Fast Euclidean Distance Transform using a Graph-Search Algorithm

ROBERTO A LOTUFO[1], ALEXANDRE A FALCÃO[2], FRANCISCO A ZAMPIROLLI[1]

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

[2] IC-Instituto de Computação
6101 - 13083-970 - Campinas, SP, Brasil
afalcao@dcc.unicamp.br

Abstract. Two new Euclidean Distance Transform algorithms are described. The algorithms are designed using a shortest path graph-search framework. The Distance Transform can be seen as the solution of a shortest path forest problem. Previous works have dealt with the Euclidean Distance Transform (EDT) and with the shortest path forest problem, but none of them have presented a EDT using the graph-search approach. The proposed algorithms are very simple and yet belong to the class of one of most efficient sequential algorithms. The algorithms easily extend to higher dimensions.

1 Introduction

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

Sharaiha [SC94] proposed a graph-based approach to compute DTs. The graph-based DT can be reduced to the problem of finding the shortest path forest with the tree root vertices as the external object contour pixels. Sharaiha described a graph-based Chamfer DT and pointed out the main advantages of using the graph-theoretic approach as opposed to the pixel-by-pixel based approaches: the graph-size is normally smaller than the image size; each pixel is normally addressed only once; the algorithm can be readily extended to higher dimensions and to different grid topologies (e.g., hexagonal grids).

This paper presents two simple and efficient exact EDT algorithms using the graph-search framework. The major difficulty to compute the EDT using a graph is in to define exact arc lengths using a small local neighborhood. The solution used in the first algorithm is to use the distance as a vector with its x and y components for the 2D case and defining appropriate arc lengths in a given neighborhood. The second EDT algorithm uses explicitly the nearest root vertex to compute the Euclidean distance of each point. This approach is equivalent to having many arc lengths between two vertices, each arc length is associated to the Euclidean distance increment with reference to a specific root vertex.

Both algorithms are some of the simplest EDT reported in the literature and yet are very efficient. Their execution time is comparable to two of the fastest EDT algorithms for serial computers, based on chain propagations: Ragnelmalm [Rag92] and Eggers [Egg98].

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

- If an appropriate neighborhood is chosen, then they give the exact discrete Euclidean Distance Transform.
- They are easily extended to higher dimensions.
- They are very simple and based on shortest path search algorithms.
- They run very efficiently in general serial computers. Their efficiency is directly dependent on the implementation of the ordered queue.

This paper is organized as follow. Section 2 describes the basic definitions and notations related to images and graphs. Section 3 discusses the Distance Transform problem as the shortest path forest problem. Section 4 describes the first

0-7695-0878-2/00 $10.00 © 2000 IEEE

Euclidean Distance Transform algorithm while Section 5 describes the second one. Section 6 discusses some issues related to the implementation and comparison to other Distance Transform algorithms. Section 7 concludes this work.

2 Definitions and notation

A *two-dimensional binary digital image* $f : E \rightarrow \{0,1\}$ is a mapping of a rectangular unit square grid of $n \times m$ points, $E = [1,\ldots,n] \times [1,\ldots,m]$, to the values 0, for background pixels or 1, for object pixels. A picture element, *pixel* $f(x,y)$ of the binary image f has value 0 or 1 at the integer *coordinates* (x,y).

Vector *addition* of two pixel coordinates p and q is given by $p+q = (p_x+q_x, p_y+q_y)$. Similarly the pixel vector *subtraction* is given by $p-q = (p_x-q_x, p_y-q_y)$.

Two pixels coordinates $p = (p_x,p_y)$ and $q = (q_x,q_y)$ are *N-adjacent* if $q \in N(p)$ where $N(.)$ is a *neighborhood relationship*. An example of adjacency representing the four horizontal and vertical neighbors of a pixel is $N_4(p) = \{p+(0,1), p+(1,0), p+(-1,0), p+(0,-1)\}$.

