Sparse Pixel Vectorization: An Algorithm and Its Performance Evaluation

Dov Dori, *Senior Member*, *IEEE*, and Wenyin Liu

Abstract—Accurate and efficient vectorization of line drawings is essential for their higher level processing. We present a thinningless Sparse Pixel Vectorization (SPV) algorithm. Rather than visiting all the points along the wire's black area, SPV sparsely visits selected medial axis points. The result is a crude polyline, which is refined through polygonal approximation by removing redundant points. Due to the sparseness of pixel examination and the use of a specialized data structure, SPV is both time efficient and accurate, as evaluated by our proposed performance evaluation criteria.

Index Terms—Vectorization, line tracking, sparse pixel vectorization, polygonal approximation, performance evaluation.

1 Introduction

Vector is a compact form of data, describing the geometry of a bar (straight line segment with nonzero width) and other line shapes using a small number of attribute values, e.g., two endpoints and line width for lines, and an additional center for circular arcs. A raster image, in contrast, conveys less semantics and requires much more space to store a large amount of pixels for the purpose of describing the same graphic object.

The role of vectorization is to convert graphic objects in a raster image to vector form. Basic vectorization concerns grouping the pixels in the raster image into raw line fragments. The resulting objects may be bars (nonzero width line segments) or polylines (chains of bars linked end to end, usually generated from curve images). These objects can be refined during the post-processing (or refinement) phase using line fitting and line extending processes.

The resulting raw vector fragments are representations of basic components of most classes of graphic objects in engineering drawings. Most line-like graphic objects can be intuitively decomposed into line fragments. Thus, an arrowhead, for example, can be approximated by a chain of several bars with increasing widths from the tip to the end, while a character consists of a number of strokes, which result in a set of bars. During graphics recognition, the resulting vectors are grouped to yield some higher level graphic object, such as an arrowhead, a character, etc.

The basic, crude vectorization is a fundamental process in the interpretation of line drawings in general and engineering drawings in particular, and is independent of later processing phases. It can also be used as a preprocessing for OCR. For advanced vectorization and other segmentation tasks to be as reliable as possible, the crude vectorization has to be as accurate as possible, i.e., preserve the original shapes of the graphic objects in the raster image to the highest extent possible.

Crude vectorization techniques have been developed in various domains, and a number of methods have been proposed and implemented. These methods, surveyed in [1] and [2], are roughly divided into two classes: thinning based methods and nonthinning based methods.

Thinning based methods [3] are applied in most of the earlier vectorization schemes [4], [5], [6], as well as in some recent methods, e.g., [7]. These methods usually employ an iterative boundary erosion process to remove outer pixels, until only a one-pixel-wide skeleton remains. A line tracking process is used to link the pixels on the skeleton to a pixel chain. A polygonal approximation procedure is then applied to convert the pixel chains to a vector, which may be a single bar or a polyline (a chain of two or more bars). The main advantage of thinning based methods is that they are capable of maintaining the line connectivity. Major disadvantages include loss of line width information, distortions at junctions, and time inefficiency.

Among the nonthinning based methods, contour based methods [2] were also popular in the early days of vectorization. Methods belonging to this class first extract image contours—the edges of the shape—between which the medial axis is then found. Contour based methods often miss pairs of contour lines at junctions, resulting in gaps that break the vectors. This disadvantage is compensated for by the preservation of line widths, which is important for postprocessing.

Other classes of nonthinning based methods that also preserve line width have been developed recently. These include run graph based methods [8], [9], mesh pattern based methods [10], [11], and the sparse pixel Orthogonal Zig-Zag (OZZ) method [12], [13], [14].

Run graph based methods seem to be able to solve the problem of keeping both the connectivity and the line width information, because they record the node areas that

- *D. Dori is with the Faculty of Industrial Engineering and Management, Technion, Israel Institute of Technology, Haifa 32000, Israel. E-mail: dori@ie.technion.ac.il.*
- *W. Liu is with Microsoft Research, Sigma Center, #49 Zhichun Road, Beijing 100080, People's Republic of China. E-mail: wyliu@microsoft.com.*

Manuscript received 11 July 1996; revised 25 Jan. 1999. Recommended for acceptance by R. Kasturi.
For information on obtaining reprints of this article, please send e-mail to: tpami@computer.org, and reference IEEECS Log Number 107764.

are junctions. A run graph representation [9] can be viewed as a semivector representation, since it uses nodes corresponding to the end points of vectors, along with a set of adjacent runs to express the digital segment between these two nodes. A polygonal approximation procedure is applied to the middle points of the set of runs representing the vector to determine its attributes. Like most vectorization methods, the run graph method is susceptible to noise and may cause distortions at junctions, because the intersection points are not precisely located during the construction of the run graph representation.

Mesh pattern based methods use many patterns of square frames sampled from the drawing and investigate only the borders of the squares for tracking decision. The mesh size in this method is very critical. If it is too large, the patterns inside the mesh are too complex to control, while if it is too small, the variety of mesh border patterns may not be large enough to distinguish among different cases. This causes a variety of problems, such as erroneously connecting broken lines with small gaps to yield one line in case the junction is completely contained inside the mesh border and merging close parallel lines if the mesh is too coarse. This method can therefore be successfully applied only for a restricted class of line drawings, in which the lines are straight and sparse.

OZZ is different from other non-thinning methods since it is based on tracking a one pixel wide path within the foreground area (usually denoted by black pixels) of a graphic object, turning orthogonally each time the path hits the edge of the area. The midpoints of the horizontal and vertical runs on the path are used to reconstruct the bar or the group of bars that approximate the graphic object. One of OZZ's advantages is its time efficiency, which is due to the sparse sampling of the image. Other advantages are OZZ's capability to partially overcome problems caused by intersections and junctions and its preservation of line width. The main disadvantage of OZZ is its imperfect approximation of arc images, in which neighboring bars often intersect or do not touch each other.

Most vectorization algorithms are capable of dealing with straight line images while inaccuracy (of shape preservation) in crude vectorization of arc images is common to many vectorization methods, especially to thinningless ones, such as those described in [8], [9], [10], [11]. Another common shortcoming of most vectorization methods is their poor junction recovery. Thinning-based methods distort junctions in spite of their connectivity preservation, while contour-based methods tend to break lines. While recent improvements [7], [8], [15], [16] include revisions and combinations of existing methods, they do not completely solve these problems. Vector fitting [8], [16] increases the vector quality at the expense of extra time processing by adjusting vectors according to their original pixels, but this is a refinement technique that can be avoided to a large extent if accurate results are obtained at the basic vectorization phase. Hori [7] claims that small, unfilled gaps in lines should be overlooked as noise. We maintain that crude vectorization must preserve such gaps, because some of them, such as those in dashed lines, are real gaps, while others, which are indeed noise that break the lines, can be filled during refinement. Our view is that maximizing the quality of vectors is the objective of the refinement and advanced graphics recognition. Crude vectorization should focus on shape preservation while striving to be time efficient.

With this in mind, we have developed the Sparse Pixel Vectorization (SPV) algorithm to cater to the requirements of shape preservation and time efficiency. Originally presented in [17], SPV is an improved version of the OZZ vectorization algorithm. Rather than visiting all the pixels of a foreground area at least once, as thinning-based algorithms do, SPV visits only a selected subset of the medial axis points. The result is a polyline, formed by tracking the sparsely visited medial axis points. The resulting vectors are refined by a polygonal approximation algorithm [18] to remove redundant points.

Like OZZ, SPV preserves the line width—a critical parameter for higher-level engineering drawing understanding—as well as an accurate medial axis and endpoints. Due to the sparse pixel examination and the use of a specialized data structure [19], the vectorization is also highly time efficient. We present a performance evaluation of the SPV algorithm on both the precision and execution time aspects.

Although a significant number of vectorization algorithms, including [1], [2], [3], [4], [5], [6], [7], [8], [9], [10], [11], [12], [13], [14], [15], [16], have been developed, the performance of most of them is known at best only from the reports of their developers, based on their own perceptual, subjective, and qualitative human vision evaluation. Objective evaluations and quantitative comparisons among them are not available. Kong et al [20], Hori and Doermann [21], Liu and Dori [22], and Phillips et al. [23] have reported performance evaluation of vectorization algorithms only recently. Of the above-mentioned protocols, only the protocol in [22] evaluates shape preservation using pixel recovery indices. We have therefore chosen to apply it to the evaluation of the SPV algorithm. Following the description of the algorithm, we present the evaluation results and discuss them.

The rest of the paper is organized as follows. Section 2 describes the details of SPV. Section 3 discusses possible refinement tasks. Section 4 briefly presents the vectorization evaluation protocol [22] we used. Section 5 presents experimental results of SPV and its evaluation. Finally, Section 6 includes a summary and proposes conclusions.

