Vector-Based Arc Segmentation in the Machine Drawing Understanding System Environment

Dov Dori, *Member, IEEE*

***Abstract*—Arcs are important primitives in engineering drawings. Along with bars, they play a major role in describing both the geometry and the annotation of the object represented in the drawing. Extracting these primitives during the lexical analysis phase is a prerequisite to syntactic and semantic understanding of engineering drawings within the Machine Drawing Understanding System. Bars are detected by the orthogonal zig-zag vectorization algorithm. Some of the detected bars are linear approximations of arcs. As such, they provide the basis for arc segmentation. An arc is detected by finding a chain of bars and a triplet of points along the chain. The arc center is first approximated as the center of mass of the triangle formed by the intersection of the perpendicular bisectors of the chords these points define. The location of the center is refined by recursively finding more such triplets and converging to within no more than a few pixels from the actual arc center after two or three iterations. The high performance of the algorithm, demonstrated on a set of real engineering drawings, is due to the fact that it avoids both raster-to-vector and massive pixel-level operations, as well as any space transformations.**

***Index Terms*—Arc segmentation, engineering drawing understanding, technical documentation automation, sparse-pixel recognition, document analysis and recognition, vectorization, raster-to-vector, Hough transform.**

I. Introduction

A typical line drawing is made up of polygons, circular arcs and other type of lines [1]. Engineering drawings, stored as scanned binary images, are the inputs to the Machine Drawing Understanding System (MDUS) [2] which is based, among other things, on the syntax of dimensions, developed in [3]. The primitive recognition phase of MDUS comprises four algorithms: orthogonal zig-zag (OZZ) for bar detection, perpendicular bisector tracing (PBT) for arc segmentation, self-supervised arrowhead recognition (SAR), and textbox location. This work concentrates on the arc segmentation algorithm.

PBT gets as input bars resulting from the OZZ algorithm [2], [4]. OZZ is a fiber-optic inspired vectorization algorithm that is particularly suitable for extracting bars from scanned, binarized engineering drawings. Since OZZ is the starting point for arc segmentation in MDUS, we describe its main principles briefly. Fig. 1 is an object-process diagram (OPD) which employs the object process analysis (OPA) methodology [5], [6]. It shows that OZZ is an aggregation of four lower-level objects: the sparse screening procedure, the OZZ Procedure, the merging procedure, and the corner correction procedure. The sparse screening procedure is the instrument for the execution of the process sparse screening, which is the first lower-level process within the bar detection process. When a black pixel is encountered, the process sparse screening starts to count the number of pixels in the black run, until enough white pixels are encountered. If the length of the run is large enough, but below a predefined threshold, the process of orthogonal zig-zagging is invoked, otherwise, the process of parallel probing is invoked. The underlying idea of the orthogonal zig-zagging process, which is the heart of the vectorization, is inspired by a light beam conducted by an optic fiber: a one-pixel-wide "ray" travels through a black pixel area, suspected to be a bar, as if the elongated black area were a conducting pipe. The ray trajectory is parallel to the drawing axes, and its course zig-zags orthogonally, changing direction by 90° each time a white area is encountered. Accumulated statistics about the horizontal and vertical sets of black run-lengths, gathered along the beam trajectory, provide data for deciding about the presence of a bar, its endpoints and its width, and enable skipping junctions. The merging procedure then performs the bar merging procedure, in which it tries to merge bars from both the slanted bar list and the parallel bar list, ob-

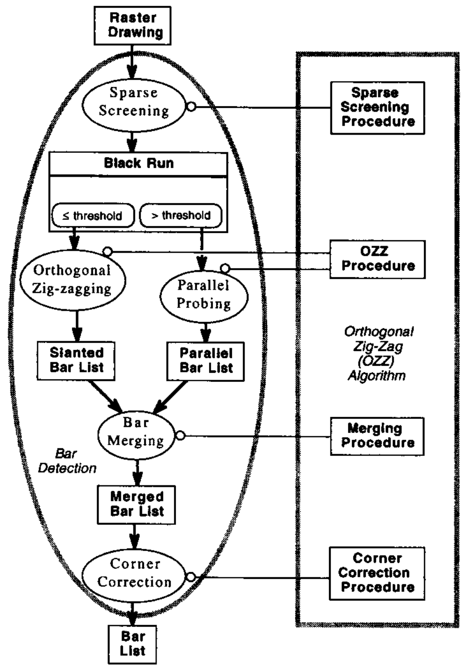


Fig. 1: Object-process diagram of the bar detection phase

Manuscript received Apr. 12, 1994; revised Apr. 4, 1995. Recommended for acceptance by R. Kasturi.

The author is with Information Systems Engineering, Faculty of Industrial Engineering and Management, Technion, Israel Institute of Technology, Haifa 32000, Israel; e-mail: dori@ie.technion.ac.il.