The Euclidean distance of two pixels at positions p and q is the magnitude of its pixel coordinates subtraction:

$$d(p,q) = (d_x,d_y) = |q-p| = \sqrt{(q_x-p_x)^2+(q_y-p_y)^2}.$$

In a binary image f, the set of *object pixels* O is given by the pixels with value 1 and the set of *background pixels*, by the complement set $\bar{O}$. The set of the *external object contour pixels* C is formed by all the background pixels which have an object pixel in its N_4 neighborhood.

A binary image f and a neighborhood relationship N can generate a *graph* $G = (V,A)$ composed of two sets V and A where V is the set of *vertices* formed by the object and its external contour pixels $V = O \cup C$; and A is the set of *arcs* $(p,q), p,q \in V$ associated to a pair of N-adjacent vertices. A graph is *weighted* if an *arc length* $w(p,q)$ is associated to each arc, and it is called *directed* if $w(p,q) \neq w(q,p)$. Two vertices p,q are *adjacent* if (p,q) is in A.

A *path* $(p \rightsquigarrow q)$ is a list of unique vertices $v_1, v_2, \ldots, v_n$ where $v_1 = p, v_n = q, (v_i, v_{i+1}) \in A$. The *path length* $D(p,q)$ in a weighted graph is given by the sum of each arc length in the path:

$$D(p,q) = \sum_{i=1}^{n-1} w(v_i, v_{i+1}).$$

Two vertices p and q are *connected* if there is at least one path between p and q. A *connected graph* is a graph where all pair of vertices are connected. A graph has a *cycle* if it has a path $p \rightsquigarrow q$ and an arc (q,p) not in the path. A *tree* is a connected graph with no cycles. A *rooted tree* is a tree which has a node s called root. A *forest* is a collection of trees.

3 Distance Transform as the shortest path forest problem

A *distance transform* is a mapping from a binary image to a *distance image*. A distance image is an image where each object pixel value represents the distance to its nearest external contour pixel.

The *shortest rooted tree path forest* is a classical problem in graph theory. The problem is to find a forest partitioning of a graph such that the sum of all shortest path lengths from any tree vertex to its root is minimum. The result is a disjoint set of shortest path trees. For simplicity we will denote it as the shortest path forest problem.

The distance transform can be formulated as a graph-theoretic problem in the following way. Build a graph from the binary image and a neighborhood relationship. Assign arc lengths to represent the desired distance metric. The tree roots are the external contour pixels of the object. Find the shortest path from each pixel to the nearest root pixel. The distance image is the image which pixel value is the path length to the nearest background pixel (root).

An example using the N_4 neighborhood relationship and the city-block distance is illustrated in Fig. 4. The binary image is formed by object and background vertices (indicated by a cross and a circle in Fig. 4a, respectively). The contour vertices (indicated by a square) are background vertices with a 4-neighbor object pixel, and represents the roots of the trees. The arcs of the graph are indicated by dashed arrows and are weighted 1 as also shown in Fig. 4a. The final minimum path forest is represented by the arrows in solid lines in Fig. 4b. The distance length (represented by the number nearest to each object vertex in Fig. 4b) is the number of arcs to reach a root vertex. Note that there may be many equivalent solutions to the minimum path forest, but the distance transform is unique.

Chessboard distance transform can be calculated similarly using a N_8 neighborhood relationship and arc lengths of 1 as shown in Fig. 4b. Chamfer distance transform can also be computed by changing the N-adjacency and setting proper arc lengths [SC94].

4 Algorithm 1: Euclidean Distance Transform

To compute the Euclidean Distance Transform using the framework of the graph-search algorithms it is necessary to define the arc lengths. The solution adopted in this first algorithm is to represent the distance by its vector projections D_x and D_y and represent the arc lengths by their x and y projections. The weights associated with the arcs leaving a vertex are displacement vectors with positive values in an 8-neighborhood as shown in Fig 4. The scalar distance of each object pixel can be computed by