2 THE SPARSE PIXEL VECTORIZATION ALGORITHM

The SPV algorithm is inspired by OZZ. However, unlike OZZ, SPV does not follow the simple zig-zag pattern by bouncing from the edges of a foreground area. Rather, it traces the medial axis points of consecutive horizontal or vertical pixel runs. In OZZ, two passes of scanning the image are required (one for extracting bars of horizontal inclinations and another for extracting bars of vertical inclinations), and there are three cases of processing (horizontal, vertical, and slanted). The bars detected in the two passes then need to be combined for the final output. The two methods visit approximately the same number of black pixels in one pass of scanning. However, SPV requires only

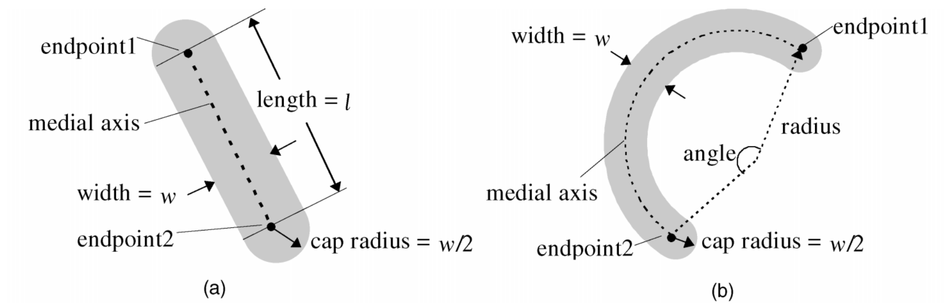


Fig. 1. (a) The ideal oval area occupied by a bar. (b) The ideal arc image.

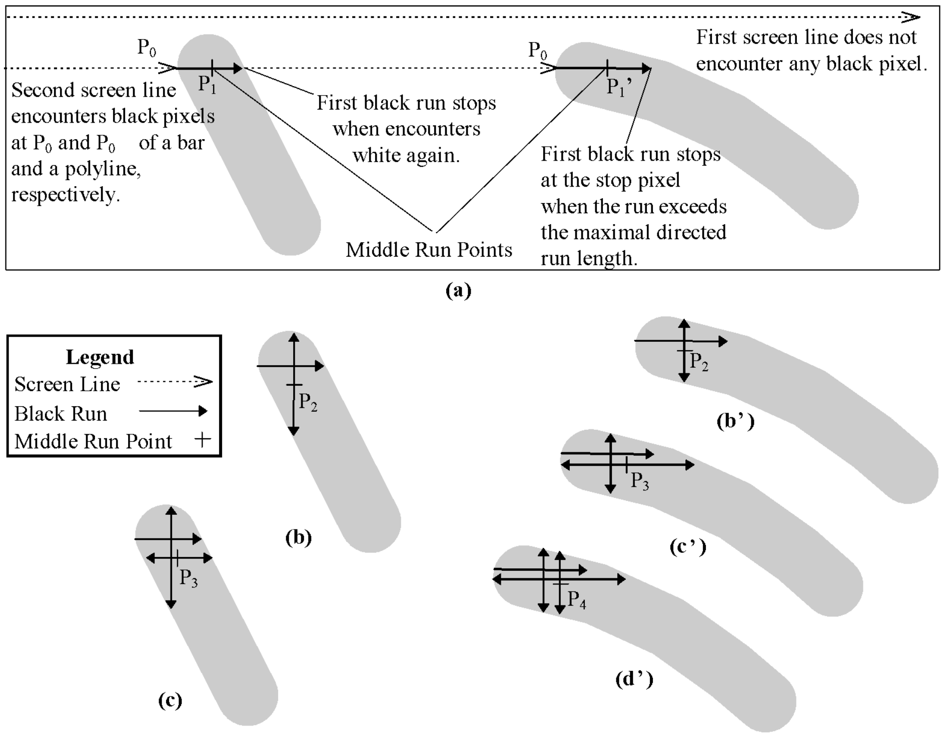

P_0
P_1
P_0
P_1'
First screen line does not encounter any black pixel.
Second screen line encounters black pixels at P_0 and P_0 of a bar and a polyline, respectively.
First black run stops when encounters white again.
First black run stops at the stop pixel when the run exceeds the maximal directed run length.
Middle Run Points
(a)
Legend
Screen Line
Black Run
Middle Run Point
P_2
(b)
P_3
(c)
P_2
(b')
P_3
(c')
P_4
(d')

Fig. 2. Illustrations of the procedure of finding the Start Medial Axis Point and the Tracking Direction. (a) The second scan (screen) line encounters two wires (a bar and a polybar) at P_0 and P_0', respectively. Two horizontal middle run points, P_1 and P_1', are obtained separately by making two horizontal runs starting from P_0 and P_0' in the wire areas. (b) Middle run point P_2 is obtained by making two vertical, opposite directed runs from P_1 and calculating the middle point of the run they formed. (c) Middle run point P_3 is obtained from P_2. P_3 is determined as the starting point of the tracking procedure because the distance between P_3 and P_2 is small enough. The Tracking direction is determined as vertical because the vertical black run at P_3 is longer than the horizontal black run at P_3. (b′) Middle run point P_2' is obtained from P_1'. (c′) Middle run point P_3' is obtained from P_2'. (d′) Middle run point P_4' is obtained from P_3'. P_4' is determined as the starting point of the tracking procedure because the distance between P_4' and P_3' is small enough. The Tracking direction is determined as horizontal because the horizontal black run is longer than the vertical black run at P_4'.

one pass and operates uniformly for all bar orientations. Hence, SPV is twice as fast as OZZ in this sense. Like OZZ, SPV solves some of the junction problems already at the crude vectorization stage. The remaining problems are solved during the refinement stage.

The terms defined below and illustrated in Fig. 1, Fig. 2, Fig. 3, and Fig. 4 are used throughout the paper to concisely describe the details of the SPV algorithm.

1) *Black pixel*—a foreground pixel of graphic objects in the image.
2) *Black pixel area*—the foreground area of graphic objects in the image.
3) *Bar*—a straight line segment with nonzero width, which occupies a (possibly slanted) black pixel area in the drawing image. The ideal area is an oval consisting of a rectangle and two semicircles, as shown in Fig. 1a.
4) *Arc*—a circular line segment with non-zero width, which occupies a (possibly closed) black pixel area in the drawing. The ideal end points are also round capped, as shown in Fig. 1b.
5) *Polybar*—a chain of two or more equal width bars linked end to end. Bar widths are considered equal if they are within a predefined tolerance.
6) *Polyline*—a polybar resulting from approximating an arc or a free curve by a crude vectorization procedure.

7) *Wire*—a generalization of bar, circular arc, and polybar, which is used to denote any line-like graphic object in the drawing image.
8) *Scan Line*—a one-pixel-wide horizontal line that crosses the drawing image from left to right, used to find the first black pixel of a black pixel area. Scan lines are spaced vertically every *n* pixels, where *n* is smaller than the minimal anticipated line width.
9) *Current Pixel*—the pixel at which the tracking has arrived at a given point in time.
10) *Pixel Visiting Direction*—one of the four directions left, right, up, or down, to which the pixel following the current pixel is visited.
11) *Directed Run*—a sequence of consecutive horizontal or vertical black pixels in the pixel visiting direction from the first encountered black pixel to a stop pixel (defined below).
12) *Directed Run Sign*—a sign which is negative when the pixel visiting direction of the directed run is left or up, and positive when it is right or down.
13) *Stop Pixel*—the first pixel along a directed run which either encounters a white pixel or exceeds a predefined maximal directed run length.
14) *Run*—the undirected, unsigned sequence of consecutive black pixels formed by linking two directed runs with opposite directions that start from a common point.
15) *Middle Run Point*—the middle point of a run. If the run length is even it is taken as the left or top of the two middle points.
16) *Length Direction*—a pixel visiting direction whose associated run is longer than the corresponding run in the orthogonal direction. The length direction is horizontal if the slant of the line at the current point is up to 45° and vertical otherwise.
17) *Width Direction*—the pixel visiting direction orthogonal to the length direction of the associated run.
18) *Tracking Step*—a directed run of a specified length in the length direction. A tracking step is taken to search for the next medial axis point from the current medial axis point.
19) *Width Run*—a run in the width direction.
20) *Length Run*—a run in the length direction.
21) *Tracking Cycle*—a cycle of finding the next middle run point, which consists of taking a tracking step, making the width run, and then obtaining the middle point of the width run.

