A contour processing method for fast binary neighbourhood operations

Lucas J. van VLIET and Ben J.H. VERWER

Pattern Recognition Group, Faculty of Applied Physics
Delft University of Technology
Lorentzweg 1, 2628 CJ Delft, The Netherlands.

Abstract: In this paper new fast algorithms for erosion, dilation, propagation and skeletonization are presented. The key principle of the algorithms is to process object contours. A queue is implemented to store the contours in each iteration for the next iteration. The contours can be passed from one operation to another as well. Contour filling and object labelling become available by minor modifications of the basic operations. The time complexity of the algorithms is linear with the number of contour elements to be processed. The algorithms prove to be faster than any other known algorithms..

Keywords: Binary neighbourhood operations, skeleton, contour processing algorithm.

1. Introduction

Binary neighbourhood operations are basic operations in image processing. Their simple description has long allowed special purpose hardware to be constructed. In those situations where such hardware is not available, one has to restrict oneself to algorithms suited for general purpose computers. On general purpose computers a straightforward implementation of the binary neighbourhood operations performs poorly, performance being measured as the total required processing time to transform an input image into an output image.

Erosion, dilation, propagation and skeletonization are often iterated several times or even as many times as required to achieve stabilization. We propose a method in which only the interesting pixels are processed. This is achieved by keeping a simple administration in each iteration to aid the next iteration.

We know of three other cases in the literature which deal with fast binary neighbourhood operations. We will compare the performance of our algorithms with the results of Young et al. (1981), and Groen & Foster (1984). Groen & Foster (1984) used a table-lookup approach in combination with a region of interest per line. They sorted pointers in order to process connected runs of pixels. Young et al. (1981) used a run-length representation of images to achieve among others binary neighbourhood operations (erosion, dilation, propagation, medial axis transform). However, their medial axis transform is so bounded by the run-length representation that a comparison with the traditional Hilditch-skeleton (1969) is uncalled for.

We were not able to compare our results directly with the results of Piper (1985), who proposed a skeletonization method for an environment based on interval coding, but we will try to indicate how his results relate to ours.

For completeness we will give the processing times achieved by straightforward implementations of the operations as well.

b_3	b_2	b_1
b_4	b_8	b_0
b_5	b_6	b_7

Figure 1. 3x3 neighbourhood

2. Description of operations

We will briefly recapitulate the definitions of erosion, dilation, propagation and skeletonization. We will follow the notation of Groen & Foster (1984).

The operations treated in this paper are based on 3x3-neighbourhoods. Erosion, dilation and propagation work with neighbourhoods (figure 1) in the image before a particular iteration started. We

L.J. van Vliet, B.J.H. Verwer, A contour processing method for fast binary neighbourhood operations,
Pattern Recognition Letters, Vol. 7, No. 1, 1988, 27-36.

will refer to these neighbourhoods as non-recursive (figure 2a). Skeletonization as proposed by Hilditch (1969) uses non-recursive, recursive and partially recursive neighbourhoods. Recursive neigbhoorhoods are neighbourhoods in the image in which the values calculated are updated immediately; partially recursive neighbourhoods consist of seven neighbours taken from a non-recursive neighbourhood and one neighbour taken from a recursive neighbourhood (figure 2c, 2d, 2e, 2f).

For a 3*3-neighbourhood Groen & Foster define the number of 4-connected neighbours Φ_4:

$$\Phi_4 = \sum_{k=0,2,4,6} b_k$$

the number of 8-connected neighbours Φ_8:

$$\Phi_8 = \sum_{k=0,\ldots,7} b_k$$

and the Hilditch crossing number γ_h (1969):

$$\gamma_h = \sum_{k=1,2,3,4} h_k$$

where b_k denotes the binary pixel value at position k (figure 1) in a the neighbourhood (figure 2) and where $h_k = 1$ if ($b_{2k-2} = 0$) and ($b_{2k-1} = 1$ or $b_{2k} = 1$) else $h_k = 0$.

We assign object pixels the value one and background pixels the value zero. Below we have listed the conditions, which, if true, cause a change in the value of the central pixel b_8.

