BINARY AND GREY-VALUE SKELETONS METRICS AND ALGORITHMS

BEN J. H. VERWER
Becton Dickinson Immunocytometry Systems
2350 Qume Drive, San Jose CA 95131-1507, USA

LUCAS J. VAN VLIET and PIET W. VERBEEK
Delft University of Technology, Faculty of Applied Physics
Pattern Recognition Group, Lorentzweg 1
2628 CJ Delft, The Netherlands

A metric defines the distance between any two points. The "natural" metrics of the digital world do not approximate the Euclidean metric of the continuous world well. Skeletonization (sometimes named topology preserving shrinking or homotopic thinning) is one example in which this leads to unacceptable results. In the present work we propose and demonstrate skeletonization using path-based metrics which are a better approximation of the Euclidean metric. Moreover, we achieve a good performance on sequential processors by processing each pixel only once in the calculations of binary (Hilditch) and grey-value (upper) skeletons.

Keywords: **Skeletonization, thinning, topology preserving shrinking, watershed upper skeleton, metric, chamfer, distance transform.**

1. INTRODUCTION

Many image processing and analyzing methods use, either implicitly or explicitly, a *metric*. A metric defines the distances between all pairs of points. Examples of operations using metrics are erosion, dilation, distance transform, local minimum and maximum filters and skeletonization. Commonly used metrics are the city block and chessboard metric, arising from the connectivity of the grid. However, in most cases the digital world should model the real world, which has a Euclidean metric. We therefore emphasize as others have,[1-4] that our digital metric should approximate the Euclidean metric as well as possible (we will define later what a good approximation means).

Secondly, we will show that the use of better metrics does not have to seriously affect our processing times. We will give algorithms for binary[5] and grey-value skeletonization which process each pixel only once.

2. PATH-BASED METRICS

A metric on a set A is a function d from $A \times A$ to R. The function has to satisfy three conditions:

- $d(\mathbf{u}, \boldsymbol{\nu}) = 0$, if and only if $\mathbf{u} = \boldsymbol{\nu}$;
- $d(\mathbf{u}, \boldsymbol{\nu}) = d(\boldsymbol{\nu}, \mathbf{u})$;
- $d(\mathbf{u}, \boldsymbol{\nu}) \leq d(\mathbf{u}, \mathbf{w}) + d(\mathbf{w}, \boldsymbol{\nu})$.

in: International Journal of Pattern Recognition and Artificial Intelligence, vol. 7, no. 5, 1993, 1287-1308.

A local approach to implement metrics[1,6,7] defines the distance between two points as a minimum path length (*path-based metrics*). The path consists of a series of vectors, each of which is taken from a limited set of vectors defined on the grid, called prime vectors.[8] In turn, each of the vectors have a distance value associated with them (named chamfer or local distances[3,4]). Figure 1 shows the minimal paths between two points. Notice that, unlike in the continuous case, more than one minimal path can exist.

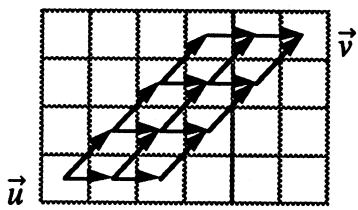


Fig. 1. The distance between u and ν is defined as the minimum path length between u and ν, using a limited number of prime vectors. Note that more than one minimum path can exist between two points.

It is easy to prove that this procedure implements a metric if the prime vectors do not include transitions of a point to itself, if the distances associated with the prime vectors are positive definite and equal for opposite vectors (see Appendix A, from Ref. 9). Montanari[1] introduced the general name quasi-Euclidean for these metrics since the error with respect to Euclidean distances can be made arbitrarily small by increasing the neighborhoods (proof can be found in Ref. 10).

Each set of local distances will lead to a certain error in the distance between two points with respect to the Euclidean distance between the points. We can optimize the local distances so as to minimize that error. For skeletonization the maximum error is an appropriate error measure (when deviations in the digital skeleton from the continuous skeleton occur for the first time).

The set of prime vectors is usually limited to a 3 by 3 or 5 by 5 neighborhood of a point (in 2 dimensions). In a 3 by 3 neighborhood there are 2 different types of prime vectors, one type connects the central pixel with one of the north, east, south or west neighbors, length d_{10}, the other with one of the north-east, south-east, south-west or north-west neighbors, length d_{11}. In a 5 by 5 neighborhood the so-called knight's move, length d_{12} is included. The other prime vectors in a 5 by 5 neighborhood can be written as two cascaded prime vectors of length d_{10} or d_{11}.

Good integer values for d_{10} and d_{11} are 5 and 7 (maximum error 4.21%[7]). The traditional city block metric can be obtained by associating the distance value 1 with d_{10} and 2 with d_{11}. The chessboard metric by associating the distance value 1 with both d_{10} and d_{11}. The maximum errors for both of these metrics is 17.16%.

We call the (5,7)-metric octagonal since "circles" in these metrics, being the set of points which lie at equal distance to one point, form octagons (note that the "circles"

of the city block metric are diamonds and that the "circles" of the chessboard metric are squares, Fig. 2). Following the same argument, metrics obtained by using 5 by 5 neighborhoods are called hexadecagonal. Good integer values for d_{10}, d_{11}, and d_{12} are 5, 7 and 11 (maximum error 1.79%), or 9, 13 and 20 (1.52%).[7]

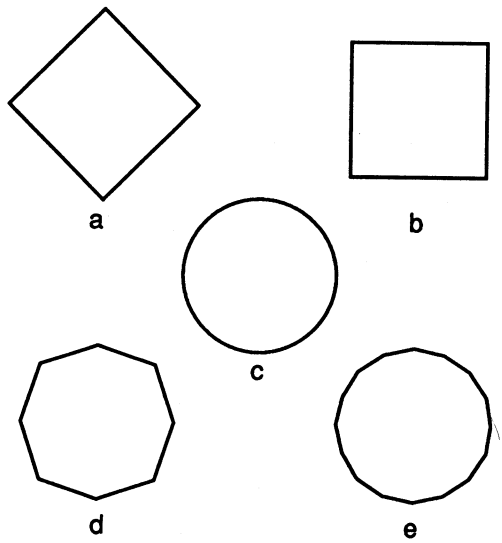


Fig. 2. "Circles" for different metrics. (a) City block (b) Chessboard (c) Euclidean (d) Octagonal (e) Hexadecagonal.