We use the terms defined above to explain the SPV algorithm in detail in the following subsections.

2.1 Finding the Starting Medial Axis Point and Tracking Direction

To start an SPV procedure, we need to find a reliable starting medial axis point, which is not affected by noise and inaccuracies at the wire's end. The starting medial axis point is found by the following procedure. When the first black pixel (P_0 in Fig. 2) is found at the point where a scan line encounters a black area, we get a first directed run by going from P_0 to the right till a stop pixel. Going back from P_0 to the left, we get a 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 few iterations are sufficient, and we can use P_3 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. 2.

After we get the starting medial axis point P_i, we also know the lengths of both the horizontal and vertical runs at 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 above 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.

2.2 The Sparse Pixel Tracking Procedure

As Fig. 3a shows, going from the first medial axis point P_3 in the tracking direction, which is vertical, we begin the main

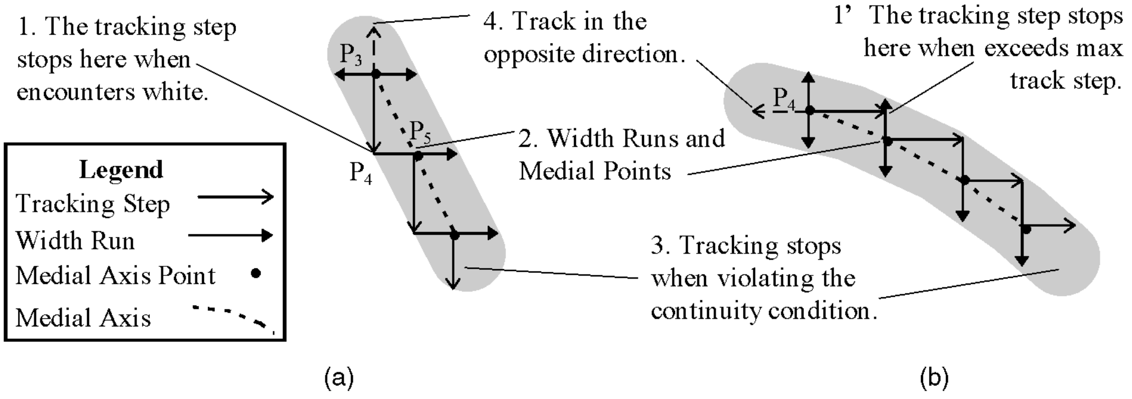
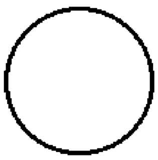
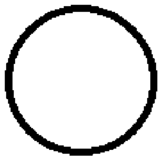
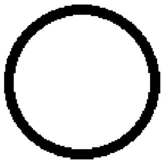
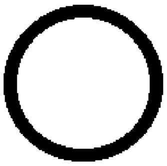
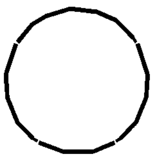
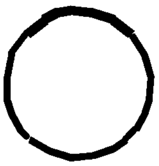
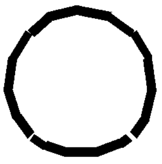
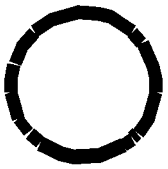

1. The tracking step stops here when encounters white.
4. Track in the opposite direction.
1' The tracking step stops here when exceeds max track step.
P3
P4
P5
P4
2. Width Runs and Medial Points
Legend
Tracking Step
Width Run
Medial Axis Point
Medial Axis
3. Tracking stops when violating the continuity condition.
(a)
(b)

Fig. 3. Illustrations of the general procedure of Sparse Pixel Tracking. (a) A vertical tracking case. (b) A horizontal tracking case.

TABLE 1
IMAGES OF CIRCLES WITH DIFFERENT WIDTHS AND THEIR VECTORIZATION BY SPV

Width	3 pixels	5 pixels	7 pixels	9 pixels
Raster Image				
SPV Vectors				

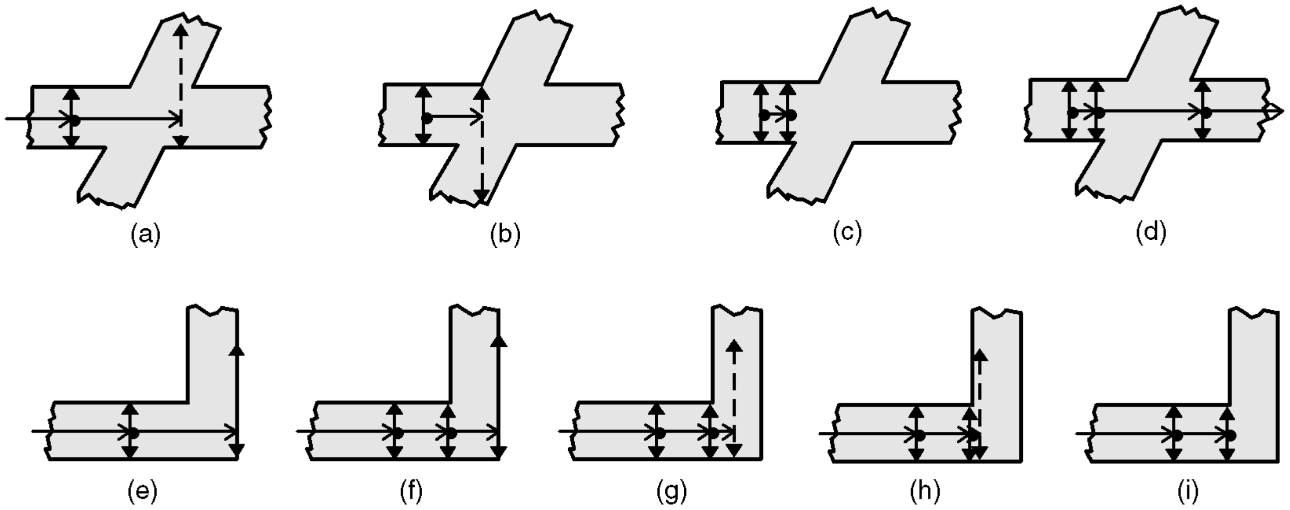


Fig. 4. Demonstration of the Junction Recovery Procedure. (a)-(d) Cross case: (a) The width run at the tracking step violates the width preservation condition. (b) The tracking point retreats to the last medial point and the tracking is repeated with half-size of the previous tracking step. The width run still violates the condition. (c) The tracking point retreats back again to the last medial point, and the tracking is repeated with half-size of the previous tracking step. This time, the width run meets the continuity condition. (d) The tracking step taken from the new medial point crosses the junction, and the width run there meets the continuity condition. (e)-(i) Corner case: (e) The width run at the tracking step violates the width preservation condition. (f) The tracking point retreats to the last medial point and the tracking is repeated with half-size of the previous tracking step. This time the width run meets the condition. Continuation of the tracking again violates the width preservation condition. (g), (h) The tracking point retreats twice to the last medial point and the tracking is repeated with half-size of the previous tracking step. Both width runs violate the continuity condition. (i) The tracking point retreats back to the last medial point and the tracking is repeated with half-size of the previous tracking step. The tracking step becomes zero and the tracking stops.

Sparse Pixel Tracking procedure for this black area. To start a tracking cycle, we take a tracking step from the last medial axis point and reach point P_4. From P_4 we make two opposite directed width runs, from which we get the (undirected) width run and its middle run point P_5 that serves as the new medial axis point. This is the end of tracking this cycle. We repeat these tracking cycles while recording and monitoring the medial axis points and the width run lengths, 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 neighborhood of some small number of adjacent medial axis points, is below some threshold, e.g., 50% of the neighborhood 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) *Length direction consistency*: The length direction is the same as that at the previous medial axis point during one tracking procedure.
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 neighborhood 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. In particular, the sparse pixel tracking procedure over an arc image stops at the four diagonal positions where the length direction changes either from horizontal to vertical or from vertical to horizontal. The resulting polyline is therefore broken at these positions into a set of polyline fragments. This phenomenon is demonstrated in Table 1, where the results of SPV on four full circles are demonstrated.

Table 1 depicts four 50-pixel-radius circle images with line widths of three, five, seven, and nine pixels, respectively. The lower row shows the SPV results. The circle whose line width is seven is broken into exactly four polylines at the four positions where the bisectors of quadrants intersect with the circle. If the gap at the break is large enough, a short bar may be found to fill in the gap, as is the case for the circles with line widths three, five, and nine. In particular, for line width nine, the polylines become more fragmentary and there is more than one bar at some break positions.