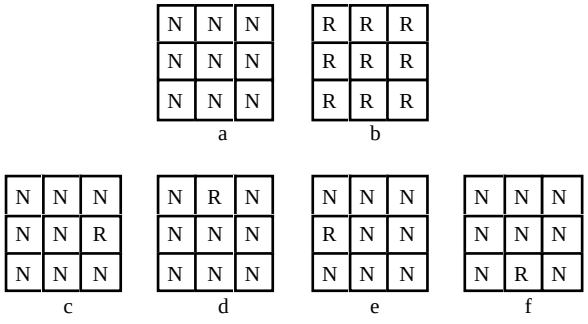


Figure 2. (a) non-recursive neighbourhood, (b) recursive neighbourhood, (c-f) partially recursive neighbourhoods. Letter N denotes a pixel from the non-recursive image. Letter R denotes a pixel from the recursive image.

Erosion

In each iteration those object pixels are removed, which were connected to at least one background pixel in the preceding iteration. The erosion therefore works with non-recursive neighbourhoods.

4-connected: $b_8 \rightarrow 0$ if ($b_8 = 1$ and $\Phi_4 < 4$),
8-connected: $b_8 \rightarrow 0$ if ($b_8 = 1$ and $\Phi_8 < 8$).

Dilation

In each iteration those pixels are added, which were connected to at least one object pixel in the preceding iteration. Again, only non-recursive neighbourhoods are used.

4-connected: $b_8 \rightarrow 1$ if ($b_8 = 0$ and $\Phi_4 > 0$),
8-connected: $b_8 \rightarrow 1$ if ($b_8 = 0$ and $\Phi_8 > 0$).

Propagation

Each iteration propagation consists of one iteration dilation of the input image (often called seed) followed by a logical AND with a mask image.

4-connected:
$b_8 \rightarrow 1$ if ($b_8 = 0$ and $m_8 = 1$ and $\Phi_4 > 0$),
$b_8 \rightarrow 0$ if ($b_8 = 1$ and $m_8 = 0$);
8-connected:
$b_8 \rightarrow 1$ if ($b_8 = 0$ and $m_8 = 1$ and $\Phi_8 > 0$),
$b_8 \rightarrow 0$ if ($b_8 = 1$ and $m_8 = 0$).

m_8 denotes the value of the pixel in the mask image which corresponds to the central pixel in the input image.

Hilditch skeletonization

Hilditch has developed a skeletonization method in which non-recursive, recursive and partially recursive neighbourhoods are used. Non-recursive neighbourhoods are used to avoid removal of end and break pixels; recursive neighbourhoods to avoid removal of single pixels and two pixel thick lines; partially recursive neighbourhoods to avoid erosion of two pixel thick lines from the end. Which partial recursive neighbourhoods have to be tested is dependent on the scan direction (see section 4).

Hilditch defined a crossing number (_h) to represent the topology. If and only if the crossing number equals one, may a pixel be removed. If the crossing number exceeds one, a pixel is a link- or vertex pixel. If the crossing number equals zero, the pixel is either a single pixel or lies in the middle of an object.

$b_8 \rightarrow 0$ if ($b_8 = 1$ and $\Phi_8 \neq 1$ and $\gamma_h = 1$).

Anchor Skeletonization

The anchor skeleton (Verbeek and Duin (1979)) is a variant of the skeleton in which some pixels are per definition skeleton pixels. These pixels are specified

in a separate image, called anchor image. The anchor skeleton is useful to connect different parts of a scene. E.g. when skeletonizing a greyvalue image greylevel after greylevel, the skeleton pixels in an upper level are 'anchored' in a lower level. Or when skeletonizing layers of printed circuit boards, connection points are anchored.

If a_8 denotes the value of the pixel in the anchor image, which corresponds to the central pixel in the input image, then the condition for change becomes:

$b_8 \rightarrow 0$ if ($b_8 = 1$ and $\Phi_8 \neq 1$ and $a_8 = 0$ and $\gamma_h = 1$).

Note that the anchor skeleton reduces to the ordinary skeleton if the anchor image is left empty. Therefore we have only implemented anchor skeletonization.

3. Principles of approach

Traditionally, binary neighbourhood operations have been implemented as table lookups. Each pixel is treated alike. We propose to take advantage of specific properties of operations to gain processing time. To process as less pixels as possible and to keep the calculations as simple as possible are the principles of our approach.

