EUROGRAPHICS '94 / M. Daehlen and L. Kjelldahl (Guest Editors), Blackwell Publishers
© Eurographics Association, 1994

Volume 13, (1994), number 3

An Integrated Line Tracking and Vectorization Algorithm

Peter R. van Nieuwenhuizen[1], Olaf Kiewiet[2], Willem F. Bronsvoort[1]

[1] Faculty of Technical Mathematics and Informatics, Delft University of Technology, Julianalaan 132, 2628 BL Delft, The Netherlands
[2] DST Deutsche System-Technik GmbH, Hans-Bredow-Strasse 20, 28307 Bremen, Germany

Abstract

A method is presented to compute a vector representation of an arbitrary line in a bi-level raster image The line may have varying line width and irregular borders. Furthermore, at line crossings and branches the user is offered the opportunity to choose from the directions in which line tracking may he continued The method avoids thinning and vectorization of chain codes. Instead, a contour follower is used at both sides of the line, and median line calculation is integrated with contour following. This leads to an efficient algorithm, in particular when individual lines have to be interactively extracted from a raster image.

Keywords: image processing, line extraction, contour following, geographical information systems.

1. Introduction

In many graphical information systems, input consists of a scanned image. Examples are scanned maps in a geographical information system and scanned engineering drawings in a CAD system. Scanning of an image results in a raster image. In order to be able to manipulate, for example transform or select, lines and other primitives in such raster images, the primitives must be extracted from the raster image and be vectorized. In the past, many techniques for extraction and vectorization of primitives from raster images have already been developed. In this paper, we will focus on the extraction and vectorization of individual lines that may have arbitrary width, rough contours, crossings and branches.

In Parker [1988] a simple method is described to convert one-pixel-thick lines to a vector representation. However, if lines may he thicker than one pixel, conversion is much more complicated. A technique useful in this case is *thinning*, which calculates a skeleton of the objects involved. The skeleton of a single line in a raster image will be called the *median line*. Many thinning methods use a bi-level raster image as input, which may be obtained by thresholding a grey tone or colour image. This thresholding process is called *segmentation.* Various segmentation methods have been described by, for example, Pavlidis and Horowitz [1974], Davis et al. [1975], and Ohlander et al. [1978]. In thinning methods, layers of pixels are iteratively erased from the border of the objects, until a one-pixel-thick skeleton results. Thinning templates are used to decide which pixels of an object have to be removed in each iteration step. If one of the templates matches the environment of a pixel, the pixel will be removed from the object. In sequential thinning methods the pixel removal job during each iteration is done pixel by pixel, whereas in parallel methods all pixels are processed simultaneously. Furthermore, some parallel thinning methods use two passes during each iteration, because otherwise two-pixels-thick lines would be removed altogether, whereas other parallel methods need only one pass [Chin et al. 1087; Wu and Tsai 1992]. Pavlidis [1982] describes a two pass parallel thinning algorithm with the possibility to label the pixels of the skeleton, in order to be able to reconstruct the original raster image from the skeleton. In Tamura [1978] some sequential and parallel thinning algorithms are compared.

The result of the thinning process consists of one or more paths of pixels, each represented by a linear array of direction codes, called a *chain code*. From a chain code, a polyline can be calculated, for instance in the way described in Landy and Cohen [1985] or Lee et al. [1990]. With thinning and vectorization of the chain codes, a whole raster image of a line drawing can be converted into vector format.

The main disadvantage of these methods is that they are computationally expensive, because each pixel in the image must be addressed several times. This is in particular inefficient if only one line must be extracted and represented by a polyline, arid not the whole raster image. This situation may occur when, for example, a line in a raster image has to be highlighted by superimposing a polyline, or when information has to he attached to a road on a map or a pipe-line on a technical drawing of a chemical plant.

A basically different approach, suitable for vectorization of both individual lines and whole images, uses a contourfollowing algorithm to find the borders of the objects in a bi-level image. The result of contour following is a chain code that describes a closed path of pixels on the border of the object. From this chain code, a polyline that encloses the object is calculated. The vectorized borders can then be used to calculate the skeleton of the object. Although this method is considerably different from thinning methods based on templates, it is also called thinning by Martinez-Perez et al. [1987]. A disadvantage of this method is that it is very complicated.

Other techniques are based on the *Hough* transform. These kinds of methods are mainly used for the detection of straight lines [Duda and Hart 1972; Princes et al. 1990] or curves, such as circles [Kimme et al. 1975]. A survey of the Hough transform can be found in Illingworth and Kittler [1988].

In Dori et al. [1993] a set of algorithms with still another approach is presented. These algorithms extract straight line segments, circular arcs and arrow heads from raster images. For line extraction, an orthogonal zig-zag method is used that proceeds from one border of the line to be tracked to the other border, alternately along horizontal and vertical lines. The main advantage of this method is that the rater image is sparsely examined, i.e. only at areas where a primitive occurs that has to be tracked and vectorized.