3. PROPAGATING DISTANCES

Several algorithms have been proposed to employ path length metrics. Two categories can be discerned. The first type of algorithm processes an image in two scans.[1,3,4,6] The second type of algorithm propagates wave fronts through the image. The propagation order can be dependent on the connectivity[1,11,18] or the metric.[5] Propagation methods which are dependent on the metric process each pixel only once in order of increasing distance. The propagation methods which are dependent on the connectivity do not in general obey this ordering principle (in some applications, such as constrained distance transformation,[11,12] these algorithms therefore cannot guarantee that each pixel is processed only once[13]). A further advantage of processing pixels in order of increasing distance is that we can do additional processing at the same time. In this paper the additional processing is conditional erosion or conditional minimum filtering. This is in contrast with another approach, which first stores all distances and then extracts the skeleton from the distance image.[2]

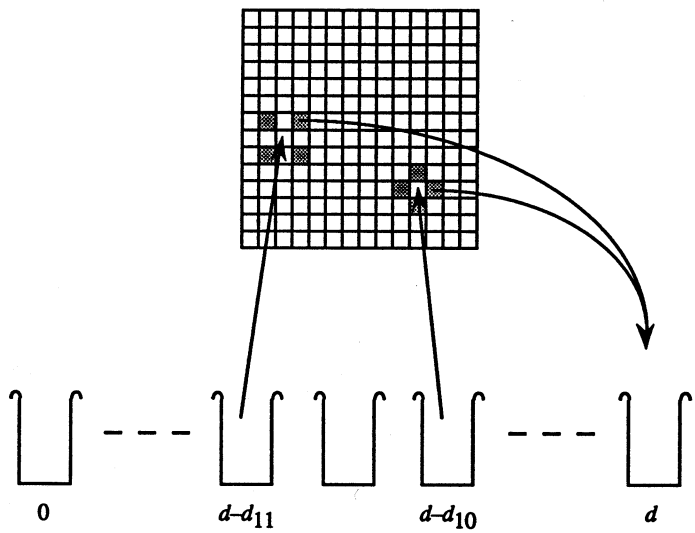


Fig. 3. The bucket algorithm. Candidate pixels with distance d are the 4-connected neighbors of pixels with distance $d - d_{10}$ and the 8-connected neighbors of pixels with distance $d - d_{11}$, the coordinates of which are stored in the buckets labeled "$d - d_{10}$" and "$d - d_{11}$". The candidates will receive distance d, and their coordinates stored in bucket 'd', if they have not already received a lower distance value. This is easily checked by comparing their distance with d. Alternatively, if the distances are not stored, a status bit will suffice, indicating "has received a distance value", which will be set for each pixel which coordinates are stored in a bucket. Not storing distance values is of course only useful if other processing is done at the same time, e.g. skeletonization, see next section.

Pixels are addressed in order of increasing distance by storing all the coordinates of the pixels[a] with the same distances in a *bucket*.[b] Pixels with a certain distance to a specified set of points (e.g. the background in a binary image or the set of lower valued pixels in a grey-value image, see Sec. 5) can be found by using the coordinates of the pixels with lower distances. Specifically, the pixels with distance d are those pixels which have not yet received a distance value and which lie to the north, east, south or west of the pixels with distance value $d - d_{10}$ or to the north-east, south-east, south-west or north-west of pixels with distance value $d - d_{11}$ or a knight's move away from pixels with distance value $d - d_{12}$. We say the latter pixels *generate* the former. Note that the coordinates one has to store are limited to the coordinates of pixels with distance values between $d - d_{12}$ and d. The procedure starts with the set to which distances have to be calculated, whose pixels have per definition the distance value 0. The procedure is outlined graphically in Fig. 3.

[a]Actually pointers to pixels are faster. In this case the edge of the image has to receive special treatment though.

[b]The algorithm described finds pixels with the same distance simultaneously, the algorithm in Ref. 13, which uses bucket sorting as well, does not.

The number of neighbors checked can be limited if directional information is used. If the coordinates of a pixel with a certain distance value are retrieved from a bucket the appropriate neighbors which can be reached via a path of shorter length from the pixel which generated the retrieved pixel need not be checked; that pixel or its relative direction was stored together with the coordinates of the retrieved pixel. Figure 4(a) shows one possible case. We are looking for pixels with distance d. The central pixel is retrieved from the bucket "$d-d_{10}$". Possible candidates for distance d are the shaded pixels. Normally we would have to check all four 4-connected neighbors. But if we know from what direction the central pixel was generated, say from the bounded box, only 1 of the 4 candidates needs to be checked. Because the other three will all have received a lower distance value already.

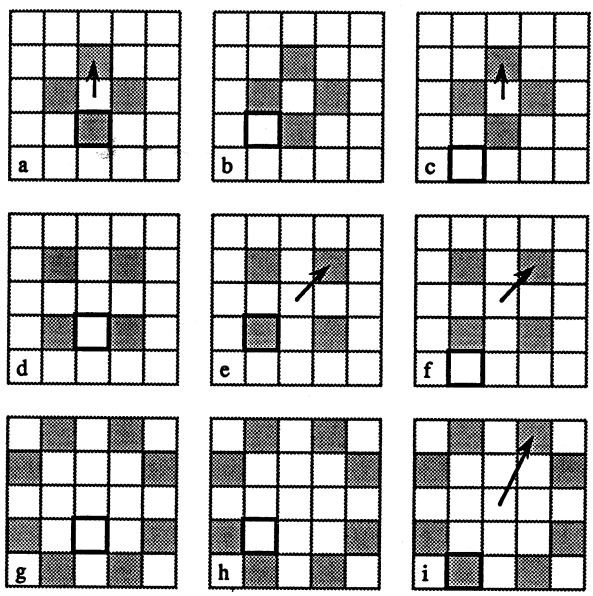


Fig. 4. The centrals pixel are the generating pixels. The pixels which generated them (their parents) are indicated by the bounded boxes. Possible successors are shaded: (abc) 4-connected, (def) 8-connected, (ghi) knight's move. The arrows indicate the successors which need to be checked. Note that the other possible candidates can be reached via other paths of shorter or equal length from the parent.

In Fig. 4 all possible cases are shown. Note that a reduction in addressing by a factor of 8 (12.5% of the total) is achieved in storing a directional field (assuming all prime vectors having the same probability: for the 10-neighbors (4-connected) and 11-neighbors (8-connected) instead of 4, on the average only $1 \times 4/16 + 0 \times 4/16 + 1 \times 8/16 = 3/4$ neighbors have to be addressed, for the 12-neighbors instead

of 8, on the average $0 \times 4/16 + 0 \times 4/16 + 1 \times 8/16 = 1/2$).[c] In the case that a pixel can be reached via two paths of equal length only one has to be evaluated. If two pixels are separated by the vector (3,1), one can either make a d_{10} transition followed by a d21 transition, or the other way around. The same applies for a (3,2) difference with d_{11} and d_{21} transitions. In skeletonization the order is important (see the next section). The choice made in drawing Fig. 4 was based on that.

4. BINARY SKELETON

4.1. Homotopic Thinning

Serra[14] gives four general conditions for discrete skeletonization:

(i) the topology should not be changed;
(ii) the result should be one pixel thick;
(iii) the result is a medial axis (object can be reconstructed);
(iv) the result is based on a digital distance.

Serra also gives two examples which show that these four conditions are too much to ask for in the digital world. Homotopic thinning is then defined as a thinning operation which keeps conditions (i) and (iv) and drops condition (iii). In pathological cases condition (ii) will also be violated (Fig. 5).[15] We have implemented homotopic thinning with the chamfer metric and with the topology preserving tests as proposed by Hilditch.[16] Note that the propagation method as proposed in the preceding section could also be used to implement other algorithms, for example the algorithm proposed by Montanari[1] which drops (i) or the algorithm proposed by Borgefors[17] which drops (ii).

