Pattern Analysis & Applications (1999)2:10–21

From Raster to Vectors: Extracting Visual Information from Line Drawings*

Liu Wenyin[1,2] and Dov Dori[3]

[1]*Microsoft Research, Beijing, PR China;* [2]*Department of Computer Science and Technology, Tsinghua University, Beijing, PR China;* [3]*Faculty of Industrial Engineering and Management, Technion – Israel Institute of Technology, Haifa, Israel*

Abstract: Vectorisation of raster line images is a relatively mature subject in the document analysis and recognition field, but it is far from being perfect as yet. We survey the methods and algorithms developed to-date for the vectorisation of document images, and classify them into six categories: Hough transform-based, thinning-based, contour-based, run-graph-based, mesh-pattern-based, and sparse-pixel-based. The purpose of the survey is to provide researchers with a comprehensive overview of this technique, to enable a judicious decision while selecting a vectorisation algorithm for a system under development or a newly developed vectorisation algorithm.

Keywords: Document analysis and recognition; Line drawings; Polygonalisation; Raster-to-vector; Thinning; Vectorisation

1. INTRODUCTION

Vectorisation, also known as raster to vector conversion, is a process that finds the vectors – straight line segments – from the raster images. Vectorisation is a widely used process in the area of Document Analysis and Recognition (DAR) as a preprocessing step for high-level object recognition, such as Optical Character Recognition (OCR) and graphic objects recognition. Basic vectorisation concerns grouping the pixels in the raster image into raw wires that are described by several attributes, such as characteristic points and line width. Advanced vectorisation includes line fitting and extending, which yields fine wires. We refer to crude vectorisation as the basic vectorisation process that takes a raster (binary) image as input and yields coarse wire fragments, which may be bars (non-zero width line segments) and polylines (chains of bars linked end-to-end).

Many crude vectorisation methods have been developed and implemented since image processing techniques were introduced more than 30 years ago. These methods can be roughly divided into six classes: Hough Transform (HT)-based methods; thinning-based methods; contour-based methods; run-graph-based methods; mesh pattern-based methods; and sparse-pixel-based methods. With the exception of HT-based methods, a typical vectorisation process consists of the following basic scheme:

1. Medial axis points sampling, or medial axis representation acquisition. This is the kernel processing for information reduction, after which only the important points that represent the medial axis are determined.
2. Line tracking, which follows (tracks) the medial axis points found in the first stage to yield a chain of points for each vector.
3. Line segment approximation or polygonalisation, which removes non-critical points from the point chains found in the second stage, and links the remaining critical points into bars and polylines.

The remaining critical points are finally used to represent the vectors extracted from the raster image. The main difference among the above-mentioned classes of vectorisation methods lies in the first two sub-processes. Several polygonalisation algorithms can be employed in the third subprocess.

In this paper, we survey the methods in each of the categories developed to-date for the crude vectorisation of document images. Crude vectorisation is a relatively mature subject in the DAR field, but it can still be improved significantly. The purpose of the survey is to provide

Received: 10 November 1998
Received in revised form: 7 January 1999
Accepted: 7 January 1999
*Presented at SSPR '98

researchers with a comprehensive overview of this technique as an aid in selecting the most appropriate vectorisation algorithm to suit the needs of the system they are developing, or help them develop a vectorisation algorithms that better suit their system's particular needs.

2. HOUGH TRANSFORM-BASED METHODS

Dori [1] discusses the application of the Hough Transform (HT) [2] in the vectorisation of straight line images by transforming spatially extended patterns in binary image data into spatially compact features in a parameter space. The transformation converts a difficult global detection problem in the image space into a more easily solved local peak detection problem in the parameter space. One way in which the HT can be used to detect straight lines is to parameterise it according to its slope and intercept. Straight lines are defined in Eq. (1):

$$y = mx + c \tag{1}$$

Every line in the (x,y) plane corresponds to a point in the (m,c) plane. Every point on the (x,y) plane can have an infinite number of possible lines that pass through it. The gradients and intercepts of these lines form a line on the (m,c) plane described by Eq. (2):

$$c = -xm + y \tag{2}$$

The (m,c) plane is divided into rectangular 'bins', which accumulate, for each black pixel in the (x,y) plane, all the pixels lying along the line in Eq. (2). When the line of Eq. (2) is drawn for each black pixel, the cells through which it passes are incremented.

After accounting for all the pixels in the image space, lines are detected as peaks in the transform space. Taking into account noise, each peak that is greater than a predefined threshold is used to form a line defined in Eq. (1). In practice, this assumed line might be a combination of several collinear line segments (bars). Hence, the pixels on the original image along the assumed line are followed so that the end points of these segments are found. The line width is also determined during the line tracking by examining the width at each pixel.

For straight-line detection, the HT visits each pixel of the image once. Therefore, its time complexity is linear with the total pixel number, which is the product of the image width and height. Since each side of the image is linear to the image resolution, we use the image resolution as the unit in analysing the time complexity of vectorisation algorithms. Hence, the time complexity of the HT-based vectorisation method is quadratic to the image resolution.

Since peaks are expected to be formed in the (m,c) plane for points whose m and c belong to broken or noisy lines in the original image, the HT can be used to detect lines in noisy images. However, since the gradients and intercepts are sampled sparsely, they may not be as precise as the original lines. Hence, the quality of lines detected is far less precise for slanted lines. This can be seen from Fig. 11(b), which is produced by an implementation of the HT-based vectorisation method by Dori [1]. Moreover, the HT-based methods can yield bars only, and cannot generate polylines.

3. THINNING BASED METHODS

Tamura [3], Smith [4] and Lam et al [5] provide comprehensive surveys of thinning algorithms. Thinning-based methods are commonly used, to the extent that most of the earlier vectorisation systems (e.g., Fahn et al [6], Kasturi et al [7] and Nagasamy and Langrana [8]), and some later revised methods (e.g. Hori and Tanigawa [9]), apply them as a first step. All of these methods employ a thinning procedure in the subprocess of medial axis points sampling to obtain the one-pixel wide skeletal pixels of the black pixel region before the line tracking subprocess takes place.

Thinning, which may also be referred to as skeletonisation, skeletonising, core-line detection, medial axis transformation or symmetric axis transformation in the literature, is a process that applies morphological operations [10] – on the input raster image and outputs one-pixel wide skeletons of black pixel areas. The skeleton of a black area is the set of pixels whose amount is the smallest, but whose topological structure is identical to the original image shape. Hence, it is much easier to operate and analyse than the original image. It is defined by Montanari [11] as the locus of the intersections of the wavefronts that propagate from inside the opposite edges of the area. Pfaltz and Rosenfeld [12] define that a skeleton is formed from the centres of maximal disks placed within the area. Davies and Plummer [13] add sufficient additional points to the skeleton defined by Pfaltz and Rosenfeld [12] such that it is connected. Based on the above definitions of skeleton, the thinning algorithms are of three groups: iterative boundary erosion (like the wavefront propagation) [14,15]; distance transform [16,17]; and adequate skeleton [13].

Iterative thinning methods employ the idea of iteratively shrinking the contour of (or removing the outside layer of pixels from) the line object, like a wavefront propagated from outside towards the inside of the object, until only (the pixels on) the skeleton (medial axis) remains. These methods, except for that of Montanari [11], which works on vector contours of the line objects, are surprisingly similar to each other, which work on pixels, as formally described by Naccache and Shinghal [14,15].