The new line tracking and vectorization method that will be described in this paper also sparsely examines the raster image, and in addition it is suitable for arbitrary lines. The method integrates contour following and median line determination, which means that during contour following the median line will be calculated. Both thinning and vectorization of the borders of the line are thus avoided. Furthermore, the output of the algorithm will not be a chain code, but a polyline will be found immediately. In section 2, the principle of the method is described. In section 3, it is discussed how the algorithm will handle difficult cases, such as variable line width and line crossings. In section 4, results are shown and concluding remarks are made.

2. Contour following and median line determination

In this section, our line tracking method, which is based on contour following, will be explained. Two contour followers, one on the left and one on the right side of the line, proceed alternately. These are used for finding the median line of the line in the raster image that has to be tracked.

We assume the processed image to be a bi-level raster image, and define the neighhours of a pixel P as the set of 8 pixels N_i surrounding P, as shown in figure 1, Furthermore, we define an 8-connected object as a set of pixels with the property that between every pair of pixels in the set there exists a path of neighbours that are all in the set. An object is 4-connected if between every pair of pixels in the set there exists a path of neighbours in $\{N_0, N_2, N_4, N_6\}$ that are in the set.

A set of line segments and curve segments in a raster image, with all its branches and irregularities, will be called a line *structure*. A line structure may contain disconnected parts. The pixels in the set are calledforeground pixels, and all other pixels of the raster image are hackground *pixels.* The foreground and background pixels are the white and black pixels respectively in the bi-level raster image.

2.1. Contour following

The contour following algorithm used in our method, is based on the algorithm in Sobel [1978]. We will describe the algorithm for finding the contour of an arbitrary 8-connected object in the counter clockwise direction. The contour will be an 8-connected set of foreground pixels, each with one or more neighbours belonging to the background. The algorithm needs a contour pixel P_0 to start with. Next a neighbour pixel S_0 of P_0 is chosen from the pixels N_i in figure 1. Now the pixels P_1 and S_1, P_2 and S_2, etc. must be determined. The pixels P_0, P_1, ..., $P_n = P_0$ constitute the contour of the object. The pixels S_i (i = 0, 1, ..., n-1) are used as starting pixels for a search amongst the neighbours of Pi.

Given P_i and Si, the next contour pixel P_{i+1} and next starting pixel S_{i+1} are chosen as follows. Search the neighbours of P_i in the counter clockwise direction, starting with Si, until a transition from background *to* foreground is found. The first foreground pixel thus found will be P_{i+1}, and the last background pixel will be S_{i+1}. This process must be continued until pixel P_0 is reached again. Figure 2 illustrates the process.

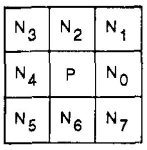


Figure 1. *The neighhours of a pixel P*

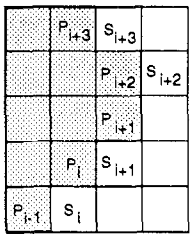


Figure 2. *Counter clockwise contourfollowing*

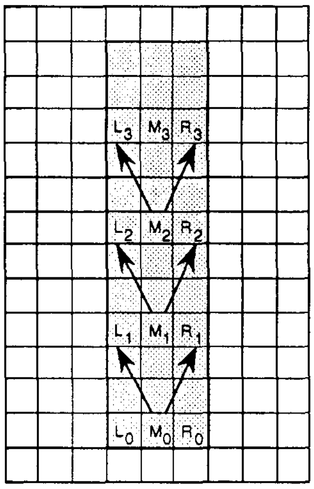


Figure 3. *Two contourfollowers*

2.2. Contour following at both sides of the line

Our line tracking method uses the contour following algorithm described above in a new way. With the left point L_0 from a pair of opposite points (L_0, R_0) on the contour of the line, contour following is started with a clockwise contour follower. After some distance has been covered. contour following at this side of the line is stopped, and the same process is started at the right side with a counter clockwise contour follower. After some distance has been covered at this side, this contour follower is stopped too. The last point found by the left contour follower and the last point found by the right contour follower are again considered as opposite points. Now the left contour follower is started again, and the whole process is repeated. In this way, pairs of opposite points (L_i, R_i) are found. The midpoint M_i of each pair will be a point in the output of the algorithm (a median point). This process is illustrated in figure 3.

The median points found with this method, should preferably be equally spaced. Furthermore, the line through two points that are considered as opposite must be approximately perpendicular to the local line direction. This means that a good criterion for one contour follower having to wait for the other is important. For this purpose we use a threshold distance. When the distance between the last found median point and the next point found by a contour follower exceeds the threshold distance, the contour follower is stopped. The threshold distance is determined by multiplying a line width estimate with a threshold factor. Good threshold factors have experimentally been determined to lie between 2 and 4. With a larger threshold factor the median line will be less accurate. whereas with a smaller threshold factor there will be a risk of both contour followers to stop at a width variation or a sharp turn (see figures 4a and 4b). In section 3, it will be shown what to do in such situations. As line width estimate, the distance between the left and right starting points L_0 and R_0 is taken (see figure 5).

2.3. Input to the algorithm

In order to be able to start the line tracking process properly, a line must be pointed out first by the user. One point on the line plus an indication in which direction the line has to be tracked is enough for this purpose. So, the input to our algorithm, besides the raster image, will be a starting point on the line, and a direction point that indicates which part of the line must be tracked (see figure 5).