$$|D| = \sqrt{D_x^2 + D_y^2}$$

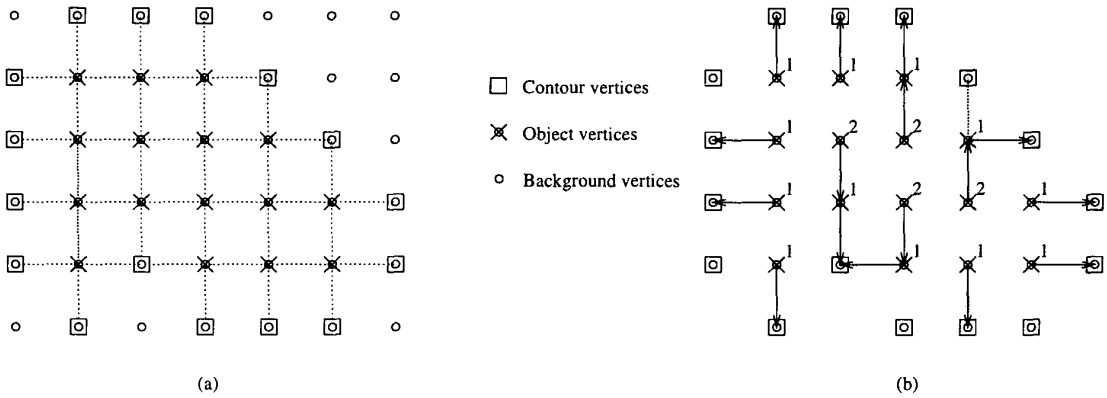

Figure 1: Distance Transform as a minimum path forest problem. City-block metric (4-connected)

$$\begin{array}{ccc} & 1 & \\ 1 & D & 1 \\ & 1 & \end{array} \qquad \begin{array}{ccc} 1 & 1 & 1 \\ 1 & D & 1 \\ 1 & 1 & 1 \end{array}$$

$(a)w_4$ $\qquad$ $(b)w_8$

Figure 2: Weights associated to the arcs of a node with distance D for: a) city-block (w_4); and b) chessboard (w_8) metrics

$$\begin{array}{ccc} (1,1) & (0,1) & (1,1) \\ (1,0) & (D_x, D_y) & (1,0) \\ (1,1) & (0,1) & (1,1) \end{array}$$

Figure 3: Weights ds_8 associated to the 8 arcs of a node with distance D_x, D_y

The accumulative distance associated to the vertex p is given by the vector addition of the accumulative distance of the 8-neighboring vertex v by the weight $ds_8(\mathrm{p}, \mathrm{v})$.

$$D(\mathrm{p}) = D(\mathrm{v}) + ds_8(\mathrm{p}, \mathrm{v})$$

The algorithm is presented in Fig. 4. This algorithm is based on Moore shortest path forest [Moo57] algorithm, which is very similar to the well known Dijkstra shortest path [Dij59].

This algorithm works with two set of vertices: temporary (T) and permanent (P). Initially all vertices are set as temporary (line 4) and as the algorithm evolves, the vertices are transformed into permanent (line 7). A remarkable property of this algorithm is that once a vertex is permanent, its distance is the final optimal shortest path. Another important property is that the order that the vertices turn from temporary into permanent generate non-decreasing distances. This algorithm has two sections: initialization (lines 1 to 4) and propagation (lines 5 to 10).

In the initialization, all vertices are set as temporary (line 4), the contour vertices have their distance assigned as 0 (line 2) and object vertices have distance assigned to infinity (line 3).

The propagation step works while there is a temporary vertex (line 5). Line 6, which is the most computation demanding part, finds a temporary vertex with the minimum distance value. We will see later that, this step can be conveniently optimized if implemented by a bucket sorting technique [Dia69]. Once a vertex with minimum distance is selected from the temporary vertices, it is transformed in a permanent vertex (line 7). Lines 8, 9 and 10 update the temporary vertices p which are 8-neighbors of the new permanent vertex v. Line 9 verifies if the distance computed through the permanent vertex v is shorter than the temporary distance associated with vertexp. If true, line

10 updates $D(\mathrm{p})$. $d_8(i)$ are the vector displacements of a pixel to its 8-neighbors and $ds_8(i)$ are their correspondent arc lengths as indicated in Fig. 4.