Following the skeleton definition, Montanari [11] considers the vector form outline (contour) of the object shape as a wavefront. The wavefront is then iteratively propagated towards the inside of the line region, with the superposition of waves not permitted. Points of the intersection of wavefronts are considered as the points of the skeleton. The pixel level outline is found by an edge detection operation, which is a very common and mature operation in computer vision [10,18]. The outline pixels are then tracked to a pixel chain, which is further vectorised by a polygonalisation procedure (which will be discussed in Section 8). Hence, the main computation is edge detection preprocessing, whose time

complexity is linear to the total number of pixels in the image, and therefore quadric to the image resolution. The polygonalisation preprocessing time is negligible compared to the edge detection. So is the wavefront propagation, which is linear to the line width (which is linear to the image resolution) and the boundary points (which is also linear to the image resolution, since the perimeter of a planar region is linear to the radius of the region). Therefore, the total time complexity is quadric.

The pixel level iterative thinning is like iterative erosion of the line object boundary. As basically proposed by Hilditch [19], the kernel procedure is moving a 3×3 window over the image and applying a set of rules to mark the centre of the window. On completion of each scan, all marked points are deleted. The scan is repeated until no more points can be removed. Coding the points in the 3×3 window is shown in Fig. 1. The marking rules are outlined by Naccache and Shinghal [14] as follows. P is marked for deletion if all the following rules are satisfied:

- P must have at least 1 white 4-connected [20] neighbours (e.g. P_{2i}, $i = 0..3$), i.e. P is an edge point.
- P must have at least 2 black 8-connected [20] neighbours (e.g. P_i, $i = 0..7$), i.e. P must not be an end.
- At least 1 of the black 8-connected neighbours of P must be unmarked.
- P must not be a break point (whose deletion disconnects two parts of a line).
- If P_2 is marked, setting P_2 white must not make P a break point.
- If P_4 is marked, setting P_4 white must not make P a break point.

The main problem of the pixel level iterative thinning algorithms is the time complexity, which is $O(wN)$, where w is the line width and N is the total number of pixels in the image. Since the line width is also linear to the image resolution, its time complexity is cubic to the image resolution. Moreover, they are prone to shape distortion at junctions like 'X' and 'T' [15], shown in Fig. 2. None of these algorithms has been proven to work for every case in terms of the accuracy of their results. Although the vector level iterative algorithm of Montanari [11] is faster, the shape distortion at junctions is a shortcoming, due to its iterative nature. Similar methods of fine tuning the iterative boundary erosion technique include justifying the marking rules [3] and varying the window size. Deutsch [21], for example, uses non-square windows, while O'Gorman [22] generalises the method to $k \times k$ sized windows. However, these modifications obtain only a small improvement in

P_3	P_2	P_1
P_4	P	P_0
P_5	P_6	P_7

Fig. 1. A pixel (P) and its 3×3 neighbourhood.

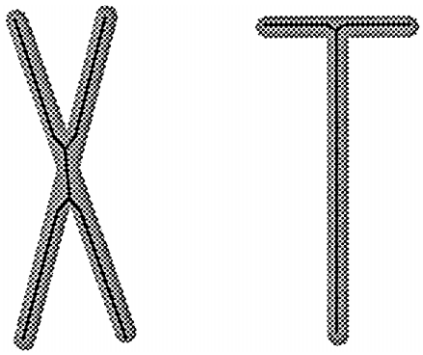

Fig. 2. Thinning distortions at junctions.

terms of speed and accuracy. For more details of iterative thinning algorithms, [3,4,5] provide thorough surveys.

Pfaltz and Rosenfeld [12] define the skeleton in a more formal way, on the basis of which the distance transform [16,23] is introduced to the thinning procedure. Rosenfeld and Pfaltz [16,23] define the distance transform of a binary image as replacing each pixel by a number indicating the minimum distance from that pixel to a white point. The distance between two points is defined by the number of pixels in the shortest 4-connected chain between the points. This transform is calculated by evaluating a function sequentially in a raster scan over the image, followed by a second function in a reverse scan.

Once the distance function has been calculated, a local maximum operation is used to find the skeleton. It has been shown to be the smallest set of points needed to reconstruct the image exactly. The distance transform and the skeleton are illustrated in Fig. 3. Haralick and Shapiro [10] also present a detailed discussion of the implementation of the distance transform operator, including recursive and non-recursive implementations.

The main problem with this algorithm, as shown in Fig. 3(c), is that the skeleton may not be connected, especially at junctions. However, the time required is of the order of the number of pixels in the image, i.e. the speed is only quadric to the image resolution. This is much faster than the iterative algorithms, which have a cubic time complexity.

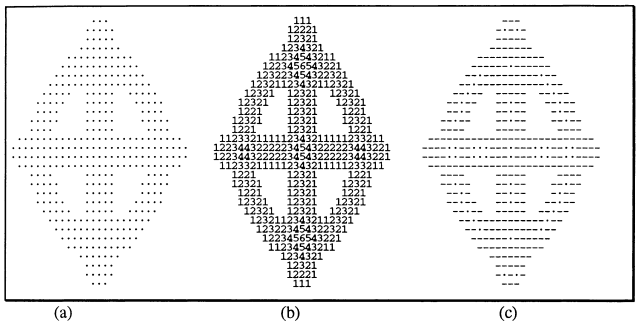

Fig. 3. Illustration of distance transform. (a) Image, (b) distance transform, (c) skeleton.

Similar to the distance transform [16, 23] for a binary image, Peleg and Rosenfeld [17] define a Min-Max Medial Axis Transform (MMMAT) for grey-scale images. The skeleton's connectivity is still not guaranteed, though the time complexity increases to the level of iterative methods. A different but faster algorithm for detecting lines and regions from grey level images was developed by Watson et al [24]. They use a Gausian filter to blur a grey-scale image before the lines are tracked along the peaks, which are supposed to be the medial axis points of the lines. Although it can distinguish regions from lines, the filter width is difficult to determine by making a trade-off between lines and regions. Therefore, it is difficult to use in the vectorisation of engineering drawings. Moreover, although it is linear to the number of pixels if a fixed filter width is used, it is also cubic to the image resolution if the filter width scales as the image resolution.

Combining the skeleton points obtained by Rosenfeld and Pfaltz [16] with those produced by a simple conventional iterative thinning algorithm, Davies and Plummer [13] define an 'adequate skeleton', and develop a composite thinning algorithm. The combination may result in skeletons with a (maximum) 2 pixel width. It is then thinned to a one pixel wide skeleton. The algorithm obtains more accurate skeletons than conventional iterative thinning algorithms, but more time is needed for the additional processing.

Generally, the objective of thinning is to reduce the data volume, such that only the topological shape of the image remains, which is size- and orientation-invariant. The result usually requires further processing. Most thinning algorithms are capable of maintaining connectivity. However, the main disadvantages are high time complexities, loss of shape information (such as line width), distortions at junctions, and false and spurious branches. Although they may be used in the vectorisation of line drawings, their main application is in the domain of OCR, in which the image size is usually small and the line width is not critical.

Performance evaluations of thinning algorithms have been carried out by Lee et al [25], Lam and Suen [26], Jaisimha et al [27] and Cordella and Marcelli [28]. Different algorithms may be suitable for different applications (e.g. OCR or line detection). For example, Lee et al [25] judge the algorithm of Tamura [3] as featuring good speed, fair connectivity and poor quality of skeleton; the algorithm of Naccache and Shinghal [15] is characterised as having good speed, good connectivity, and also a good quality of skeleton. However, they examine these thinning algorithms only from the viewpoint of OCR. Developers who wish to use these thinning-based vectorisation algorithms may refer to the above references for detailed comparisons of the algorithms.

The skeletons produced by the thinning procedure are still in bit-mapped form, and need to be vectorised for further processing. The one-pixel wide skeletal pixels are followed and linked to a chain by a line tracking subprocess. Polygonalisation can then be applied to convert the pixel chain to a polyline (or a single bar – a 'monoline'), which contains only the critical points. The polygonalisation methods are discussed in detail in Section 8.

