metric must be leveraged to minimize the computational cost and guarantee an exact EDT. One popular solution was initially introduced by Saito et al. [4]. It is based on the lower envelope of a set of parabolas, from which the EDT of a given line can be deduced (see Fig.2).

The main contribution of this article is a new formulation of this strategy. While maintaining its separability and linear complexity, it leverages the discrete nature of digital shapes to merge successive parabolas into line segments (see Fig.2). As such, it allows to generate the exact same result (i.e. the exact EDT) while significantly reducing the computational cost. Additionally, the resulting algorithm is extremely short and simple to implement in arbitrary dimensions, thus favoring its use on a wide range of real-time contexts, especially when GPU-based implementation is not an option. Note that this last point explains why most recent works, such as [5] or [6] are not compared to the proposed solution.

To better understand the theoretical and algorithmic underpinnings of this contribution, the rest of the article is organized as follows: In Section 2, a brief history of exact EDT is presented, with particular emphasis on algorithms based on dimensional reduction. In Section 3, preliminary concepts are introduced, including general definitions related to EDT computation, and its relation to the lower envelope of a set of parabolas. In Section 4, the core contribution is presented, both in terms of theory and effective implementation. Section 5 is dedicated to the experimental validation, where the gain of the proposed algorithm is quantified with respect to the best state-of-the-art algorithms, before concluding this work in Section 6.

2 State of the art

Early DT algorithms were based on *raster scan* [1] or *ordered propagation* [7], and considered city-block or chessboard discrete metrics. The issue of Euclidean DT computation was thereafter investigated more than a decade later. Although fast to compute, the algorithms proposed in [8, 9] were only approximations, and therefore required a costly post-processing step [10, 11]. For a complete overview of the approximate method, we re-

fer the reader to the state-of-the-art proposed by Fabbri et al. [2] or the technical survey proposed in [12] for implementation details. In the following, we will focus on exact EDT algorithms.

From the early 90's, exact EDT algorithms based on *dimensional reduction* became popular. In such approaches the basic idea is to first compute the 1D DT for each row (or column) independently, and then use this intermediate result in a second phase to compute the 2D DT. The non trivial part is the second step, where various algorithms have been proposed to minimize the computational cost and guarantee an exact EDT. There are three main variants of this approach guaranteeing linear complexity with respect to the discrete grid size. The first is based on Voronoi diagrams (VD). Instead of computing the VD explicitly (which is time consuming, and in no case linear), Breu et al. [13] proposed an EDT algorithm that efficiently determines the intersection between an image line and the VD, without constructing it explicitly. This approach was later improved by Guan et al. [14], who took advantage of the fact that adjacent points tend to have the same nearest boundary. Although the concepts are exactly the same, more efficient algorithms were then successively proposed by Maurer et al. [15] and Wang et al. [16] (who also introduced a recursive generalization to higher dimensions). The second variant is based on mathematical morphology. After Shih et al. [17] showed that the EDT can be computed by a single gray-scale morphological erosion of the input shape, Lotufo et al. [18] proposed a strategy to decompose the structuring element into a set of 1D elements, thus leading to an independent scanning algorithm. The last variant uses parabola intersections, as originally introduced by Saito et al. [4]. The central idea is to speed up the second phase (and possibly additional phases) by computing the lower envelope of a set of parabolas, from which the EDT of a given line can be deduced (see Section 3). This approach has been greatly improved, reformulated and enhanced over the years ([3], [19], [20]), and has led to a set of algorithms with exactly the same complexity (i.e. linear in the grid size), but with a variable constant term.

3 EDT computation using lower envelope of parabolas

3.1 General idea

For the sake of simplicity, and given that the following can easily be generalized to higher dimensions, let's consider a two- dimensional binary image input, $I = \mathcal{X} \cup \mathcal{X}^c$ of dimension $m \times n$ (m rows, n columns), where $\mathcal{X}$ denotes the shape, and $\mathcal{X}^c$ the background. The EDT of I is a 2D grid $\mathcal{D}_\mathcal{I} = \{\mathcal{D}(\mathbf{x})\}$ storing for each pixel $\mathbf{x} = (x_1, x_2)$ its distance $\mathcal{D}(\mathbf{x})$ to the nearest background point:

$$\forall \mathbf{x} \in I,\ \mathcal{D}(\mathbf{x}) = \min_{\mathbf{y} \in \mathcal{X}^c} \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2} \qquad (1)$$

This formulation provides an efficient computation process using two steps :

- First, each row of index l is independently considered as a 1D signal to generate a one dimensional EDT $G = \{g(\mathbf{x})\}$:

$$\forall \mathbf{x} = (l, x_2),\ g(\mathbf{x}) = \min_{(l, y_2) \in X^c} |x_2 - y_2| \qquad (2)$$

- Second, each column of index c is scanned to deduce the distance transform $\mathcal{D}(\mathbf{x})$ by solving:

$$\forall \mathbf{x} = (x_1, c),\ \mathcal{D}(\mathbf{x})^2 = \min_{1 \leq l \leq m} \{(x_1 - l)^2 + g(l, c)^2\} \qquad (3)$$

As discussed in [4], the *min* operation in the second step is equivalent to a calculation of the lower envelope of a set of parabolas. Let us consider a given column c as a one-dimensional signal. Each row l defines a parabola $\mathcal{F}_l(i) = (i - l)^2 + g(l, c)^2$ representing the (squared) distance between a point along c and the closest boundary point from (l, c) along row l. Consequently, the lower envelope of the set $\mathcal{F} = \{\mathcal{F}_l\}$ of all parabolas exactly defines the EDT for column c.

