Fast Multidimensional Parallel Euclidean Distance Transform based on Mathematical Morphology

ROBERTO A LOTUFO
FRANCISCO A ZAMPIROLLI

FEEC-Faculdade de Engenharia Elétrica e de Computação
6101 - 13083-970 - Campinas, SP, Brasil
{lotufo,fz}@dca.fee.unicamp.br

Abstract. This paper presents a novel Euclidean distance transform algorithm formulated under the Mathematical Morphology approach. The distance transform is an erosion by a structuring function dependent on the distance metric used. To achieve high speed performance, the squared Euclidean distance structuring function is decomposed into a family of four one-dimensional two-point structuring functions. The erosion algorithm is based on a propagation scheme which resulted in a overall Euclidean distance transform algorithm very simple to code and understand, yet with speed performance compared to the Chamfer 3-5-7 sequential raster and anti-raster algorithm.

1 Introduction

Distance Transform (DT) is a powerful image processing transformation that assigns to pixels of a binary image its distance to the background pixels. It can be used for a variety of binary image operations such as computing Voronoi regions, image classification, skeletons, dilation, erosion, binary image interpolation, matching etc. The DT was first introduced by Rosenfeld and Pfaltz [RP68]. The most natural metric for computing distance in most applications is the Euclidean metric, mainly because of its rotation invariance property. However, due to the difficulty to implement efficient algorithms for the Euclidean Distance Transform (EDT), many researchers developed algorithms to compute approximate EDT and the most popular is known as Chamfer metric DT algorithm [Bor86]. Its popularity is due to its simplicity, good speed performance and reasonable approximation to the EDT. The few exact EDT algorithms reported in the literature are either inefficient or complex to implement and understand.

In 1992, Ragnemalm [Rag92] described a propagation algorithm to compute the EDT. Recently, Eggers [Egg98] improved the speed of the propagation algorithm but increased the complexity of the coding.

Shih and Mitchel, in 1992 [SM92], have shown that the DT can be computed by a morphological erosion of the input image by a structuring function given by the negative of the distance to the origin. Later, in 1994, Huang and Mitchel [HM94] arrived at an efficient computation of the square of the EDT by decomposing the structuring function by a sequence of 3×3 different structuring functions.

The work by Mitchel and collaborators has opened a door to study and classify the diversity of distance transform algorithms available in the literature. As there are many ways to implement efficiently the morphological erosion using parallel, recursive and propagation algorithms and using different ways to decompose the structuring function, Zampirolli and Lotufo [ZL00] proposed a classification of the Distance Transforms algorithms by analyzing which erosion algorithm and which structuring function decomposition scheme was used in each paper. As an additional result of this classification, a new EDT algorithm, described here, use a one-dimensional decomposition of the 3x3 structuring functions and a fast 1-D directional propagation erosion algorithm. The algorithm is separable such that higher dimensions EDT can be computed from the 1-D EDT algorithm, exhibiting the same parallelism behavior of higher dimensions FFT algorithms.

The main features of our proposed algorithm are summarized as follows:

- Exact discrete Euclidean Distance Transform.
- Well suited to higher dimensions.
- Code simplicity when compared to other efficient EDT algorithms.
- One of the fastest known algorithm for general serial computers. Its speed is comparable to the recursive Chamfer distance transform implementation.
- Efficient parallelization.

This paper is organized as follows. Section 2 reviews the computation of the Distance Transform using a gray-scale morphological erosion. Section 3 presents efficient ways to decompose the squared EDT structuring functions and shows two erosion implementations, parallel and by propagation. In Section 4, we discuss the speed analysis

1530-1834/01 $10.00 © 2001 IEEE

and show comparisons with other distance transform algorithms. Finally, Section 5 give the conclusions.