The operations described in the introductory remarks have in common the existence of a region of interest which consist of pixels on or connected to object contours. We propose to use a queue to address these pixels. In the initializing phase of the operations pointers to contour pixels are queued. During the iterations pixels which pop from the end of the queue are processed and the neighbours to be processed in the next iteration are pushed to the front of the queue.

A queue is most easily implemented as an array with two pointers to denote the 'front' and the 'end' of the queue. The pointers are incremented modulo the size of the array each time a pointer is read at the 'front' or written at the 'end'; shifting inside the queue is not necessary.

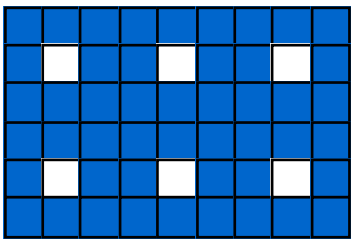


Figure 3. Pathological image pattern in which 8/9th of the total number of pixels belong to the contour.

Robustness however demands a queue-length of at least 8/9th of the size of the image to be processed to be able to store the maximum number of contour pixels which can occur in an image (see figure 3). Taking into account the fact that each pixel only needs to be queued once for the operations involved, moderate extra memory costs will simplify the administration to a large extent: if one increases the queue size to the size of the image to be processed the modulo requirement vanishes. We have adopted this approach for the algorithms presented in the next section.

If memory allocation should pose problems it is possible to stick to the 'modulo' approach. The queue size then has to be determined heuristically. Apart from pathological cases a queue size of about 20% of the size of the image is sufficient.

The second principle, to keep the calculations as simple as possible, is a derivative of the first. At the point where only contour pixels are processed the need for calculations can vanish. For example, if a contour pixel is removed in an erosion, all object neighbours are subject to removal the next iteration and any calculation in the next iteration would be redundant. Only the skeleton requires additional computations, induced by the need to hold break and end pixels.

From hereon we will speak of 'queueing a pixel' rather than the more precise 'queueing a pointer to a pixel'.

4. Algorithms

Introduction

The algorithms start with a call to a routine in which all contour pixels are queued. The skeleton uses a special routine because candidate pixels for removal may not be set in the anchor image. After the queue initialization, the operations proceed along different but comparable paths. Either the neighbours of the queue pixels are processed and queued or the pixels on the queue are processed themselves and their neighbours are queued. Most often the first method is used to prevent pixels from being queued twice without having to use labels. In the case of skeletonization the second method is more natural because the recursive and the non-recursive image must be available.

Connectivity

In binary neighbourhood operations connectivity is crucial. Pecht (1985) has shown that the shape of structuring elements can induce redundancy after one

or more iterations. For the 8-connected erosion and dilation only diagonal neighbours have to be considered after the first iteration. In the propagation redundancy is not present because of the constraints introduced by the mask image. Skeletonization erodes objects 4-connected and in the 4-connected case naturally only the horizontal and vertical neighbours have to be processed.

Queue initialization

The queue is initialized by filling it with the contour pixels of the objects in the image. A contour pixel is an object pixel which is connected to at least one background pixel.

We have implemented and compared two methods. It proved to be faster to check only object pixels in the detection of the contour if the 8-connected contour is desired or if less than fifty percent of the image is object. Otherwise it is faster to check all pixels with a scan method as proposed by Groen & Foster (1984).

The processing time of the scan method is image independent, while the processing time of the object-oriented method is image dependent. The scan method is implemented by a table-lookup. The table-entry is composed of the 3*3 neighbourhood. Three neighbours in the scan-direction are read for each pixel. In the entry of the previous pixel the upper three bits are stripped off, the result is shifted up three bits and the new values are added. In the detection of the 8-connected contour use of the scan-code is not advantageous.

Erosion

Our erosion starts with a call to the initialization procedure. In the first iteration all queued pixels are removed (i.e. the value of the pixels is changed from one to zero). In the next iterations the object neighbours of the pixels on the queue are removed and queued. Because of the sequential procedure each pixel will only be queued once. During the iterations queued pixels are counted in order to know how many pixels must be popped in the next iterations (the number of queued pixels is implicitly derived at the end of each iteration by taking the difference of the pointers to the 'front' and the 'end' of the queue.)

For the 4-connected erosion the horizontal and vertical neighbours are taken into account and for the 8-connected erosion the diagonal neighbours (see remarks in the section on connectivity). To prevent propagation outside the image, we have chosen to remove the edge of the image beforehand. If the edge is removed after calling the initialization routine, objects will not be eroded from the edges of the image; if it is called before, the objects are eroded from the edges as well.