3.2 EDT Algorithm based on parabolas lower envelope

Several works ([3], [19], [21], [20]) have thus focused on the definition of efficient algorithms to compute such a lower envelope. They are extremely close to each other, both in term of formulation and performance. According to our tests (see Section.5), the one proposed by [20] slightly outperforms the others. Algorithms 1 and 2 present the corresponding pseudo-code for both steps.

The first algorithm (Algo.1), corresponding to the one-dimensional EDT computation, is common to all solutions (including ours), and is simply a direct application of Eq.2. The second algorithm (Algo.2) is central to the proposed contribution, and therefore requires some explanation. For each column $j \in [0, n[$ (line 1), the core idea is to determine the ordered subset of $\mathcal{F}$ contributing to its lower envelope, using a single scan of each cell. For this, it uses two arrays v and z to store for each parabola k of this subset, *i)* the horizontal coordinate of its summit ($v[k]$), and *ii)* the starting position of its contribution to the lower envelope ($z[k]$), corresponding to its intersection with the previous parabola

Algorithm 1: Computing g (first dimension)

input : Binary image $I = X \cup X^c$ of size $m * n$
output : 1-dimensional distance transform $g(\mathbf{x})$ for each $x \in I$

1 **for** *i=0 to m-1* **do**
2 **if** $(i,0) \in X^c$ **then**
3 $g(i,0) = 0$
4 **else**
5 $g(i,0) = \infty$
6 **for** *j=1 to n-1* **do**
7 **if** $(i,j) \in X^c$ **then**
8 $g(i,j) = 0$
9 **else**
10 $g(i,j) = g(i,j-1) + 1$
11 **for** *j=n-2 to 0* **do**
12 **if** $g(i,j+1) < g(i,j)$ **then**
13 $g(i,j) = g(i,j+1) + 1$

Algorithm 2: Computing $\mathcal{D}$ from g (second dimension)

input : 1-dimensional distance transform $g(\mathbf{x})$, for each $\mathbf{x} \in I$
output : Euclidean distance transform $\mathcal{D}$

1 **for** *j=0 to n-1* **do**
2 $k = 0$, $v[0] = 0$, $z[0] = -\infty, z[1] = +\infty$
3 **for** *i=1 to m-1* **do**
4 $w = 1 + s(v[k], i)$ (Eq.4)
5 **while** $w \leq z[k]$ **do**
6 $k = k - 1$
7 $w = 1 + s(v[k], i)$ (Eq.4)
8 $k = k + 1, v[k] = i, z[k] = w, z[k+1] = \infty$
9 **for** *i=0 to m-1* **do**
10 **while** $z[k+1] < i$ **do**
11 $k = k + 1$
12 $\mathcal{D}^2(i,j) = (i - v[k])^2 + g(v[k], j)^2$

$(k-1)$ (see Table.1 for an illustration). Both arrays are initialized with the first parabola of the column (line 2). At a given step of the algorithm, corresponding to a cell $i \in [1, m[$ (line 3), v and z contain k parabolas defining a temporary lower envelope. To determine whether or not the parabola $\mathcal{F}_i$ contributes to the last one, its intersection $s(v[k], i)$ with $\mathcal{F}_{v[k]}$ is calculated :

$$s(v[k], i) = \frac{i^2 - v[k]^2 + g(i,j) - g(v[k], j)}{2(i - v[k])} \quad (4)$$

Let $w = 1 + s(v[k], i)$ denote the closest integer higher than this intersection coordinate (line 4). Two cases are then to be considered:

- w is higher than $z[k]$ (line 8): as illustrated Fig.3(left), it means that $\mathcal{F}_{v[k]}$ still contributes to the lower envelope of $\mathcal{F}$ (from $z[k]$ to $w-1$). $\mathcal{F}_i$ is then added to the list such that $z[k+1] = w$, and $v[k+1] = i$.
- w is lower than or equal to $z[k]$ (lines 5 to 7): as illustrated Fig.3(center), it involves that the contribution to the lower envelope of $\mathcal{F}_{v[k]}$ is entirely covered by the one of $\mathcal{F}_i$. Therefore, $\mathcal{F}_{v[k]}$ is removed, and the process is repeated with $\mathcal{F}_{v[k-1]}$.

Once each cell has been considered, v and z entirely define the lower envelope of $\mathcal{F}$. The last step is thus to iterate through the column, and compute for each cell its Euclidean distance, based on the parabola associated to its position (lines 9 to 12).