5 Algorithm 2: Euclidean Distance Transform

Another way to compute the Euclidean Distance Transform using the graph-search framework, is to represent the arc lengths implicitly by computing the nearest root of each vertex. This approach is very close to the minimum forest problem which searches for minimum distance from a vertex to its root vertex. So, $r(p)$ is the nearest root vertex of a pixel at the coordinate p, and $|p-r(p)|$ is the Euclidean distance associated to that pixel. The algorithm first initializes the nearest root vertex of the contour vertices and then propagates them to inner pixels in a manner where the distances are increasing. The algorithm is shown in Figure 5. This algorithm works in the same graph-search framework and it is very similar to algorithm 1. The main differences are in lines 8, 9 and 10.

In the initialization, all vertices are set as temporary and the nearest root of the contour vertices are themselves, so their distance is 0. All other vertices have their nearest root assigned to an infinity coordinate value so their distance is infinite.

The propagation step starts at line 5 and runs while there is a temporary vertex. Line 6 finds a vertex with the minimum distance to its nearest root, from all of the temporary vertices and this vertex becomes permanent in line 7. Lines 8, 9 and 10 update the nearest root of the temporary vertex x which is a 8-neighbor of the new permanent vertex v. Line 9 verifies if the distance from x to the nearest root of v is shorter than to its nearest root. If true, line 10 updates the nearest root of x to be the same as v.

This algorithm can also be seen as the classical shortest forest path algorithm proposed by Moore [Moo57]. The main differences of the proposed algorithm and the classical one, rely on lines 9, and 10. In the classical algorithm, $w(\mathrm{v},\mathrm{x})$ is the arc length linking vertices v to x. $D(p)$ is the distance of node p to its nearest root:

```
8.      for each x ∈ N8(v) and flag(x) = T
9.       if D(v) + w(v,x) < D(x)
10.        D(x):=D(v) + w(v,x)
```

As $D(\mathrm{v}) = |\mathrm{v} - \mathrm{r}(\mathrm{v})|$, the Boolean expression of line 9 above can be rewritten as:

$$|\mathrm{v} - \mathrm{r}(\mathrm{v})| + w(\mathrm{v},\mathrm{x}) < |\mathrm{x} - \mathrm{r}(\mathrm{x})|.$$

From Fig 5, the vector addition $|\mathrm{v} - \mathrm{r}(\mathrm{v})| + w(\mathrm{v},\mathrm{x}) = |\mathrm{x} - \mathrm{r}(\mathrm{v})|$, so line 9 can be finally rewritten as:

$$|\mathrm{x} - \mathrm{r}(\mathrm{v})| < |\mathrm{x} - \mathrm{r}(\mathrm{x})|$$

which is the Boolean expression used in the proposed algorithm. We can also note that although there are 8 arcs leaving each vertex, due to the nature of the algorithm to generate increasing distances, the tests are done on the arcs connecting permanent vertices to temporary ones where the arc length $w(\mathrm{v},\mathrm{x})$ is always positive. This is a requirement for the classical graph search algorithm to find the optimum path.

Selection of the neighborhood

Both algorithms 1 and 2 require a neighborhood for the distance propagation. Cuisenaire and Macq [CM99] have demonstrated that the discrete Voronoi regions using the Euclidean metric are not connected sets. This means that depending on the particular binary image configuration, a given neighborhood is required for those algorithms to determine the exact Euclidean distance. Cuisenaire and Macq report a table with the size of the neighborhood and the maximum distance in the image. The eight neighborhood will always give correct results for distance less than 169 and a 7x7 neighborhood, for distance less than 2404. For the complete table, see [CM99].

6 Implementation

The computational efficiency of the EDT algorithms presented in the previous sections are very dependent on the search for a vertex with minimum distance of line 6. When the graph is sparse, as it is in the distance transform algorithm, one of the best implementations is to use an ordered queue of distances. There are many implementations based on this idea using different ordered queue algorithms such as bucket sorting (Dial), d-Heap, Fibonacci heap and radix heap. For this particular case where the graph is large, sparse and the variance of the arc costs is small, Dial's implementation is recommended because it is easy to implement and has excellent empirical behavior [AMO93].

Speed analysis and comparison

The main motivation of the efficient ordered propagation algorithm presented in [Rag92] is to avoid visiting a pixel several times. Ideally each pixel should be visited only once. The graph-based algorithms presented here use the same principle, but in the well known shortest path search framework.