A pseudo code of the algorithm can be found in the appendix.

Dilation

The dilation also starts with a call to the initializing procedure. Contour pixels which are 4-connected to the background are queued, independent of the type of connectivity of the operation because all background pixels can be reached from these pixels.

In the first iteration all background pixels, 4- or 8-connected to the queued pixels dependent on the connectivity of the operation, are added (the value is changed from zero to one) and queued. Again each pixel will only be queued once.

The next iterations are identical to the first apart from the fact that the 8-connected dilation only accesses the diagonal neighbours as described previously.

A pseudo code of the dilation can be found in the appendix.

Propagation

The propagation is the same as the dilation with one extra check. In the initialization seed contour pixels are queued after which the seed image is initialized with the logical AND between the seed and the mask image. In the iterations added neighbours have to be set in the mask image.

The propagation is often used to remove objects connected to the edges of an image. This is easily achieved in our algorithm by adding to the initialization a processing of the edges.

Hilditch skeleton

The most complex binary neighbourhood operation is skeletonization. A skeleton algorithm which treats the objects contour by contour and which uses recursivity to its advantage is the Hilditch skeleton.

Hilditch (1969) uses a fixed scan direction (top-down, left to right) and the newly found values for the north and west neighbours are put in the neighbourhoods in the original image to avoid removal of two pixel thick lines. In our skeletonization algorithm, in which a fixed scan direction does not exist (figure 4), all 4-connected neighbours have to be tested in partially recursive neighbourhoods (see figure 2, section 2).

The implementation of the Hilditch skeleton uses three binary images:

- an input image;
- an anchor image, containing the pixels which are by definition skeleton pixels;
- a change image, storing the pixels changed in a current iteration.

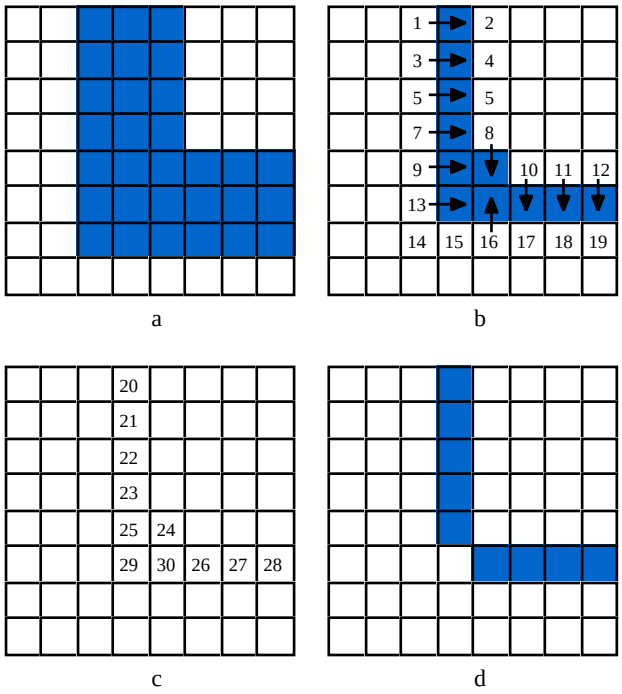


Figure 4. Hilditch skeletonization. The order of processing is image dependent.

The Hilditch skeleton is 8-connected. Hence, the image has to be eroded 4-connected. The queue is initialized with unanchored 8-connected contour pixels. In the first iteration each queued pixel is removed if it is not a break or end pixel in the original neighbourhood, not a break pixel in the recursive neighbourhood and not a break pixel in the four partially recursive neighbourhoods. Of course each of the latter has to be tested only if that neighbour is set in the change image.

If a pixel may be removed it is set in the change image. Detected break and end pixels are anchored because: 'once a break pixel, always a break pixel' and 'once an end pixel, always an end pixel'.

When all queued contour pixels have been processed, the input image is updated by removing the pixels set in the change image. These pixels can quickly be addressed by passing through the queue a second time. Simultaneously a new queue is built. All unanchored object pixels, 4-connected to the changed pixels, are queued and temporarily anchored to prevent double queueing. This temporary anchor is removed during the testing phase of the next iteration which is, apart from that, the same as in the first iteration. Note that the temporary anchor can be discerned from the original anchor because the temporary anchor pixels are queued and the original anchor pixels are not. The skeletonization procedure finishes after a desired number of iterations or when stabilization occurs.