Since we have used a sequential processor to implement the bucket algorithm it is advantageous to use a recursive[d] thinning algorithm. Hilditch[16] proposed an algorithm which uses 3 by 3 neighborhoods to test if removal of the central pixel will change the topology of the objects being thinned.

Different types of nonrecursive and (partially) recursive neighborhoods are used to achieve different goals: nonrecursive neighborhoods to preserve topology and to avoid erosion of one pixel thick lines from the end; recursive neighborhoods to reduce two pixel thick lines to one pixel thick lines; partially recursive neighborhoods to avoid erosion of two pixel thick lines from the end. For a fixed scan direction

[c]This procedure is applicable with any number of dimensions, with savings improving dramatically with dimension. For example, in three dimensions with a 5 by 5 by 5 neighborhood: 100-neighbors (sometimes called 6-connected neighbors, but denoting neighbors by vectors is less confusing) have 54 relevant successors on a total of 6 successors times 98 possible directions. 110-neighbors 108, 111-neighbors 56, 210-neighbors 120, 211-neighbors 72 and 221-neighbors 24:

$$\frac{54 + 108 + 56 + 120 + 72 + 24}{6 + 12 + 8 + 24 + 24 + 24} \times 98 = 4.5\%$$

[d]A recursive neighborhood is a neighborhood in which the values of the pixels are the values as known at the time of calculation, not what they were when the iteration started.

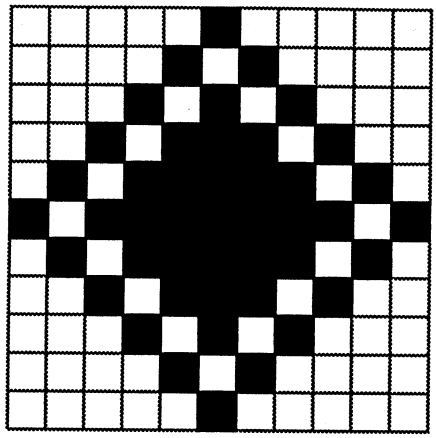


Fig. 5. Pathological image in which conditions (i) and (ii) are in conflict.

from top-left to bottom-right two partially recursive neighborhoods are required, in which the values of the north and the west neighbor are substituted recursively. In an earlier paper[18] we showed that four partial recursive neighborhoods have to be used if a fixed order cannot be guaranteed: successively recursive values of all 4-connected neighbors have to be tested in the otherwise nonrecursive neighborhood.

4.2. Basic Idea

The basic idea in using improved, path-based metrics is to remove pixels in order of quasi-Euclidean distance to the background. We uncouple the metric from the connectivity of the grid. The connectivity of the grid is essential as it preserves the topology of the object. But there is no need, as we have seen in the previous section, to let the connectivity determine the metric and the processing order.

The propagation method of Sec. 2 addresses pixels in order of quasi-Euclidean distance. In a continuous world this would lead to "perfect" skeletons, perfect with respect to the conditions listed in Sec. 4.1. In a discrete world however, removing pixels in order of increasing distance does not always result in a skeleton which lies in the middle of an object, where middle is defined according to the metric used. Figure 6 gives an example. The cause of this effect is the anisotropy of the grid. In diagonal directions the sampling frequency is lower. We have found that a small modification of the algorithm solves the problem. Pixels generated from 4-connected skeleton pixels should not be removed. The motivation is that pixels generated from pixels which are skeleton pixels, *casu quo* lie in the middle of the object, are bound to be skeleton pixels as well. Only 4-connected pixels should be taken into account since otherwise whole areas would become skeleton: two points are linked by a number of paths, not by only one as in the continuous world (see

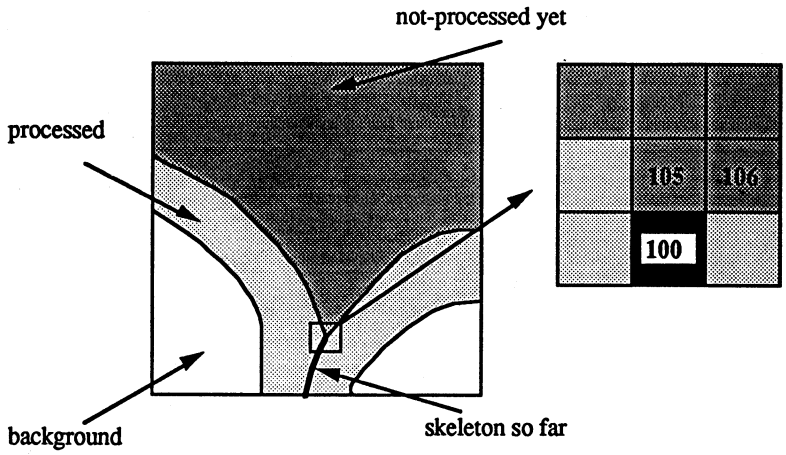


Fig. 6. Removing pixels in order of increasing distance does not lead to skeletons which lie in the middle of an object: diagonals are preferred directions (pixel valued "105") is removed instead of the pixel valued "106", since it has a lower value: but 105 lies closer to the middle of the object.[2]

Fig. 1). This also requires a special order in which successors are generated. In the case of paths of equal length the paths which starts with the largest chamfer distance is chosen. In that case 4-connected neighbors are the first to generate successors.

The pseudo code of the algorithm can be found in Appendix B.

4.3. Results

The example of Fig. 7 highlights the improvements of the hexadecagonal skeleton compared to the city-block skeleton. As expected, the more isotropic the algorithm, the more isotropic the result.

Figure 8 lists the processing times required to process a number of 256 by 256 test images. All timings were performed on a SUN Sparcstation IPX.

The city-block skeleton uses the algorithm published in Ref. 18. It is faster when applied to the random image because it takes a worst case approach to memory management. It preallocates a buffer large enough to store all the coordinates of all the points. In the case of a random image all the object pixels are very close to the background pixels so that approach is advantageous; the hexadecagonal algorithms allocates memory in chunks and needs an amount in the order of the length of the contours in image, since only the coordinates of the contours are stored. This is normally negligible compared to the memory the image itself occupies.

For both algorithms the processing times are roughly linear to the number of object pixels to be processed in the images. This is as expected since each pixel is processed only once (apart from the first scan to find the object pixels, which accounts for the 0.2 seconds necessary to process an empty image).