2 Definitions and notation

Let $\mathbf{Z}$ be the set of integer numbers, $\mathbf{E} \subset \mathbf{Z}^2$ the image domain and $K = [0, k] \subset Z$ and interval of integer numbers representing the possible gray-scale values in the image. The gray-scale erosion operator, $\varepsilon_b : K^{\mathbf{E}} \to K^{\mathbf{E}}$, is defined as [Hei91]:

$$\varepsilon_b(f)(x) = \min\{f(y) \dot{-} b(y - x) : y \in (B + x) \cap \mathbf{E}\}, \quad (1)$$

where $f \in K^{\mathbf{E}}$, $x \in \mathbf{E}$, $B \in \mathcal{P}(\mathbf{Z}^2)$ ($\mathcal{P}(\mathbf{E})$ is the set of parts of $\mathbf{E}$ and B is called *structuring element*), $B + x = \{y + x,\ y \in B\}$ (translation of B by x) and b is a *structuring function* defined in B by $b : B \to Z$.

2.1 Euclidean distance

The squared Euclidean distance between two pixels $p = (p_1, p_2)$ and $q = (q_1, q_2)$, p and $q \in \mathbf{Z}^2$ is given by:

$$d_e^2(p, q) = (q_x - p_x)^2 + (q_y - p_y)^2.$$

We define *distance function of a pixel x to a set X*, as:

$$d(x, X) = \min\{d(x, y) : y \in X\}.$$

The *squared Euclidean distance function* (denoted by $\Psi_e^2(f)$ or simply EDT^2) is defined as:

$$\Psi_e^2(f)(x) = d_e^2(x, \{y \in \mathbf{E} : \mathbf{f(y) = 0}\}), \quad (2)$$

that assigns to each pixel the minimum squared distance to the image background.

2.2 EDT by erosion

Shih and Mitchel [SM92] have shown that the EDT^2 can be exactly computed by the morphological erosion of the binary input image with values 0 and ∞ using a structuring function b_e:

$$\Psi_e^2(f) = \varepsilon_{b_e}(f), \quad (3)$$

where the structuring function is given by:

$$b_e(x) = -d_e^2(x, \mathcal{O}), x \in \mathbf{E} \oplus \mathbf{E}, \quad (4)$$

where $\mathcal{O} = (0, 0)$.

A useful property of this particular erosion is its idempotency:

$$\varepsilon_{b_e}(\varepsilon_{b_e}(f)) = \varepsilon_{b_e}(f). \quad (5)$$

3 Efficient erosion algorithms

There are two ways to improve the efficiency of the erosion algorithms. By decomposing the structuring function in smaller structuring functions and by using different erosion algorithms, where the main classifications are parallel, sequential and propagation.

3.1 Structuring function decomposition

The direct application of the structuring function b_e is inefficient. A usual decomposition scheme, applied to connected structuring elements is to use the Minkowski addition, in the following way [Ser82, SM92]. If b_e can be written as:

$$b_e = b_1 \oplus b_2 \oplus b_3 \cdots \quad (6)$$

then,

$$\varepsilon_{b_e}(f) = \cdots \varepsilon_{b_3}(\varepsilon_{b_2}(\varepsilon_{b_1}(f))), \quad (7)$$

where $\oplus$ is the gray-scale Minkowski addition:
$\forall x \in B_i \oplus B_j$,

$$(b_i \oplus b_j)(x) = \max\{b_i(y) + b_j(x - y) : y \in (\breve{B}_j + x)\}, \quad (8)$$

where $j = 1, \ldots k$, $\breve{B}_j = \{x \in \mathbf{E} : -x \in B_j\}$ is the reflection of B_j, and $B_i \oplus B_j$ is the set Minkowski addition[1].

It is important to note that the order of the erosion application does not matter as the Minkowski addition is commutative.

Shih and Mitchel have shown that b_e used in the EDT^2 can be decomposed in a sequence of decreasing 3×3 structuring functions b_i:

$$b_i = \begin{bmatrix} -4i+2 & -2i+1 & -4i+2 \\ -2i+1 & \mathbf{0} & -2i+1 \\ -4i+2 & -2i+1 & -4i+2 \end{bmatrix}, \quad (9)$$

where the origin, at the center, is marked in bold and $i \in \{1, 2, \ldots\}$.

