# Region Growing Euclidean Distance Transforms

Olivier Cuisenaire

Telecommunication and Remote Sensing Laboratory
Université Catholique de Louvain, Belgium

## Abstract

By propagating a vector for each pixel, we show that nearly Euclidean distance maps can be produced quickly by a region growing algorithm using hierarchical queues. Properties of the propagation scheme are used to detect potentially erroneous pixels and correct them by using larger neighbourhoods, without significantly affecting the computation time. Thus, Euclidean distance maps are produced in a time comparable to its commonly used chamfer approximations.

## 1. Introduction

Distance maps are images where the value of each pixel of the foreground is the distance to the nearest pixel of the background. Generating such maps using a Euclidean distance metric is a complex problem since a direct application of this definition usually requires an excessive computation time.

The development of fast algorithms for producing Distance Transforms (DT), as approximations of the Euclidean distance maps, has allowed their applications in various fields such as chamfer matching, registration of medical images [6], generation of morphological skeletons or active contour models.

Numerous DT algorithms have been proposed, offering various trade-offs between computation time and quality of the approximation of the Euclidean metric. The DT in the literature belong to two categories: the Chamfer DT originally proposed by Borgerfors in [2] and the Vector DT proposed by Danielsson in [1]. These algorithms rely on a number of raster scans over the image, although some parallel implementations have also been proposed [4]. In this paper, we present a region-growing Vector DT where pixels are scanned by increasing value of the distance.

In section 2 we present the principles of existing DT. In section 3, the region growing new algorithm is described. In section 4, the quality of the approximation is increased by using larger neighbourhoods for a subset of potentially erroneous points. In section 5, we analyse the types and values of errors made by our algorithm and compare it with CDT. In section 6 the complexity of the algorithm is considered.

## 2. A review of Distance Transforms

Chamfer Distance Transforms (CDT) are commonly used to approximate the Euclidean metric. They are based on the assumption that the value of the distance for each pixel can be computed from of the values of its neighbours plus a mask constant.

CDT are usually produced in 2 raster scans over the image, using half of the neighbour pixels as a mask for each scan. The simplest CDT are the chessboard and city-

block DT using masks a and b of fig 1. G. Borgerfors introduced better approximations with mask c in [2] and mask d in [3].

There are two sources of errors in CDT. First, lines are approximated by segments following the main directions available in the mask, i.e. every 45° for the CDT 3-4 and every 22.5° for CDT 5-7-11. Secondly, the values of the mask constants are approximated by integers which leads to errors even in the main directions of mask.

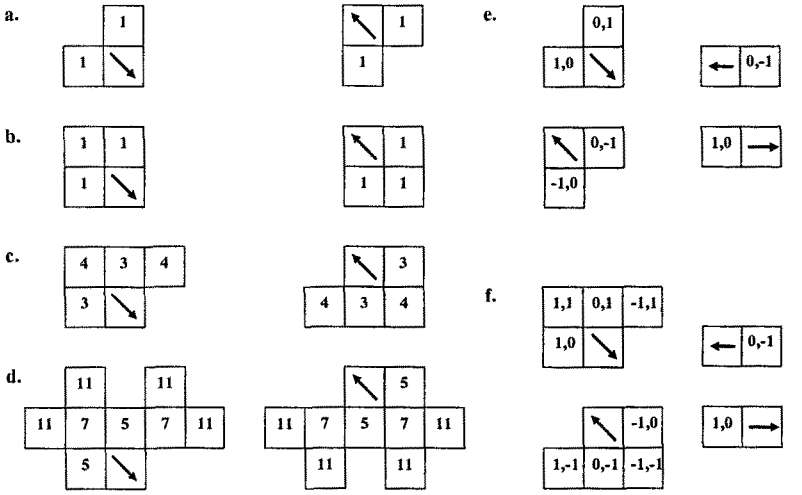


**Fig. 1.** Masks of the various DT: a) chessboard - b) City-block - c) Chamfer 3-4 - d) Chamfer 5-7-11 - e) 4SSED - f) 8SSED.

To produce better approximations of the Euclidean metric, one should transmit more information from pixels to their neighbours. In [1], Danielsson proposes to propagate a vector localising the nearest background pixel or NBP.