The Hilditch skeleton without end pixels proved to be relatively time consuming compared to the Hilditch skeleton with end pixels. Without endpixel condition a pixel which is a breakpixel in some iteration will be a candidate pixel for removal in all next iterations and therefore all one pixel thick lines need to be queued over and over again.

A solution is to obtain the skeleton without end pixels from the skeleton with end pixels by eroding from the end pixels until closed contours or single pixels remain. The anchor which has been filled with detected skeleton pixels has to be restored before the erosion from the end pixels can start.

If objects connected to the edge should stay connected to the edge, the edge is anchored and made object, else the edge is made background before calling the initialization procedure.

A pseudo code of the Hilditch skeletonization can be found in the appendix.

Combinations of operations

Binary neighbourhood operations can be combined to compose higher level operations. Examples of higher level operations are opening (erosion followed by dilation), closing (dilation followed by erosion), separation of objects with and without holes (skeleton without end pixels followed by pepper removal and propagation) or removing small objects (erosion followed by propagation).

These higher level operation can be programmed efficiently with our algorithms by passing the region of interest from one operation to another. Between the operations the image need not be scanned for the contours.

The queue itself cannot be passed on straightforwardly. Some operations end with a queue containing the background contours, others with a queue containing the object contours. Moreover the queue has to be shifted to the start again. Interaction routines to build a queue containing object contour pixels from a queue of background contour pixels and vice versa are easy to construct. In the first case the object neighbours of the queued pixels become the new queue. In the second case the background neighbours of the queued pixels.

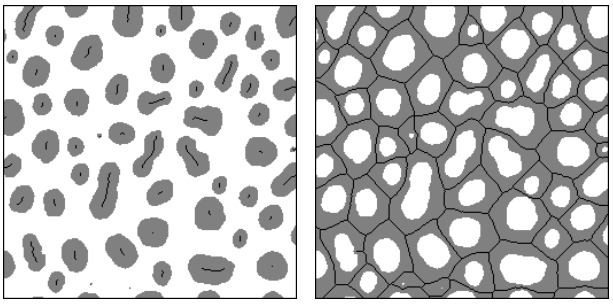
The processing time drops dramatically if the queue is passed on. The interaction routine is of time complexity linear in the number of contour elements, whereas a complete initialization is of time complexity linear in the number of image pixels.

Contour filling and object labelling

Propagation of interesting pixels (contours) can be used to fill contours and to label objects. In the case of contour filling, a pixel inside a contour is queued and subsequently the area inside the contour is filled by queueing and adding all connected background pixels inside the contour. In the case of object labelling, the image is sequentially scanned. As soon as a non-labelled object pixel is encountered, this pixel is queued and all connected object pixels are labelled subsequently as in the contour filling case. Forks, spirals, etc. do not pose any problems as opposed to a method which scan the image. The time to label an image is proportional to the number of object pixels.

5. Results

We have used two 2562 test images to compare the speed of our algorithms, referred to as the VVV-algorithms, with the CLP-algorithms of Groen & Foster (1984) and the PXY-algorithms of Young et al. (1981). The test images were a thresholded image containing gold particles in glass (figure 5a) and the background of this image (figure 5b). The objects are of the shape frequently encountered in biomedical cell analysis, the background is used to observe the behaviour of the algorithms when a lot of holes are present in an object. Depending on the operation, the results on completely filled or empty images are given as well, as are the results of straightforward implementations.



a) b)

Figure 5. (a) Thresholded image containing gold particles in glass plus the skeleton. (b) Background of the same image plus the skeleton.

The computer used in this experiment was a MicroDutch. This is a VME-bus system built around a Motorola 68020 with a clock frequency of 12.5 MHz., running under UNIX V.2. All programs have been written in C.

Piper (1985) has proposed a method to implement the Hilditch skeleton using interval coding. In his experiments he used a VAX 11/750 running under UNIX which is three to four times slower than the MicroDutch we used. Nevertheless, the performance of his method can be estimated. Assume that his system is four times slower than ours and that his image (Piper (1985), figure 5.b) is comparable to our figure 5.b. Hence, the estimated processing time of his algorithm on our system on the image of figure 5.b is 10 seconds, twice the time of our algorithm.