IEEECS Log Number P95101.

tained from the OZZ Procedure. The resulting merged bar list then undergoes a process of corner correction, which fills in missing black pixels in corners. The final outcome is the object bar list. MDUS uses the IGES standard [7] as the format of the output files of each module. Arc segmentation, which is the main theme of this work, is described in detail in the rest of the paper. After it has been completed, bars, which are linear approximations of arcs, are excluded from the IGES file, while the detected arcs are added to the file.

II. Arc Segmentation Methods

Existing methods for arc segmentation can be divided into two groups. The first group includes methods to detect circular objects that are based on Hough transform, while the second group is motivated by the need to recognize objects in a scene. Algorithms in the first group attempt to locate circles in (possibly noisy) images by a certain transformation. Algorithms in the second group are primarily concerned with the extraction of meaningful features from objects by estimating their edge curvature. The features extracted from the object contour can be used for object classification and recognition. In this section we briefly survey several methods from each group and discuss their suitability for the purpose of arc segmentation in engineering drawings.

Hough transform (HT) [11], [12] is a conventional method for object extraction from binary images. HT is normally used for arc segmentation in cases of isolated points that potentially lie on circles or circular arcs. A circle can be described by (1), where (a, b) is the circle center and r is the radius.

$$(x-a)^2 + (y-b)^2 = r^2 \qquad (1)$$

HT uses this equation to map each (x, y) image point into all parameter points which lie on the surface of an inverted right angled cone, whose apex is at $(x, y, 0)$. The circle parameters a, b, and r are identified by the intersection of many conic surfaces, as shown at the top of Fig. 2. This is done by examining the peaks in a 3D accumulator array. This 3D accumulator peak detection is in general too costly to be applicable. To reduce the dimensionality of the problem from 3 to 2, the Adaptive Hough Transform (AHT) method [13] was developed. To locate the circle center, AHT incorporates the constraint that the vectors, which are normal to the circle boundary, must all intersect at the circle center (a, b) [14]. Knowing the image point (x, y) and $\tan\theta$ then leaves only a and b as parameters to be found (see bottom of Fig. 2). Mapping (x, y, θ) into the 2D parameter space (a, b) produces a straight line. The intersection of many of these lines indicates the circle center in the image. The radius of the circle can then be found by histogramming $r = (x-a)^2 + (y-b)^2$ and locating the largest peak in the r histogram. Although AHT converts circle detection into 2D peak detection, it still requires pixel-by-pixel image operations and demands a large memory space to accumulate the frequency of each circle parameter. Locating peaks in accumulators is also a heavy task. To make HT more applicable, other works, including [15], [16], and [17], have attempted to present optimized variations. However, since they are derived from the original HT, they also require considerable time and space.

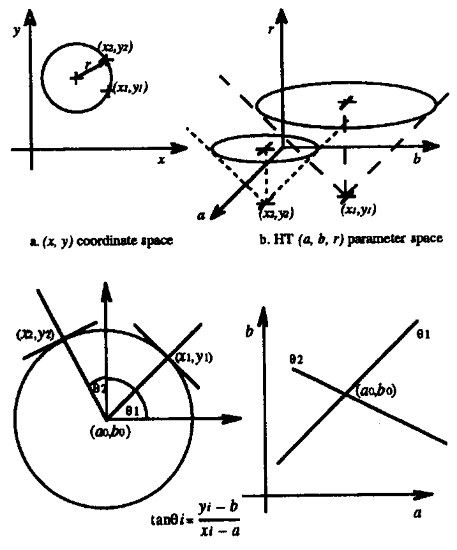


Fig. 2. The coordinate and HT parameter spaces for a circle (top) and finding the circle center by AHT (bottom)

Other arc segmentation methods, including [18], [19], [20], and [21], belong to the curvature estimation group. Motivated by object recognition, the aim of these algorithms is to extract meaningful features from objects by estimating their edge curvature. Whereas Hough-based methods can handle isolated points, the common requirement of curvature estimation algorithms is that the input be a digital curve, i.e., a one-pixel-wide line. This requirement implies a heavy preprocessing phase, such as edge detection (for gray-level images) or thinning (for line drawings). Asada and Brady [18] propose the curvature primal sketch to represent significant changes in curvature along the bounding contour of a planar shape. They define a set of primitive parametrized curvature discontinuities and derive expressions for their convolution with the first and second derivatives of a Gaussian. They interpret the significant changes in curvature at various scales. The input is a bounding contour of the object to be recognized, while the output may be a semantic network or a filtered response graph. Rosin and West [19] describe a method of segmenting curves in images into a combination of circular arcs and straight lines. It is an extension of a method proposed by Lowe [22], which analyzes arbitrary curves and produces a high-level straight line description. In [22], each curve is segmented by splitting it at the maximum deviation from the approximating straight line. The extension in [19] finds the best fit of a combination of arcs and straight lines to the data. The input to the algorithm is a digital line, hence the image must be preprocessed by an edge detector to extract connected pixels or use boundary descriptions from chain-coded binary images. O'Gorman [20], [21] proposes to carry out feature extraction by estimating the curvature along digital lines and determining the features from the curvature plots. For the difference of slopes (DOS) approach, curvature at a point is estimated as the angular difference between the slopes of two line segments fit to the data before and after the point. After finding the curvature of each point in the data, the curvature plot is usually smoothed to reduce noise. The specialized difference of slopes method DOS$^+$ [21] proposes to use a nonzero, small positive gap between the two segments that estimate the lines between which the angle is measured. DOS$^+$ has been shown to be effective for estimating curvature in terms of signal detectability. In [20], two approaches for curvature estimation are compared: the DOS$^+$ method and the Gaussian smoothing of the second derivative. DOS$^+$ was shown to over perform Gaussian Smoothing for small signal angle and high noise. Here, too, the input is assumed to be a chain of data points.