2.3 Junction Recovery

When one or more of the first three continuation conditions is violated, the Sparse Pixel Tracking procedure pauses and a Junction Recovery Process starts using the Junction Recovery Procedure, as exemplified by the cross and the corner in Fig. 4. 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 violates one of the above conditions, a new iteration is reapplied with the tracking step half-sized. The iterations halt when the tracking step becomes zero, as in Condition 4. If at the new medial axis point all the conditions are met again, then the general tracking procedure continues from the new medial axis point with the normal tracking step. By so doing, the procedure may overcome uneven areas, where the width runs are significantly different, as demonstrated in Fig. 4a, Fig. 4b, Fig. 4c, and Fig. 4d. 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 Fig. 4e, Fig. 4f, Fig. 4g, Fig. 4h, and Fig. 4i.

2.4 The Polygonal Approximation

The result of the sparse pixel tracking procedure is a chain of points that are approximately on the medial axis of the black area. Although this chain may be regarded as a vector representation of the corresponding black area, some intermediate points are redundant because they lie (approximately) on the straight line segment formed by connecting their neighbors. These points should be removed from the chain in order to represent the polyline with the fewest edges (or vertices) that approximate the original shape while maintaining the shape to the closest extent possible. This is done by polygonal approximation. We use Sklansky and Gonzalez's method [18]. This method cannot guarantee the minimum number of critical points due to one-pass local optimization (maximal number of local noncritical points), but it is very time efficient and the criticality of the remaining points is guaranteed.

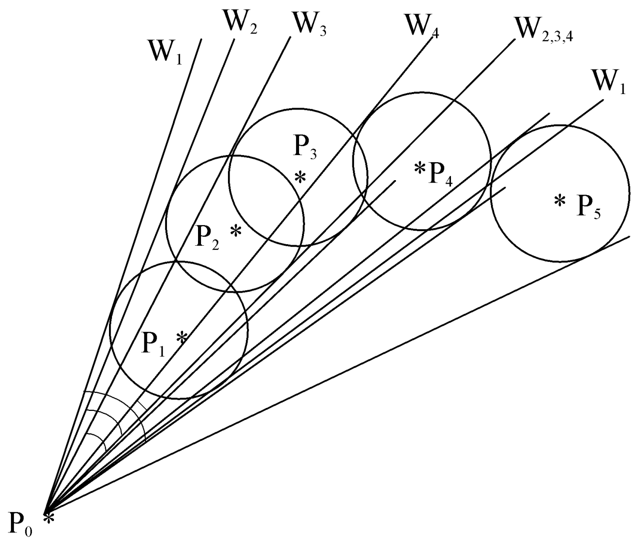


Fig. 5. Sklansky and Gonzalez's method for polygonal approximation.

To preserve the original shape, a value denoted by ε is predefined to constraint the procedure. The procedure finds as few critical points as possible from the point chain, such that the maximal distance of any noncritical point between every pair of consecutive critical points to the chord joining these two points is less than ε. The criticality of a point is then determined by its distance to the line formed from its two neighbor critical points on both sides. One endpoint of the chain is set as the first critical point for initialization. Since the critical points are not known in advance, each critical point is found by going ahead from a known critical point each time, and finding the next critical point and the intermediate noncritical points simultaneously. This is done by finding the farthest point whose distance from the line linking it with the current critical point is less than ε. The farthest point is the new critical point and the intermediate points are noncritical. This way we find as many noncritical points as possible, so only few critical points are left.

In Sklansky and Gonzalez's method, a wedge formed by two tangents from the start (critical) point (such as P_0 in Fig. 5) 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, such that it forms the intersection of the wedges from all points so far. When the wedge shrinks to zero, the last point to fall inside the wedge will be marked as critical and taken as the new initial point. Fig. 5 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 that 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.

The time complexity of this method is linear in general. The worst case complexity is $O(n^2)$, and it occurs when all

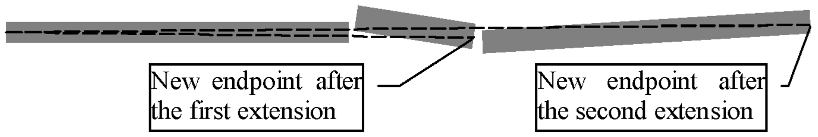

New endpoint after the first extension
New endpoint after the second extension

Fig. 6. Stepwise bar extension

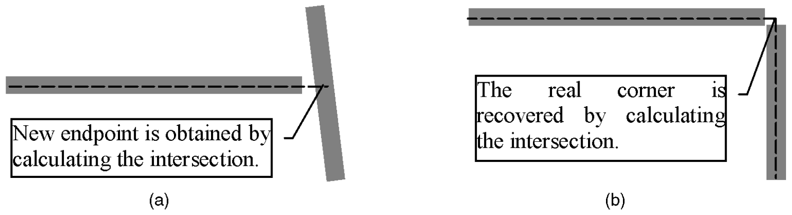

New endpoint is obtained by calculating the intersection.
(a)
The real corner is recovered by calculating the intersection.
(b)

Fig. 7. Illustration of adjusting the bar's endpoint to its orthogonal line.

points are critical and the criticality examination for each point has to continue to the last point in the chain. This case is rare, or even impossible in smooth curves.

The value of ε determines how well the shape of the original chain will be preserved, that is, it determines the accuracy of the edges and the number of vertices of the polyline. The smaller the value, the larger the number of vertices. ε is set to a small number of pixels. In polygonal approximation of engineering drawing lines, ε can be set as much as half of the line width, as used in the refinement procedure. For best results, we have used one pixel in the current implementation. To increase ε, the line width should be known. The ceiling of the total width run/$\sqrt{2}$ is the lower limit of the line width, because any bar makes at most 45° with the tracking direction.

The result of the polygonal approximation is a polyline (which may be a bar if only two vertices are left) without real line width. To find the polyline's width, the width run at each vertex of the polyline is projected on the normal at that point, and the average of these projected runs is taken as the line width of the polyline.

3 Optional Refinement Processes

For line drawings, the refinement stage may include the following three tasks:

1) linking collinear line segments that have similar line width;
2) correcting defects at junctions; and
3) refining the vector endpoints, line width, and other attributes values.

Linking short line segments into longer ones may yield the correct line width and overcome some junction problems. Defects at junctions, such as corners and branches, are subject to special processing, in which the precise intersection points are calculated.

As noted, the vectors resulting from the SPV algorithm may be broken. This problem can be overcome during refinement. Two line segments are linked if the following conditions are met.

1) Width similarity: The width difference of the two line segments is not significant.
2) Proximity: The endpoints of the two segments are close enough.
3) Colinearity: The distance from the joining point to the straight line formed by joining the farther pair of endpoints is less than a predefined value ε.

While this operation is similar to testing the criticality of the intermediate points of the polyline in polygonal approximation, it is done in a different way than the method of Sklansky and Gonzalez. The line width is the weighted average of the two bars with the weights being their lengths. The broken lines are linked in a stepwise extension manner, as shown in Fig. 6. An initial bar is selected and is gradually extended using adjacent bars in the direction pointing from its current endpoint outside along its slant.

The final endpoint of a bar that touches another bar is adjusted by extending the endpoint of the touching bar to the intersection with the touched bar, as illustrated in Fig. 7a. The endpoints of two bars that end at the same point are adjusted as shown in Fig. 7b.

Converting polylines with low enough curvature to straight line segments can also be done during the refinement. The endpoints of the polyline are taken as the only two critical points, and the distances from the intermediate points to the chord linking these two points are used to test their criticality, as shown in Fig. 8. It is also possible to repeat the polygonal approximation using Sklansky and Gonzalez's method, but with a bigger ε, to further reduce the number of vertices on the polyline. To avoid cumulative errors, the value of ε is set small during the polygonal approximation. In the refinement procedure, the value of ε is set more loosely, and it can be as much as half the line width of this polyline. If only two points remain, the polyline is converted to a straight line.

The line widths of the resulting vectors may be statistically classified into a thinner group and a thicker group, as defined by the two major standards for engineering draw-

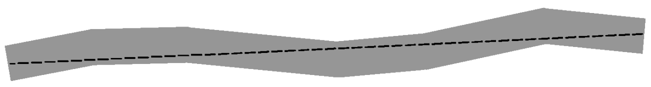


Fig. 8. Converting a polyline to a bar.

ings—ISO and ANSI. This classification is also very useful during postprocessing and recognition of higher level graphical objects, which leads to the required separation of the annotation layer from the geometry layer in engineering drawings.

Line drawings that contain circular arcs require arc segmentation as part of the refinement. For this purpose, we developed an incremental arc segmentation algorithm [24], which takes as input the vectors resulting from any basic vectorization process, such as the SPV algorithm.