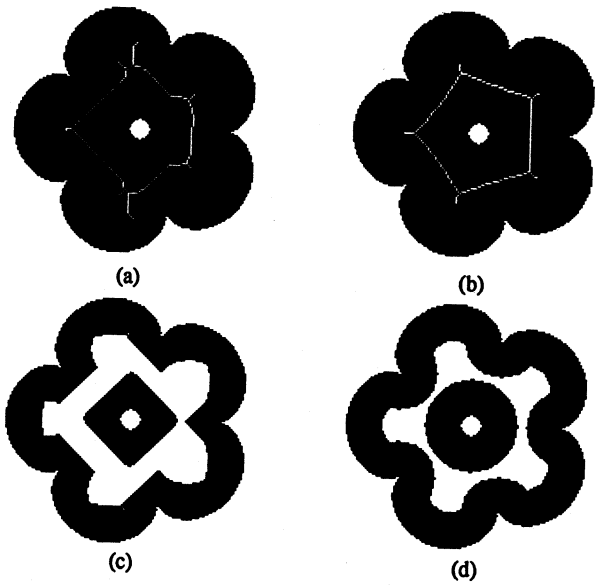


Fig. 7. An example highlighting the importance of metrics in skeletonization. (The circle in the middle of the images is part of the background) (a) Skeleton using city-block (b) Skeleton using hexadecagonal metric (c) Intermediate image — after 20 iterations — during calculation of (a) (d) Intermediate image — after distance 20 — during calculation of (b).

5. GREY-VALUE SKELETON

The natural analogy of skeletons in grey-value images are watersheds. Watersheds divide the images in domains of attraction of rain falling over the grey-value landscapes. A watershed can be obtained from a grey-value skeleton by pruning the end pixels from a skeleton while preserving the original topology. Dyer and Rosenfeld[19] made a first approach to generalize the concept of connectivity to functions (grey-value images) where the topology is dictated by the landscape (sinks and summits). Goetcharian[20] contributed to this field by translating the Arcelli[21] algorithm to grey-value functions using local minimum and maximum filters resulting in the so-called lower skeleton. A survey and formal description of grey-value skeletons is presented by Serra[14].

According to Serra, two functions can be considered homotopically equivalent if the supports of divides (sinks) and channels (summits) are jointly homotopic. The cells created by the divides and channels should have the same relationship to each other with respect to inclusion and intersection in both functions. Each sink constitutes a watershed tile and is surrounded by a divide. By duality, in the complement of the image each summit belongs to a hill and is surrounded by a channel.

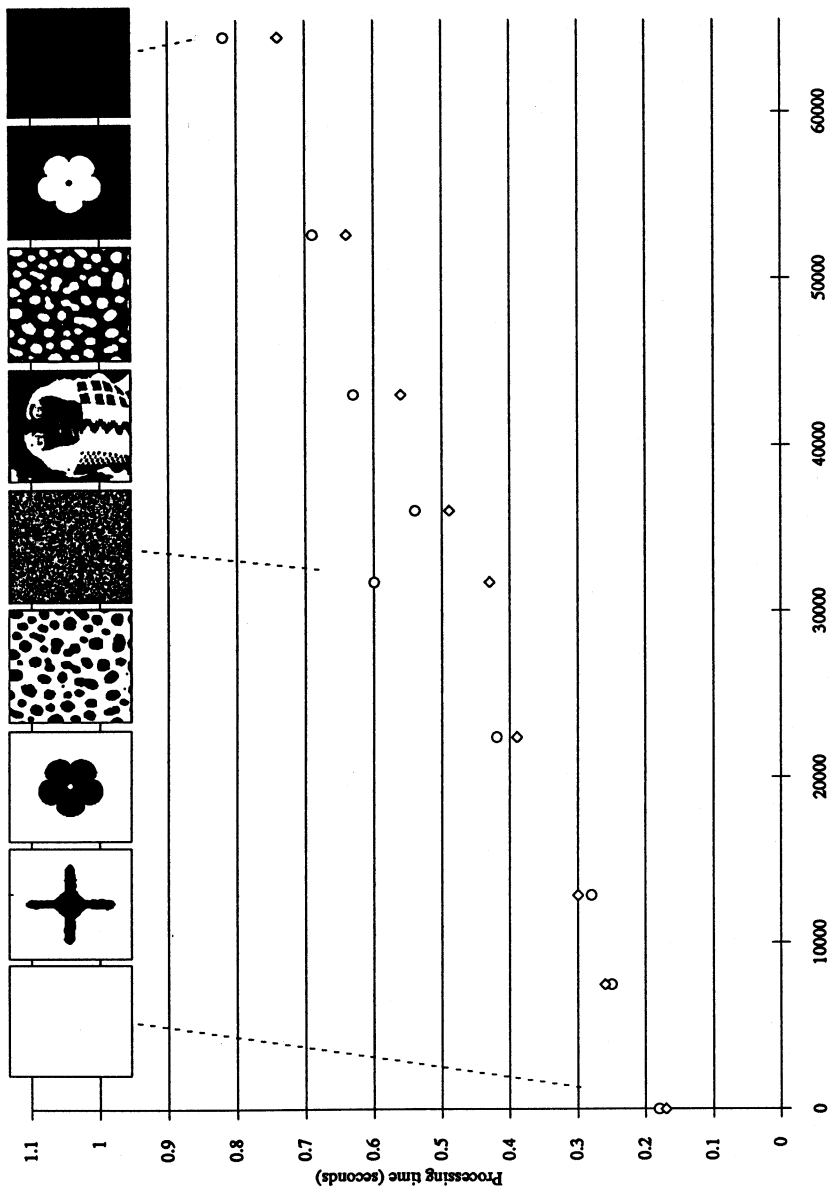


Fig. 8. Processing times for test images (0 = hexadecagonal, ◇ = city-block) on a Sparcstation IPX. The black pixels are the object pixels.

Grey-value thinning can be achieved by applying binary operations to the results of thresholding at subsequent grey-values. Depending on the processing order we can distinguish two types of grey-value thinning[14]: lower thinning and upper thinning (which correspond by duality respectively to lower thickening and upper thickening).

5.1. Lower Skeleton

The lower skeleton owes its name to the fact that the resulting divides do not always follow the surface of the grey-value landscape. Due to the order in which the pixels are processed from the highest grey level to the lowest — the divides can penetrate the landscape. When a ridge surrounds a watershed tile (a crater), all pixels belonging to that watershed tile get assigned the lowest value present in the valley.

Due to the processing order from the highest to the lowest value, pixels present in a certain grey level are also present in all subsequent levels. This implies that pixel values can be changed more than once and multiple passes through the image are required to achieve the final result.

Algorithms

Lower thinning can be implemented in two principle ways: The first approach makes multiple passes through the image and applies a skeleton test to pixels at all levels. The second works strictly at one level and makes exactly one pass through the image per grey level.

Goetcharian[20] gives an example of the first approach. He extended the Arcelli[21] skeleton — which is a parallel algorithm — to grey-value images. For each iteration, a set of eight masks have to be applied to the image. This must be repeated until stabilization occurs. This algorithm is mainly intended for processor arrays.

Along the same lines, a straightforward extension of the Hilditch skeleton[16] to grey-value images can be made by applying a threshold to the current 3 by 3 neighborhood at the level of the central pixel. If the Hilditch test applied to the thresholded values allows removal of the current pixel, the value of the central pixel is replaced by the minimum value of its 4-connected neighbors. Unfortunately, one cannot employ a fast implementation of the Hilditch skeleton to speed up calculations.