The motivation of arc segmentation in MDUS is to explicitly define where arcs exist in the drawing and determine their parameters, including angle (which may vary from several to 360°), radius, center, endpoints, and, last but not least, line width. Line width is very important in higher-level drawing understanding, because geometry lines are required by drafting standards (ISO and ANSI) to be twice as thick as annotation lines. Hough- based methods may be effective in gray-level images, where circles are sought and noise may be present. On top of their computational complexity, they are not suitable for detecting arcs with small angles, because the peaks that arcs with small angles produce are not high enough to distinguish them from noise. Detected arcs require postprocessing to define their endpoints, because Hough-based algorithms define only the arc center and radius. Finally, the width of each detected arc has to be somehow related to the width of the signal in the transform domain. This does not seem to be a trivial problem, and it is relevant only if the input to the Hough-based algorithm is not a result of a contour extraction preprocessing—thinning or edge detection—which causes all contours and lines to lose their original width and become one-pixel wide. Curvature estimation methods assume that the input is a digital line, implying that the image must undergo some contour extraction preprocessing. Not only does this prerequisite put a heavy computational burden on a system, but it also causes line drawings to lose important information about their original line widths. Since curvature estimation methods are initially driven by the need to extract features that emerge from the combination of straight lines and circular arcs, they are not even concerned with the extraction of the more basic arc parameters center and radius, which Hough-based methods attempt to determine.

In summary, existing arc segmentation methods do not provide adequate means for extracting arcs from engineering drawings. First, none of the methods detects line width. Second, their main focus is either on detecting circles or wide-angle arcs form noisy images or extract features from curvature estimation for object recognition. Rather than using one of the existing methods, we take advantage of the knowledge about the bars detected in the previous Bar Detection phase of MDUS. Some of these bars, which are linear approximations of arcs, provide the basis for an arc segmentation algorithm that is both effective and efficient.

Before describing the details of the algorithm, let us consider an example of arcs segmented by the algorithm. Fig. 3 shows on the left hand side a 300 DPI scan of one orthogonal view taken from an engineering drawing. The right hand side shows a magnified portion of the drawing, in which the detected arcs are drawn in thin dotted lines with the arc centers and edges denoted by little white circles to show the segmentation results. The results are complete and accurate in spite of the presence of many bars crossing the arcs in various directions.

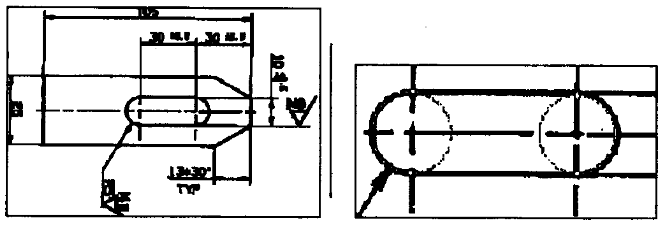


Fig. 3. A drawing scanned at 300 DPI (left) and PBT arc segmentation results.

III. The Perpendicular Bisector Tracing Arc Segmentation Algorithm

The perpendicular bisector tracing (PBT) arc segmentation algorithm starts with clustering bars that potentially approximate arcs into bar chains. The arc center is refined by recursively finding triples of points along the arc and computing the intersection of the corresponding perpendicular bisectors of the chords these points define. The details of the algorithm follow.

A. Clustering Candidate Bars

Using the extracted bar list resulting from OZZ, we can view the image as approximated by bars of known locations and widths. Fig. 4 is another scan of an engineering drawing. Bars extracted from this drawing are shown in Fig. 5. The left hand side of Fig. 6 shows the portion of interest of Fig. 5, which contains arcs. The right hand side is a magnification of the same portion in Fig. 8, after OZZ has been applied to extract bars. This is the starting point for PBT.

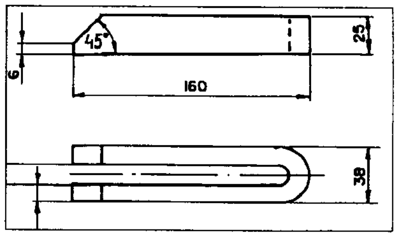


Fig. 4. A portion of the drawing "Horseshoe 1" scanned at 300 DPI.