It should be noted that, for a shape defined in $\mathbb{Z}^n$, Algo.2 is simply repeated for each dimension except the first one (i.e. $n-1$ times).

4 EDT Algorithm based on parabolas and line segments

4.1 General idea of the proposed method

A simple analysis of Algo.2 shows that the main computational load is induced by lines 4 and 7, where the intersection between two parabolas is determined (Eq.4). The core idea behind the proposed approach is to reduce the frequency of this operation. To achieve this goal, two properties induced by the context are leveraged.

- The considered shapes are defined on discrete grids. Thus, due to sampling, adjacent cells often tend to have the same one-dimensional distance transform g. As illustrated in Fig.2 (bottom left), for a given column, the corresponding lower envelope is then formed of a large number of identical parabolas, up to a shift along the axis.
- The distance transform is also defined on a discrete grid. In other words, as long as the resulting EDT is exact for each cell, the lower envelope calculated for each column does not have to be exact for any continuous point on the grid domain.

Based on these properties, this work proposes to approximate adjacent parabolas of the same height by a single line segment connecting their summit, as illustrated in Fig.2. This approach has two major advantages. First, as explained previously, this reduces the number of entities composing the lower envelope for each column, and therefore the number of intersections (lines 4 and 7) to be computed. Second, the construction of such line segments can be performed using a fast-forward process, which allows to entirely skip lines 4 to 8 of Algo.2 for the corresponding cells.

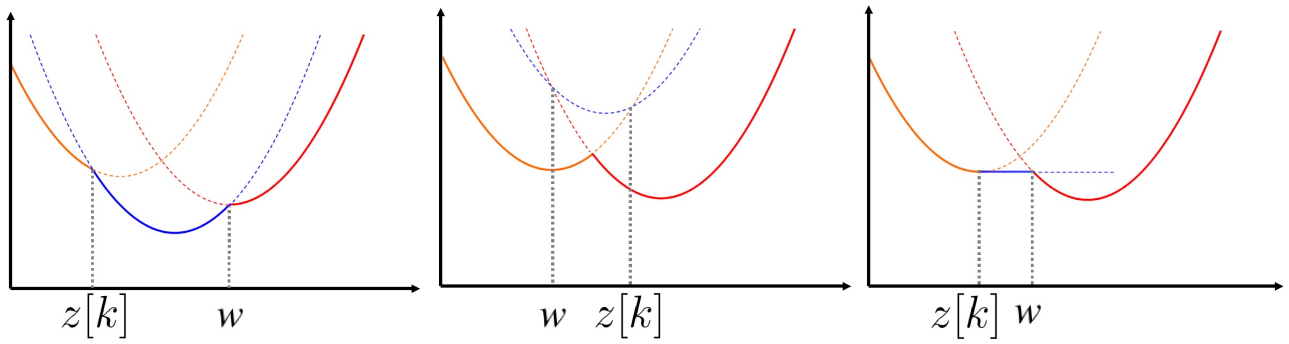


Fig. 3 The three cases to be considered when adding the red parabola to the lower envelope: (left) When $z[k] <= w$, the parabola k (in blue) still contributes to the lower envelope; (center) when $z[k] > w$, the parabola k is totally eclipsed by the red one. It is thus removed from the lower envelope; and (right) when the element k is a line segment, the process is exactly the same, except that w is calculated based on Eq.6 instead of Eq.4.

4.2 Lower envelope computation

Similarly to the original work, the lower envelope of a given column c is built incrementally by successively considering the parabolas $\mathcal{F}_i$ for each $i \in [0, n[$, and, if necessary, by adding a single one as a parabola, or several ones as a line segment. The success of this process depends on answering the following questions:

- *When should a line segment be added ?* Consider when a parabola $\mathcal{F}_l$ (computed from row l and defined by the tuple $\{z[k], v[k]\}$) has just been added. This parabola marks the beginning of a line segment when *i)* the beginning of its area of influence $z[k]$ is inferior or equal to its center $v[k]$, and *ii)* at least the following two parabolas have the same height as $\mathcal{F}_l$:
$$\begin{cases} z[k] \leq v[k] \\ \exists L \geq 2 \mid \forall j \in [l, l+L], g(j,c) = g(l,c) \end{cases} \quad (5)$$
The first condition is required, because when not respected, the summit of $\mathcal{F}_l$ does not belong to the lower envelope, and thus cannot be the beginning of a line segment.
- *How to efficiently add a line segment?* Assuming the two previous conditions are respected, a fast forward from row l to row $l+L$ is performed (i.e., all the parabolas in between are not considered). Two elements are then added to the lower envelope:
  - A line segment from l to $l+L$ with height $h = g(l,c)^2$ to encapsulate all the required information for the skipped parabolas,
  - A parabola starting from and centered on $l+L$ to ensure the transition with the rest of the lower envelope.