4. CONTOUR-BASED METHODS

Aiming at lowering down the computational burden of thinning, another group of vectorisation algorithms tries to reduce the data volume before sampling the medial axis points. The main idea is finding the shape – the edges (or contour) of the line object first, and then calculating the middle points of the pair of points on two opposite parallel edges. This group of vectorisation algorithms sample and track (follow) the medial axis points simultaneously. This is different from thinning-based algorithms, which do line tracking after all the medial axis (skeleton) points have been sampled. The main computationally intensive operation in these methods is edge detection and polygonalisation. The time complexity of edge detection is linear to the pixel volume (i.e. quadric to the image resolution). Polygonalisation is only linear to the pixels on the contour, as discussed in Section 8. Hence, this group of vectorisation algorithms has a quadric complexity, and is much faster than thinning-based algorithms. Moreover, the line width is also much easier to obtain, which is very important for higher level drawing interpretation. Edge detection is a common and mature operation in computer vision. Some edge detectors can be found in the books of Haralick and Shapiro [10] and Nalwa [18].

Assuming that the edges are found and polygonalised, the medial axis points of two approximately parallel edges are defined by Jimenez and Navalon [29] to be the midpoints of the perpendiculars projected from one side to the other, as shown in Figs 4(a) and 4(b). Starting with a 'well behaved' edge (which is a vector), a linear search finds the nearest edge on the other side of the object, and several perpendiculars are projected from one to the other. Subsequent edges are easier to find simply by following the edge until a junction point is found.

The problem with all the algorithms of this type is how to deal with junctions. There are two main problems associated with junctions. The first is shown in Fig. 4(c), in which an intersection at a small angle forms a merging junction, which is most likely to be missed during tracking. In the second case, shown in Fig. 4(d), a cross intersection is encountered. Joining up the lines meeting here is problematic. It is important that the algorithm be robust and capable of dealing with all line shapes without misjudging a junction and producing an incorrect skeleton. Hence, it is inappropriate for use in the vectorisation of curved and multi-crossing lines.

Based on the same idea, Shapiro et al [30] start with a non-polygonalised, pixel-form edge. The midpoint of two pixels on opposite edges of the object is taken as being a point on the skeleton. The edges are followed such that the width is minimised, thus preventing scanning of the inside of a bend from getting out of synchronisation with the outside. The problem at junctions remains unsolved, since the object is assumed to have a simple backbone with branches, but these branches are not permitted to have sub-branches. This makes the algorithm unsuitable for most applications, but it does provide some useful ideas for a more general solution.

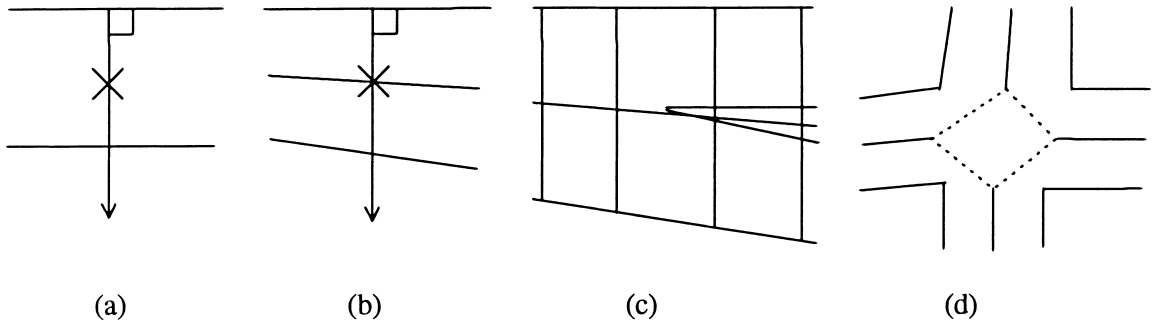

Fig. 4. Illustration of parallel edge pair based vectorisation. (a) Midpoint of parallel edges, (b) midpoint of approximate parallel edges, (c) missed junction, (d) confused junction.

5. RUN GRAPH-BASED METHODS

Extending the ideas of Di Zenzo and Morelli [31] and Boatto et al [32], Monagan and Roosli [33] more formally define the run graph as a semi-vector representation of a raster image before line detection and other segmentation. It is sufficient for structural representation of the line-like images, and efficient for line extraction, information preserving and easier to operate.

Several cases of run graph representation for raster images are illustrated in Fig. 5. A run has a direction, which can be either horizontal or vertical. It is a maximal sequence of black pixels in its direction. Hence, a *run* can be defined using the quadruplet

$$R = \{d, c_{d'}, b_d, e_d\}, b_d \leq e_d$$

where d is referred to as the *direction* of the run, which can be either 0 for horizontal or 1 for vertical, $c_{d'}$ is referred to as the *orthogonal coordinate*, which is the coordinate of the run in the direction orthogonal to the run direction, i.e. $c_{d'}$ is the row number if d is horizontal, or the column number if d is vertical, b_d is referred to as the *begin (directional) coordinate*, which is the coordinate of the first pixel of the run in the run direction, and e_d is referred to as the *end* (*directional*) *coordinate*, which is the coordinate of the last pixel of the run in the run direction. A vertical run R1 and a horizontal run R2 are illustrated in Fig. 5. A run can also be expressed by two endpoints, in which case, the direction is inferred from the two points' coordinates. If their x coordinates are equal, the direction is vertical, and if their y coordinates are equal, the direction is horizontal.

A *subrun* is a part of a run. Runs A and B are *adjacent* if they have the same direction, the difference of their orthogonal coordinates is 1, and the difference of their maximal begin coordinates and minimal end coordinates is less than or equal to 1. If A and B are adjacent and A's orthogonal coordinate is smaller than that of B, A is called *predecessor* of B, and B is called *successor* of A. A run is *regular* if it has only one predecessor and only one successor, otherwise, it is *singular*. Two runs are *conjugate* if they are orthogonal and overlap one and only one pixel, i.e. they cross. A vertical run is a *short* run if it is regular and not longer than all its conjugates. A horizontal run is a *short* run if it is regular and shorter than all its conjugates.

An *edge area* consists of a maximal sequence of adjacent short runs in the same direction. An *extreme area* consists of only one run that has no adjacent run on one side. A *junction area* consists of a maximal sequence of adjacent vertical runs/subruns of pixels belonging neither to vertical nor to horizontal short runs. Monagan and Roosli [33] introduce the concept of a *touching point*, which does not actually cover any pixel point in the image. A *touching point* is formed between two adjacent runs if they do not overlap in their direction, or between two orthogonal runs if they do not overlap but touch each other end-to-end, as shown in Fig. 5. However, this definition is contradictory to the definition of edge area, since two touching and adjacent runs are parts of an edge area. Removing this definition, a run graph RG can be formally defined as follows:

$$RG = \langle V, E \rangle$$

where V is a set of nodes (vertices), which are either junction areas or extreme areas, and E is a set of edges, which are edge areas that link nodes.

According to Boatto et al [32], the procedure of constructing a run graph of an image is as follows. The first step is to build both horizontal and vertical simple run graphs, which consist of only horizontal runs and vertical

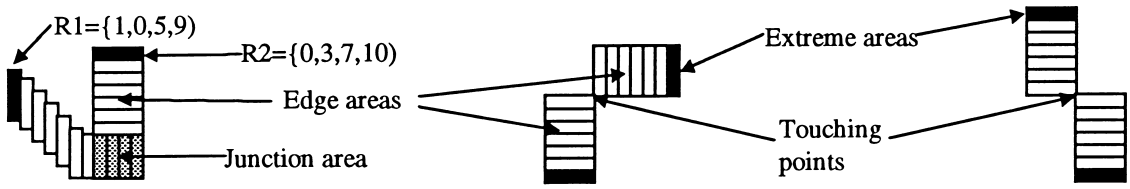

Fig. 5. Illustrations of run graph representation of a raster image.

runs, respectively. Secondly, edges are built as sets of adjacent regular short runs. The remaining pieces of the image, encoded as lists of vertical runs and subruns, are the node areas.