The second approach allows an implementation better suited for sequential machines. Duin and Verbeek[22] used a binary *anchor* skeleton with the Hilditch connectivity test as described in the section about binary skeletons. The *anchor* skeleton as proposed by Verbeek and Duin[23] is a variant of the skeleton that requires two input images, one containing the binary mask to be skeletonized and one with pixels that are per definition skeleton pixels (the anchor). The lower skeleton can be obtained by skeletonizing a grey-value image level by level, starting at the highest grey level. The skeleton pixels of an upper level are anchor pixels in a lower level.

Initially the binary input image, the anchor image and the grey-value output image are cleared. For all grey levels — starting with the highest level — the following steps have to be taken:

- Threshold the input image at the current level and store the result in the binary input image.
- Apply the anchor skeleton to the binary input image.
- Find all new skeleton pixels and set their values in the output image to the current level. When operating at the level of a sink, all pixels belonging to that watershed tile are surrounded by a divide in the anchor image. This anchor prevents those pixels from being removed. This way all points of the watershed tile receive the sink's value.
- Copy the intermediate skeleton result to the anchor image.

The algorithm produces a grey-value image which is homotopic equivalent to the original image. The binary quasi-Euclidean skeleton presented in the preceding section can be easily modified to an anchor skeleton as demonstrated by Van Vliet and Verwer.[18]

5.2. Upper Skeleton

As opposed to the lower skeleton the divides produced by the upper skeleton always run over the surface of the grey-value landscape. Consequently, this technique can be used for the detection of ridges and ruts. The upper skeleton starts at the lowest level and completes all operations on that level before moving on to the next grey level. Using this characteristic we propose an algorithm for upper thinning in which each pixel is processed only once.

Algorithms

In an initialization pass through the image the coordinates of all pixels are stored in a list. Then the list is sorted by increasing grey level, allowing pixels to be processed in order of grey level quickly. Since the coordinates of each pixel are stored only once, and pixels are processed only when they are retrieved from the list, each pixel will be processed only once as with the binary skeleton.

Initially all pixels in the binary output image are labeled "skeleton". For all grey levels starting with the lowest level a twofold procedure is applied.

First, insert new holes in the binary output image for all sinks at the current grey level, i.e. at places where all 4-connected neighbors have a value greater or equal than the pixel to be processed. This can be considered an initialization step for each grey level. Note that direct addressing allows processing of pixels of interest only. The image is not scanned.

Secondly, if necessary, the algorithm of Sec. 4, Appendix B is applied, on plateaus of pixels with the same grey-value. The points on the plateau are ordered, as shown in Fig. 9,[14] by increasing distance to the lower levels. The algorithm again only addresses pixels of interest by propagating over the plateau from the pixels

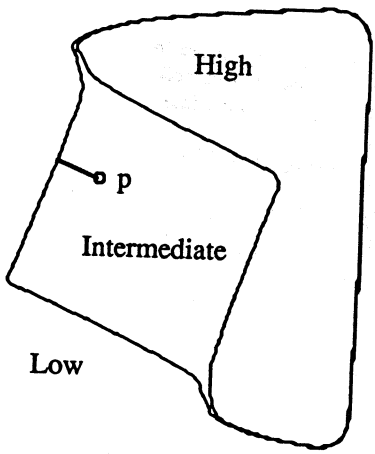


Fig. 9. On a plateau points which are closer to a lower level are considered lower than points which are further away from the lower levels.

connected to lower levels. If a pixel is removed from the binary image its grey value is replaced by the minimum value of its 4-connected neighbors.

Since the resulting skeleton will be 8-connected, the pixels of a watershed tile should be 4-connected. Pixels reached with the d_{11} and d_{12} prime vectors may only be processed if the 4-connected pixels in-between are also part of the same level (Fig. 10). This complication makes the directional information introduced in Sec. 2 redundant, so the simple version of the propagation method is used.

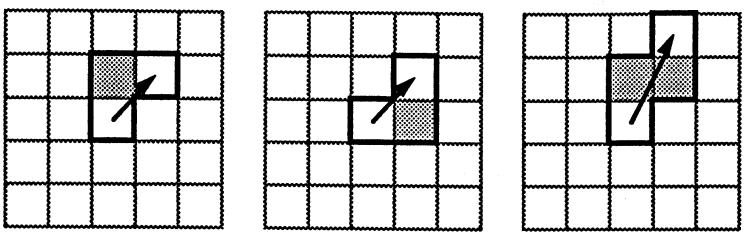


Fig. 10. (a) and (b) For a d_{11} transition, one shaded pixel should be on the plateau (4-connected path outlined) (c) For a d_{21} transition, both shaded pixels should be on the plateau.

The algorithm produces a binary output image containing the divides and a grey value image that is homotopic equivalent to the input image. Apart from the first pass each pixel is processed once. Appendix C gives a pseudocode of the algorithm.

Figure 11(a) shows an electron microscope image containing gold particles embedded in glass. Grey value skeletons are very sensitive to minor local variations in grey level producing spurious craters and divides in the resulting skeleton. To