This requires the use four scans. Masks e of fig 1.corresponds to 4SED (4 neighbours Sequential Euclidean Distance) and masks f to 8SED. In 3 dimensions, 6 scans are needed. The increased number of scans and the higher complexity of vector comparisons makes the Vector Distance Transform (VDT) significantly slower than CDT.

Using VDT, most pixels are error-free, but some errors occur for particular background pixel configurations, when the basic assumption - that the closest background point to a pixel is also the closest point to one of its neighbours - is not satisfied.

## 3. Region Growing Algorithm

Our method uses the 4SED mask, but instead of using the raster scans, pixels are considered by increasing value of the distance. This is implemented with a data structure called hierarchical queues (HQ), illustrated at fig 3. The queue labelled i in the HQ contains the pixels for which i is the square of the distance to their NBP. With a Euclidean metric, this value is an integer. For each pixel in the HQ, its location and its NBP are stored. An image called map is also created, containing the square of the distance at each pixel. In what follows, our aim is to compute this map.

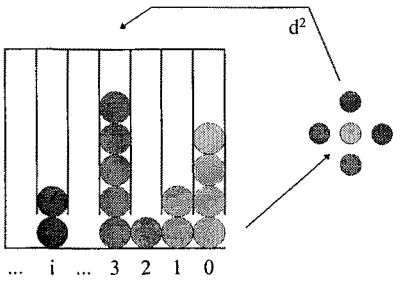


**Fig. 2.** Hierarchical Queues are made of a collection of FIFO queues. In-going elements enter any of the queues, outgoing elements are taken from the smaller numbered non-empty queue.

The HQ is initialised with queue 0 containing the background pixels and all other queues empty. The map is initialised with zero for background pixels and the maximum integer value everywhere else.

Pixels are then treated in the HQ order. For each pixel, we consider that its neighbours have the same NBP. If this leads to a smaller value of the distance than stored in the map, the map is updated with this value and the neighbour is inserted in the HQ. In the HQ, pixels are treated by increasing value, which on the map corresponds to the border of a growing region centred on the background pixels.

More formally, the algorithm is written

*Initialisation:* *HQ(0) is the list of background pixels*
*HQ(i) empty for all* $i \neq 0$
*map[p] is maxint for all pixel p*
*map[p] is 0 for background pixels*

```
Main:   while HQ not empty
        {       get p from HQ
                for each neighbour n of p
                {       d=dist²(n, NBP(p))
                        if d<map[n]
                        {       map[n]=d
                                add n to HQ(d) with NBP(n)=NBP(p)
        }       }       }
```

Note that this algorithm allows pixels to be first mislabelled with a wrong NBP and later corrected as closer to another. Nonetheless, the HQ scan processes the correction before the initial error since the distance is smaller hence the correction is in a queue of smaller label. Errors are therefore not propagated.

Also, with the HQ scan there is no need of propagating the information back to pixels of a smaller value. Therefore, we only consider the part of the neighbourhood which decreases none of the components of the relative position of the NBP. In most cases, only one quadrant of the mask will need to be considered, which reduces the search to 2 pixels our of 4 for 4SED, 3 out of 8 for 8SED, ...

## 4. Use of Larger neighbourhoods

As we will see in next section, VDT produces errors only for few pixels, with specific background pixel configurations. With 4SED for instance, the largest relative error is illustrated at fig 7.a, where the information "closer to pixel B" cannot be propagated to pixel x. This could be solved by using the mask of 8SED allowing the information to be transmitted diagonally so that x could be reached from the pixel labelled 2, as suggested by the arrow.

In general, two main strategies can be envisaged to reduce the errors of a DT. First, one can use larger neighbourhoods for the masks. This leads to a significant increase of the computation time, which is proportional to the product of the size of the mask by the number of pixels. The second method, proposed in [5], consists of increasing the amount of information transmitted from pixel to pixel. εVDT(0) stores the list of all NBP instead of only one of them. εVDT(1) also includes nearly NBP in the list. It produces an error-free map, but is orders of magnitude slower than 4SED or 8SED.

We propose to use the first approach and to quicken it significantly. The distance map is first computed quickly but roughly with the smallest possible mask (4SED). Then, larger neighbourhoods are used to correct errors made with this first scan. Fortunately, these corrections only need to be made for a small subset of pixels. Indeed, after using 4SED, most pixels are error-free. And the erroneous pixels share some properties we can use to restrict the set of points to treat with larger masks.