4 Vectorization Performance Evaluation Protocol

To evaluate the SPV algorithm, we have applied the general performance evaluation protocol for line detection algorithms [22], described briefly below. The protocol consists of two levels of performance indices of the evaluated algorithm: the pixel level and the vector level. At each level, the detection rate and the false alarm rate are calculated. The pixel level indices are the pixel level detection rate (D_p) and pixel level false alarm rate (F_p). They rate the shape preservation (pixel recovery) capability of the algorithm, and are defined in (1) and (2), respectively.

$$D_p = \frac{\left|P_g \cap P_d\right|}{\left|P_g\right|} \tag{1}$$

$$F_p = 1 - \frac{\left|P_g \cap P_d\right|}{\left|P_d\right|} \tag{2}$$

P_g is the set of all the black pixels in the ground truth image, i.e., the original image to be vectorized, and P_d is the set of all the black pixels detected by the vectorization algorithm.

Combining the Pixel Detection Rate and Pixel False Alarm Rate, the resulting Pixel Recovery Index is defined in (3).

$$PRI = \alpha D_p + (1-\alpha)\left(1-F_p\right) \tag{3}$$

$0 \le \alpha \le 1$ is the relative importance of the detection, and $1 - \alpha$ is the relative importance of the false alarm. In this paper, we use $\alpha = 0.5$, which is the default that is employed when no information is provided regarding the relative importance of detection versus false alarms.

The vector level performance indices are defined as follows. For the case of SPV, we consider each pair of ground truth and detected vector as being matched if they overlap each other either fully or partially. The vector detection quality of the overlapping segments of every detected line and the ground truth line are measured by the detection accuracy using a number of criteria, including the endpoint locations, their offsets, and the line width. The vector detection quality of these overlapping segments, as well as their lengths, are accumulated for both the ground truth lines and the corresponding detected lines. We use these quantities to calculate three indices:

1) the Basic Quality, (Q_b), which is the length-weighted sum of the vector detection quality of overlapping segments;
2) the Fragmentation Quality, (Q_{fr}), which is the measure of the detection fragmentation and/or consolidation; and
3) the Total Quality, (Q_v), which is the product of Q_b and Q_{fr}, for both the ground truth lines and their corresponding detected lines.

Using these values, we calculate three other indices:

1) the total Vector Detection Rate, D_v, which is the length weighted sum of Q_v of all ground truth lines;
2) the Vector False Alarm Rate, F_v, which is the length weighted sum of $1 - Q_v$ of all detected lines; and
3) the resulting Vector Recovery Index, $VRI = (D_v + 1 - F_v)/2$.

Since we consider crude vectorization in general and SPV in particular to account only for shape preservation, we evaluate it using the pixel level indices only.

As discussed in Section 1, there are many different algorithms for basic vectorization as well as for refinement. Combination of the choices of these two subprocesses yields an even larger number of vectorization schemes. High performance of basic vectorization facilitates good refinement, but low performance of some basic vectorization algorithm does not mean a suitable refinement process cannot be adopted to obtain a high performance final result. We therefore wish to evaluate each one of the two subprocesses separately.

The Vector Recovery Index is suitable for evaluating the vector list that is the output of the refinement. Since SPV is a crude vectorization algorithm that yields only bars and polylines, images that contain arcs and other styles of lines cannot be adequately evaluated using the vector level indices.

Time efficiency is evaluated by reporting the elapsed time of running the program for each drawing. Since drawings vary in size, density, and complexity, we use two drawing-independent time evaluation factors to evaluate the time efficiency of vectorization algorithms: time per (black) pixel and time per vector.

5 Experimental SPV Results and Their Evaluation

The SPV algorithm has been implemented in C++ on SGI Indy (IRIX5.3), Sun (Solaris2.5) workstations and PC (Windows 95). The code for Sun (Solaris) is available from the ftp address given in [25]. Fig. 9, Fig. 10, Fig. 11, Fig. 12, Fig. 13, Fig. 14, Fig. 15, and Fig. 16 display the results of applying SPV on drawings of different nature. The (approximate) time evaluation of the SPV process of these drawings is given in Table 2. The two time performance evaluation indices—time per (black) pixel and time per vector—are also provided in Table 2. We use (1) and (2) to evaluate the shape preservation capability of SPV. The shape preservation evaluation of basic vectorization of Fig. 9, Fig. 10, Fig. 11, Fig. 12, Fig. 13, Fig. 14, Fig. 15, and Fig. 16 is also given in Table 2. For technical reasons, only the top left part of the drawing that can be viewed on a single screen was considered in the calculation.

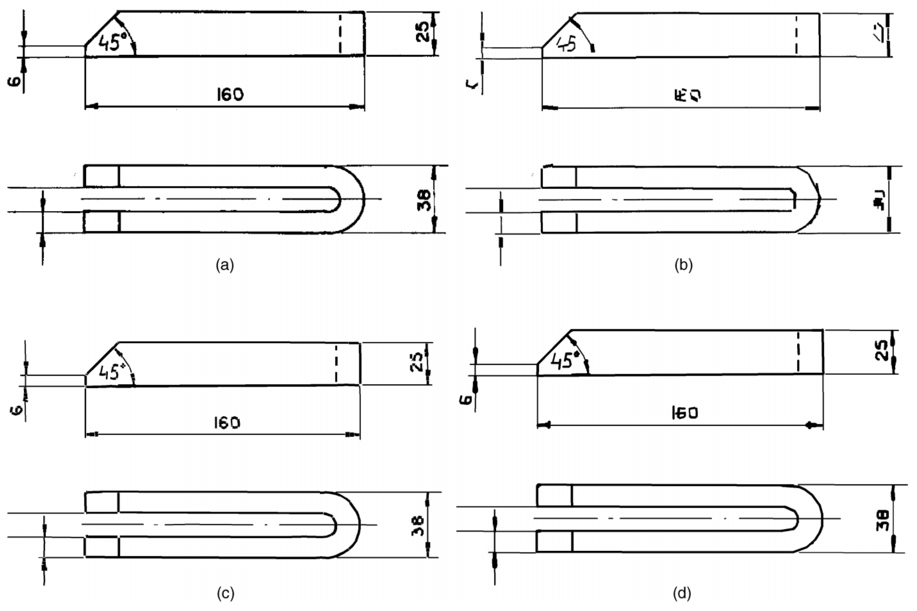


Fig. 9. Result of SPV compared with OZZ. (a) Original drawing. (b) Vectorization by OZZ. (c) SPV results. (d) SPV and refinement results.

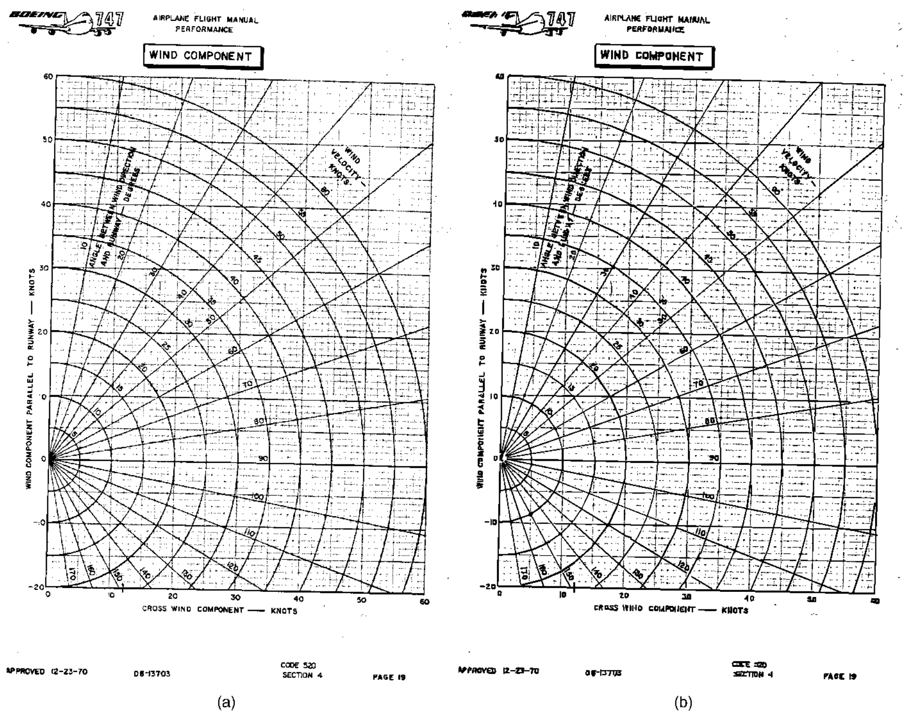
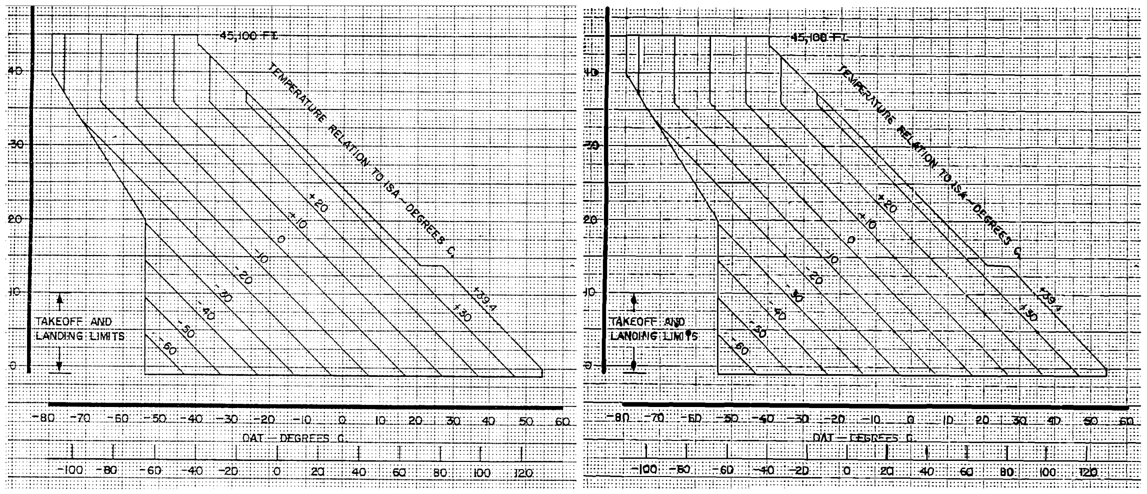