In this paper, we call the attention that b_i can be further decomposed in four one-dimensional 2-point structuring functions, two in the vertical directions, North (b_{Ni}), and South (b_{Si}), and two in the horizontal directions, East (b_{Ei}), and West (b_{Wi}):

$$b_{Ni} = \begin{bmatrix} -2i+1 \\ \mathbf{0} \end{bmatrix}, \quad b_{Ei} = \begin{bmatrix} \mathbf{0} & -2i+1 \end{bmatrix},$$
$$b_{Si} = \begin{bmatrix} \mathbf{0} \\ -2i+1 \end{bmatrix}, \quad b_{Wi} = \begin{bmatrix} -2i+1 & \mathbf{0} \end{bmatrix}. \quad (10)$$

[1] Note that the same symbol is used for the set addition and numerical addition.

The structuring function for the EDT^2 can be decomposed in:

$$b_e = \cdots \oplus b_{N2} \oplus b_{N1} \oplus \cdots \oplus b_{S2} \oplus b_{S1} \oplus \cdots \cdots \oplus b_{W2} \oplus b_{W1} \oplus \cdots \oplus b_{E2} \oplus b_{E1}. \quad (11)$$

This decomposition leads to two important consequences. Firstly the EDT^2 can be separated in simpler 1-D erosions, which brings independence to the computation of each image line or column. Secondly, the structuring functions are reduced to two-point directional structuring function, allowing simple and efficient algorithm implementations.

This means that the EDT^2 can be computed by eroding each column of the image by $b_{N1}, b_{N2}, \ldots$ until stability using the idempotency property, then similarly eroding the columns by $b_{S1}, b_{S2}, \ldots$, then eroding the rows by $b_{E1}, b_{E2}, \ldots$, and finally, eroding the rows by $b_{W1}, b_{W2}, \ldots$.

3.2 Parallel erosion

In the parallel algorithms, the pixels are processed independently of the way the pixels are scanned. The output pixels depend only on the input image pixels and on the structuring function.

The parallel erosion algorithm can be written in pseudo code as:

```
Function  g = eroPar(f, b)
  for all x ∈ E in parallel
    g(x) = min{f(y) − b(y − x) : y ∈ (B + x)};
```

where the input and output images are f and g respectively, and b is the structuring function. This leads to the following EDT^2 algorithm, which is one of the simplest exact Euclidean distance in the literature:

```
Function g = edt(f)
  {f is assumed to be a binary image with values 0 and M,
  where M is the maximum possible squared distance
  in the image}
  for each column c
    {First step, vertical erosions}
    for b = 1, 3, 5, 7, ... until stability
      for each row r
        N(r, c) = min{f(r, c), f(r − 1, c) + b};
    for b = 1, 3, 5, 7, ... until stability
      for each row r
        S(r, c) = min{N(r, c), N(r + 1, c) + b};
  for each row r
    {Second step, horizontal erosions}
    for b = 1, 3, 5, 7, ... until stability
      for each column c
        E(r, c) = min{S(r, c), S(r, c + 1) + b};
    for b = 1, 3, 5, 7, ... until stability
      for each column c
        g(r, c) = min{E(r, c), E(r, c − 1) + b};
```

The inefficiency of this algorithm is due to unnecessary scanning in the areas of the image where the erosion is not affected. Better efficiency can be accomplished with propagation algorithms.

3.3 Erosion by propagation

The idea of the erosion algorithm by propagation is to process only the neighborhood of the pixels that may change in the erosion. This set of pixels is called *front* or *border* of f, denoted by ∂f_b. The efficiency of the propagation algorithm is increased when a sequence of erosions is computed. In this case, the front for the next erosion is computed during the previous erosion.