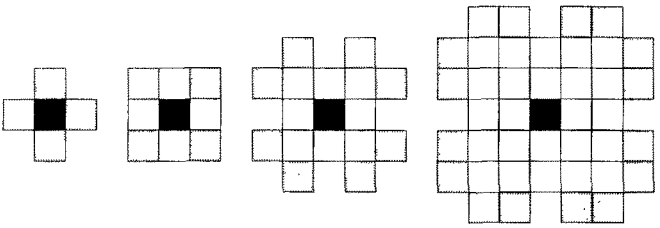

**Fig. 3.** a sequel of increasing size neighbourhoods: from left to right: 4SED mask, 8SED or 3x3 mask, 5x5 mask, 7x7 mask. Light grey pixels are those added to the smaller mask to create the larger one.

First, we consider neighbourhoods $N_0$, $N_1$, ... $N_i$ ... of increasing size, such as illustrated at figure 6. We define an $err_i$ pixel for $N_i$ a pixel in the map for which the values computed using $N_i$ and $N_{i+1}$ are different. An $end_i$ pixel is one that was not propagated while using the algorithm described in section 3 with $N_i$. $end_i$ pixels can easily be detected while running the algorithm.

Once we have a map created using $N_i$, we want to correct some errors to make it as good as if it had been created with $N_{i+1}$. For this, we only need to correct the $err_i$ pixels. We use the following property: Any $err_i$ pixel is a $N_{i+1}$ neighbour of either another $err_i$ pixel or of an $end_i$ pixel. This can easily be proved ab absurdum. Therefore, the $N_{i+1}$ mask only needs to be propagated from $end_i$ pixels and from corrected $err_i$ pixels. This leads to the following algorithm for 2 neighbourhoods, which can be ex-

tended immediately for any number of neighbourhoods

```
while HQ1 not empty
{   get p from HQ1
    for each N1 neighbour n of p
    {       d=dist²(n, NBP(p) )
            if d<map[n]
            {       map[n]=d
                    add n to HQ1(d) with NBP(n)=NBP(p)
    }       }
    if no neighbour n was put in HQ1 (p is endi)
        add p to HQ2(map[p])
}
while HQ2 not empty
{   get p from HQ2
    for each N2 neighbour n of p
    {       d=dist²(n, NBP(p) )
            if d<map[n]
            {       map[n]=d
                    add n to HQ2(d) with NBP(n)=NBP(p)
}   }       }
```

This can further be improved by noticing that each neighbourhood provides a perfect map up to a certain value. There is no need to use larger neighbourhoods for smaller distances. For instance, the smallest possible error with 4SED is illustrated at figure 7.a. The pixel labelled 2 is $end_{4sed}$. It should used to reach pixel x with the 8SED mask. Hence $end_{4sed}$ pixels labelled 0 or 1 should not be considered while using 8SED to improve a map created with 4SED. The smaller value for which 8SED shall be used is 2.

The thresholds for some masks in 2 and 3 dimensions can be found in tables 1 and 2. In 3D, the background pixels should be included in the HQ2 to take into account the fact that the direct neighbourhood can produce errors even with $d^2=0$.

## 5. Error Analysis

The error analysis for our DT compared to the Euclidean metric is similar to the analysis made in [1]. In this paper, we consider, for each mask, the smallest values for which a pixel can be mislabelled. It also corresponds to the largest relative error. The configurations of NBP leading to those errors for 4SED and 8SED are illustrated in fig 7 and numerical results found by extensive search are listed in tables 1 and 2.

They should be compared with the relative errors obtained with CDT: 8% for CDT 3-4 and 2% for CDT 5-7-11. This ranks the 4SED mask between the two chamfer methods and 8SED or any larger mask as better than any published CDT.

Furthermore, VDT finds the correct value for most pixels. The number of erroneous pixels is image dependant. In the case of figure 9, less than 300 out of 65500 points,

or 0.5% are erroneous when using the 4SED mask. Typically, errors are located in areas where the "propagation front" is shrinking while expansion areas are error-free since no information loss occurs there. In particular, for a single background pixel (fig. 8), the 4SED mask provides a perfect Euclidean map.