Fig. 10. Result of vectorization by SPV. (a) Original drawing. (b) SPV results.

Fig. 9 is the vectorization result of a real life clean drawing with a typical density (5%), compared with the vectors resulting from OZZ. Fig. 9a is the original drawing. Fig. 9b is the vectorized drawing by OZZ. As we can see, the arcs are vectorized by OZZ as overlapping bars. Some short arcs belonging to characters are also missing. Fig. 9c is the result of SPV without refinement. Here, all vectors approximately maintain their original shape. This can also be confirmed by the high detection rate, 96.8%, in Table 2. The false alarm rate is mostly due to the polylines that approximate arcs. After refinement, some fragmentary vectors are linked into longer, more straight, integrated ones, and endpoints of lines are extended to their ideal positions, as shown in Fig. 9d. Fig. 10 and Fig. 11 are performance graphs of the Boeing 747 airplane, which are very noisy and have high densities (about 20%) due to the grid background of the graphs. Fig. 12, Fig. 13, Fig. 14, Fig. 15, and Fig. 16 are other kinds of real life engineering drawings tested by the SPV algorithm, which demonstrate the suitability of SPV for a variety of engineering drawing types.

To automatically test the vector quality along with the pixel detection of SPV, we download two test images and



(a) (b)

Fig. 11. Result of vectorization by SPV. (a) Original drawing. (b) SPV results.

TABLE 2
PERFORMANCE EVALUATION OF THE SPARSE PIXEL VECTORIZATION ALGORITHM ON INDY

Fig. #	Size (pixels)	Noise Level	Complexity	# Black Pixels	Pixel Density	Time on Indy (sec)	Time (μs) per pixel	# detected vector	Time (ms) per vector	Detection Rate (D_p)	False Alarm Rate (F_p)
9	1568X899	low	low	69371	4.9%	3	43.2	164	18.3	0.968	0.146
10	1992X3036	high	high	684240	11.3%	92	134.4	25368	3.6	0.837	0.181
11	1768X1500	high	high	370282	14.0%	56	151.2	20347	2.8	0.780	0.230
12	344X644	high	medium	26002	11.7%	3	115.4	213	14.1	0.883	0.164
13	456X430	high	medium	20064	10.2%	2	99.6	185	10.8	0.869	0.142
14	1520X1111	high	high	120351	7.1%	12	99.7	1279	9.4	0.848	0.189
15	792X636	low	low	29851	5.9%	3	100.5	228	13.2	0.873	0.221
16	600X1012	low	low	41333	6.8%	3	72.6	286	10.5	0.940	0.159
Average							102.1		10.3	0.875	0.179

their ground truths. These were taken from the web page developed by Chhabra and Phillips [26] for the second graphics recognition contest held during the *Second IAPR Workshop on Graphics Recognition*, Nancy, France, 1997 and presented in the contest report [27]. The first image, ds08.tif, is similar to that of [27, Fig. 2] while the second one, ds33.tif, is similar to [27, Fig. 1].

Since there were several classes of graphic object types in these drawings, for which we do not have performance evaluation protocols, we separated bars and arcs from the drawings and used only them in the evaluation. Table 3 shows the performance evaluation results after SPV, after refinement and after arc segmentation.

Examining the performance following the refinement, we see that due to refinement the vector detection rates for both drawings are improved by 1%, while the PRI of the bar part of ds33.tif is decreased by 1%.

The reason for this decrease in the PRI value is that although some missing pixels can be recovered after SPV, the straightening of the lines decreases the number of matching pixels. Arc segmentation greatly improves the vector level performance while the pixel level performance indices may be smaller than before.

As Table 2 shows, SPV is very fast. We have tested SPV with other real life, noisy drawings. On the average, the SPV vectorization takes about 3-5 seconds on an Indy workstation for a 1,000 $\times$ 1,000 pixel size drawing with moderate line densities of about 5%-10%. The average time spent on recovering one vector and one pixel in each drawing are also shown in Table 2. The time/vector factor is larger on high quality drawings, in which the vectors are longer and their number is small. This factor is not very suitable for evaluating the time efficiency of the vectorization algorithm, because the vector attributes, length and width, vary from one drawing to another. However, the time/pixel does not vary significantly from drawing to drawing. In general, the higher the pixel density, the higher is the time/pixel factor, because high density pixel drawings usually contain more line intersections. Drawings with a lot of background "noise", such as the two graphs of Fig. 10 and Fig. 11, yield numerous small vectors whose recovery time is small. We have also tested the relative time spent on the refinement stage. The results, presented in Table 4, show that the refinement time is smaller by a factor of 20 to 70 than the SPV time.

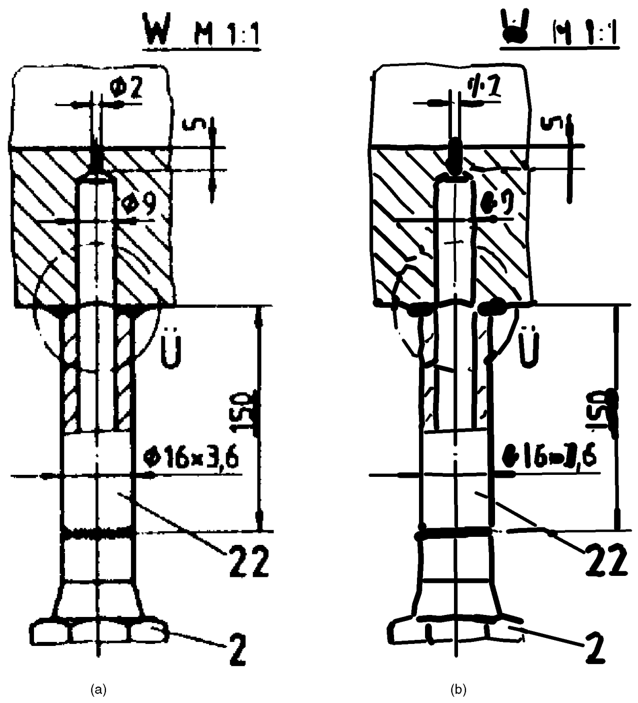


Fig. 12. Result of vectorization by SPV. (a) Original drawing. (b) SPV results.

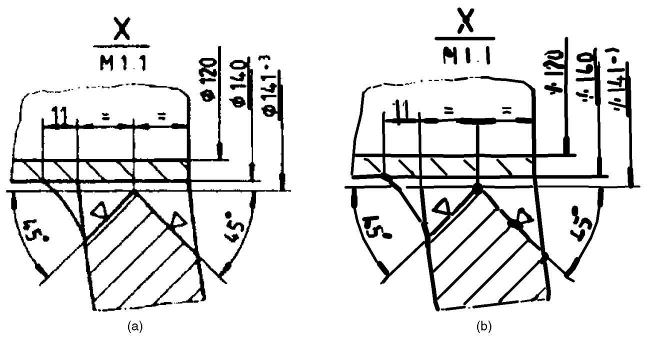


Fig. 13. Result of vectorization by SPV. (a) Original drawing. (b) SPV results.