Below is the pseudocode of the erosion by propagation.

```
Function [g, ∂g_b] = eroPro(f, b, ∂f_b)
  {g and ∂g_b are output parameters}
  g = f;
  for all x ∈ ∂f_b
    for all y ∈ (B + x) ∩ E
      if g(y) > f(x) − b(x − y)
        g(y) = f(x) − b(x − y);
        if y ∉ ∂g_b,  set_in(∂g_b, y);
```

where $set\_in(\partial g_b, y)$ is the function that inserts y in the set ∂g_b. Observe that the pixel is inserted in the front only once. It is possible to generalize the erosion by propagation to use a sequence of erosions by a non-crescent family of structuring functions, i.e., $b_1 \geq b_2 \geq \cdots \geq b_k$.

For the erosions using the 2-point decomposed family of structuring functions, it is possible to further improve the efficiency of the propagation algorithm. These improvements are different for each of the two steps: the first vertical erosion and the second horizontal erosion.

For the first vertical erosion, the pseudo code is given below:

```
Function edt1(f)
  {f is input and output, H is image height}
  for each column c
    b = 1;
    for each row r = 2, ..., H
      if f(r, c) > f(r − 1, c) + b
        f(r, c) = f(r − 1, c) + b;
        b = b + 2;
      else
        b = 1;
    b = 1;
    for each row r = H − 1, H − 2, ..., 1
      if f(r, c) > f(r + 1, c) + b
        f(r, c) = f(r + 1, c) + b;
        b = b + 2;
      else
        b = 1;
```

As the input image is binary, the front can be propagated in the same direction of the structuring function using a sequential processing so that the pixel under processing is a function of the previous modified pixel. In this way the vertical erosion can be computed using a single raster and an anti-raster scan.

For the second part, the horizontal erosion, it is not possible to make a sequential processing wave, but it is possible to avoid the copying of the images and process the image in place as long as the order of processing be the opposite order of the front propagation. When using the East direction, the raster order must be West, so that the modified pixel will not be used in that scan. This is achieved using a FIFO queues, two for each direction. The speed efficiency of this algorithm is improved by the fact that only the front pixels are processed until stability. The algorithm for the second part is given below in pseudo code:

```
Function edt2(f)
  {f is input and output, W is image width}
  for each row r
    for c = W − 1 until 1 step −1,  insertQueue(Eq, c);
    for c = 2 until W,  insertQueue(Wq, c);
    b = 1;
    while (notEmptyQueue(Wq) or
                notEmptyQueue(Eq))
      while notEmptyQueue(Eq)
        c = fromQueue(Eq);
        if f(r, c + 1) > f(r, c) + b
          f(r, c + 1) = f(r, c) + b;
          if c + 1 < W
            insertQueue(Eq2, c + 1);
      while notEmptyQueue(Wq)
        c = fromQueue(Wq);
        if f(r, c − 1) > f(r, c) + b
          f(r, c − 1) = f(r, c) + b;
          if c − 1 > 1
            insertQueue(Wq2, c − 1);
      b = b + 2;
      Wq = Wq2;
      Eq = Eq2;
```

We call the attention that the queues used in this algorithm have a maximum size of the width of the image. For best performance, these queues can be easily implemented by fixed length integer vectors.

Example

To better illustrate the EDT^2 algorithm using erosions, a simple example f with a 4x4 image is shown below. The result of the vertical erosion by the structuring function B_v is shown next. Finally, f_1 is the first horizontal erosion and f_2 is the second horizontal erosion and the final squared Euclidean distance transform. In the results of the horizontal erosions, the pixels inserted in the propagation queue are marked in bold.