Given the starting point and the direction in which to follow a line, two points L_0 and R_0 on the contour of the line, which will be used as the initial points for the left and right contour followers, must be determined. These points must be situated opposite to each other. The midpoint M_0 of these two points is taken as the first median pint.

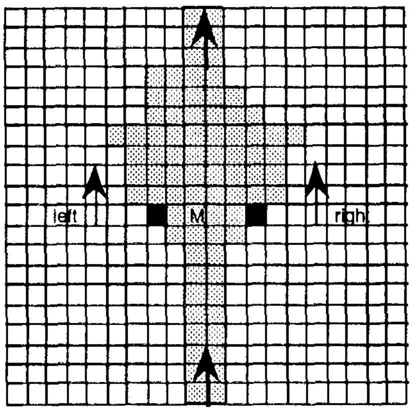


Figure 4a. *Width variation*

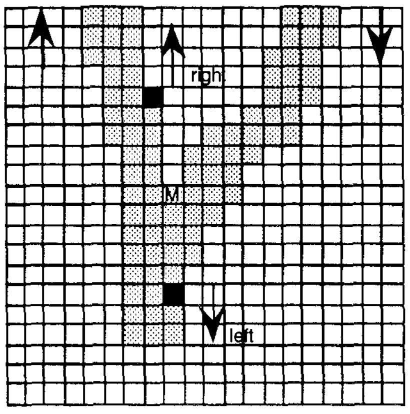


Figure 4b. *Sharp turn*

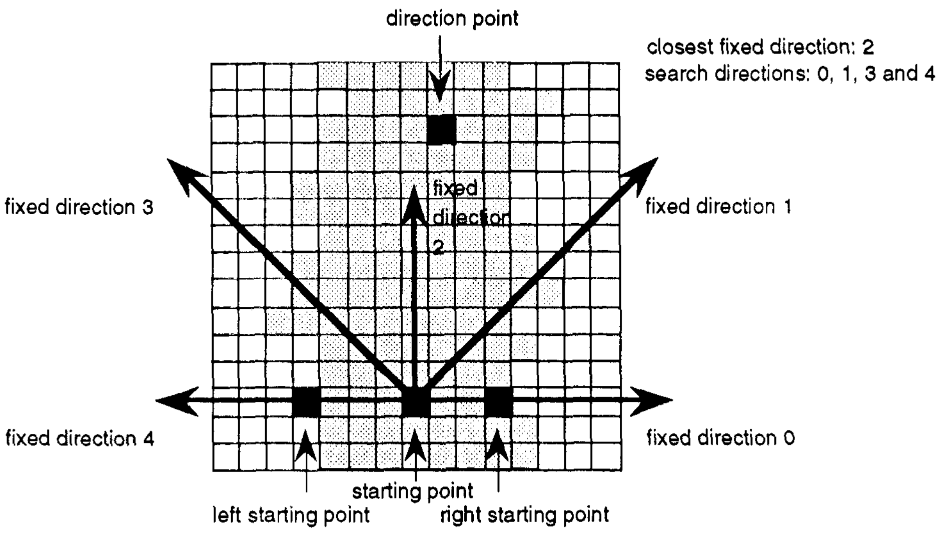


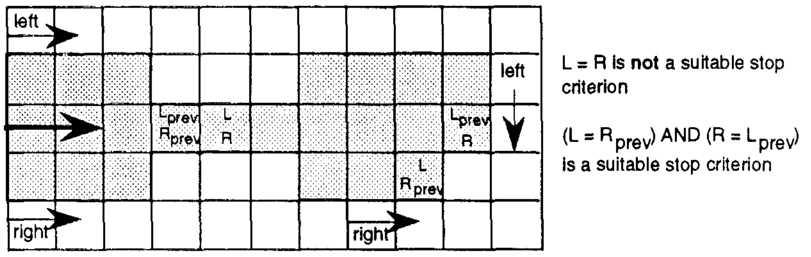
Figure 5. *Starting point, direction point, and left and right starting points*

The points L_0 and R_0 may be found in the following way. Eight fixed directions are distinguished, numbered 0 to *7;* five of these directions are shown in figure 5. The fixed direction *cfd* closest to the direction in which the line must be tracked, is determined. Then R_0 will be the closest contour point in one of the search directions *(cfd-2)* mod 8 and *(cfd-1)* mod 8, and L_0 is determined in the same way from the search directions (cfd+1) mod 8 and *(cfd+2)* mod 8.

As already indicated, the left contour follower will start with $P_0 = L_0$. For several-pixels-thick lines S_0 may be an arbitrary neighbour of P_0, but not for one-pixel-thick lines, because there contour pixels on the left and the right contour coincide, and the contour might be followed in the wrong direction. Instead, S_0 should be the first background pixel on the line from the starting point to L_0. The same applies to the right contour follower.