The speed performance of both algorithms depend very much on the implementation of the ordered queue algorithm. In both cases the Euclidean distance transform algorithm is linear with respect to the number of pixels in the image.

Measuring the speed performance of distance transform algorithms is a difficult task. A complexity analysis, although useful, does not represent the effective speed of a particular algorithm implementation. In addition to

r = function EDT1(f)

input: binary image f with

- $O = \{p|f(p) = 1\}$ (object pixels)
- $C = \{p|f(p) = 0$ and $p + i \in O, i \in N_4(p)\}$ (external contour C pixels)
- $V = O \cup C$ (all vertices of the graph)

output: D(p), such that |D(p)| is the Euclidean distance from p to the nearest background pixel in C.

auxiliary variables:

- $d_8(i) = \{(0,1),(1,1),(1,0),(1,-1),(0,-1),(-1,-1),(-1,0),(-1,1)\}$
- $ds_8(i) = \{(0,1),(1,1),(1,0),(1,1),(0,1),(1,1),(1,0),(1,1)\}$
- flag(p): Boolean array indicating T (temporary) or P (permanent) label.

```
1.  D = function EDT1(f)
2.     D(p) := (0,0); p ∈ C
3.     D(p) := (∞,∞); p ∈ O
4.     flag(z) := T, z ∈ V
5.     while ∃ z ∈ V|  flag(z)=T
6.      find v ∈ V such that |D(v)| is minimum, and flag(v)=T
7.      flag(v):= P
8.      for each p := v + d8(i), i=1 to 8 and flag(p)=T
9.       if |D(v) + ds8(i)| < |D(p)|
10.        D(p) := D(v) + ds8(i)
```

Figure 4: Algorithm 1: Euclidean Distance Transform

Algorithm	Img 1	Img 2	Img 3	Img 4	Img 5	Img 6	Img 7
Eggers	3.98	3.09	3.39	3.25	3.21	3.83	4.78
Algorithm 1	4.98	6.51	5.00	3.91	5.68	4.12	5.88
Ragnemalm	16.5	9.26	20.17	28.42	95.33	13.54	32.11

Table 1: Speed performance based on the speed of the 5x5 Chamfer implementation for seven different images of size 512x512.

`r = function EDT2(f)`

input: binary image `f` with

- $O = \{\mathtt{p}|\mathtt{f}(\mathtt{p}) = 1\}$ (object pixels)
- $C = \{\mathtt{p}|\mathtt{f}(\mathtt{p}) = 0 \text{ and } \mathtt{p} + i \in O, i \in N_4(\mathtt{p})\}$ (external contour C pixels)
- $V = O \cup C$ (all vertices of the graph)

output: `r(p)`, array with the coordinates of the nearest root pixel from `p` such that $|\mathtt{p} - \mathtt{r}(\mathtt{p})|$ is the Euclidean distance from `p` to the nearest root $\mathtt{r}(\mathtt{p}) \in C$.

auxiliary variables:

- $N_8(\mathtt{p}) = \{\mathtt{p} + (0,1), \mathtt{p} + (1,0), \mathtt{p} + (-1,0), \mathtt{p} + (0,-1), \mathtt{p} + (-1,-1), \mathtt{p} + (1,1), \mathtt{p} + (-1,1), \mathtt{p} + (1,-1)\}$
- `flag(p)`: Boolean array indicating `T` (temporary) or `P` (permanent) label.

```
1.  r = function EDT2(f)
2.    r(p) := p; p ∈ C
3.    r(p) := (∞,∞); p ∈ O
4.    flag(z) := T, z ∈ V
5.    while ∃ z ∈ V|  flag(z)=T
6.     find v ∈ V such that |v − r(v)| is minimum, and flag(v)=T
7.     flag(v):= P
8.     for each x ∈ N8(v) and flag(x)=T
9.      if |x−r(v)| < |x−r(x)|
10.       r(x) := r(v)
```