The line extraction procedure takes as input such a run graph. The shape of each node is then refined by a heuristic procedure (run splitting) that attempts to minimise the area of the node and maximise the lengths of the connected edges. The midpoints of the short runs in the edge areas are taken as the skeleton points, which further undergo polygonalisation to produce the final polylines (or bars).

While the line extraction is made efficient due to working on the semi-vector run graph representation, the preprocessing (i.e. the run graph construction) also takes time for visiting each black pixel on the image at least once. This implies that the complexity of the run graph-based vectorisation is quadric to the image resolution. Disadvantages of the method also include inaccurate intersection points due to coarse locations of junction areas, false (undesirable) junction areas resulting from run direction changes, or at noise points on uneven edges. Hence, this method, too, is not appropriate for curved line vectorisation.

6. MESH PATTERN-BASED METHODS

Mesh patterns were first introduced by Lin et al [34] to detect characteristic patterns in diagrams, e.g. logic connection diagrams. A connection diagram understanding system is then developed based on this method. The basic idea is to divide the entire image using a certain mesh, and to detect characteristic patterns by only checking the distribution of the black pixels on the border of each unit of the mesh. A control map for the image is then prepared using these patterns. Finally, the extraction of long straight-line segments is performed by analysing the control map. Figure 6 shows the principle of the mesh pattern-based line detection method. In Fig. 6(a), the image is divided into square meshes, which are defined by an equal proper mesh size, *n*. Each unit mesh is analysed according to the pixels on the one-pixel wide border only. It is then labelled according to its characteristic pattern, identified in comparison to a known one in the pattern database. The image is then represented by a control map, in which each unit mesh in the original image is replaced with its characteristic pattern label.

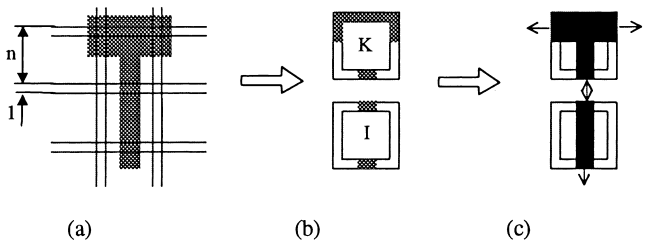

Fig. 6. Illustration of mesh pattern-based line extraction method. (a) Image and meshes, (b) mesh pattern labels and the control map of the two central meshes of the image, (c) lines extracted by analysing the control map.

In Fig. 6(b), the central part of the image in Fig. 6(a) is represented by a control map consisting of two meshes, whose labels are 'K' and 'I', respectively. The lines are recovered and tracked from mesh-to-mesh in the control map by analysing the characteristic patterns of the meshes, as shown in Fig. 6(c). In their characteristic pattern database, Lin et al [34] defined and labelled 51 known characteristic patterns. The other unknown complicated patterns are labelled with question marks. Such areas are further processed by a more complex detailed procedure during the control map analysis. This procedure scans every pixel in these areas, and every black pixel is labelled as being a line pixel or a feature point.

Vaxivere and Tombre [35,36] extended the mesh pattern-based method for mechanical engineering drawings analysis and recognition. They use dynamic meshes. Complicated (question-mark-labelled) meshes are further split into several smaller but known mesh patterns, whose shape may be non-square. Their vectorisation result is a data structure describing the image as a collection of segments of different types (e.g. thick line, thin line and contour of black blob) and junctions between these segments. The method may obtain good performance when applied to sparse, straight and long lines. The mesh pattern-based line detection methods are fast due to their sparse pixel access. Since only the pixels on the mesh border are visited, their time complexity is linear to one side of the image, i.e. linear to the image resolution. The access ratio, which is the ratio of the number of accessed pixels to the total number pixels in the image, is about $2/n$, if all meshes are labelled with known labels. However, if no extra memory is used for the borders of the neighbour meshes, each pixel accessed may be accessed twice, since it is on the border of two neighbouring meshes. As claimed by Lin et al [34], the access ratio for $n = 16$ pixels is about 2/15 to 3/15. The access time increases as the number of question-mark-labelled meshes increases. Lin et al [34] also claim that an optimum mesh size is just a little larger than the maximum line width in the image, in which case, the processing time is about 55% of the processing time of the complete application of the detailed processing procedure to the entire image.

However, the mesh size is hard to control. A big mesh size introduces more question-mark-labelled meshes, which requires much more processing time. A small mesh size increases the access ratio, and may also make the line extraction difficult. Moreover, it is not suitable for the detection of more complex line patterns, such as arcs and discontinuous (e.g. dashed or dash-dotted) lines. As claimed by Lin et al [34], the proper mesh size, *n*, should be larger than the maximum width of the line segments, but smaller than the minimum interspace between two line segments on the image. In this case, in the areas where only long, straight line segments exist, each side of a unit mesh border mostly intersects one line segment of the image. Moreover, *n* should be smaller than the smallest (shortest) line segment of the image, so that the background area judgment can be

simply performed on the control map by detecting the characteristic patterns labelled with blank labels. This excludes the dot segments, whose lengths can be as small as their widths, or even shorter. Dot segments may be missed during line tracking. The nature of investigating only the borders of meshes may link lines with gaps that are shorter than the mesh size. This may be a big advantage in some cases, but a big disadvantage in others, since it is desirable to fill the gaps of a broken line due to noise, while gaps of dashed lines should not be eliminated. In noisy drawings, this may turn out to be a non-trivial challenge.

7. SPARSE PIXEL-BASED METHODS

Orthogonal Zig-Zag (OZZ) is the first in a new family of vectorisation algorithms, developed by Dori et al [1,37,38]. Like mesh sampling [34–36], OZZ samples the image sparsely. The basic idea of OZZ is to track the course of a one-pixel wide 'beam of light', which turns orthogonally each time it hits the edge of the area covered by the black pixels, such as a bar area. The midpoint of each run, which is the intersection of the light beam and the area within the area, is recorded, as shown in Fig. 7. If a run is longer than a predefined threshold, e.g. 30 pixels [37], the run stops there, an orthogonal run is made and its midpoint is recorded. This may happen when tracking along a nearly horizontal or vertical area. A horizontal case is shown in Fig. 7.

The details of the OZZ vectorisation algorithm are as follows. A horizontal screen line, which moves down n (e.g. $n = 10$) pixels each time, goes from left to right of the image. When it encounters a black pixel, it enters in the area and an OZZ procedure begins. The screening continues until a white pixel is encountered (i.e. the light beam hits the edge of the area) or the travelling length within the black area (which is referred to as a run) exceeds a predefined threshold. If the edge of the area is hit, the midpoint of the run is recorded, the screening trajectory turns orthogonally within the black area, and the OZZ procedure continues, as shown in the slant case of Fig. 7. If the run exceeds a predefined threshold, the screening stops, and two new beams of light are emitted orthogonally from the stopping point, one to left and the other to right. When they hit the edges of the black area, a new run is defined from one edge of the black area to the other as the union of the two collinear screening runs. The midpoint of this joint run is recorded, and serves as the source of a new light beam that is orthogonal to the joint run. The OZZ procedure continues from this newly emitted light, as shown in the horizontal case of Fig. 7. After the OZZ procedure hits the edge of the line, i.e. an end of the black area on one side, another OZZ procedure starts. It is performed by emitting a beam of light orthogonal to the first screen line from the point where that line hit the black area coming from outside for the first time, if a slant area is visited, as shown in the slant case of Fig. 7. These midpoints are followed during the light tracking, and used as the medial axis points of the bar image. The midpoints are further processed by polygonalisation, so that the output of OZZ is a set of bars obtained by breaking the point chain where necessary. Each detected black area is 'coloured' to avoid spending time for detecting the same area more than once.