2.4. Stop criterion

The algorithm may terminate in several ways. First the algorithm must stop when the end of the line is reached. Let the last point found by the left contour follower be L, and the last point found by the right contour follower be R, then L = R may at first sight seem to be a good test for detection of the end of the line. However, with this test a problem occurs if a line has parts that are only one pixel thick. The left and right contour followers will find the same contour points on these parts of the line, whereas the end of the line has not yet been reached. This problem can be solved **by** storing the last two points found by each contour follower (L, L_{prev}, R and R_{prev}), and using the stop criterion: (L = R_{prev}) and (R = L_{prev}), which is illustrated in figure 6. This test always works when the end of a line is reached.



left
left
L_{prev}
R_{prev}
L
R
L_{prev}
R
L
R_{prev}
right
right
L = R is not a suitable stop criterion
(L = R_{prev}) AND (R = L_{prev}) is a suitable stop criterion

Figure 6. *Criterion to determine whether a line has been tracked to the end*

The second case in which the algorithm must stop, occurs if a line is closed. The contour followers will then return to the points they started. By storing the left and right starting points, and in each step comparing these with the points found by the contour followers, this can be detected.

There are some other situations in which the tracking process, as described until now, will stop. In these cases the line has not been tracked to an end, and line tracking must he continued. These cases will he handled in section 3.

A line tracked with the algorithm described above is shown in figure 7. A threshold factor of 2 was used.

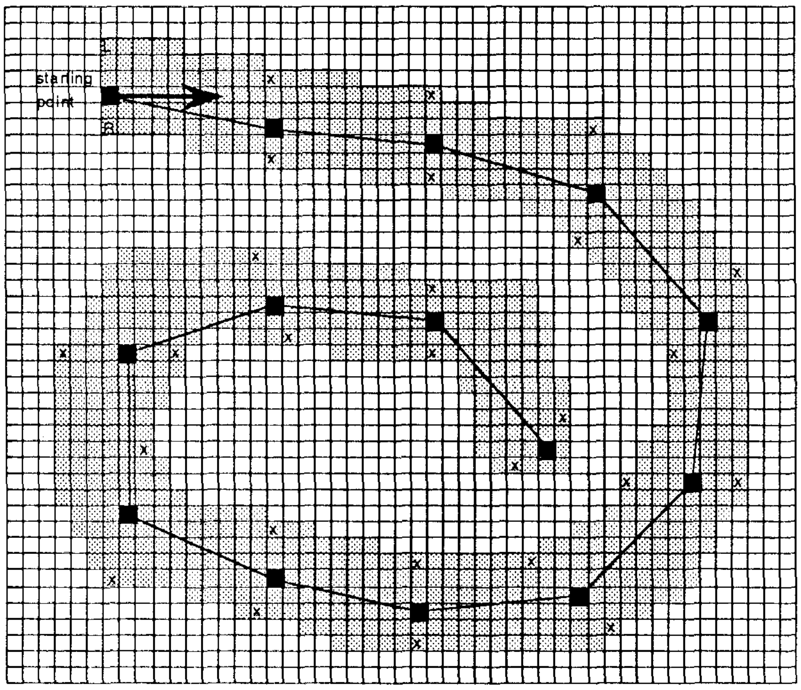


Figure 7. *A line tracked with the algorithm (with threshold factor 2)*

3. Handling irregular lines and branches

In many applications lines will not have smooth contours. Although small variations in line width can be handled by the method described in the preceding section, strong variations in line width will cause the algorithm to stop, as shown before in figure 4. This cannot be prevented altogether, because it could even happen that a line runs into a filled area, in which case it is in fact required that the algorithm stops.

There are, however, other important situations that will wrongly cause the algorithm to stop, namely line crossings and branches. This is shown in figure 8.

The idea behind our line tracking method is that only one line is tracked, so at line crossings and branches, one branch has to be chosen to continue. The line tracking algorithm must identify all branches and offer these to the user. This task is accomplished by a search of the environment of the point, where the algorithm stopped, with circles. After identification of all branches, the user must choose one of these, and the line tracking process is continued.

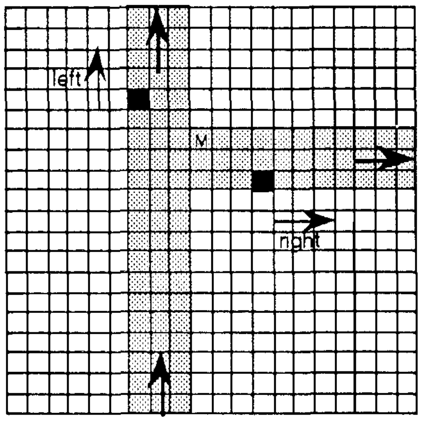


Figure 8. *Line structure with a branch*

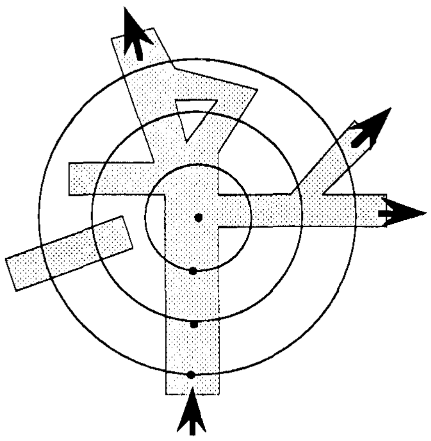


Figure 9. *Circles for circle following process*

3.1. Circle following

During line tracking, situations such as branching and line width expansion will occur occasionally. All these cases have in common that the contour followers are not able to continue. This can be detected very easily, because the same median point is found repeatedly. So, as soon as a median point equal to the previous median point has been found, a special situation must be handled.