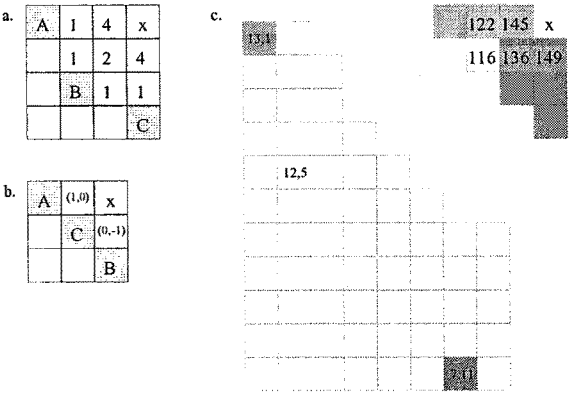


**Fig. 4. a)** 4SED error: pixel x is assigned a squared distance value of 9 rather than 8 as A or C are mistaken as closer to x than B. **b)** New type of error: pixel x is assigned a value of 4 instead of 2 if pixels are scanned in a specific order A,B, C **c)** using 8SED, pixel x is wrongly assigned $170=7^2+11^2=13^2+1^2$ instead of $169=12^2+5^2$

| Mask | $d^2$ correct | $d^2$ assigned | d correct | relative error | $d^2$ |
|---|---|---|---|---|---|
| 4SED | 8 | 9 | 2.89 | 6,1% | 2 |
| 8SED | 169 | 170 | 13 | 0,3% | 116 |
| 5x5 | 964 | 965 | 31.05 | 0.05% | 778 |
| 7x7 | 2404 | 2405 | 49.03 | 0.02% | 2089 |

**Table 1**: Smallest errors for each mask size. The last column ($d^2$) gives the value from which the non-propagating pixels should be propagated with a larger mask.

| Mask | $d^2$ correct | $d^2$ assigned | d correct | relative error | $d^2$ |
|---|---|---|---|---|---|
| 6SED | 3 | 4 | 1,73 | 15,5% | 0 |
| 3x3x3 | 49 | 50 | 7 | 1,0% | 24 |
| 5x5x5 | 228 | 229 | 15,1 | 0,2% | 146 |
| 7x7x7 | 626 | 627 | 25,02 | 0,08% | 441 |

**Table 2**. Same as table 1 for 3D distance maps.

Other differences between VDT and CDT are illustrated at fig 8. First, the shape of iso-distance curbs: CDT gives polygons of 8 or 16 sides, VDT a perfect circle. This influences applications such as the generation of skeletons [1]. Secondly, CDT only allow 8 or 16 directions for the gradient, VDT a continuous range of directions. This influences convergence domains and speed in gradient-based minimisation as in [6].

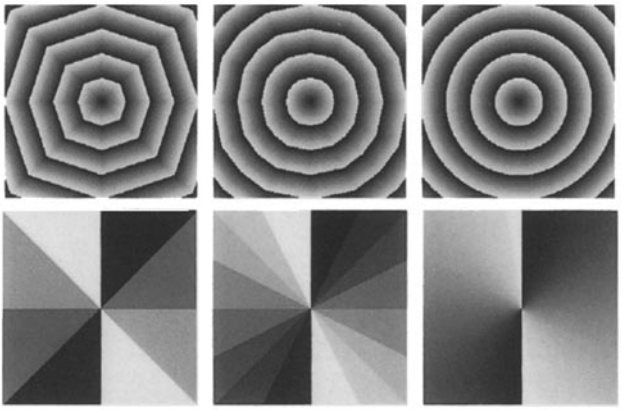

**Fig. 5.** From left to right: CDT 3-4 / CDT 5-7-11 / EDT. Up: distance from a central pixel. Colours are mod. 30 for better visibility. Down: direction of the gradient of those maps.

## 6. Complexity Analysis

The usual method for assessing the complexity of a DT is to consider the number of comparisons per pixel required by the algorithm. Unfortunately, in order to compare our algorithm to a CDT, this is not a valid method since the type of comparison is different (scalars for CDT, vectors for VDT) and their number is image dependant.

Furthermore, hierarchical queues are dynamic data structures. Hence, a significant part of the time is used for dynamic memory allocation. This makes the comparison of CDT and our algorithm dependant on the relative efficiency of each type of operation, and thus machine-dependant.