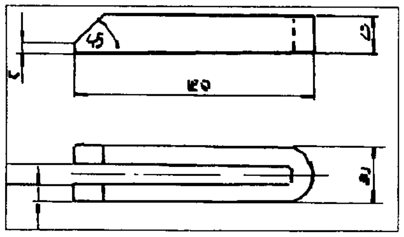


Fig. 5. Bars segmented from "Horseshoe 1" as a result of the OZZ algorithm.

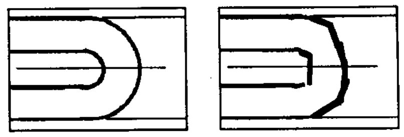


Fig. 6. Arcs before and after bar extraction by OZZ.

Tracing a chain of consecutive bars gives a preliminary clue for the potential existence of an arc and its two endpoints. PBT therefore first examines the list of bars extracted by OZZ. Minimal and maximal length thresholds can optionally be applied during this examination to filter out bars that are unlikely to be on an arc. These two thresholds are closely related to the minimal and maximal radii that can be detected. If no limit is put on the upper bound of the length of the bars which can be candidates for chains, then we should be able to detect arcs of any radius, provided at least two bars were detected from the chain. This, in turn, depends on two things: the magnitude of the arc angle and the width of the line used to draw the angle. A wide angle ensures that it be approximated by at least two bars, because OZZ does not allow that more than 20% of the pixels traversed along the medial axis of any bar be white. For the very same reason, an arc with a given radius and angle, which is drawn in a thick line, is less likely than an arc with the same radius and angle, but with a thinner line, to be approximated by at least two bars. Hence, the wider the angle and the thinner the line, the more likely it is that the arc will be detected. As for the lower bound on the length of candidate bars, it, too, can be ignored if we wish to detect arcs of small radii. In any case, OZZ already filters out bars that are shorter than a minimal bar length parameter. Filtering bars through the upper and/or lower bound threshold indeed speeds up the search for chains, but it should only be applied if we can be certain that arcs above or below some radius cannot be present in the type of drawings we are processing. Those bars that have close endpoints and similar width are clustered into bar-chains. Each bar-chain is a candidate for representing an arc in the raster drawing. For each such chain, the two extreme edges, belonging to the two extreme bars in the chain, are defined as a pair of preliminary arc endpoints. Since OZZ provides linear approximations for arcs, each preliminary endpoint is just an estimation of the arc edge, used to start up the arc segmentation process.

B. Finding the Arc Curvature Direction

If an arc of less than 360° exists, then following the arc can be done uniquely either clockwise or counterclockwise. For example, in Fig. 7, going from A to B along the bar-chain can be done only clockwise. To determine this direction, let $\mathbf{V}_i = (Dy_i, Dx_i)$ and $\mathbf{V}_{i+1} = (Dy_{i+1}, Dx_{i+1})$ be two consecutive vectors in the bar-chain, where Dy_i and Dx_i are the y and x components of $\mathbf{V}_i$, respectively. Let α_i and α_{i+1} be the angles of inclination of $\mathbf{V}_i$ and $\mathbf{V}_{i+1}$, respectively. We then have:

$$\tan\alpha_i = Dy_i/Dx_i \tag{2}$$

$$\tan\alpha_{i+1} = Dy_{i+1}/Dx_{i+1} \tag{3}$$

$$\tan\theta_i = \tan(\alpha_{i+1} - \alpha_i) = (\tan\alpha_{i+1} - \tan\alpha_i)/(1 - \tan\alpha_i \tan\alpha_{i+1}) \tag{4}$$

Substituting (2) and (3) into (4), the angle θ_i between $\mathbf{V}_i$ and $\mathbf{V}_{i+1}$ is obtained in (5).

$$\theta_i = \arctan[(Dy_{i+1}\, Dx_i - Dy_i\, Dx_{i+1})/(Dy_i\, Dx_{i+1} + Dy_{i+1}\, Dy_i)] \tag{5}$$

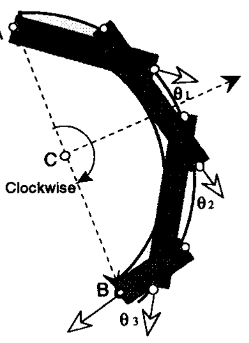


Fig. 7. Determination of the trace direction.

To determine the arc direction, we compute the partial sums in (6), where n is the number of bars in the bar-chain.

$$\alpha_k = \sum_{i=1}^{k} \theta_i \qquad \text{for } k = 1, \ldots, n-1 \tag{6}$$

As long as the absolute value of α_k keeps growing along with k, the curvature is consistent and the summation continues. Reduction in this absolute value implies that the curvature direction has changed. When this occurs, a new endpoint is taken for the bar-chain, while the remaining $(n - k)$ bars in the list form a new bar-chain. The trace direction is then determined by the sign of α_k.