When such a situation occurs, the environment of the last found median point will be examined. This is done by defining a series of circles with this median point as centre, and with radii determined by the distances to the previous median points (see figure 9). These circles are used for a process we call *circlefollowing.*

For each circle, this circle following is executed. First, circle points are derived with the Bresenham circle algorithm [Hearn and Baker 1986], which has been slightly adapted to obtain a 4-connected circle instead of an 8-connected one. This 4-connectivity is needed, because an 8-connected circle and an 8-connected thin line crossing that circle, do not necessarily have pixels in common. Next, starting with the median point on the circle, the circle is followed pixel after pixel. Every circle pixel that belongs to the contour of the line structure, except the first and the last, is saved. The saved contour points for several cases are shown in figure 10. The points are grouped in pairs of opposite points during the circle following process. At the first point of a pair, the circle enters the line structure, and at the next point (the opposite point), it exits again. If a circle enters the line structure and leaves it again at the next pixel, the same pixel must be saved both as the left point and the opposite point. All pairs of points are marked with a circle number.

Furthermore, the first and the last contour pixel on one of the circles (the intersection points of the second circle and the contour of the already tracked part ofthe line in figure 10) are saved separately for the purpose of branch construction, as will be shown in the next subsection. The first and the last contour points of all other circles are not saved, because these belong to the part of the line that *has* already been tracked.

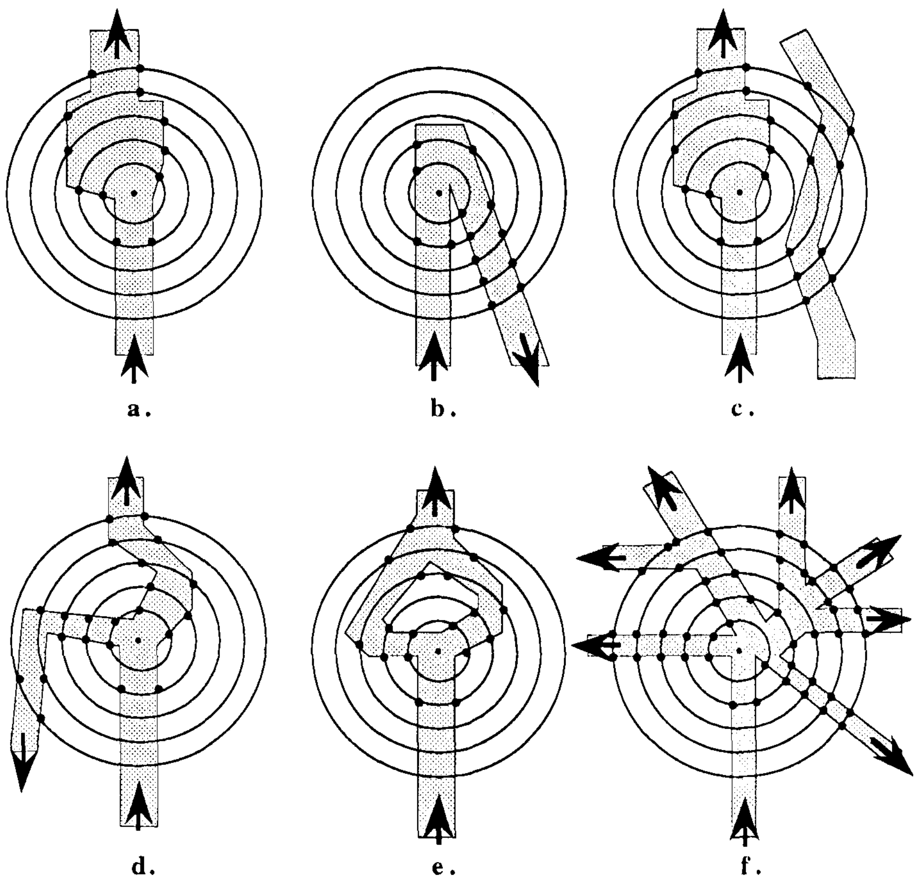


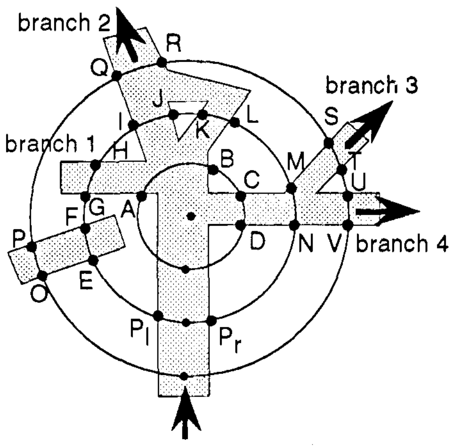
Figure 10. *Several situations that must he handled by circle. following*

3.2. Construction of branches

In the next step of our algorithm, the contour points found by the circle following method are used to determine branches. If only two contour points have been found on the largest circle, only one branch is found (figures 10a, 10b and 10e). Branch construction is here still necessary in order to find extra median points inside the largest circle. If, however, more than two points have been found on the largest circle, the line structure splits up in several lines in different directions (figures 10d and 10f), or a disconnected line structure lies close to the line that must be tracked (figure 10c). In these cases, branch construction will detect all branches and ask the user to choose one for continuation of the line tracking process. Median points on this branch are added to the output.