$$f = \begin{pmatrix} \infty & \infty & 0 & \infty \\ \infty & \infty & \infty & \infty \\ \infty & 0 & \infty & \infty \\ 0 & \infty & \infty & \infty \end{pmatrix}, \quad B_v = \begin{pmatrix} -5 \\ -3 \\ -1 \\ 0 \\ -1 \\ -3 \\ -5 \end{pmatrix},$$

$$f_v = f \ominus B_v = \begin{pmatrix} 9 & 4 & 0 & \infty \\ 4 & 1 & 1 & \infty \\ 1 & 0 & 4 & \infty \\ 0 & 1 & 9 & \infty \end{pmatrix}, \tag{12}$$

$$B_{h1} = \begin{pmatrix} -1 & 0 & -1 \end{pmatrix},$$

$$f_1 = f_v \ominus B_{h1} = \begin{pmatrix} \mathbf{5} & \mathbf{1} & 0 & \mathbf{1} \\ \mathbf{2} & \mathbf{1} & 1 & \mathbf{2} \\ 1 & 0 & \mathbf{1} & \mathbf{5} \\ 0 & 1 & \mathbf{2} & \mathbf{10} \end{pmatrix}, \tag{13}$$

$$B_{h2} = \begin{pmatrix} -3 & 0 & -3 \end{pmatrix},$$

$$f_2 = f_1 \ominus B_{h2} = \begin{pmatrix} \mathbf{4} & 1 & 0 & 1 \\ 2 & 1 & 1 & 2 \\ 1 & 0 & 1 & \mathbf{4} \\ 0 & 1 & 2 & \mathbf{5} \end{pmatrix}. \tag{14}$$

4 Speed analysis and comparison

The first step of the algorithm requires a raster and an anti-raster scan with the neighborhood of two pixels each. The speed performance of the second step is more complex. The best case situation requires a raster and an anti-raster scan with neighborhood of two pixels, which occurs with an image with equal columns, where no horizontal propagation is required. In a typical case, the speed will depends on how many times the pixel goes into the propagation queue. The worst case happens with a square image with a diagonal line of zeros. In this situation the number of times a pixel is inserted in the queue is in average $W/4$ times, where W is the width of the image. Although this makes this algorithm proportional to the square of the image dimensions, for typical images, the pixel appears in the queue approximately 1.3 times. In conclusion, counting the first and second steps, the algorithm requires 9.3 pixel accesses for real case images. The performance of the algorithm degrades when the image has large distances to diagonal features.

For illustration, we show a speed comparison using other four different Distance Transform algorithms in Table 1. LZ is the algorithm proposed here, $Eggers$ [Egg98],

	$img1$	$img2$	$img3$	$img4$	$img5$
LZ	0.033	0.029	0.031	0.171	0.054
Egg	0.226	0.140	0.168	0.260	0.213
Cha	0.068	0.021	0.021	0.068	0.036
Dan	0.055	0.048	0.049	0.053	0.051
Box	0.080	0.023	0.022	0.079	0.042

Table 1: Time in seconds of several algorithms applied to five different images (see text).

	64K	256K	1M	4M
LZ	0.016	0.090	0.390	1.69
Box	0.017	0.088	0.368	1.45

Table 2: Time in seconds of the proposed algorithm and the chess-board DT applied to the real image ($img5$) replicated by a factor of 4, 16, 32 and 64 to illustrate the scaling behavior of the algorithm for a typical image.

is an exact EDT based on propagation and Cha is the Chamfer 5-7-11 [Bor86]. Although this algorithm is an approximation of the EDT, we include it here as it is one of the most popular distance transform algorithms. Dan is one of the first Euclidean DT algorithms that although it is not exact, it is still well known among the community. Finally, Box is a distance transform using the chess-board metric using a graph searching algorithm presented by the authors in [LFZ00]. This algorithm is much faster than the first part of the linear-time EDT algorithm recently published by O. Cuisenaire [Cui99], using graph searching techniques.

The images used are of size 256×256, where $img1$ is an image with a single background pixel at the center, $img2$ is an image with random squares of different sizes, $img3$ is an image of random circles of different sizes, both with 20% of foreground pixels, $img4$ is an image with a diagonal background line and $img5$ is a real image. The experiments of Table 1 were made in a Sun Ultra 5 SparcStation, 270MHz, 128MB RAM and the algorithms coded in the ANSI C.