C. Finding a Third Point on the Arc

Based on the Euclidean geometry theorem, which states that three points on a plane uniquely determine a circle, we wish to find three points to start the arc detection process. Two preliminary arc endpoints are provided by the bar-chain edges. Using these two points, the algorithm tries to locate a third point along the presumed arc. The five steps of a naive version of this process are shown and explained in Fig. 8. A perpendicular bisector is constructed for the segment AB and tracing is done along it. When the bisector encounters a run of black pixels, the potential arc is assumed to be met. The point F returned by this procedure lies midway between D, the arc entry point, and E, the arc exit point.

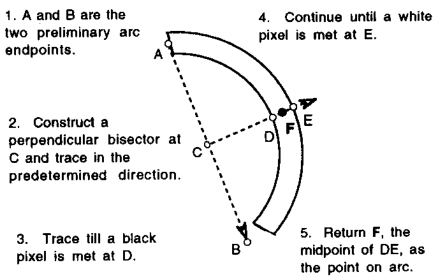

1. A and B are the two preliminary arc endpoints.
4. Continue until a white pixel is met at E.
A
2. Construct a perpendicular bisector at C and trace in the predetermined direction.
C
D
F
E
3. Trace till a black pixel is met at D.
B
5. Return F, the midpoint of DE, as the point on arc.
Fig. 8. Detecting the third point along the candidate arc.

Fig. 9 shows the location of the third point originating from the two initial, inexact arc endpoints A and B, and the two exact arc endpoints, C and D. In spite of the difference between the correct trace direction and the one actually pursued (due to the incorrect estimation of the preliminary endpoints A and B rather than C and D), for arcs up to 180°, the traced third arc point E is close to arc point F, found for the exact arc endpoints C and D. To prove this, let us first assume that A and C are close enough to be considered the same point, such that the large distance is only between B and D. The resulting distance between E and F for arcs up to 180° can be no more than half the distance between B and D. To see why, consider a 180° arc, such that AB is an approximate diameter, and CD is a precise one. Let O be the middle of CD (the arc center). Triangle DBC is about similar to triangle OEF and about twice as big, because CD is the diameter and OF is the radius. Therefore, EF = 0.5BD. For arcs less than 180°, the small triangle OEF is smaller than half the big one, DBC, meaning that E is closer to F more than 0.5BD. When we remove the assumption that C and A are very close, but assume instead the more relaxed assumption that points A and B are on the same side of chord CD, the situation improves, because the big triangle is now bigger than before. Since the small triangle is about the same size relative to the previous situation, the similarity ratio between the two triangles is now larger, forcing the distance between E and F to be smaller.

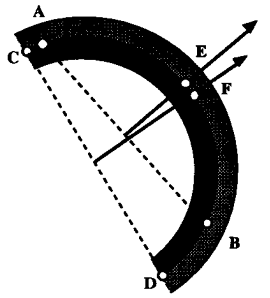


Fig. 9. The location of the third point (E and F) originating from inexact and exact (AB and CD) arc endpoints, respectively.

D. Arc Endpoint Refinement for One-Bar Chains

A common situation in engineering drawings is demonstrated in Fig. 10: For small, 90° arcs, in which the radius-to-width ratio of the arc is not large enough, OZZ is likely to extract only one bar from the arc area. If the tangent bars are long enough to exceed the length threshold for being candidate members of an arc bar-chain, as in Fig. 10, the bar-chain would consist of just one bar. This situation occurs either when the arc has a low radius/width ratio, or if the preliminary arc endpoints A and B in Fig. 10 are too remote from the actual endpoints of the arc. In this case, the naive version of the algorithm, which is based on chain determination, would fail to detect the arc.

For this situation, we need an improved approximation of the two preliminary arc endpoints. This is done in a procedure based on the observation that OZZ detects straight bars more accurately than bars that approximate arcs. Due to this, the endpoints C and D of the long bars tangent to the arc mark the arc endpoints more accurately than A and B. Accordingly, C and D replace A and B as the preliminary endpoint candidates. Next, line CD is stretched in two directions beyond its two endpoints C and D, till either a white-pixel area is found, or a threshold is exceeded. Then, from the two new extreme points E and F of the stretched line, black pixels are searched to both directions perpendicular to line CD. As demonstrated in Fig. 10, this search yields points G and H. The trace direction is then the reverse of the direction of vector from E to G and it yields the arc point L. The two other arc endpoints are M and N—the middle points of EG and FH, respectively. If the radius is even smaller, and especially if the line is relatively thick, it may be the case that the arc is not approximated even by one bar. Instead, the two bars, which are supposed to be perpendicular, are shifted toward each other such that they form an angle slightly larger than 90°. A situation like this can be detected already when trying to perform the corner correction procedure within the OZZ Algorithm by observing at least one of the following two phenomena:

1) The angle between the two bars is somewhat larger than 90°, which is very uncommon in engineering drawings (an angle is normally either 90° or else significantly different than 90°, e.g., 60° or 120°).
2) The square region where the two bars are expected to meet each other does not contain enough black pixels, as it should if the two lines form a real corner.

The current version of OZZ does not support this refinement.