PXY-algorithms

The PXY-algorithms work on run-length coded images. The run-codes are stored in pxy-tables. A pxy-table contains the addresses of both the start and the stop of each run. Processing a pxy-table means processing only object contours.

Monadic and dyadic point operations (NOT, AND, OR, XOR) are the basic operations in the PXY-algorithms. A dilation is implemented by shifting the runs in a pxy-table up, down, left and right and by a logical OR between the original image and the shifted image. An erosion is implemented as a dilation of the background.

The strength of the PXY-algorithms is the possibility to use the output pxy-table of one operation as input for a next operation. A disadvantage is the large amount of memory space required for complicated images.

CLP-algorithms

The CLP-package of Groen & Foster does not process whole images, but only regions of interest. During one iteration the candidate pixels for the next iteration are stored in a list. Per row the candidate pixels are sorted to obtain runs of successive pixels. In processing these runs a table-lookup is used. The table-index is filled according to the scan-code, so only three new neighbours have to be read for each pixel on a line.

The computational overhead introduced by sorting is large. Therefore a simple sub-optimal procedure often proves to be as fast: instead of building a complete candidate pixel list, only the minimum and maximum column value of the changed pixels are used in setting the region of interest.

The advantages of the methods are their general applicability for binary neighbourhood operations and their limited use of memory. A disadvantage is the time consuming sorting in the optimal CLP-procedure, as will become clear in the next section.

Measurements

The results are summarized in three tables. Table 1 contains the processing times of figure 5a, table 2 of figure 5b and table 3 of empty and filled images.

Table 1. Times (in seconds[1]) required to process image of Figure 5a. VVV times include 0.6 seconds to build a queue with contour pixels. PXY times include 0.4 seconds to build a PXY-table and 0.2 seconds to convert the output PXY-table to an image.

	10 iterations 4-connected erosion	10 iterations 4-connected dilation	Skeleton with end pixels	Skeleton without end pixels	Propagation from edge
VVV[1]	0.9	1.1	3.4	3.9	0.5
CLP	8.9	12.7	12.0	12.0	6.4
PXY	1.5	2.3	–	–	–
SFW	7.7	9.4	22.3	40.3	32.3

Table 2. Times (in seconds[1]) required to process image of Figure 5b. VVV times include 0.8 seconds to build a queue with contour pixels. PXY times include 0.4 seconds to build a PXY-table and 0.2 seconds to convert the output PXY-table to an image.

	10 iterations 4-connected erosion	10 iterations 4-connected dilation	Skeleton with end pixels	Skeleton without end pixels	Propagation from edge
VVV[1]	1.3	1.1	5.3	5.9	1.2
CLP	12.8	9.1	19.5	19.5	34.7
PXY	2.4	1.5	–	–	–
SFW	9.4	7.7	34.6	35.1	97.1

Some comments on the experiments. We have only compared the 4-connected algorithms. The results for the 8-connected case show the same trend. The PXY-propagation was not available. The VVV-skeleton without end pixels consisted of a skeleton with end pixels followed by a pass through the image to detect all non-break pixels and subsequent erosion from these pixels. The initialization uses the first method of section 4. The VVV-propagation from the edge used a redundant pass through the image to test for other seeds than the edge. If this pass is omitted the processing time becomes 0.5 second less. The CLP-algorithms we used were the optimal algorithms as described by Groen & Foster (1984).

Table 3. Processing times (in seconds[1]) to clear a completely filled image by erosion or to fill an almost empty image by dilation of one pixel.

	10 itera-tions 4-connected dilation	Skeleton without end pixels
VVV[1]	2.0	1.4
CLP	32.0	55.6
PXY	3.7	4.8
SFW	114	105

The straightforward implementation (SFW) of the skeleton is a scan-code algorithm. At each pixel a table entry is composed of the neighbours by shifting and refreshing three new pixel values. The recursive and partially recursive neighbourhoods are constructed from this entry as well. In the straightforward implementation of the other operations for each pixel the conditions as described in section 2 are evaluated.

The VVV-algorithms calculate a skeleton 3 to 4 times faster than the CLP-algorithms. The dilations and erosions are an order of magnitude faster than the CLP-algorithms and almost 1.5 to 2 times as fast as the PXY-algorithms. The processing times of the straightforward implementations show that intelligent algorithms have their benifits, but only if the overhead is kept small.