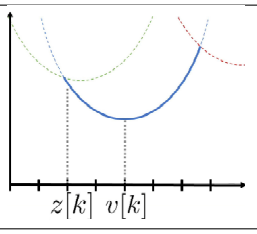
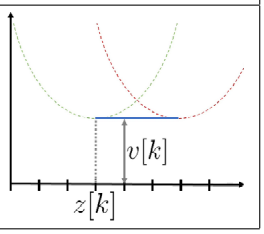
  Notice that the whole process, while potentially considering a large number of parabolas, does not require any calculation.
- *How to efficiently add a parabola?* When $\mathcal{F}_l$ does not belong to a line segment, it has to be added on the lower envelope by itself. This process leads to evaluate its intersection with the last element to determine if the latter should be removed, and iterates until this is not the case. Two cases are then to be considered:
  - **Case 1:** the last element is a parabola: Eq.4 is used, exactly as in [20] (lines 4 to 7 of Algo.2).
  - **Case 2:** the last element is a line segment. Let h be its height. In this case, the intersection with $\mathcal{F}_l$ is obtained by solving the second-order system $\mathcal{F}_l = h$, and keeping the lowest solution (i.e. the one on the left of the parabola). It is then given by:
    $$s_2(h,l) = l - \sqrt{h - g(l,c)^2} \quad (6)$$
    As in the parabola/parabola case, let $w = 1 + s_2(h,l)$ be the nearest integer higher than this intersection. When the start of the line segment follows the intersection (i.e. when $z[k] > w$), it is completely occluded by the parabola, and is therefore removed from the lower envelope. Otherwise, as illustrated (Fig.3(right)), the line segment still contributes to the lower envelope (from $z[k]$ to $w-1$). The parabola is then added to the list such that $z[k+1] = w$ and $v[k+1] = i$.

4.3 Efficient implementation

In the original version of the algorithm (Algo.2), the lower envelope is represented with two arrays v and z storing respectively for each parabola k: *i)* the horizontal coordinate of its summit ($v[k]$), and *ii)* the starting position of its zone of influence ($z[k]$). Given its computational and memory efficiencies, this representation is adapted and completed to fit the proposed combination of parabolas and line segments. First, a third array t is added to store the nature of a given object k

(parabola or line segment). Arbitrarily, $t[k]$ is set to 0 when k is a parabola, and to 1 otherwise. Second, when the object k is a line segment, the array v is recycled to store the (unique) value of g along it (i.e. the height of the line segment). Please refer to Table.1 for a clear understanding of this representation.

Table 1 Proposed representation of the lower envelope. For parabolas, v and z are exactly the same as in [20].

	Parabola	**Line segment**
v[k]	horizontal location of the parabola's center	height of the line segment
z[k]	beginning of the zone of influence	beginning of the zone of influence
t[k]	0	1
		

The structure of the proposed algorithm is given in (Algo.3). To start, the fast forward process is implemented (lines 4 to 10). Once the presence of a line segment has been asserted (lines 4 and 6), it is added in line 7 and the process ends (lines 8 and 9) by adding the last parabola (line 10). Notice that when the test (line 6) is not valid (i.e. when the next parabola of a potential line segment is not at the same height), the algorithm jumps to line 10 and simply adds $\mathcal{F}_i$ as a parabola without computing its intersection with $\mathcal{F}_{i-1}$. The rest of the algorithm, i.e. lines 11 to 20, follows exactly the same structure as the original algorithm (Algo.2). The only differences lay in lines 15 and 20, to hand over the cases where the current considered element of the lower envelope is a line segment. In particular, the intersection with $\mathcal{F}_i$ is calculated (Eq.6) on line 15, while line 20 describes how to determine the DT $\mathcal{D}^2(i,j)$ of a line segment, simply given by its height $v[k]$.

Algorithm 3: Proposed algorithm

input : 1-dimensional distance transform $g(\mathbf{x})$, for each $\mathbf{x} \in I$
output : Euclidean distance transform $\mathcal{D}$

1 **for** *j=0 to n-1* **do**
2 $k = 0$, $v[k] = 0$, $z[k] = -\infty$, $z[k+1] = +\infty$, $t[k] = 0$, $w = 0$, $i = 1$
3 **while** $i < m$ **do**
4 **if** $g(i-1,j) == g(i,j)$ *and* $w < i$ **then**
5 $i = i + 1$
6 **if** $i < m$ *and* $g(i-1,j) == g(i,j)$ **then**
7 $k = k + 1$, $z[k] = i - 1$, $v[k] = g(i,j)$, $t[k] = 1$
8 **while** $i < m$ *and* $g(i-1,j) == g(i,j)$ **do**
9 $i = i + 1$
10 $k = k + 1$, $z[k] = i - 1$, $z[k+1] = \infty$, $v[k] = i - 1$, $t[k] = 0$
11 $w = 1 + s(v[k], i)$
12 **while** $w \leq z[k]$ **do**
13 $k = k - 1$
14 **if** $t[k] == 0$ **then**
15 $w = 1 + s(v[k], i)$
16 **else**
17 $w = 1 + s_2(v[k], i)$
18 $k = k + 1$, $z[k] = w$, $z[k+1] = \infty$, $v[k] = i$, $t[k] = 0$, $i = i + 1$
19 **for** *i=0 to m-1* **do**
20 **while** $z[k+1] < i$ **do**
21 $k = k + 1$
22 **if** $t[k] == 0$ **then**
23 $\mathcal{D}^2(i,j) = (v[k] - j)^2 + g(v[k], j)^2$
24 **else**
25 $\mathcal{D}^2(i,j) = v[k]$