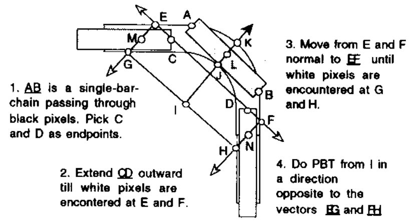

E
A
M
K
C
G
L
J
1. AB is a single-bar-chain passing through black pixels. Pick C and D as endpoints.
I
D
B
F
3. Move from E and F normal to EF until white pixels are encountered at G and H.
H
N
2. Extend CD outward till white pixels are encontered at E and F.
4. Do PBT from I in a direction opposite to the vectors EG and FH

Fig. 10. Finding three preliminary points for a 90° arc connected to two long tangent bars.

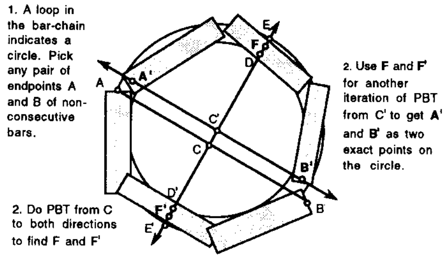

1. A loop in the bar-chain indicates a circle. Pick any pair of endpoints A and B of non-consecutive bars.
E
F
D
A
A'
C'
C
B'
D'
F'
E'
B
2. Use F and F' for another iteration of PBT from C' to get A' and B' as two exact points on the circle.
2. Do PBT from C to both directions to find F and F'

Fig. 11. Detection of points on a circle.

E. Circle Detection

A circle is detected if a loop is found while examining the bar chain. Since a circle has no endpoints, we select any two endpoints A and B, which belong, respectively, to any two non-consecutive bars in the bar-chain (see Fig. 14). Starting from C, the middle point of AB, PBT is carried out to both directions perpendicular to AB. Since tracing to both directions will encounter points on the circle, it is no longer necessary to determine the trace direction first. The two points found serve as two new arc points for a second application of PBT, which is done in the same way as the detection of an open arc, except that the trace direction is arbitrary.

F. Recursive Arc Center Location

Having found three preliminary arc points, a preliminary center of the arc, O_0 in Fig. 12, is computed as the intersection of the two perpendicular bisectors of the segments AC and BC. To verify that the bar chain indeed approximates an arc and to refine the location of its center if it indeed exists, we use (A, C) and (B, C) as two new endpoint pairs. Application of PBT to each one of these two point pairs yields the two new arc points D and E, respectively. The two point triples (A, C, D) and (B, C, E) are used to compute two more arc center estimations, O_1 and O_2, respectively. The distance differences among O_0, O_1, and O_2 are checked and compared to a predetermined center dispersion threshold.

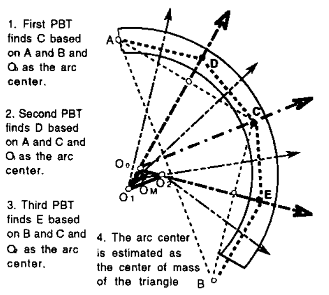

1. First PBT finds C based on A and B and O_0 as the arc center.
2. Second PBT finds D based on A and C and O_1 as the arc center.
3. Third PBT finds E based on B and C and O_2 as the arc center.
4. The arc center is estimated as the center of mass of the triangle
A
D
C
E
B
O_0
O_1
O_M
O_2

Fig. 12. The first iteration of the recursive arc center computation.

If the center estimates of the potential arc are remote from each other by more than the center dispersion threshold, the conclusion is that the bar chain does not represent an arc, and further checking of this chain is aborted. Otherwise, it is likely that the chain is indeed a result of a linear approximation of an arc. This is due to the fact that if A and B were wrong in the first place, i.e., if they were not approximate arc endpoints, the probability that O_0, (obtained from A, B and C), O_1 (obtained from A, C, and D), and O_2 (obtained from B, C, and E), would be close enough to each other by pure chance is very low. Hence, if the dispersion threshold criterion is met, the center of mass O_A of the triangle formed by O_0, O_1, and O_2 is returned as the first approximation of the detected arc center and the first iteration of the recursion ends. If, however, the chain is not a result of an arc, and O_0, O_1, and O_2 just happen by chance to be close to each other, the triangle mass center found in this iteration and those found in subsequent iterations will not converge as they should in case of a circular arc. For the second iteration, since points D and E in Fig. 9 are likely to be more precisely located on the arc than points A and B (which are just bar-chain endpoints), the former pair is taken as the new endpoints, and the process illustrated in Fig. 9 is performed recursively. The halting condition is either the satisfaction of an accuracy criterion for two consecutive detected centers, or proximity of the two newly detected arc points. The recursion depth depends on the radius/width ratio and on the curvature of the arc.