To illustrate the behavior of the algorithm with images of different sizes, the real image ($img5$) was replicated by a factor of 4, 8, 16 and 64 and a comparison was made using the Box chess-board DT. This time the execution time was measured in a Pentium III, 750MHz, 128MB RAM notebook.

From the results presented in Table 1, we can conclude that the proposed algorithm has a speed performance comparable to some of the best known Euclidean distance transform approximation algorithms.

5 Conclusions and comments

We have shown a novel exact multidimensional parallel Euclidean Distance Transform algorithm based on morphological erosions by family of decomposed 1-D directional structuring functions of size 2. The 1-D structuring function decomposition allows independence of lines and columns making the algorithm very suitable for parallel processing and easily extended for higher dimensions. The erosion by propagation algorithm uses a fixed size propagation queue allowing simple and efficient implementation.

We have also confirmed that the framework of Mathematical Morphology is very suitable to the understanding and designing of efficient distance transform algorithms. The pseudo-code presented in this work is one of the simplest EDT algorithms reported in the literature, both for coding and understanding, with a typical speed performance of about 9 accesses per pixel.

For future work, we will implement a multidimensional version of the algorithm and investigate the quadratic behavior of the worst case condition of the proposed algorithm and compare it to a recently published paper [MRH00] which claims a linear time behavior using similar 1-D propagations.

6 Acknowledgments

Francisco A. Zampirolli is supported by FAPESP Ph.D. scholarship under process n. $98/06641 - 6$.

References

[Bor86] G. Borgefors. Distance transformations in digital images. *Computer Vision, Graphics and Image Processing*, 34:344–371, 1986.

[Cui99] O. Cuisenaire and B. Macq. Fast Euclidean distance transformation by propagation using multiple neighborhoods. *Computer Vision Image Understanding*, 76, 163–172, 1999.

[Egg98] H. Eggers. Two fast Euclidean distance transformations in Z^2 based on sufficient propagation. *Computer Vision and Image Understanding*, 1998.

[Dan80] P.E.. Danielson. Euclidean Distance Mapping *Computer Vision and Image Understanding*, 1980.

[Hei91] H. J. A. M. Heijmans. Theoretical aspects of gray-level morphology. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(6),568–582, 1991.

[HM94] C. T. Huang and O. R. Mitchell. A Euclidean distance transform using gray scale morphology

decomposition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16, 443–448, 1994.

[LFZ00] R. Lotufo and A. Falcão and F. Zampirolli. Fast Euclidean distance transform using a graph-search algorithm. In *SIBGRAPI2000*, Gramado-RS-Brazil, 17-20 October 2000.

[MRH00] A. Meijster and J.B.T.M. Roerdink and W.H. Hesselink. A general algorithm for computing distance transforms in linear time. J. Goutsias, L. Vincent, D.S. Bloomberg (eds.) Mathematical Morphology and Its Applications to Image and Signal Processing, Kluwer, 2000.

[Rag92] I. Ragnemalm. Neighborhoods for distance transformations using ordered propagation. *CVGIP: Image Understanding*, 1992.

[RP68] A. Rosenfeld and J. L. Pfalz. Distance functions on digital pictures. *Pattern Recognition*, 1:33–61, 1968.

[Ser82] J. Serra. *Image Analysis and Mathematical Morphology*. Academic Press, London, 1982.

[SM92] F. Y. C. Shih and O. R. Mitchell. A mathematical morphology approach to Euclidean distance transformation. *IEEE Transactions on Image Processing*, 1:197–204, 1992.

[ZL00] F. Zampirolli and R. Lotufo. Classification of the distance transformation algorithms under the mathematical morphology approach. In *SIBGRAPI2000*, Gramado-RS-Brazil, 17-20 October 2000.