5 Comparative results

5.1 Methodology

A complete benchmark was conducted to compare the proposed approach with the fastest existing CPU-based algorithms for exact EDT computation. They consist of the solutions offered by:

- Felzenszwalbe et al.[20], which performs an independent scanning based on the lower envelope of parabolas as detailed in Section.3;
- Hirata[3] which is similar to [20] with a slightly different algorithm for constructing the lower envelope;
- Maurer et al.[15] which performs independent scanning based the partial construction of the Voronoi diagram;
- Lucet et al.[22] which performs an independent scanning based on the Legendre Conjugate;
- Schouten et al.[23] that uses a combination of ordered propagation, raster scanning and independent scanning.

With the exception of the last one, all algorithms are directly comparable to the proposed approach, as they are relatively easy to implement, separable, generalizable to higher dimensions, and linear in the number of pixels. The FEED algorithm introduced by Schouten et al. [23] is much more complicated to implement and

Table 2 Computation time results on the [2] dataset, for the algorithms considered in the benchmark

images	Computation Time (in $ns/pixel$) per algorithm											
	Proposed		Felzenswalb		Hirata		Maurer		Lucet		Schouten	
	ave	rms	ave	rms	ave	rms	ave	rms	ave	rms	ave	rms
rotating line	7.86	1.67	7.64	1.61	8.07	1.69	8.20	1.70	6.11	0.3	19.90	8.11
rotating line, inverse	4.20	0.05	6.69	0.07	8.79	0.08	8.65	0.197	9.03	0.07	1.79	0.05
inscribed circle	8.99	0.26	9.23	0.36	9.90	0.14	10.70	0.31	11.98	4.81	44.88	21.68
inscribed circle, inverse	5.09	0.25	7.20	0.17	9.04	0.19	9.09	0.35	12.95	4.62	3.82	0.53
random pixels (see Fig.5)	18.92	6.50	19.21	5.19	18.60	4.15	19.92	5.11	23.10	3.50	18.52	8.73
random squares	6.69	1.51	7.97	0.81	9.29	0.32	9.33	0.40	17.59	0.80	7.10	3.83
point in corner	5.76	0.12	7.81	0.14	9.45	0.14	8.89	0.11	10.35	2.91	1.86	0.07
point in corner, inverse	4.00	0.15	6.43	0.17	8.71	0.20	8.58	0.16	12.18	2.95	1.74	0.05
Lenna edges	16.34		15.42		14.30		15.95		17.05		13.68	
Lenna edges, inverse	7.51		9.07		10.56		10.77		16.36		4.74	

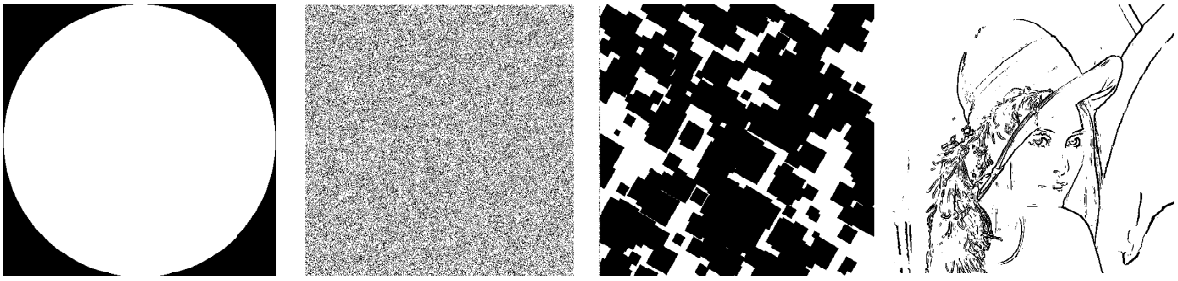


Fig. 4 Samples of the considered shapes extracted from Fabbri's dataset. From left to right: inscribed circle, random pixels (50%), random squares (75%), Lenna edges (inverse). The images have different complexities to test the algorithms in different situations and their dependency on image content.

2.

configure. Furthermore, it is not linear in the image size and has exponential complexity when dealing with higher dimensions (according to Schouten et al.[24]). However, the authors have shown its excellent performances on 2D shapes. Although, we do not consider it to be in the same scope as the proposed algorithm, it was still added to the benchmark. Except for FEED, for which we directly used the source code provided by the authors (about 500 optimized lines with many implementation tricks), all algorithms were re-implemented according to their formulation, and optimized with all known strategies. In particular, as stated in [23], the order of processing (i.e. first the rows and then the columns, or vice-versa) does not affect the accuracy of the result, but it influences the execution time. For all algorithms, the fastest implementation was therefore selected, which in our case was column-row ordering.