We therefore choose a heuristic approach, using CDT and our method on test images several types of workstations. In average, our DT with the 4SED mask is 1,5 slower than CDT 3-4 and similar to CDT 5-7-11. In three dimensions, it is between 1.05 and 3 times slower than CDT 3-4-5, depending on the workstation.

These good results - despite the complexity of the basic operation -are explained by the small amount of comparisons required. On the test images, our method requires an average of 2,1 and 3,35 comparisons per pixel in 2 and 3 dimensions respectively, compared with 8 and 26 for CDT. In d dimensions, CDT requires $o(d^3)$ comparisons while our method requires o(d) comparisons whose complexity grow like o(d). Hence the global complexity growth like $o(d^2)$. Furthermore, the algorithm can be stopped at any step and provide a partial map where all pixels within a certain distance have been computed. This is of interest for the last steps of minimisation methods where all points of interest are close to their targets.

Let us now consider the multi-mask algorithm of section 4. The additional cost for using the larger mask can be estimated from the number of $end_{4sed}$ pixels. This number is image related. For instance, with the 256x256 image of fig. 9, out of the 65536 pixels treated with the 4SED mask, 8894 are considered for the 8SED mask, 3765 for the 5x5 mask and 2282 for the 7x7 mask. This leads to 65536*2,1 + 8894*1 +

3765*2 + 2282*4 =163177 comparisons if, for the larger masks, only one quadrant of the pixels highlighted at fig 3 are used. The additional cost of using the 7x7 mask instead of 4SED is less then 20%, while using this 7x7 mask with the single-mask algorithm of section 3 represents an additional cost of around 400%.

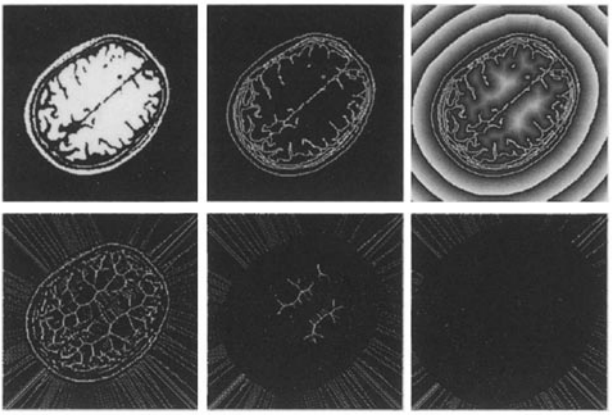

**Fig.6.** Up: a typical image (left), its edges (centre) and the Euclidean map (right) to these edges. Colours are displayed modulo 25 for better visibility. Down: set of $err_{4SED}$ pixels used for 8SED (left, threshold at $d^2=2$), 5x5 (threshold at 116) and 7x7 masks (threshold at 778).

## 7. Conclusion

We have developed an algorithm computing Euclidean distance maps in a time similar to the chamfer DT approximations. It is progressive since it can be stopped at any time and provide a sensible result, either by computing only within a certain distance, or improving the map by progressively using larger masks.

Finally, the method seems easily adaptable to other metrics and in particular to non-isotropic grids or higher dimensions. This and the use of the distance maps to generate skeletons of binary objects is the subject of future research.

## Acknowledgement

Olivier Cuisenaire's work is funded by Belgium's F.R.I.A

## References

1. P.E. Danielsson, Euclidean Distance Mapping, *CGIP* 14, 1980, 227-248.
2. Borgerfors, Distance Transformations in Arbitrary Dimensions, *CVGIP* 27, 1984, 321-345
3. G. Borgerfors, Distance Transformations in Digital Images, CVGIP 34, 1986, 344-371
4. H. Embrechts and D. Roose, A parallel Euclidean Distance Transformation Algorithm, *CVIU* 63, 1996, 15-26
5. J. Mullikin, The vector distance transform in two and three dimensions, *CVGIP* 54(6), 1992, 526-535
6. O. Cuisenaire, J.Ph. Thiran, B.Macq, Ch. Michel, A. De Volder and F. Marques, Automatic Registration of 3D MR Images with a Computerised Brain Atlas, *SPIE Medical Imaging 1996*, SPIE vol. 1710, 438-449 .