6 Conclusion

We have presented, demonstrated and evaluated a thinningless vectorization algorithm, the Sparse Pixel Vectorization (SPV), which is an improved version of OZZ. SPV improves the OZZ method in the following three aspects.

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 the scanning is needed and combination of the two passes is avoided. It is therefore faster than OZZ.
3) A junction recovery procedure is introduced wherever a junction is encountered during line tracking.

The prominent feature of the algorithm is that it examines a small portion of the black pixel population. This results in fast vectorization without compromising the recognition quality. It therefore serves as a good basis for higher level object recognition and refinement tasks in line drawings and technical documents in general and in mechanical engineering drawings in particular.

The SPV algorithm is evaluated using the protocol described in [22]. Experimental results on several kinds of drawings with different noise levels and densities have yielded good performance in terms of both time efficiency and shape preservation. SPV can be used for crude vectorization of any class of line drawings and technical documents, such as forms [28]. SPV has two main shortcomings. One is its junction recovery capability. Although SPV can overcome junctions, not all can be overcome. The other is that arcs are vectorized as polylines that are broken at the four special positions. However, this does not hamper the arc segmentation result, as shown in [24]. Overall, SPV's advantages clearly outweigh its disadvantages, and we recommend that it be adopted for use as the vectorization algorithm of choice in graphics recognition systems.

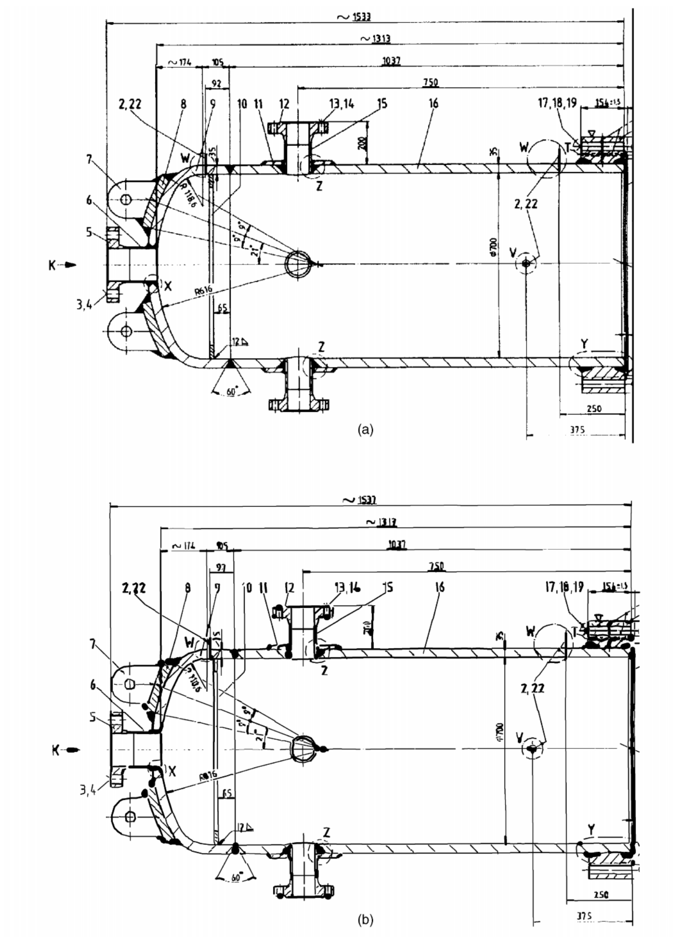


Fig. 14. Result of vectorization by SPV.(a) Original drawing. (b) SPV results.

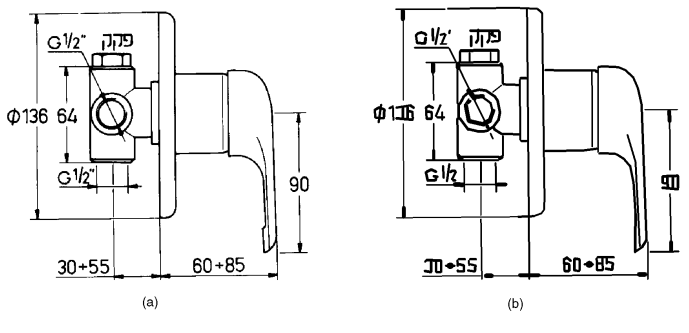


Fig. 15. Result of vectorization by SPV. (a) Original drawing. (b) SPV results.

TABLE 3
PERFORMANCE EVALUATION OF SPV, REFINEMENT, AND ARC SEGMENTATION ON THE TWO CONTEST IMAGES

Performance Indices / Image (part)		After SPV						After Refinement						After Arc Segmentation					
		Pixel Level			Vector Level			Pixel Level			Vector Level			Pixel Level			Vector Level		
		D_p	F_p	PRI	D_v	F_v	VRI	D_p	F_p	PRI	D_v	F_v	VRI	D_p	F_p	PRI	D_v	F_v	VRI
ds08.tif	Bar part	0.99	0.02	0.99	0.77	0.26	0.76	0.99	0.02	0.99	0.78	0.27	0.76						
	Arc part	0.89	0.27	0.81	0.28	0.58	0.35							0.86	0.27	0.79	0.96	0.04	0.96
ds33.tif	Bar part	1.00	0.05	0.98	0.90	0.12	0.89	1.00	0.05	0.97	0.91	0.12	0.89						
	Arc part	0.94	0.33	0.81	0.39	0.23	0.55							0.98	0.31	0.83	0.89	0.11	0.89

The performance evaluation metrics and the evaluation protocol we have devised and used in this work are objective tools that can be applied to quantitatively evaluate any vectorization algorithm. Future work should employ this

TABLE 4
TIME PERFORMANCE OF SPV AND REFINEMENT

Image	Graphic object type	SPV time (seconds)	Refinement time (seconds)
ds08.tif	bars	6.43	0.33
	arcs	7.85	0.11
ds33.tif	bars	1.98	0.05
	arcs	3.51	0.11

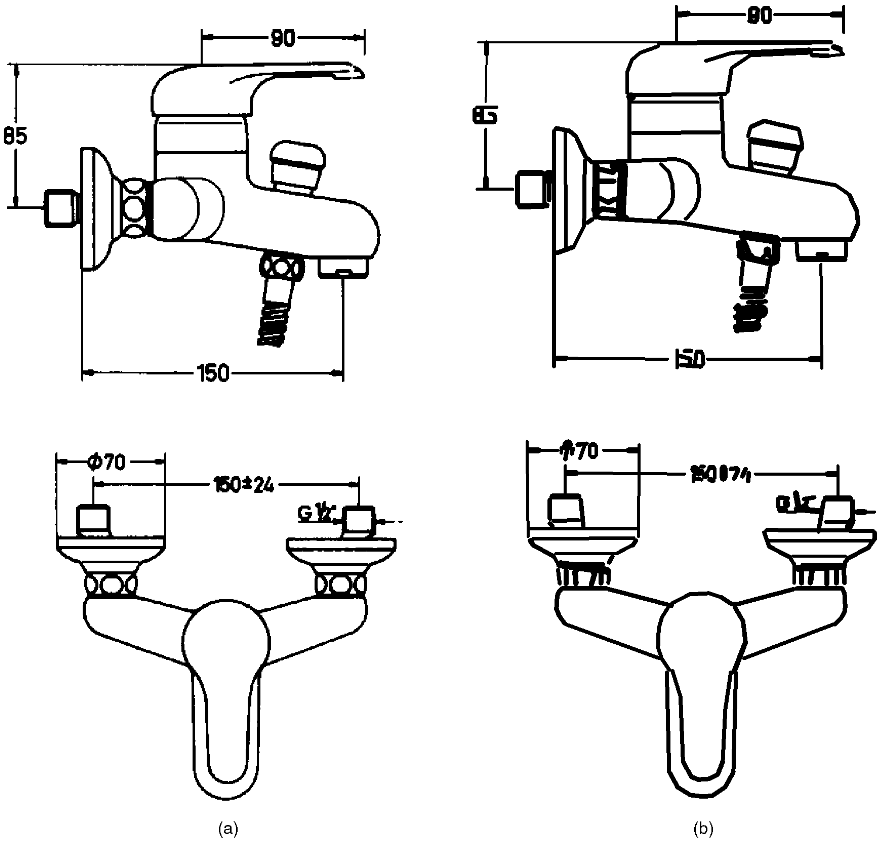


Fig. 16. Result of vectorization by SPV. (a) Original drawing. (b) SPV results.

evaluation protocol on a set of customary vectorization algorithms to obtain an objective assessment of their relative advantages and disadvantages.