Figure 5: Algorithm 2: Euclidean Distance Transform

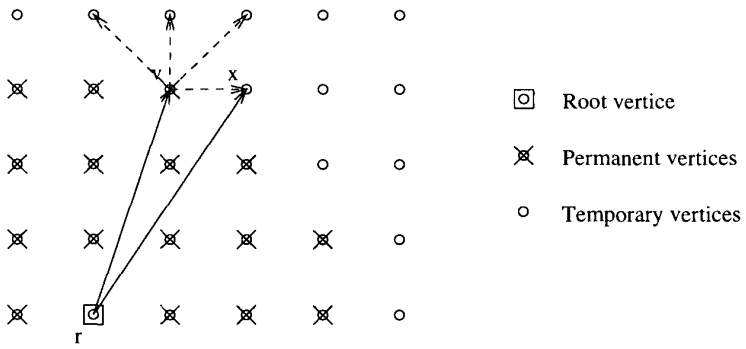

Figure 6: Distance of point x (temporary), based on point v (permanent). Root point is r

this, differences in computer architecture, compiler optimization can lead to very different speed performance. The speed analysis made in [Rag92] is by counting the number of memory access per pixel. The 5x5 Chamfer requires about 28 memory accesses per pixel while the ordered propagation requires about 8 memory accesses. Comparing the speed analysis of algorithm implementation the results are much favored to the Chamfer distance algorithm as the program implementation has much less instructions, is simpler than the ordered propagation, and the raster nature of the Chamfer is more appropriate to compiler optimization and modern general purpose CPUs.

In this paper we compared the speed of our own implementation of the following algorithms: 5x5 Chamfer, Ragnemalm's and Eggers'. We list in Table 1 the speed of the algorithms divided by the speed of the 5x5 Chamfer algorithm for seven different binary images of size 512 x 512. From the table, we can notice that our proposed algorithm has similar efficiency as the Eggers' algorithm.

7 Conclusions and comments

We have presented in this work, a novel formulation of the Euclidean Distance Transform algorithm using the shortest path graph-search framework. Two algorithms were presented: the first propagates the shortest Euclidean distance and the second propagates the node that achieves the shortest Euclidean distance. The first algorithm is useful for distance calculation whereas the second is useful for the determination of Voronoi regions and skeleton approaches.

The main advantages of the proposed algorithms are the fact of being a a novel algorithm for Euclidean Distance Transform based on graph-search algorithm; one of the simplest algorithms reported in literature, but yet yielding very good speed performance; and easily extended to higher dimensions.

The only drawback of the algorithm is the neighborhood dependency with the largest distance size of the output image. Despite of that we found that in most 512 x 512 images the use of 3x3 neighborhood gave the exact Euclidean distance. But if one requires exact result for any image configuration, using a 7x7 neighborhood gives the Euclidean Distance for any image of size 1700 x 1700.

8 Acknowlegments

This project is partially supported by FAPESP. The second author is supported by CNPq, grant n. 300698/98-4.

References

[AMO93] R.K. Ahuja, T.L. Magnanti, and J.B. Orlin. *Network Flows - Theory, Algorithms and Applications*. Prentice Hall, 1993.

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

[CM99] O. Cuisenaire and B Macq. Fast euclidean distance transformation by propagation using multiple neighborhoods. *Computer Vision and Image Understanding*, 76(2):163–172, November 1999.

[Dia69] R. Dial. Algorithm 360: Shortest path with topological ordering. *Comm. Assoc. Comput. Mach.*, pages 632–633, 1969.

[Dij59] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, pages 269–271, 1959.

[Egg98] H. Eggers. Two fast euclidean distance transformations in z^2 based on sufficient propagation. *Computer Vision and Image Understanding*, 69(1):106–116, 1998.

[Moo57] E.F. Moore. The shortest path throught a maze. *Proc. Internat. Symposium on the Theory of Switching, Part II, Harvard university press, Cambridge, MA*, pages 285–292, 1957.

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

[RP68] A. Rosenfeld and J. Pfaltz. Distance functions in digital pictures. *Pattern Recognition*, pages 33–61, 1968.

[SC94] Y. Sharaiha and N. Christofides. Graph-theoretic approach to distance transformations. *Pattern Recognition Letters*, 1994.