To guarantee full control over these assessments and encourage the comparison of the present work with yet-to-be-released algorithms, all source codes (including algorithms and benchmark implementation) can be found on github[1]). Finally, although all the algorithms (except for FEED) are separable, they were implemented without any parallelization on a laptop equipped with an Intel Core i7-5600U CPU @ 2.60GHz. Their execution time could thus be easily reduced on any modern processor with a multi-core architecture.

[1] https://github.com/RomMarie/EnhancedDT

5.2 Fabbri et al. dataset

The first dataset considered here was introduced by Fabbri et al.[2] in a comparative survey to review state-of-the-art sequential 2D EDT algorithms. It considers a variety of artificial 2d binary shapes (see Fig.4), and is designed to highlight the pros and cons of each approach. The results of this dataset are presented in Table 2. Overall, they confirm those presented in [23]. The FEED algorithm outperforms other solutions (including the presented work) in most scenarios, whilst the algorithm proposed by [22] is generally the slowest. Additionally, although the theoretical foundations of both works are exactly the same, the algorithm proposed by [20] is significantly faster than the one initially introduced by [3], with an average gain of around 10%. It validates the algorithmic choice discussed earlier.

Another global observation depicted by Table 2 is the good performance of the proposed approach, as it outperforms other comparable algorithms (except FEED), ranging from 13% for the solution proposed by Felzenswalb, to 41% for Lucet et al. algorithm. However, it is worth noting that this statement is not true for 3 scenarios,

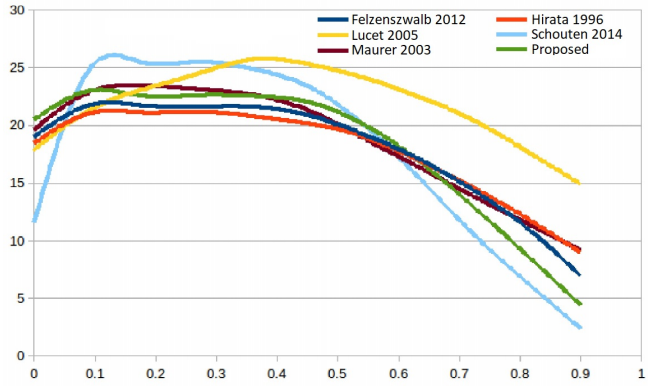


Fig. 5 Evolution of the computational cost (in ns/pixel) for each considered algorithm, when increasing the number of background pixels (randomly generated). Although slower in the beginning, the proposed algorithm (and FEED) outperforms other comparable solutions when the background is sufficiently connected.

namely *random pixels, rotating line and Lenna edges* which require specific attention. Consider for example the case of *random pixels*. In this scenario, a set of images are generated by setting an increasing percentage of random pixels as the background. As detailed in Fig.5, the proposed algorithm is slower than [20] and [3] until about 70% of the image is filled with background points. This behavior clearly highlights the overload induced by mixing line segments with parabolas. As the points are randomly generated, the background is not connected. To generate line segments, the algorithm has to search for non-existent adjacent parabolas, which increases the computational cost by at most 5% (compared to [20] and [3]). From 70%, the background pixels become highly connected. This cost is then largely counterbalanced by the gain induced by the line segments now found, and the proposed approach begins to significantly outperform other algorithms. Finally, we also observe that the FEED algorithm suffers from the same drawback, but to a much greater extent.

Following the same reasoning, the proposed algorithm is also slightly slower when applied on Lenna edges (by definition, edges are thin, and therefore rarely generate line segments) and on the rotating line. Note however that for all these scenarios, the difference remains very small.

5.3 The Kimia's dataset

Although the Fabbri et al. dataset considers a wide range of shapes, with varying geometric properties, it remains artificial and not representative of real-world application cases. Therefore, we also compared the algorithms considered on the dataset proposed in [25]. It is a collection of 216 binary images representing real world entities (objects, animals, ...). The computational cost obtained by averaging a hundred runs of each algorithm on each image is presented in Fig.6.

Overall, the hierarchy in terms of execution time between the considered algorithms is consistent across the entire dataset. More specifically, the FEED algorithm remains the fastest, followed by the proposed solution. The latter is in average 20% faster than the method it extends (i.e. Felzenswalb work), and about 35% faster than Hirata's algorithm. Finally, the other algorithms are significantly slower. Please notice that, as shown in Fig.6, the considered shapes are extremely different in terms of topology and geometry. Although it has a clear influence on the computational cost, the performance order of the algorithms remains unchanged throughout the dataset.

5.4 Discussion

In this section, six algorithms (including the proposed one) have been compared in terms of computational cost. It clearly shows that FEED [23] outperforms the others, whilst the proposed solution consistently comes at the second place. At this point, the reader might thus question the interest of this work. Together with the quantitative comparison, a qualitative reflection is further added.