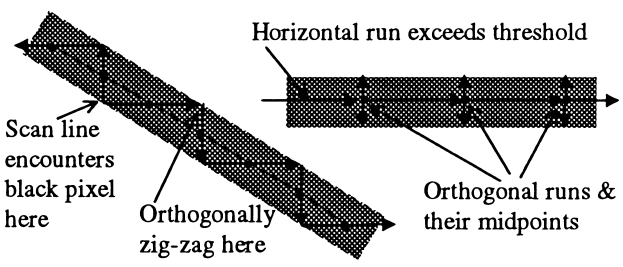

Fig. 7. The principle of the Orthogonal Zig-Zag (OZZ) vectorisation algorithm.

After the horizontal pass is finished, a vertical pass is similarly performed. In the vertical pass, the vertical screen line goes from top down every n pixels. After the vertical pass is over, the bars found in the two passes of combined, and overlapped bars are omitted.

OZZ is time-efficient due to the sparse sampling of the image. As shown in Fig. 7, the number of pixels visited by OZZ is linear to the sum of the image's width and height. Hence, it is linear to the image resolution. However, it is also noise-sensitive. Moreover, it is designed to yield bars only. Hence, curve images are vectorised to a set of bars that may either overlap each other at their ends or generate false gaps. Therefore, it does not perform well in vectorising arc images, as shown in Fig. 11(c).

Elaborating on the OZZ idea, Liu and Dori [39] developed the Sparse Pixel Vectorisation (SPV) algorithm. SPV improves the OZZ method in the following ways: (1) The general tracking procedure starts from a reliable starting medial axis point found by a special procedure for each black area; (2) a general tracking procedure is used to handle all three cases of OZZ, i.e. horizontal, vertical and slant. Therefore, only one pass of screening is needed, and the combination of the two passes is avoided, making SPV faster than OZZ; and (3) a junction recovery procedure is applied wherever a junction is encountered during line tracking. The three procedures are presented in detail below.

To start the SPV algorithm, a reliable starting medial axis point is found first, which is not affected by noise and inaccuracies at the wire's end. The first black pixel (P_0 in Fig. 8) is found at the point where a screen line encounters a black area. Going from P_0 to the right until a stop pixel, we get the first directed run. Going back from P_0 to the left, we get the zero length run. From these two horizontal black runs we get the horizontal middle run point P_1. Starting from P_1, we get the vertical middle run point P_2 by finding two opposite vertical black runs. In a similar manner, we then get the horizontal middle run point P_3 from P_2. We repeat the process until the distance between P_i and P_{i-1} is less than some predefined error, which is usually 1 and at most 2 pixels. P_i is defined as the starting medial axis point. In practice, only a few iterations are

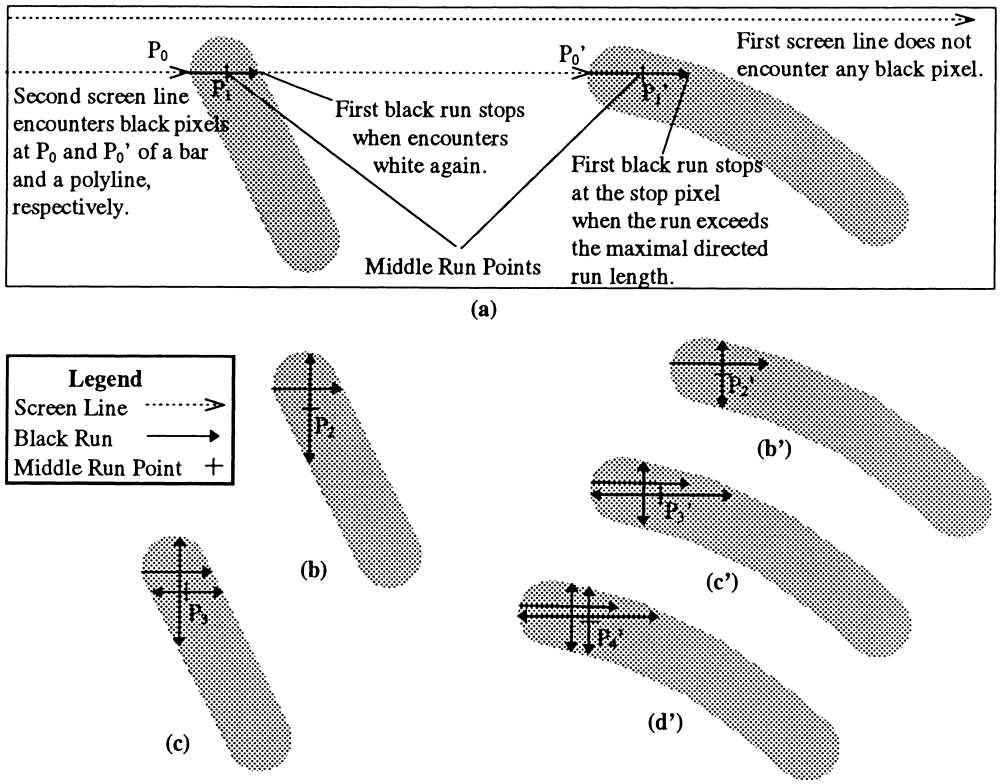

Fig. 8. Illustration of the procedure of finding the start medial axis point and the tracking direction.

needed, so P_3' can be used as the first medial axis point for a vertical bar, and P_4' as the first medial axis point for a horizontal bar, as shown in Fig. 8.

The lengths of both the horizontal and vertical runs are known at the first medial axis point P_i. If the horizontal run is longer than the vertical run, the bar's inclination is more horizontal than it is vertical (i.e. its slant is less than 45°), in which case, the length direction is set as horizontal, otherwise the length direction is defined to be vertical. The width direction is defined to be orthogonal to the length direction. If the length direction is horizontal, the tracking is done first to the right, then to the left, and if the length direction is vertical, the tracking is done first downwards, then upwards.

As Fig. 9(a) shows, going from the first medial axis point P_3 in the tracking direction, which is vertical, the main Sparse Pixel Tracking procedure for this black area starts. To start the tracking cycle, a tracking step is taken from

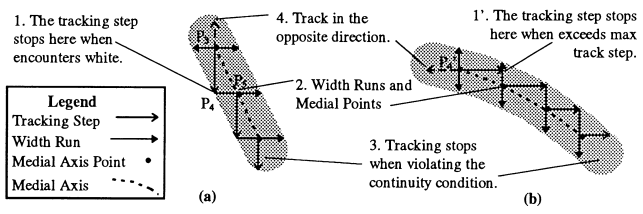

Fig. 9. Illustrations of the general procedure of the sparse pixel tracking. (a) A vertical tracking case, (b) a horizontal tracking case.

the last medial axis point, which reaches point P_4. From P_4 two opposite directed width runs are made. From these runs, the (undirected) width run and its middle run point, P_5, are obtained. P_5 serves as the new medial axis point. This concludes one tracking cycle. These tracking cycles repeat while the medial axis points and the width run lengths are recorded and monitored, as long as all of the following four continuation conditions are satisfied:

1. *Width preservation*. The largest difference among line widths, represented by the width runs found during tracking, along a neighbourhood of some small number of adjacent medial axis points, is below some threshold, e.g. 50% of the neighbourhood average width.
2. *Sole occupancy*. The medial axis point should not be in an area occupied by another vector that has already been detected.
3. *Direction consistency*. The pixel visiting direction is the same as the pixel visiting direction of the previous tracking cycle.
4. *Positive tracking step length*. The length of the tracking step is greater than zero.