REFERENCES

[1] J. Jimenez and J.L. Navalon, "Some Experiments in Image Vectorization," *IBM J. Res. Develop.*, vol. 26, pp. 724-734, 1982.
[2] R.W. Smith, "Computer Processing of Line Images: A Survey," *Pattern Recognition*, vol. 20, no. 1, pp. 7-15, 1987.
[3] H. Tamura, "A Comparison of Line Thinning Algorithms From Digital Geometry Viewpoint." *Proc. Fourth Int'l Joint Conf. Pattern Recognition*, pp. 715-719, Kyoto, Japan, 1978.
[4] C.S. Fahn, J.F. Wang, and J.Y. Lee, "A Topology-Based Component Extractor for Understanding Electronic Circuit Diagrams," *Computer Vision, Graphics and Image Processing*, vol. 44, pp. 119-138, 1988.
[5] R. Kasturi, S.T. Bow, W. El-Masri, J. Shah, J.R. Gattiker, and U.B. Mokate, "A System for Interpretation of Line Drawings," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 12, no. 10, pp. 978-992, Oct. 1990.
[6] V. Nagasamy and N.A. Langrana, "Engineering Drawing Processing and Vectorization System," *Computer Vision, Graphics and Image Processing*, vol. 49, no. 3, pp. 379-397, 1990.
[7] O. Hori and S. Tanigawa, "Raster-to-Vector Conversion by Line Fitting Based on Contours and Skeletons," *Proc. ICDAR93*, pp. 623-626, Tsukuba, Japan, 1993.
[8] L. Boatto et al., "An Interpretation System for Land Register Maps," *IEEE Computer*, vol. 25, no. 7, pp. 25-32, July 1992.
[9] G. Monagan and M. Roosli, "Appropriate Base Representation Using a Run Graph," *Proc. ICDAR93*, pp. 623-626, Tsukuba, Japan, 1993.
[10] X. Lin, S. Shimotsuji, M. Minoh, and T. Sakai, "Efficient Diagram Understanding With Characteristic Pattern Detection," *Computer Vision, Graphics and Image Processing*, vol. 30, pp. 84-106, 1985.
[11] P. Vaxiviere, and K. Tombre, "Celesstin: CAD Conversion of Mechanical Drawings," *IEEE Computer*, vol. 25, no. 7, pp. 46-54, July 1992.
[12] I. Chai and D. Dori, "Orthogonal Zig-Zag: An Efficient Method for Extracting Lines From Engineering Drawings," C. Arcelli, L. P. Cordella and G. Sanniti di Baja, eds., *Visual Form*. New York: Plenum Press, 1992, pp. 127-136.
[13] D. Dori, Y. Liang, and I. Chai, "Spare Pixel Recognition of Primitives in Engineering Drawings," *Machine Vision and Applications*, vol. 6, pp. 69-82, 1993.
[14] D. Dori, "Orthogonal Zig-Zag: An Algorithm for Vectorizing Engineering Drawings Compared With Hough Transform," *Advances in Eng. Software*, vol. 28, no. 1, pp. 11-24, 1997.

[15] A.J. Filipski and R. Flandrena, “Automated Conversion of Engineering Drawings to CAD Form,” *Proc. IEEE*, vol. 80, no. 7, pp. 1,195-1,209, 1992.

[16] M. Roosli and G. Monagan, “Towards a High Quality Vectorization,” *Proc. GREC95*, pp. 44-52, University Park, Pennsylvania, 1995.

[17] W. Liu and D. Dori, “Sparse Pixel Tracking: A Fast Vectorization Algorithm Applied to Engineering Drawings,” *Proc. ICPR96, Volume III: Robotics and Applications*, pp. 808-811, Vienna, 1996.

[18] J. Sklansky and V. Gonzalez, “Fast Polygonal Approximation of Digitized Curves,” *Pattern Recognition*, vol. 12, pp. 327-331, 1980.

[19] W. Liu, D. Dori, L. Tang, and Z. Tang, “Object Recognition in Engineering Drawings Using Planar Indexing,” *Proc. GREC95*, pp. 53-61, University Park, Pennsylvania, 1995.

[20] B. Kong et al., “A Benchmark: Performance Evaluation of Dashed-Line Detection Algorithms,” *Graphics Recognition—Methods and Application*, R. Kasturi and K. Tombre, eds., Lecture Notes in Computer Science, vol. 1,072. New York: Springer, 1996, pp. 270-285.

[21] O. Hori and D. Doermann, “Quantitative Measurement of the Performance of Raster-to-Vector Conversion Algorithms,” R. Kasturi and K. Tombre, eds., *Graphics Recognition—Methods and Application*. Lecture Notes in Computer Science, vol. 1,072. New York: Springer, 1996, pp. 57-68.

[22] W. Liu and D. Dori, “A Protocol for Performance Evaluation of Line Detection Algorithms,” *Machine Vision Applications*, vol. 9, no. 5/6, pp. 240-250, 1997.

[23] I.T. Phillips, J. Liang, A. Chhabra, and R.M. Haralick, “A Performance Evaluation Protocol for Graphics Recognition Systems,” K. Tombre and A. Chhabra, eds., *Graphics Recognition—Algorithms and Systems*. Lecture Notes in Computer Science, vol. 1,389. New York: Springer, 1998, pp. 372-389.

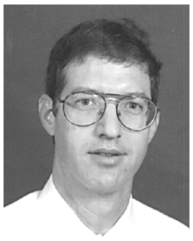
[24] W. Liu and D. Dori, “Incremental Arc Segmentation Algorithm and Its Evaluation,” *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 20, no. 4, pp. 424-431, Apr. 1998.

[25] ftp.technion.ac.il/pub/supported/ie/dori/MDUS/sunmdus.gz.

[26] A. Chhabra and I. Phillips, Web pages for the Second International Graphics Recognition Contest—Raster to Vector Conversion: http://graphics.basit.com/iapr-tc10/contest.html.

[27] A. Chhabra and I. Phillips, “The Second International Graphics Recognition Contest—Raster to Vecter Conversion: A Report,” K. Tombre and A. Chhabra, eds., *Graphics Recognition—Algorithms and Systems*. Lecture Notes in Computer Science, vol. 1,389. New York: Springer, 1998, pp. 390-410.

[28] J.-Y. Yoo, M.-K. Kim, and Y.-B. Kwon, “Information Extraction From a Skewed Form Document in the Presence of Crossing Characters,” K. Tombre and A. Chhabra, eds., *Graphics Recognition—Algorithms and Systems*. Lecture Notes in Computer Science, vol. 1,389. New York: Springer, 1998, pp. 139-148.



Dov Dori received his BSc in industrial engineering and management from the Technion, Israel Institute of Technology, in 1975; the MSc in operations research from Tel Aviv University in 1981; and the PhD in computer science from Weizmann Institute of Science, Rehovot, Israel, in 1988. Between 1987 and 1990, he was assistant professor at the Department of Computer Science, University of Kansas. Between 1996 and 1998, he was head of the Area of Information Systems Engineering. He has been a faculty member at the William Davidson Faculty of Industrial Engineering and Management, Technion, since 1991. His research interests include systems development methodologies, information systems engineering, computer-aided software engineering, and document analysis and recognition. Dr. Dori has developed the Machine Drawing Understanding System (MDUS) and the Object-Process Methodology (OPM). In 1995, he won first place in the Dashed Line Recognition Contest held during the *First IAPR Workshop on Graphics Recognition* at Pennsylvania State University. In 1997, he won the Hershel Rich Technion Innovation Award for the development of OPCAT-Object-Process CASE Tool, which supports OPM. Dr. Dori is associate editor of *IEEE Transactions on Pattern Analysis and Machine Intelligence* and the *International Journal of Document Analysis and Recognition* and is on the editorial board of the *International Journal of Pattern Recognition and Artificial Intelligence*. He is the coeditor of two books and the author of more than 60 journal papers and book chapters and more than 60 conference publications. He is a senior member of IEEE and a member of the IEEE Computer Society, ACM, and IAPR.

W. Liu received his BEngg and MEngg in computer science from the Tsinghua University in 1988 and 1992, respectively, and his PhD in information management engineering from the Technion, Israel Institute of Technology, in 1998. Dr. Liu now is a researcher at Microsoft Research, China. His research interests include automated engineering drawing interpretation, computer vision and pattern recognition, software engineering, object-oriented programming, object-process methodology, artificial intelligence, and computer graphics. Dr. Liu has 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 third prize in the First International Java Programming Contest (ACM Quest for Java 97), sponsored by ACM and IBM in 1997.