Normally, no more than two to three iterations are required to obtain accuracy of the center location within few pixels. The reason for this is that in the ideal case, any triple of points on a circular arc should yield exactly the same center. As we have proven in the section on finding the third point on the arc (see Fig. 9), locating the third point is at least twice as accurate as the location of the less accurate of the two approximated arc endpoints. Hence, the center location accuracy in each iteration improves on that in the preceding iteration by a factor of at least 2. Thus, after three iterations we improve the accuracy of the center location by a factor of at least 8. If the initial approximation of the arc center was, for example, 50 pixels away from the actual one, the center location after three iterations will be only 6 pixels or less away form the real center location. In practice, if the original arc is accurate and not very thick, the improvement rate is even quicker, because the factor 2 improvement per iteration is for 180° arcs, while most arcs are less than that.

Arcs in engineering drawings are frequently accompanied by annotation. In particular, they may end with one or two arrowheads, as in the 45° dimension-set in Fig. 4, or interleave with center (dash-dotted) lines, as in Fig. 3. Therefore, when the PBT algorithm performs tracing to find a point on the arc, it is not sufficient to simply take the first black pixel encountered. Rather, the pixels along the trace path may be black before the arc is actually starting to be crossed, or even right from the beginning of the tracing, as shown in Fig. 13. Hence, the non-naive version of PBT checks the width of the black area traversed to find the entry and exit points of the arc. AB, CD, EF, GH, IJ, and KL are short line segments within black-pixel area perpendicular to the trace direction. Before the trace enters the arc, there are at most little length differences among the segments KL, IJ, and GH. After entering the arc, however, there is an abrupt increase in the width measured perpendicular to the trace direction (line EF in Fig. 13). The arc entry point is the middle of EF. From this point on, the width starts to decrease, until it levels again at AB, whose middle is the arc exit point. The third point in the triplet (in addition to M and N) is in the middle between the entry and exit points, shown in Fig. 13.

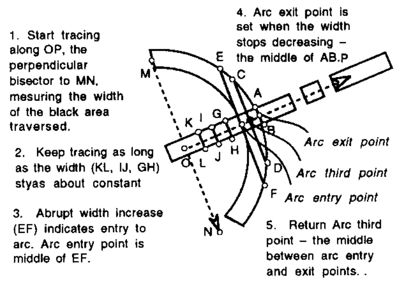

1. Start tracing along OP, the perpendicular bisector to MN, mesuring the width of the black area traversed.
2. Keep tracing as long as the width (KL, IJ, GH) styas about constant
3. Abrupt width increase (EF) indicates entry to arc. Arc entry point is middle of EF.
4. Arc exit point is set when the width stops decreasing – the middle of AB.P
5. Return Arc third point – the middle between arc entry and exit points. .
Arc exit point
Arc third point
Arc entry point
M
E
C
A
K
I
G
B
O
L
J
H
D
F
N

Fig. 13. Finding the third point on arc when tracing is done within a black area.

G. Continuity Preservation

Since the arc detection starts with a pair of approximated endpoints, the correct positions of the two endpoints of the detected arc should be recalculated after its center and radius have been obtained. Continuity is a very useful feature in engineering drawings, and it may be helpful in determining the arc endpoints. As Fig. 17 demonstrates, each one of the four horizontal bars connected to the edges of the two 180° arcs should be tangent to the arc at the point where the bar meets the arc. However, as we have learned from experience, when continuity preservation is applied for the sake of arc endpoint determination without additional considerations, it can potentially lead to results that are worse than those obtained without requiring the preservation of continuity. The reasons for this are that first, not every arc is continued by a bar which is tangent to the arc, so enforcing continuity without applying additional checks can lead to severe distortions. Second, even if continuity should be preserved and has to be restored, a decision must be made as to whether to fix the meeting point at the edge of the bar or at the edge of the arc or somewhere in between. When we applied continuity while stipulating that the arc edge be the fixed point, the right edge of the bottom bar of Figs. 14 and 16 was raised relative to the left edge, because the detected arc lower edge was higher than the detected bar right edge. This caused the bottom bar to lose its parallelism to the other three bars, making the overall result less accurate and visibly less pleasing. On the other hand, deciding that the bar edge be the fixed point is not always the right thing to do either, because this may require to change the radius of the arc, such that it may deviate from the arc in the original raster drawing. This deviation, in turn, may potentially be corrected by other operations such as a slight change in the location of the arc center. In other drawings, the continuity preservation requirement caused severe distortions of previously obtained reasonable results.

These consequences imply, that to take advantage of continuity, such that its benefits outweigh potential distortions or departures from the original drawing, a sophisticated decision mechanism should be developed. This mechanism must contain rules pertaining to such considerations as parallelism preservation and the extent of deviation of bar and arc endpoints and orientations from the original raster file, below which it is likely that continuity was meant to be present in the first place. Fig. 17 shows the detected circles on which the arcs lie, drawn at a width of one pixel on the original raster drawing. The demonstrated detection accuracy of the arcs is high, even though the preliminary endpoints were quite off, as can be deduced from the corresponding bar-chains shown in Fig. 8. Note, however, that the two arc centers C_1 and C_2 do not quite coincide as they should. The decision as to whether or not make these two arcs be concentric is another example for high-level corrections, similar to those suggested by continuity preservation, that can be made only after taking into account a host of geometric, contextual and semantic considerations.