The first three conditions are usually violated at a junction, a black area neighbourhood of a cross, a branch or a corner. A free (isolated) end of a line may also violate the first or the fourth condition. Curves may cause violations of the third condition, i.e. the direction consistency.

When one or more of the first three continuation con-

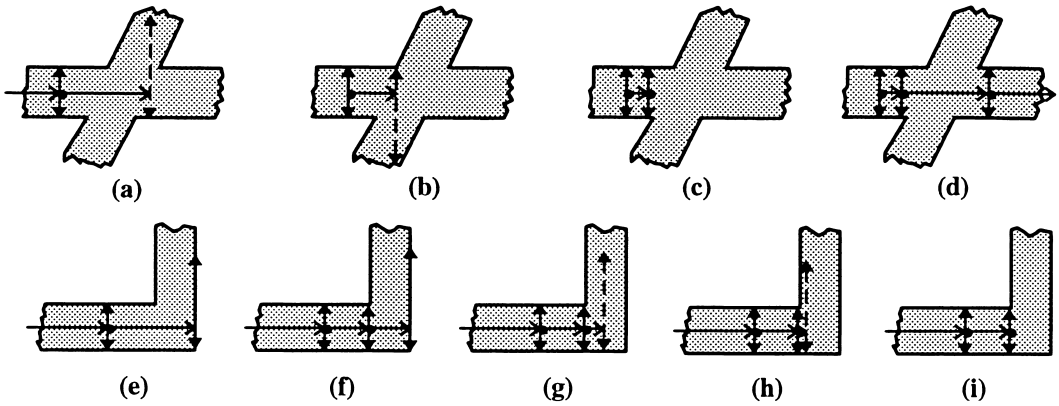

Fig. 10. Demonstration of the junction recovery procedure. (a)–(d) Cross case, (e)–(i) corner case.

ditions is violated, the Sparse Pixel Tracking procedure pauses, and a Junction Recovery Process starts using the Junction Recovery Procedure, as exemplified by a cross and a corner in Fig. 10. Junction recovery is an iterative procedure consisting of three steps: (1) retreating to the last medial axis point; (2) adjusting the tracking step length by half-sizing the length of the current tracking step; and (3) testing the conditions at the new position. If the test fails, i.e. the new medial axis point also violates one of the above conditions, a new iteration is reapplied with the tracking step half-sized again. The iterations halt when the tracking step becomes zero, as in the fourth condition. If all the conditions are met again at the new medial axis point, then we continue the general Sparse Pixel Tracking procedure, using the normal tracking step from the new medial axis point. By so doing, the procedure may overcome some uneven area, where the width runs are significantly different, as demonstrated in Figs 10(a–d). If the tracking step length becomes zero, the tracking stops at the last medial axis point, as in the case of a corner, shown in Figs 10(e–i).

Figure 11(d) has shown the SPV improvement to the OZZ algorithm and the HT-based method. SPV is used in the Machine Drawing Understanding System [40], which provides a good preprocessing basis for graphics recognition. This is verified by the automatic evaluation as high speed and good quality using the protocol proposed by Liu and Dori [41]. SPV has also been successfully used for line detection by Yoo et al [42].

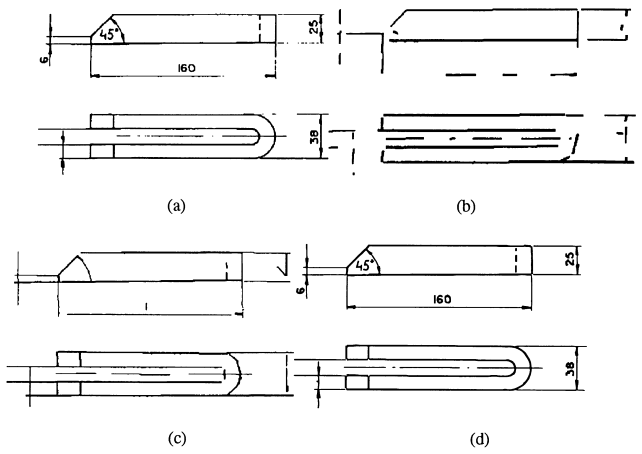

Fig. 11. Comparison of HT, OZZ and SPV results. (a) Original image, (b) HT-based vectorisation, (c) OZZ vectorisation, (d) SPV.

8. POLYGONALISATION ALGORITHMS

The result of the line tracking procedure is a chain of points (or polyline) that are on the approximate medial axis of the black area. Although it can also be regarded as a vector representation of this black area, some intermediate points are redundant because they are (approximately) on the straight line segments formed by their neighbours. To obtain the most concise vector representation of this black area, these points should be removed from the chain list. This should result in a polyline with the fewest edges (or vertices) to approximate the original one, while maintaining the original shape. A polygonal approximation procedure, or polygonalisation, does this. Most vectorisation methods apply polygonalisation to the tracked coarse polyline. To preserve the original shape, a value denoted by ϵ is predefined as a parameter to decide which points should be removed, and therefore to constrain the precision of the resulting polyline. As the ϵ value gets smaller, there are more vertices, but the edge accuracy is improved. ϵ can be set to be 1–2 pixels to preserve the original shape as much possible. In polygonal approximation of lines in engineering drawings, however, ϵ can be set up to half of the line width.

The criticality of a point in a point chain is determined by its distance to the line formed from its two neighbouring critical points on both sides. Polygonalisation leaves as few critical points in the point chain as possible. The removal criterion is that the distance between each non-critical point P and the line joining the two neighbouring critical points on both sides of P is less than ϵ. However, the best set of critical points is usually not known in advance. The determination of critical points is the main procedure of polygonalisation.

The various polygonalisation algorithms can be classified into two groups. The first class of methods includes those

of Montanari [43] and Hung and Kasvand [44], who use a global method to examine the criticality of points. The second class of methods, which includes those of Sklansky and Gonzalez [45] and Jimenez and Navalon [29], applies local optimisation. Here, the polygonalisation is done by going ahead from a new critical point (one endpoint of the chain is a critical point for initialisation) each time, and finding as many intermediate non-critical points as possible before the next critical point is found. Montanari [43] uses an iterative method in the polygonalisation procedure. On each iteration the critical points are moved so that as many points as possible are collinear, subject to the constraint that no point may move outside a small circle around its original position, whose radius is ϵ. Intermediate collinear points are then removed. Convergence is claimed within 10 iterations for most cases. The polyline generated has the smallest perimeter, but the iterative nature of the method makes it slow. Moreover, the remaining critical points are not at their original position. This may not be allowed in some applications. For instance, exact vertex locations are essential for arc approximation.

Hung and Kasvand [44] start with a perfectly 8-connected thin binary line, and calculate some defined functions for each pixel on the line. A complex set of rules is then applied to these pixels to determine the criticality of each pixel. Due to the global nature, this method produces a good approximation to the original line. The computation time is linear to the number of original points on the line. However, the requirement of 8-connectivity avoids its application to other forms of tracked point chains, such as the sparse point chains resulting from OZZ and SPV.