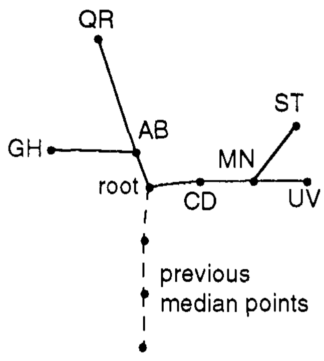
In the initialization step of branch construction, all points saved by circle following, are marked with branch number 0. Next, a clockwise contour follower is started from the separately stored first contour point on one of die circles (P_1 in figure 11). Every point found by this contour follower is checked against the saved points. If a contour point matches one of these points, that point is marked with branch number 1 (the initial value of a branch number counter). If a point found by the contour follower has been saved and belongs to the largest circle, the contour follower jumps over to the other point **of** the pair (the opposite point), and contour following is continued with this point as the new starting point. Whereas the contour follower first climbed into a branch, it will now climb down again along the other side of the branch. When, after finding one or more opposite points, a first point of a pair is found again, this indicates that climbing into the next branch has been started, and the branch number counter is incremented. This process is repeated for every branch, and each saved point encountered during this process is marked with the current branch number. The contour following process is finished when the separately stored last contour point on one of the circles (point P_r in figure 11) has been reached.

The process is illustrated in figure 11. The clockwise contour follower starts at point P_l and finds points A and G while climbing into the first branch. Then point H is found, while climbing down again along the other side of the branch. During this process, points A, G and H are all marked with branch number 1. Points I and Q are found during climbing the second branch. From point Q, which is on the largest circle, a jump *is* made to point R (the opposite point of Q), and points L and B are found while climbing down the second branch. Points I, Q, R, L and B are all marked with branch number 2. This process continues until point P_r has been reached. Points E, F, O, P, J and K are never reached, because these are contour points of another line structure and a small hole. Figure 11b shows the data structure for the saved points after point P_r has been reached.



first point	branch number (bnf)	opp. point	branch number (bno)	circle number
A	1	B	2	1
C	3	D	4	1
E	0	F	0	2
G	1	H	1	2
I	2	J	0	2
K	0	L	2	2
M	3	N	4	2
O	0	P	0	3
Q	2	R	2	3
S	3	T	3	3
U	4	V	4	3

candidate branches for continuation of the line tracking process: branches 2, 3 and 4



a. *Circles and contour points* **b.** *Data structure* **c.** *Median point tree*

Figure 11. *Construction of brunches*

The branch numbers of the marked points correspond to a tree of pairs of points. In the tree in figure 11c the median point of each pair of marked points is shown. A pair of points, and thus its median point, belongs to all the branches that could be followed by traveling from the root through that median point. Let *bnf* and *bno* be the branch numbers added to the first point and the opposite point of a pair. This pair of points then belongs to the branches numbered bnfup to and including *bno*, if $0 < bnf \leq bno$, because the first point was found while climbing branch *bnf*, and the opposite point was found while climbing down branch *bno*. If not $0 < bnf \leq bno$, the pair of points does not belong to a branch. In figure 11, pair (A, B) belongs to the branches 1 and 2, pairs (C, D) and (M, N) belong to branches 3 and 4, and pairs (E, F), (I, J), (K, L) and (O, P) do not belong to any branch, whereas all other pairs of points belong to one branch.

After all branches have been determined, the candidate branches for continuation of the line tracking process are presented to the user. Candidate branches are only those branches that are climbed up to the largest circle. All other branches are too short, and only represent slight irregularities in the line structure. In figure 11, only branches 2, 3 and 4 are presented to the user.

The median points of all pairs of saved points belonging to the selected branch are added to the output of the line tracking process. *So,* in fact one of the paths from the root to a leaf of the median point tree shown in figure 11c, is added to the output. The pair of points on the largest circle will be used as the new left and right starting pints. Furthermore, the distance between these two points is taken as the new line width estimate.

3.3. Handling small gaps in lines

In scanned images, the quality of the line structures is not always sufficient for the contour following algorithm. A line must be 8-connected, because otherwise it cannot be tracked to die end. Especially for lines that are only a few pixels thick, small gaps in the line may occur as a result of the scanning process. A simple method to skip those small gaps has been developed in order to make the algorithm more robust. When the line tracking algorithm stops, because the end of the line is supposed to be reached, this method checks whether this is really the case.