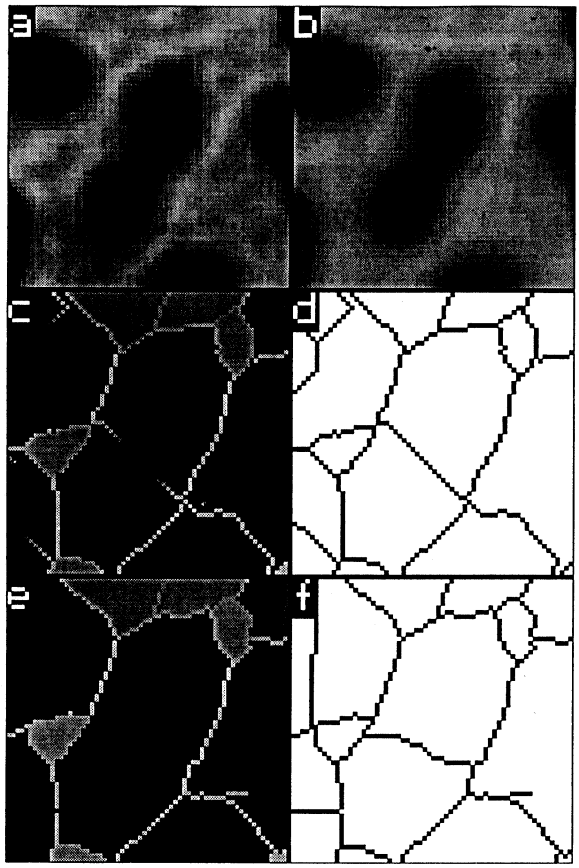
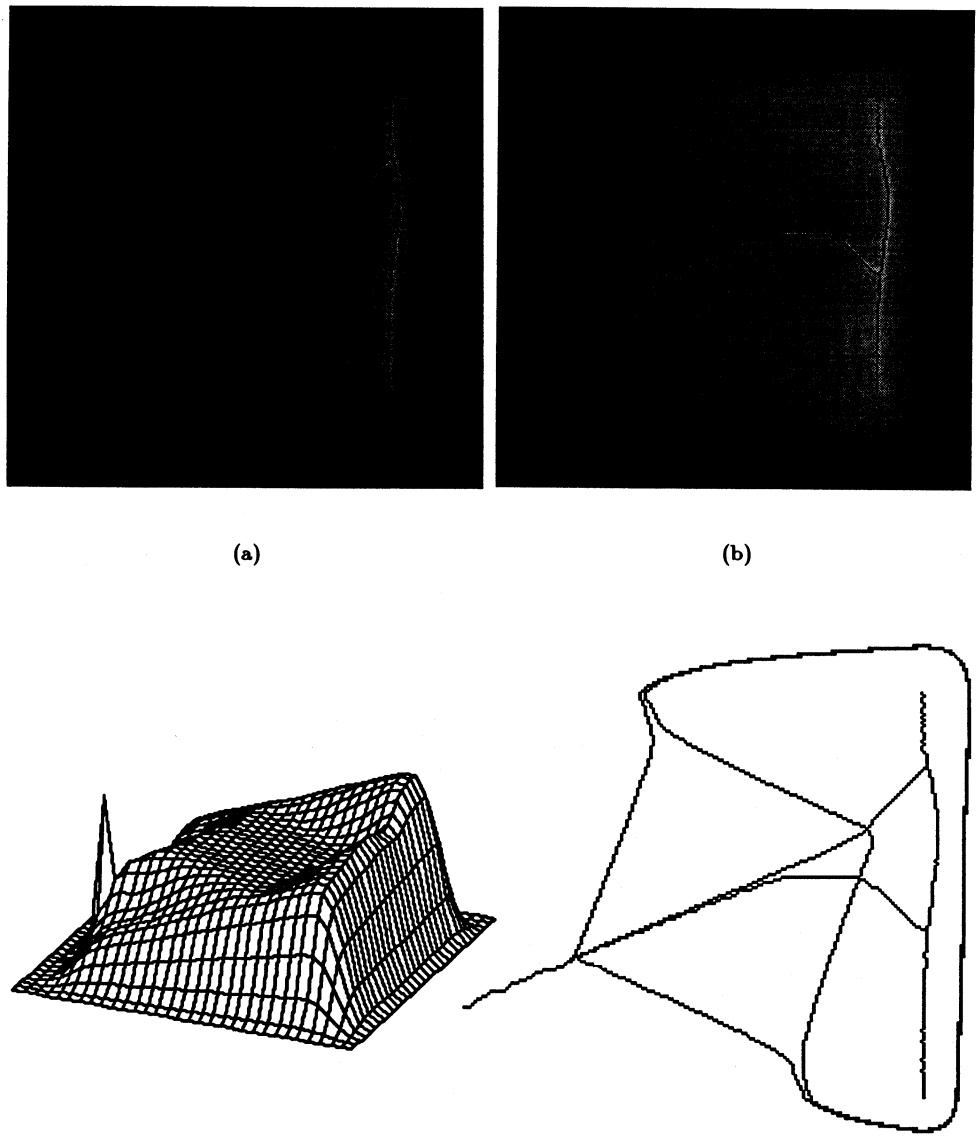


Fig. 11. (a) 64 x 64 image of gold particles in glass after contrast sketch (b) Smoothed with $\sigma = 2.25$ (c) Upper skeleton (d) Ridges of upper skeleton (e) Lower skeleton (f) Ridges in lower skeleton: the arrow points at a position in the lower skeleton where the ridge goes into the landscape of the original, as can be seen by the low grey value on the ridge.

reduce this side-effect we filtered the image with a low pass filter before skeletonization (Fig. 11(b)). The difference between the upper and lower skeletons is indicated in Figs. (c) through (f). In Fig. 11(e) it is clearly visible that the ridge penetrates the landscape, whereas is does not in Fig. 11(c).

5.3. Isotropy

The position of the grey value skeletons is mainly dictated by the grey levels. The combination of the grey levels with the skeletonization can give rise to preferential directions (see Fig. 6 of the binary skeleton and read the distances there as grey levels). This explains the preference for diagonals in Fig. 11(c). On plateaus the



(a) (b)

(c) (d)

Fig. 12. (a) Artificial landscape with plateau; black shows the ridges found with the upper skeleton using a hexadecagonal metric. (b) Same using a city block metric (c) Original, notice the peak in the lower left corner which is the start of the ridge (d) Top view showing the error introduced by the city-block metric.

metric starts to play its role. Figure 12 shows an artificial image from which the same conclusion as in Sec. 4 can be drawn: the metric is important and should be uncoupled from the topology testing.

5.4. Processing Times

The processing times for the various algorithms will differ considerably. To obtain the upper skeleton each pixel in the image is processed only once. Therefore the processing times are linear in the number of pixels — $O(N^2)$ — and independent of the number of grey levels.

The two strategies for the lower skeleton are of a higher order of complexity. The first method requires O(N) passes through the image making the algorithm $O(N^3)$. The second method needs as many passes as there are different grey levels present in the image. The complexity of this algorithm is $O(N^2\ 2^{\text{bitsPerPixel}})$.

To illustrate the remarkable speed of our implementation of the upper skeleton we compared the processing times of our upper skeleton with the processing times of the lower skeleton using the anchor skeleton. A straightforward implementation of the upper skeleton — a complete pass through the image for each grey level — will be of the same order of complexity as the lower skeleton. The processing times on a SUN SPARC station IPX are given in Table 1.

Table 1. Processing times of upper and lower skeleton in seconds on a SUN SPARC station IPX.

	Upper skeleton (s)	Lower skeleton (s)
gold (256 x 256) 8 bits	1.6	121.1
gold (128 x 128) 8 bits	0.4	28.9
gold (128 x 128) 4 bits	0.4	1.9

6. CONCLUSIONS

We have shown the importance of metrics in image processing and applied it to one specific type of application, skeletonization. Using a path based metric, better approximations of the Euclidean metric can be achieved. Processing times are low because the number of pixels addressed is kept to a minimum. This applies to both binary skeletonization and the upper grey-value skeleton.

ACKNOWLEDGMENTS

This research was partially supported by the Dutch Government as part of the SPIN-FLAIR-II program "Delft Intelligent Assembly Cell" and by the Netherlands Foundation of Medical Research, MEDIGON NWO, grant no. 900-538-016.

APPENDIX A

Recall the conditions for a metric:

- $d(\mathbf{u}, \boldsymbol{\nu}) = 0$, if and only if $\mathbf{u} = \boldsymbol{\nu}$; (1)
- $d(\mathbf{u}, \boldsymbol{\nu}) = d(\boldsymbol{\nu}, \mathbf{u})$; (2)
- $d(\mathbf{u}, \boldsymbol{\nu}) \leq d(\mathbf{u}, \mathbf{w}) + d(\mathbf{w}, \boldsymbol{\nu})$. (3)

- Define a set P of prime vectors on the grid:

$$P = \{\mathbf{p}_1, \ldots, \mathbf{p}_l\} \tag{4}$$

- Define a path Q as a sequence of prime vectors:

$$Q = (\mathbf{p}_{i_1}, \ldots, \mathbf{p}_{i_m}),\ 1 \leq i_k \leq l \tag{5}$$

- Assign local distances d_i to the prime vectors $\mathbf{p}_i$ and define the length L of the path Q as:

$$L(Q) = \frac{1}{s} \sum_{k=1}^{m} d_{i_k} \tag{6}$$

where s is a constant, real-valued scale factor, useful if we constrain the local distances d_i to be integer valued. If only relative distances are used, as in skeletonization, s is not of importance.