6. Conclusion

We have implemented and tested algorithms for binary neighbourhood operations. The operations are erosion, dilation, propagation and skeletonization. The algorithms process only object contours by storing pointers to relevant pixels.

Our new algorithms are the fastest in all tests. Compared to the PXY-erosion and -dilation of Young et al. (1981), the gain in processing time is small, but since these algorithms do not offer a reasonable skeleton, the large difference with the next available

[1] In 1998 (10 years later) the processing times listed in table 1-3 have been reduced by more than a factor of 100 using a Pentium II or PPC G3 at 250 MHz.

alternative, the CLP-algorithms of Groen & Foster (1984), is more important.

From the basic operations other operations can easily be derived, such as contour filling and object labeling. If operations are combined, e.g. opening and closing, initialization is redundant and the algorithms become even faster.

Acknowledgement

We thank I.T. Young, R.C. Peverini, P.W. Verbeek, P.J. van Otterloo, F.C.A. Groen and N.J. Foster for making their software available to us, and TPD - TNO (Institute of Applied Physics Delft) for using their computer system.

References

F.C.A. Groen and N.J. Foster, (1984), A fast algorithm for Cellular Logic operations on sequential machines, Pattern Recognition Letters 2, 333-338.

C.J. Hilditch, (1969) Linear Skeletons from Square Cupboards, In: B. Meltzer and D. Mitchie, Ed., Macine Intelligence 4, University Press Edingburgh, 404-420.

J. Pecht, (1985) Speeding Up Successive Minkowski Operations with Bitplane Computers, Pattern Recognition Letters 3, 113-118.

J. Piper, (1985), Efficient Implementation of skeletonisation using interval coding, Pattern Recognition Letters 6, 389-398.

P.W. Verbeek and R.P.W. Duin, (1981), personal communication.

I.T. Young, R.C. Peverini, P.W. Verbeek and P.J. van Otterloo, (1981), A New Implementation for the Binary and Minkowski Operators, Computer Graphics and Image Processing 17, 189-210.

Appendix

The appendix contains the pseudo code of the erosion, dilation and the Hilditch skeleton.

Erosion *4-connected / 8-connected*

```
begin
    make image edge zero
    initialize the queue with pointers to pixels 4- / 8-connected to the background
    for all pointers on queue do
        remove pixel pointed to
    endfor
    while ( old_queue not empty and still iterations to go ) do
        for all pointers on old_queue do
            for all horizontal & vertical / diagonal neighbours do
                if neighbour = 1 then
                    neighbour = 0
                    put pointer to neighbour on new_queue
                endif
            endfor
        endfor
        old_queue = new_queue
    endwhile
end
```

Dilation *4-connected / 8-connected*

```
begin
    make image edge one
    initialize the queue with pointers to pixels 4-connected to the background
```

```
    for all pointers on queue do
        for all 4-connected/8-connected neighbours do
            if neighbour = 0 then
                neighbour = 1
                put pointer to neighbour on new queue
            endif
        endfor
    endfor
    while ( old_queue not empty and still iterations to go ) do
        for all pointers on old_queue do
            for all horizontal & vertical / diagonal neighbours do
                if neighbour = 0 then
                    neighbour = 1
                    put pointer to neighbour on new_queue
                endif
            endfor
        endfor
        old_queue = new_queue
    endwhile
end
```

Hilditch Skeleton

```
begin
    make image edge anchor
    initialize the queue with pointers to the 8-connected unanchored contour pixels
    while ( old_queue not empty and still iterations to go ) do
        for all pointers on old_queue do
            if ( γ_h = 1 in non-recursive, recursive and partially recursive
            neighbourhoods and Φ_8 ≠ 1 in non-recursive neighbourhood ) then
                set pixel in change image
            else
                anchor pixel
            endif
        endfor
        for all pointers on old_queue do
            if pixel pointed to is set in change image then
                remove pixels in change, mask and anchor image
            endif
            for all horizontal & vertical neighbours do
                if ( neighbour = 1 and neighbour ≠ anchor ) then
                    anchor the neighbour temporary
                    put pointer to neighbour on new_queue
                endif
            endfor
        endfor
        old_queue = new_queue
    endwhile
end
```