Sklansky and Gonalez [45] developed a polygonalisation algorithm using the idea of gradually shrinking a wedge to find the next local critical point. The principle is illustrated in Fig. 12. In Sklansky and Gonzalez's method, a wedge formed by two tangents from the start critical point (such as P_0 in Fig. 12) to circles with radius of ϵ around each new point is used to limit the possible positions of the next critical point. The points outside this wedge are out of the domain constrained by ϵ. The wedge is modified with each new point, so that it forms the intersection of the wedges from all points so far. When the wedge shrinks to a line, the last point to fall inside the wedge will be marked as critical and taken as the new initial point. Figure 12 illustrates this method. Point P_1 yields the wedge W_1, P_2 yields W_2 and P_3 yields W_3. Note that W_3 shares a line with W_2, since the tangents to P_3 do not fall entirely within the wedge W_2. So does W_4 for the same reason, but unlike P_3, which falls inside W_2, P_4 does not fall inside W_3. So when the wedge for P_5 is outside W_4 and the scanning stops, P_3 is marked as critical because it is the last one which fell within the wedge.

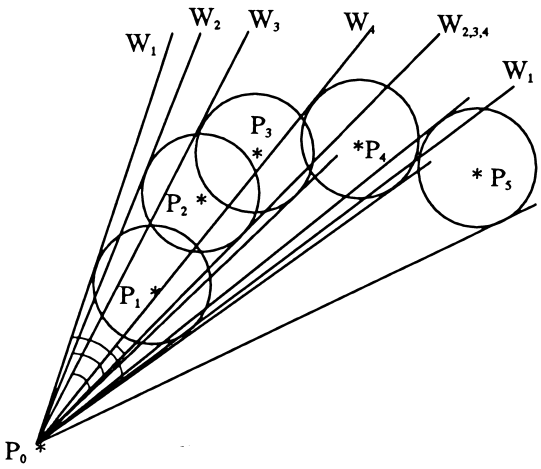

Fig. 12. Sklansky and Gonzalez's method for polygonal approximation.

Jimenez and Navalon [29] also use a local optimisation similar to Sklansky and Gonzalez [45]. However, their algorithm relies on the following assumption. Given a start critical point P_i and the next candidate critical point P_j for $i < j$, the function $D(i,j)$ is defined as

> $D(i,j)$ = maximum of the distances of point P_k (for $i < k < j$) to the line segment joined by P_i and P_j

They assume that $D(i,j)$ is non-decreasing while j increases. Therefore, a binary search method can be used in the polygonalisation procedure. For each P_i, the points P_j for $j = i+2^m$, where $m = 0, 1, 2, \ldots$, are checked in sequence if $D(i,j)$ is greater than the error. The last point P_j, whose $D(i,j)$ is less than ϵ is critical, and the points between P_i and P_j are removed. The time complexity is likely to be logarithmic due to the binary search. However, the assumption is not always true. Therefore, it is limited to convex polylines.

Among the above four polygonalisation algorithms, that of Sklansky and Gonzalez [45] is suitable for all applications. Although it cannot guarantee the minimum number of critical points due to its one-pass local optimisation, it is very time efficient, and the criticality of the remaining points is guaranteed. Moreover, the algorithm is theoretically sound and quite simple to implement. In general, the time complexity of this method is linear to the number of the original vertices. This is confirmed by the experiments of Dunham [46], who also shows that the number of remaining critical points is the smallest from among the nine algorithms tested. We use Sklansky and Gonzalez's [45] polygonalisation to process the point chain obtained from the SPV algorithm.

After the fine polyline has been determined, the line width can be calculated from the average of the width at each intermediate point of the polyline. The procedure can be applied to all groups of vectorisation methods. If the width information is not recorded, as is the case with some thinning-based methods, the original image should be consulted to get the width at these points. Other vectorisation methods keep the width information during line tracking, simplifying the final width calculation. For example, in SPV, the width run at each vertex of the polyline is projected on the normal line at that point, and the weighted average of these projected runs is taken as the line width of the polyline.

Table 1. Comparison of vectorisation method features

Method	Subprocess sequence*	Time complexity	Quality of line geometry	Line width preservation	Junction recovery	Image constraints	Applications Examples
HT based	HT,t	quadric	poor	no	yes	sparse, straight	Dori [1]
Iterative thinning	s, t	cubic	high	no	no	clean, thin	Fahn et al [6], Kasturi et al [7], Nagasamy and Langrana [8]
Contour	Edge detection, t	quadric	poor	yes	no	straight	Jimenez and Navalon [29]
Run-graph	run-graph construction, t	quadric	poor	yes	no	straight	Boatto et al [32]
Mesh	s while t	linear	poor	yes	yes	sparse, long	Vaxivere and Tombre [35]
OZZ	s while t	linear	poor	yes	yes	straight	Dori et al [38]
SPV	s while t	linear	good	yes	yes		Liu and Dori [40], Yoo et al [42]

*s means medial axis points sampling and t means line tracking

9. SUMMARY

Vectorisation is the most fundamental operation in document analysis, recognition and interpretation of line drawings in general, and technical documents in particular. It has to be implemented using a vectorisation method that best suits the needs of the system. Good methods should preserve the shape information, which includes line width, line geometry and intersection junction, as much as possible. This information is important for post-processing. To be of practical use in engineering systems, the vectorisation method should be fast enough. This paper has provided a survey of the various vectorisation methods useful for document analysis systems developers when selecting the most adequate method. To compare the vectorisation methods discussed in this paper, we list their main features in Table 1.

While a few benchmark-like contests have been organised around particular topics like dashed lines recognition [47] and engineering drawing recognition performance in general, no benchmarking-based performance evaluation and analysis has been conducted on vectorisation. This is in contrast with the centrality and importance of this process as the foundation of any line drawing recognition system. Conducting such a benchmark test will raise an important contribution to the domain of pattern recognition in general, and document analysis and recognition in particular.

References

1. Dori D. Orthogonal Zig-Zag: an Algorithm for Vectorizing Engineering Drawings Compared with Hough Transform. Advances in Engineering Software 1997;28(1):11–24
2. Hough PVC. A method and means for recognizing complex patterns. USA Patent 3,096,654, 1962
3. Tamura H. A Comparison of line thinning algorithms from digital geometry viewpoint. Proceedings of the 4th International Conference on Pattern Recognition, Kyoto, Japan, 1978, pp 715–719
4. Smith RW. Computer processing of line images: A survey. Pattern Recognition 1987;20(1):7–15
5. Lam L, Lee SW, Suen CY. Thinning methodologies – A comprehensive survey. IEEE Transactions on Pattern Analysis and Machine Intelligence 1992;14(9):869–887
6. Fahn CS, Wang JF, Lee JY. A topology-based component extractor for understanding electronic circuit diagrams. Computer Vision, Graphics and Image Processing 1988;44:119–138
7. Kasturi R, Bow ST, El-Masri W, Shah J, Gattiker JR, Mokate UB. A system for interpretation of line drawings. IEEE Transactions on Pattern Analysis and Machine Intelligence 1990;12(10):978–992
8. Nagasamy V, Langrana N. Engineering drawing processing and vectorisation system. Computer Vision, Graphics and Image Processing 1990;49(3):379–397
9. Hori O, Tanigawa S. Raster-to-vector conversion by line fitting based on contours and skeletons. Proceedings 2nd International Conference on Document Analysis and Recognition, Tsukuba, Japan, 1993, pp 623–626
10. Haralick RM, Shapiro L. Computer and Robot Vision. Addison Wesley, 1992.
11. Montanari U. Continuous skeletons from digitized images. Journal of the ACM 1969;16:534–549
12. Pfaltz JL, Rosenfeld A. Computer representation of planar regions by their skeletons. Communications of the ACM 1967;10:119–125
13. Davies ER, Plummer APN. Thinning algorithms: a critique and a new methodology. Pattern Recognition 1981;14:53–53
14. Naccache NJ, Shinghal R. An investigation into the skeletonization approach of Hilditch. Pattern Recognition 1984;17:279–284
15. Naccache NJ, Shinghal R. SPTA: a proposed algorithm for thinning binary patterns. IEEE Transactions on System, Man, and Cybernetics 1984;14:409–418
16. Rosenfeld A, Pfaltz JL. Sequential operations in digital picture processing. Journal of the ACM 1966;13:471–494