- Let the length of the empty sequence be zero:

$$L(\emptyset) = 0 \tag{7}$$

- Define the chamfer distance d_c between two points $\mathbf{u}$ and $\boldsymbol{\nu}$ on the grid as the minimum path length of all paths between $\mathbf{u}$ and $\boldsymbol{\nu}$ (Fig. 1):

$$d_c(\mathbf{u}, \boldsymbol{\nu}) = \min_{Q}\{(L(Q)\} \tag{8}$$

The local distances d_i are generally chosen to minimize the difference between $d_c(\mathbf{u}, \boldsymbol{\nu})$ and the Euclidean distance $d_e(\mathbf{u}, \boldsymbol{\nu})$:

$$d_e(\mathbf{u}, \boldsymbol{\nu}) = \sqrt{\sum_{j=1}^{N} (u_j - v_j)^2} \tag{9}$$

where u_j and v_j are the coordinates of $\mathbf{u}$ and $\boldsymbol{\nu}$ in the space with dimension N.

Let $\mathbf{p}_i$ be a prime vector connecting point $\mathbf{a}$ with point $\mathbf{b}$. Then $\mathbf{p}_{i'}$, with local distance $d_{i'}$, is defined as the prime vector connecting $\mathbf{b}$ with $\mathbf{a}$.

Theorem 1. If $\forall i : d_i > 0$ and $d_i = d_{i'}$, then $d_c(\mathbf{u}, \boldsymbol{\nu})$ is a metric.

Proof condition (1).
If $\mathbf{u} = \boldsymbol{\nu}$, $d_c(\mathbf{u}, \boldsymbol{\nu}) = 0$ by definition (7) and (8), if $\mathbf{u} \neq \boldsymbol{\nu}$, the path from $\mathbf{u}$ to $\boldsymbol{\nu} \neq \emptyset$ and $d_c(\mathbf{u}, \boldsymbol{\nu}) > 0$ since $\forall i\, d_i > 0$.

Proof condition (2).
If $Q_1 = (\mathbf{p}_{i_1}, \ldots, p_{i_m})$, $1 \leq i_k \leq l$ is a minimum path from $\mathbf{u}$ to $\boldsymbol{\nu}$, then we can construct a path Q_2 from $\boldsymbol{\nu}$ to $\mathbf{u}$: $Q_2 = (\mathbf{p}'_{i_m}, \ldots, p'_{i_1})$, $1 \leq i_k \leq l$. If Q_2 is a minimum path, $d_c(\mathbf{u}, \boldsymbol{\nu}) = d_c(\boldsymbol{\nu}, \mathbf{u})$.

Suppose Q_2 is not a minimum path but $Q_3 = (\mathbf{p}_{j_1}, \ldots, \mathbf{p}_{j_n})$, $1 \leq j_k \leq l$, from $\boldsymbol{\nu}$ to $\mathbf{u}$ is. Then we can construct a path $Q_4 = (\mathbf{p}'_{j_n}, \ldots, \mathbf{p}'_{j_1})$, $1 \leq j_k \leq l$, from $\mathbf{u}$

to ν. $L(Q_4) = L(Q_3) < L(Q_2) = L(Q_1)$. Contradiction since Q_1 was a minimum path.

Proof condition (3).
Let Q_1 be the minimum path from $\mathbf{u}$ to ν. If $Q_2 = (\mathbf{p}_{i_1}, \ldots, \mathbf{p}_{i_m})$, $1 \leq i_k \leq l$ is a minimum path from $\mathbf{w}$, and $Q_3 = (\mathbf{p}_{j_1}, \ldots, \mathbf{p}_{j_n})$, $1 \leq i_k \leq l$ is a minimum path from $\mathbf{w}$ to ν, then the concatenation $Q_4 = Q_2Q_3 = (\mathbf{p}_{i_1}, \ldots, \mathbf{p}_{i_m}, \mathbf{p}_{j_1}, \ldots, \mathbf{p}_{j_n})$, has a length $d_c(\mathbf{u}, \mathbf{w}) + d_c(\mathbf{w}, \nu)$.

Suppose $d_c(\mathbf{u}, \nu) > d_c(\mathbf{u}, \mathbf{w}) + d_c(\mathbf{w}, \nu)$, then $L(Q_1) > L(Q_4)$, but Q_1 was a minimum path. Contradiction.

APPENDIX B

// Pseudo code for binary skeleton

```
for all background pixels            // initialization
    label pixel "treated"
    if pixel is 4-connected to object pixel
        store coordinates of pixel in bucket "0"
        store direction "none" in bucket "0"
    endif
endfor

for d is d10 to infinity

    if all buckets are empty then exit

    // first find all pixels with distance value d
    for all generating pixels in bucket "(d − d10) mod (d12+1)"
        recall direction generating pixel
        if direction generating pixel is "none" then      // background pixel
            for successor is all 4-connected neighbors
                if successor is not labeled "treated" then
                    label successor "treated"
                    store coordinates successor in bucket "d mod(d12+1)"
                    store direction successor with respect to
                        generating pixel in bucket "d mod (d12+1)"
                endif
            endfor
        else
            let successor be 4-connected neighbor to which dashed arrow in
                Fig. 4(a) points (dependent on the direction of generating pixel)
            if successor exists and is not labeled "treated"
                label successor "treated"
                store coordinates of successor in bucket "d mod (d12+1)"
                store direction successor with respect to generating
```

```
                    pixel in bucket "d mod (d12+1)"
                endif
            endif
        endfor
        for all pixels in bucket d − d11 analogous (see Fig. 4(b)).
        for all pixels in bucket d − d12 analogous (see Fig. 4(c)).

        // check topology conditions
        for all pixels in bucket d
            if pixel may be removed without changing topology
                label pixel "candidate for removal"
                // label used as recursive value in neighborhood
            endif
        endfor
        for all pixels in bucket d      // update image
            if pixel is labeled "candidate for removal"
                label pixel "removed"
            endif
        endfor
endfor
```