Implementation difficulty: We spent a considerable amount of time implementing all the benchmark algorithms. As such, we can affirm that among them, the easiest were those proposed by Felzenswalb [20] and Hirata [3], followed by ours, Maurer [15], and finally Lucet [22]. Although a C++ source code is made available by the authors for FEED (as supplementary material), it still consists in around 500 very dense lines, making it by far the most difficult to implement. This is, in our opinion, a major drawback for this algorithm, because its use in a different context (with another programming language or in higher dimension) is far from straightforward. This is certainly not the case for the proposed algorithm.

Parameter tuning: Unlike the other five methods, FEED's performance relies heavily on tuning six parameters. In the source code provided in the supplementary material, it is stated that *"A way to do it is to start with an initial educated guess and then vary P1 to obtain minimal time. Then repeat this with P2 to P6, followed by going back from P5 to P1"*. Although we may have our doubts as to what an "initial educated guess" should be, the results presented in this article for FEED are the ones obtained after spending time working on P1-P6 to obtain the best possible execution

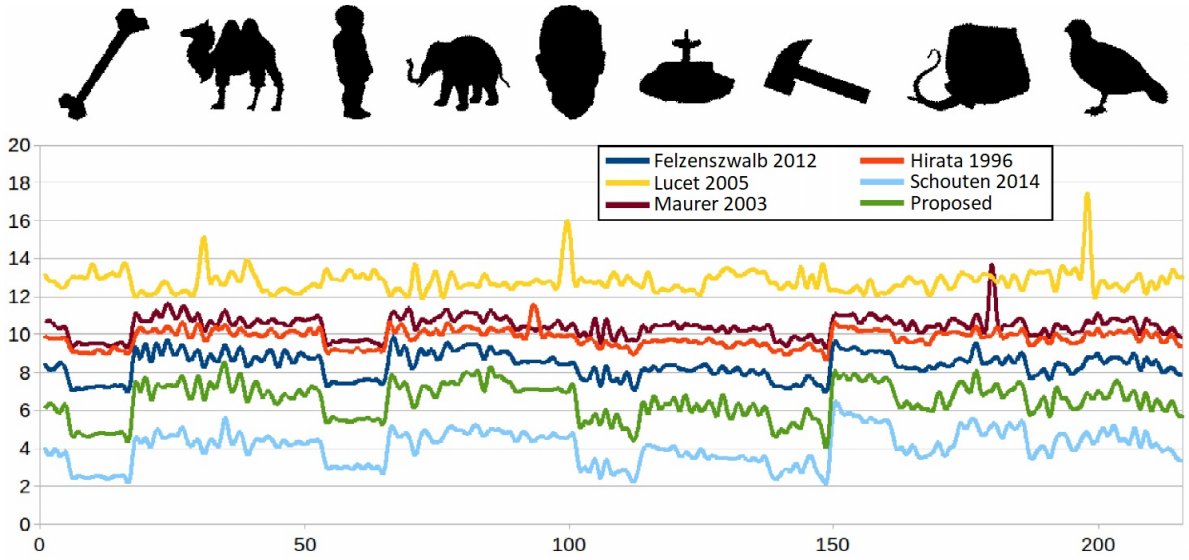


Fig. 6 Computational cost (in ns/pixel) of each considered algorithm, when applied on the 216 images of Kimia's dataset. Note that, although the FEED algorithm is clearly faster, the proposed approach consistently outperforms the other state-of-the-art solutions.

time. As pointed out in [23], the difference between default values and fine-tuning is about 44%, which is huge. While fine-tuning is clearly a strength of FEED (since it optimizes the performance of the algorithm for a given context), it also increases the difficulty of actually using it effectively.

Generalization to higher dimensions: Among the six considered algorithms, only those proposed by Lucet [22] and Schouten [23] cannot be directly generalized to higher dimensions. As described in [24], FEED requires major adaptations (and a new set of parameters) to work efficiently in 3D, and is not yet designed to work in arbitrary dimensions.

Based on these points, the proposed algorithm clearly seems to be an interesting alternative, especially when working in arbitrary dimensions.

6 Conclusion

In this article, we presented a major reformulation to the well-known solution for the Euclidean Distance Transform computation initially introduced by Saito et al. [4]. It leverages the discrete nature of the input shape and the output result to limit the frequency of evaluating the intersection between two parabolas, which is a computationally expensive operation. The resulting algorithm remains linear in the shape size, but allows the constant term to be reduced significantly.

We proposed a comprehensive analysis and benchmarking of several similar state-of-the-art algorithms. Although slower than the FEED algorithm (which is, to our knowledge, the fastest at this stage), it is both easy to implement and easily scalable to higher dimensions. In fact, our ongoing work uses this algorithm as input for robot path planning in 4-dimensional space, where FEED is definitely not an option.

Conflict of interest

The authors declare that they have no conflict of interest.