17. Peleg S, Rosenfeld A. A Min-max medial axis transformation. IEEE Transactions on Pattern Analysis and Machine Intelligence 1981;3:208–210
18. Nalwa VS. A Guided Tour of Computer Vision. Addison-Wesley, 1993
19. Hilditch CJ. Linear skeletons from square cupboards. Machine Intelligence 1969;4:403–420
20. Rosenfeld A. Connectivity in digital pictures. Journal of the ACM 1970;17:146–160
21. Deutsch ES. Thinning algorithms on rectangular, hexagonal and triangular arrays. Communications of the ACM 1972;15:827–837
22. O'Gorman L. k × k thinning. Computer Vision, Graphics and Image Processing 1990;51:195–215
23. Rosenfeld A, Pfaltz JL. Distance functions in digital pictures. Pattern Recognition 1968;1:33–61
24. Watson LT, Arvind K, Ehrich RW, Haralick RM. Extraction of lines and regions from greytone line drawing images. Pattern Recognition 1984;17:493–507
25. Lee S, Lam L, Suen CY. Performance evaluation of skeletonization algorithms for document image processing. Proceedings 1st International Conference on Document Analysis and Recognition, Saint-Malo, France, 1991, pp 260–271
26. Lam L, Suen CY. Evaluation of thinning algorithms from an OCR viewpoint. Proceedings 2nd International Conference on Document Analysis and Recognition, Tsukuba, Japan, 1993, pp 287–290
27. Jaisimha MY, Haralick RM, Dori D. A methodology for the characterization of the performance of thinning algorithms. Proceedings 2nd International Conference on Document Analysis and Recognition, Tsukuba, Japan, 1993, pp 282–286
28. Cordella LP, Marcelli A. An alternative approach to the performance evaluation of thinning algorithms for document processing applications. In: Kasturi R, Tombre K (eds). Graphics Recognition – Methods and Applications (Lecture Notes in Computer Science, 1072). Springer-Verlag Berlin, 1996, pp 13–22
29. Jimenez J, Navalon JL. Some experiments in image vectorisation. IBM Journal of Research and Development 1982;26:724–734
30. Shapiro B, Pisa J, Sklansky J. Skeleton generation from x-y boundary sequences. Computer Graphics and Image Processing 1981;15:136–153
31. Di Zenzo S, Morelli A. A useful image representation. Proceedings 5th International Conference on Image Analysis and Processing, Word Scientific Publishing, Singapore, 1989, pp 170–178
32. Boatto L et al. An Interpretation System for Land Register Maps. IEEE Computer 1992;25(7):25–32
33. Monagan G, Roosli M. Appropriate base representation using a run graph. Proceedings 2nd International Conference on Document Analysis and Recognition, Tsukuba, Japan, 1993, pp 623–626
34. Lin X, Shimotsuji S, Minoh M, Sakai T. Efficient diagram understanding with characteristic pattern detection. Computer Vision, Graphics and Image Processing 1985;30:84–106
35. Vaxiviere P, Tombre K. Cellestin: CAD conversion of mechanical drawings. IEEE Computer 1992;25(5):46–54
36. Vaxiviere P, Tombre K. Subsampling: A structural approach to technical document vectorisation. In: Dori D, Bruckstein A (eds), Shape, Structure and Pattern Recognition (Proceedings of the IAPR Workshop on Syntactic and Structural Pattern Recognition, Nahariya Israel, 1994). World Scientific, 1995, pp 323–332
37. Chai I, Dori D. Orthogonal Zig-Zag: An efficient method for extracting lines from engineering drawings. In: Arcelli C, Cordella LP, Sanniti di Baja G (eds), Visual Form. Plenum Press, 1992, pp 127–136
38. Dori D, Liang Y, Dowell J, Chai I. Spare pixel recognition of primitives in engineering drawings. Machine Vision and Applications 1993;6:79–82
39. Liu W, Dori D. Sparse pixel tracking: a fast vectorisation algorithm applied to engineering drawings. Proceedings 13th International Conference on Pattern Recognition, Volume III: Robotics and Applications, Vienna, Austria, 1996, pp 808–811
40. Liu W, Dori D. Automated CAD Conversion with the machine drawing understanding system. Proceedings 2nd IAPR Workshop on Document Analysis Systems, Malvern, PA, 1996, pp 241–259
41. Liu W, Dori D. A protocol for performance evaluation of line detection algorithms. Machine Vision Applications 1997;9:240–250
42. Yoo J-Y, Kim M-K, Kwon Y-B. Information extraction from a skewed form document in the presence of crossing characters. In: Tombre K, Chhabra A (eds). Graphics Recognition – Algorithms and Systems (Lecture Notes in Computer Science, 1389). Springer-Verlag, 1998, pp 139–148
43. Montanari U. A note on the minimal length polygonal approximation to a digitized contour. Communications of the ACM 1970;13(1):41–47
44. Hung SHY, Kasvand T. Critical points on a perfectly 8- or perfectly 6-connected thin binary line. Pattern Recognition 1983;16:297–284
45. Sklansky J, Gonzalez V. Fast polygonal approximation of digitized curves. Pattern Recognition 1980;12:327–331
46. Dunham JG. Optimum uniform piecewise linear approximation of planar curves. IEEE Transactions on Pattern Analysis and Machine Intelligence 1986;8(1):67–75
47. Dori D, Liu W, Peleg M. How to win a dashed line detection contest. In: Kasturi R, Tombre K (eds), Graphics Recognition – Methods and Applications (Lecture Notes in Computer Science, 1072). Springer-Verlag, Berlin, 1996, pp 286–300

Liu Wenyin is a Researcher at Microsoft Research, China, and a faculty member at the Department of Computer Science and Technology, Tsinghua University, Beijing, China. He received his BEngg and MEngg. in Computer Science from the Tsinghua University in 1988 and 1992, respectively, and his DSc in Information Management Engineering from Technion – Israel Institute of Technology, in 1998. His research interests include automated engineering drawing interpretation, pattern recognition, software engineering, object-oriented programming, object-process methodology, artificial intelligence and computer graphics. Liu Wenyin developed the Machine Drawing Understanding System (MDUS), and won first place in the Dashed Line Recognition Contest held during the First IAPR Workshop on Graphics Recognition at Pennsylvania State University in 1995. He also won a third prize in the First International Java Programming Contest (ACM Quest for Java'97), sponsored by ACM and IBM, in 1997.

Dov Dori has been a faculty member at The William Davidson Faculty of Industrial Engineering and Management, Technion, Israel Institute of Technology since 1991. He received his BSc in Industrial Engineering and Management from the Technion in 1975, his MSc in Operations Research from Tel Aviv University in 1981, and his PhD in Computer Science from the Weizmann Institute of Science, Rehovot, Israel, in 1988. His research interests include document analysis and recognition, computer vision and pattern recognition, information systems engineering, systems development methodologies and computer-aided software engineering. Dr Dori is Associate Editor of *IEEE Transactions on Pattern Analysis and Machine Intelligence*, and is on the editorial board of the *International Journal of Document Analysis and Recognition* and of the *International Journal of Pattern Recognition and Artificial Intelligence*. He is senior member of the IEEE and a member of the IEEE Computer Society, ACM and IAPR.

Correspondence and offprint requests to: L. Wenyin, Microsoft Research, Sigma Center, #49 Zhichun Road, Beijing 100080, PR China.