APPENDIX C

```
// Pseudo code for upper (grey-value) skeleton

make histogram of image
allocate an array Ag for each grey-level to store the coordinates of pixels of
that grey-level
    // length of Ag known from histogram

for all image pixels      // initialization
    if grey value g of pixel != 0
        label pixel "skeleton"   // will remove these everywhere except from
        real skeleton pixels
        put coordinates of pixel in array Ag
    endif
endfor

for all grey values g, starting with 1 to maximum grey-value

    for all pixel with grey value g (found in array Ag)
        if pixel 4-connected to lower grey value
            store coordinates of pixel in bucket 0   // edge of plateau to
            lower level
        else
            remove label "skeleton"   // if local minimum, permanently,
            else temporarily
    endfor
```

```
// first determine ordering of pixels on plateau
for d is d10 and up, until buckets are empty
    // find all pixels with distance value d
    for all pixels in bucket "(d − d10) mod (d12 + 1)"
            // first one is 0, the modulo is to reuse memory space
        for successor is all 4-connected neighbors
            if grey value of successor = g // successor is on plateau
                label successor "skeleton" // next usage of same bit
                store coordinates successor in bucket "d mod(d12 + 1)"
                store coordinates successor in Ag // 2nd usage
            endif
        endfor
    endfor
        idem for all pixels in bucket "(d − d11) mod (d12 + 1)"
            // but make sure connecting pixels are on plateau too, Fig. 10
        idem for all pixels in bucket "(d − d12) mod (d12 + 1)"
        save number of pixels with distance d found
            // so we can trace in Ag where the different distance sequences start
    endfor // all pixels with grey value g, except local minima, have their "skeleton"
    bit set again
    // now skeletonization
    for d is 0 to maximum distance on plateau
        for all pixels with distance d // stored in Ag
            if pixel may be removed without changing topology
            // check neighbors "skeleton" and "candidate for removal" labels
                label pixel "candidate for removal"
                // label used as recursive value in neighborhood
            endif
        endfor
        for all pixels with distance d // go through Ag again
            if pixel is labeled "candidate for removal"
                remove label "skeleton"
                remove label "candidate for removal"
                assign pixel minimum of the grey values of its 4-connected
                neighbors
            endif
        endfor
    endfor
endfor
```

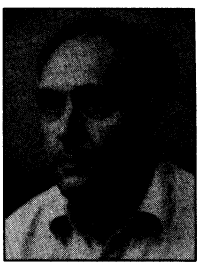
REFERENCES

1. U. Montanari, "A method for obtaining skeletons using a quasi-Euclidean distance", *J. ACM* **15** (1968) 600–624.
2. L. Dorst, "Pseudo-Euclidean skeletons", *Proc. Int. Conf. on Pattern Recognition*, Paris, 1986, pp. 286–288.
3. G. Borgefors, "Distance transformations in arbitrary dimensions", *Comput. Vision Graphics Image Process.* **27** (1984) 321–345.
4. G. Borgefors, "Distance transformations in digital images", *Comput. Vision Graphics Image Process.* **34** (1986) 344–371.
5. B. J. H. Verwer, "Improved metrics in image processing applied to the Hilditch skeleton", *Proc. Int. Conf. on Pattern Recognition*, Rome, 1988, pp. 137–142.
6. A. Rosenfeld and J. L. Pfaltz, "Distance functions in digital pictures", *Pattern Recogn.* **1** (1968) 33–61.
7. B. J. H. Verwer, "Local distances for distance transformations in two and three dimensions", *Pattern Recogn. Lett.* **12** (1991) 671–682.
8. I. Ragnemalm, "Generation of Euclidean distance maps", Linköping University, 1990.
9. B. J. H. Verwer, "Distance transforms", PhD. thesis, Delft University of Technology, 1991.
10. M. Yamashita and T. Ibaraki, "Distances defined by neighborhood sequences", *Pattern Recogn.* **19** (1986) 237–246.
11. J. Piper and E. Granum, "Computing distance transformations in convex and non-convex domains", *Pattern Recogn.* **20** (1987) 599–615.
12. P. W. Verbeek, L. Dorst, B. J. H. Verwer and F. C. A. Groen, "Collision avoidance and path finding through constrained distance transformation in robot state space", *Intell. Auton. Syst.*, preprint, Amsterdam, 1986, pp. 634–641.
13. B. J. H. Verwer, P. W. Verbeek and S. T. Dekker, "An efficient uniform cost algorithm applied to distance transforms", *IEEE Trans. Pattern Analysis Mach. Intell.* **11** (1989) 425–429.
14. J. Serra, *Image Analysis and Mathematical Morphology*, chp. XII, Academic Press, London, 1982.
15. C. Arcelli, "Pattern thinning by contour tracing", *Comput. Graph. Image Process.* **17** (1981) 130–144.
16. C. J. Hilditch, "Linear skeletons from square cupboards", in *Machine Intelligence*, vol. 4, eds. B. Meltzer and D. Mitchie, University Press Edinburgh, 1969, pp. 404–420.
17. G. Borgefors and G. Sanniti di Baja, "Skeletonization the distance transform on the hexagonal grid", *Proc. Int. Conf. on Pattern Recognition*, Rome, 1988, pp. 504–507.
18. L. J. van Vliet and B. J. H. Verwer, "A contour processing method for fast binary neighborhood operations", *Pattern Recogn. Lett.* **7** (1988) pp. 27–36.
19. C. R. Dyer and A. Rosenfeld, "Thinning algorithms for grey-scale pictures", *IEEE Trans. Pattern Anal. Mach. Intell.* **1** (1979) 88–89.
20. V. Goetcharian, "From binary to grey-level tone image processing by using fuzzy logic concepts", *Pattern Recogn.* **12** (1980) 7–15.
21. C. Arcelli, L. Cordella and S. Levialdi, "Parallel thinning of binary pictures", *Electron Lett.* **11** (1975) 148–149.
22. R. P. W. Duin and P. W. Verbeek, "Grey value skeleton", Delft University of Technology (TN/FI-PH), internal rep., 1979.

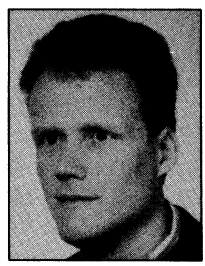
23. P. W. Verbeek and R. P. W. Duin, "Anchor skeletonization", Delft University of Technology (TN/FI-PH), internal rep., 1979.

Received March 1992; revised July 1992.

Ben J. H. Verwer studied applied physics at the Delft University of Technology. He obtained his Ph.D. in 1991. He is currently working for Becton Dickinson Cellular Imaging Systems, where he specializes in image processing for medical applications.



Piet W. Verbeek (1941) studied physics at Leyden University, his Ph.D. thesis (1973) was on quantum statistics and systems theory of magnetic relaxation. In 1973 he joined the Pattern Recognition Group of the Department of Applied Physics at Delft University of Technology. Since 1974 he works on Image Processing. Some topics: 3D skeletonization (1978), cell nucleus texture analysis (1979), texture segmentation (1980), alpha-pull (1981), video speed range sensor system (since 1985), max-min filtering (1988), distance transform and robot collision avoidance (1986–91), measurement in grey images (since 1985), image telephony (since 1990).



Lucas J. van Vliet was born in Schiedam, The Netherlands, on January 12, 1965. He received the M.Sc. degree in applied physics from the Delft University of Technology, Delft, The Netherlands, in 1988. In 1987 and 1988 he spent two periods (5 months and 2 months) as a postgraduate researcher at the University of California, San Francisco and at the Biomedical Sciences Division of Lawrence Livermore National Laboratory where he conducted research in the field of chromosome analysis.

Currently he is completing his Ph.D. thesis entitled "Grey-scale Measurements in Multi-Dimensional Digitized Images". His research interests include quantitative microscopy, image filtering and image analysis with emphasis on the accuracy and precision of measurements in digitized images.