In order to examine the surrounding of the supposed end of the line, shields are built around the end. If (L_n, R_n) is the last pair of opposite points found by the contour followers, the left contour follower proceeds from L_n to R_n, and finds a series of points that we will call *shield* 0. *Shield I* consists of all 8-connected neighbours of *shield* 0 that do not belong to the line structure. All subsequent shields can be found by the rule: *shield* i consists of all 4-connected neighbours of the pixels of *shield i-1* that do not belong to *shield i-2* or *shield i-1.* With the outermost shield the surrounding is examined, just as with the circles in subsection 3.1. If overlap with the line structure occurs, line tracking is restarted. The maximum gap width that can be skipped in this way is determined by the number of shields used. Figure 12 shows the end of a line, and *shield 1* and *shield* 2.

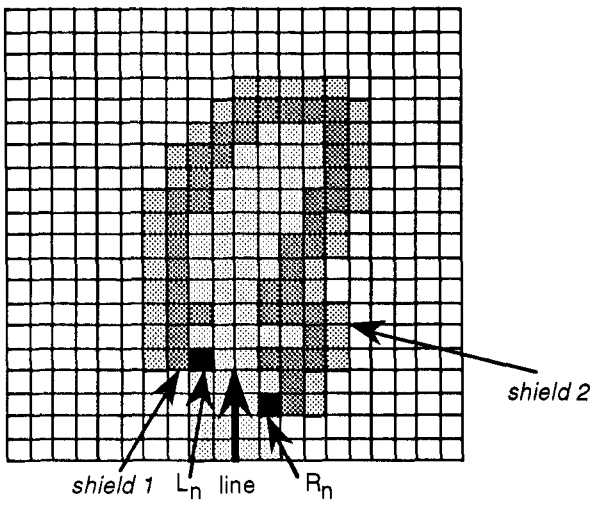


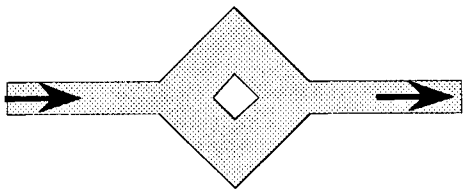
Figure 12. *Shields around the end of a line*

3.4. Output dependence on parameter settings

The output of the algorithm can be influenced by the following parameters:

- the number of circles taken at a point: *n*
- the threshold factor: *t*
- the maximum line width expansion factor: *e*
- the maximum gap width: g.

Parameter *e* is used in order to effect that line tracking is stopped at a point of extreme line width expansion, i.e. at a point where the new line width estimate exceeds *e* times the initial line width estimate. The influence of the parameter settings on the output of the algorithm is explained with the example line structure in figure 13. The table gives all possible interpretations of this line structure by the algorithm.

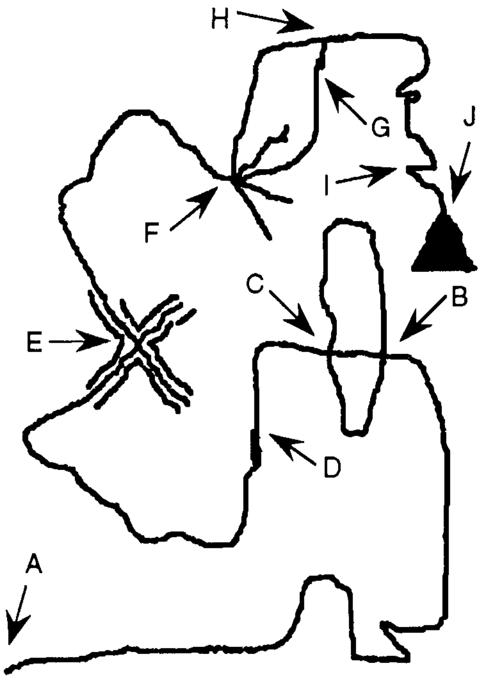


e	*t*	*n*	interpretation
small			line runs into an area
large	large		hole and line width expansion ignored
large	small	small	line branching
large	small	large	line width expansion

Figure 13. *Example line structure with different interpretations*

4. Results and conclusions

The algorithm presented in this paper has been implemented on a PC with an Intel 80386 microprocessor running on 33 MHz. It has been tested extensively with test images containing many special cases, including line width variations, sharp turns, disconnected parts of a line structure near the tracked line, small gaps, and all kinds of branching.