References

1. Rosenfeld, A., Pfaltz, J.L.: Distance functions on digital pictures. Pattern Recognition **1**(1), 33–61 (1968)
2. Fabbri, R., da Fontoura Costa, L., Torelli, J.C., Bruno, O.M.: 2d euclidean distance transform algorithms: A comparative survey. ACM Comput. Surv. **40**(1), 2:1–2:44 (2008)
3. Hirata, T.: A unified linear-time algorithm for computing distance maps. Information Processing Letters **58**(3), 129–133 (1996)
4. Saito, T., Toriwaki, J.I.: New algorithms for euclidean distance transformation of an n-dimensional digitized picture with applications. Pattern Recognition **27**(11), 1551 – 1565 (1994)
5. Leal, J., Ramírez-Torres, J., Barron-Zambrano, J., Diaz-Manriquez, A., Nuño-Maganda, M.A., Saldivar-Alonso, V.: Parallel raster scan for eu-

clidean distance transform. Symmetry **12** (2020). DOI 10.3390/sym12111808
6. Manduhu, M., Jones, M.W.: A work efficient parallel algorithm for exact euclidean distance transform. IEEE Transactions on Image Processing **28**(11), 5322–5335 (2019)
7. Montanari, U.: A method for obtaining skeletons using a quasi-euclidean distance. Journal of the ACM (JACM) **15**, 600 – 624 (1968)
8. Danielsson, P.E.: Euclidean distance mapping. Computer Graphics and Image Processing **14**(3), 227–248 (1980). DOI https://doi.org/10.1016/0146-664X(80)90054-4
9. Borgefors, G.: Distance transformations in arbitrary dimensions. Computer Vision, Graphics, and Image Processing **27**(3), 321–345 (1984). DOI https://doi.org/10.1016/0734-189X(84)90035-5
10. Cuisenaire, O., Macq, B.M.: Fast euclidean distance transformation by propagation using multiple neighborhoods. Comput. Vis. Image Underst. **76**, 163–172 (1999)
11. Shih, F.Y., Wu, Y.T.: Fast euclidean distance transformation in two scans using a 3×3 neighborhood. Computer Vision and Image Understanding **93**(2), 195–205 (2004)
12. Strutz, T.: The distance transform and its computation (2021)
13. Breu, H., Gil, J., Kirkpatrick, D.G., Werman, M.: Linear time euclidean distance algorithms. IEEE Trans. Pattern Anal. Mach. Intell. **17**(5), 529–533 (1995)
14. Guan, W., Ma, S.: A list-processing approach to compute voronoi diagrams and the euclidean distance transform. IEEE Transactions on Pattern Analysis and Machine Intelligence **20**(7), 757–761 (1998)
15. Maurer, C.R., Qi, R., Raghavan, V., Member, S.: A linear time algorithm for computing exact euclidean distance transforms of binary images in arbitrary dimensions. IEEE Transactions on Pattern Analysis and Machine Intelligence pp. 265–270 (2003)
16. Wang, J., Tan, Y.: Efficient euclidean distance transform algorithm of binary images in arbitrary dimensions. Pattern Recognition **46**(1), 230–242 (2013)
17. Shih, F.Y., Mitchell, O.R.: A mathematical morphology approach to euclidean distance transformation. IEEE Trans. Image Processing **1**(2), 197–204 (1992)
18. Lotufo, R., Zampirolli, F.A.: Fast multidimensional parallel euclidean distance transform based on mathematical morphology. In: 14th Brazilian Symposium on Computer Graphics and Image Processing (SIBGRAPI 2001), 15-18 October 2001, Florianopolis, Brazil, pp. 100–105 (2001)
19. Meijster, A., Roerdink, J.B.T.M., Hesselink, W.H.: A general algorithm for computing distance transforms in linear time. In: International Symposium on Mathematical Morphology and its Applications to Image and Signal Processing, ISMM 2000, Palo Alto, CA, USA, June 26-28, 2000, pp. 331–340 (2000)
20. Felzenszwalb, P.F., Huttenlocher, D.P.: Distance transforms of sampled functions. Theory of Computing **8**(1), 415–428 (2012)
21. Coeurjolly, D., Montanvert, A.: Optimal separable algorithms to compute the reverse euclidean distance transformation and discrete medial axis in arbitrary dimension. IEEE Trans. Pattern Anal. Mach. Intell. **29**(3), 437–448 (2007)
22. Lucet, Y.: A linear euclidean distance transform algorithm based on the linear-time legendre transform. In: Second Canadian Conference on Computer and Robot Vision (CRV 2005), 9-11 May 2005, Victoria, BC, Canada, pp. 262–267 (2005)
23. Schouten, T.E., van den Broek, E.L.: Fast exact euclidean distance (FEED): A new class of adaptable distance transforms. IEEE Trans. Pattern Anal. Mach. Intell. **36**(11), 2159–2172 (2014)
24. Schouten, T., C. Kuppens, H., van den Broek, E.L.: Three dimensional fast exact euclidean distance (3d-feed) maps. Pattern Recognition Letters - PRL **6066** (2006). DOI 10.1117/12.643721
25. Sebastian, T.B., Kimia, B.B.: Curves vs. skeletons in object recognition. Signal Processing **85**(2), 247–263 (2005)