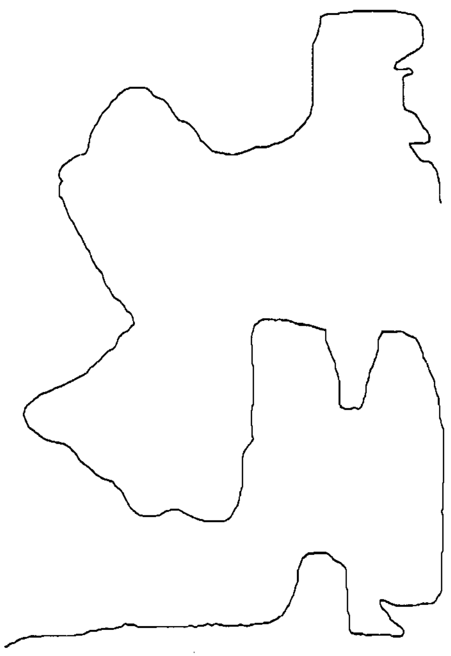


Figure 14a. *Test input raster image* **Figure 14b.** *Output*

Figure 14a is a 500 x 700 raster image of a line structure containing several of these special cases. It has been tracked with the parameter settings: $n = 4$, $t = 2$, $e = 3$. Line tracking started at point A. The situations at point B up to and including J were handled with branch construction. At points D, G and I, line tracking was continued without user interaction, because only one branch was found. At point J, line tracking was stopped. At all other points, candidate branches were presented to the user. Figure 14b shows the result of the tracking process. The tracking time was approximately 14 seconds. The output polyline consisted of 561 vectors.

With $t = 4$, the tracking time was approximately 13 seconds, 300 vectors were generated, and the output was hardly distinguishable from figure 14b. For all $t \geq 2$, tracking time is almost the same, which could be expected because most of the processing time is used by the contour followers and the distance covered by these is independent of t. If $t < 2$, contour following stops even at slight irregularities and circle following is needed much more times. This causes the processing time to increase considerably. If $t > 4$, quality of the polyline will be less, because in turns the line will tend too much to the inner side.

The algorithm generates the same number of median pints on straight parts of the line as in turns, although a large straight part could be represented by a single vector. Therefore, the number of points in the polyline representation could be reduced, without quality loss. Reduction methods are based on both the length of and the angle between vectors of the polyline.

The new algorithm has shown to be a robust and efficient method for vectorization of lines in a raster image, in particular when individual lines have to be tracked. It can handle lines with irregular borders, skips short dead end branches and small holes, and offers the user a choice between several branches when needed.

The algorithm will be incorporated in a geographical information system in the near future, but it could also be used in other applications that can benefit from vectorizing individual primitives in raster images.

Acknowledgements

We thank D.K. Steemann for his support of the graduation project of Olaf Kiewiet at DST, which has led to the idea and implementation of the algorithm presented in this paper, and F.W. Jansen for his comments on an earlier version of this paper.

References

Chin RT, Wan HK, Stover DL, and Iverson RD (1987) A one-pass thinning algorithm and its parallel implementation. *Computer Vision, Graphics, and Image Processing* 40(1): 30-40

Davis LS, Rosenfeld A, and Weszka JS (1975) Region extraction by averaging and thresholding. *IEEE Transactions on Systems, Man, and Cybernetics* 5(3): 383-388

Dori D, Liang Y, Dowell J, and Chai I (1993) Sparse-pixel recognition of primitives in engineering drawings. *Machine Vision and Applications* 6: 69-82

Duda RO, and Hart PE (1972) Use of the Hough transformation to detect lines and curves in pictures. *Communications of the ACM* 15(1): 11-16

Hearn D, and Baker MP (1986) *Computer Graphics.* Prentice-Hall, Englewood Cliffs

Illingworth J, and Kittler J (1988) A survey of the Hough transform. *Computer Vision. Graphics, and Image Processing* 44(1): 87-116

Kimme C, Ballard DH, and Sklansky J (1975) Finding circles by an array of accumulators. *Communications ofthe ACM* 18(2): 120-122

Landy MS, and Cohen Y (1985) Vectorgraph coding: efficient coding of line drawings. *Computer Vision, Graphics, and Image Processing* 30(3): 331-344

Lee KJ, Shirai Y, and Kunii TL (1990) Attribute-grammar based approach to vector extraction from a raster image. *Proceedings CG International '90 Computer Graphics around the World,* pp 225-239

Martinez-Perez MP, Jimenez J, and Navalon JL (1987) A thinning algorithm based on contours. *Computer Vision, Graphics, and Image Processing* 39(2): 186-201

Ohlander R, Price K, and Reddy DR (1978) Picture segmentation using a recursive region splilting method. *Computer Graphics and Image Processing* 8: 313-333

Parker JR (1988) Extracting vectors from raster images *Computers & Graphics* 12(1): 75-79

Pavlidis T, and Horowitz SL (1974) Segmentation of plane curves. *IEEE Transactions on Computers* C-23(8): 860-870

Pavlidis T (1982) An asynchronous thinning algorithm. *Computer Graphics and Image Processing* 20: 133-157

Princen J, Illingworth J, and Kittler J (1990) A hierarchical approach to line extraction based on the I-lough transform. *Computer Vision, Graphics, and Image Processing* 52(1): 57-77

Sobel I (1978) Neighborhood coding of binary images for fast contour following and general binary array processing. *Computer Graphics and Image Processing* 8: 127-135

Tamura H (1978) A comparison of line thinning algorithms from digital geometry viewpoint. *Proc. Fourth Intern. Joint Conf. on Pattern Recognition,* pp 715-719

Wu RY, and Tsai WH (1992) A new one-pass parallel thinning algorithm for binary images. *Pattern Recognition Letters* 13(10): 715-723