H. The Effect of Small Holes and Islands on Arc Detection

The input for PBT is the list of bars obtained from OZZ. OZZ has a "white noise" parameter, which denotes a certain, small number of white pixels that must be counted by OZZ when traversing through a black pixel area before it concludes that the edge of the line has been encountered. Using this parameter, even if the line has small holes, or even breaks that cause discontinuities, it is still detected as one long bar, provided that the original width of the line does not change significantly. This improves the detection accuracy of bars in general and of those that approximate arcs in particular. The same parameter is used by PBT to prevent it from prematurely concluding that the arc exit point has been reached or from prematurely aborting the search for the arc entry point. A similar "black noise" parameter is used by PBT when traversing a white area to find a third point on the arc, as explained in Fig. 13. It functions to prevent noise that looks like small islands of black pixels, caused due to crossing lines, dust in the scanning, small ink stains, etc., from misleading the search for the arc entry point.

I. Merging Multiple Arcs

Two concentric arcs are quite common, as Fig. 14 shows. A necessary condition for concentric arcs to be identical is that their radii and endpoints are close enough. As is the case with bars in OZZ, a single arc in a drawing may sometimes be detected more than once, due to some particular bar segmentation by OZZ, which results in more than one bar chain for the same arc. A case of this nature is shown in Fig. 15, where two arcs with different radii and centers—ADB and CDE—are detected for the same actual arc. The reason for getting more than one bar chain is explained in the next section. Every pair of detected arcs is checked for possible overlapping. This is done by comparing the values of the arc parameters—center, radius, and the two endpoints of each arc—to the corresponding values of the other arc. Merging two such arcs is done in three situations. The first situation occurs when the centers, radii and the two endpoints of each arc is close enough (relative to a proximity parameter) to the corresponding points of the other arc. In this case, the merged arc center, radius and endpoints are simply the averages of the corresponding points of the two detected arcs. The second situation is when three selected points of one arc, two of which are the arc endpoints, are located within a proximity parameter from the medial axis of the other arc. In this case, the smaller arc is considered to be contained within the larger one, and is discarded. This is the case described at the top part of Fig. 15. The third situation occurs when two arcs partially overlap. This can be detected by noting that two arcs have similar radii and centers, while each arc has exactly one endpoint that lies close enough to the other arc. In this case, the extreme arc endpoint of each one of the two arcs, which lies outside the other arc, serves as the bar-chain endpoint and is input to a new application of PBT.

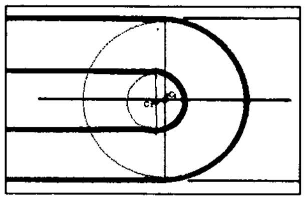


Fig. 14. The two 180° almost concentric geometry arcs detected in "Horseshoe 1."

The proximity parameter, used to determine whether a point on one arc is "close enough" to the other arc should not be too large, as this may cause two concentric circles or arcs, such as the ones denoting threading, to be erroneously merged. The arcs may also be concentric and with the same radius, but still non-overlapping. This is possible in the case of a dashed arc or a dashed circle. When such an arrangement is found, it may serve as a starting point for detection of dashed arc or circle. Appendices A through D provide an annotated C code of the main routines of the PBT algorithm.

IV. Artificial Line Drawing Tests

To further evaluate the performance of the PBT algorithm, we examine artificial line drawings. One such drawing is depicted at the top of Fig. 15, where two arcs were detected, where in fact only one arc is present.

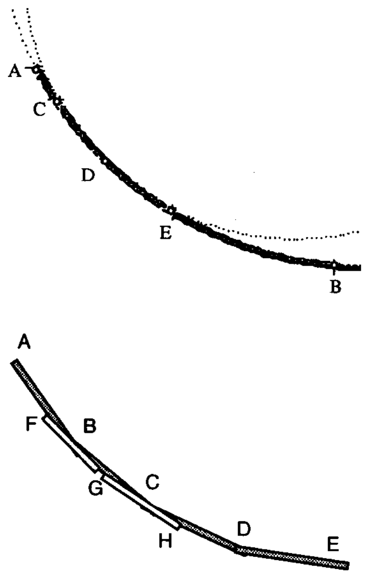


Fig. 15. An arc which is detected as two arcs (top) and the two bar chains which are the reason for this multiple arc detection (bottom).

As the bottom part of Fig. 15 shows, a situation like this can occur when OZZ approximates the same arc by overlapping bars, which lie in such an arrangement that when they are combined into bar chains, two bar chains result. One chain is made of the four bars AB, BC, CD, and DE, and the other-of the two bars FG and GH. Even though OZZ has a bar merging routine, it did not merge the overlapping bars, because the merging condition for two partially overlapping bars, such as AB and FG, is that at least one endpoint of each bar be within the rectangular area of the other bar. Hence, while endpoint B of bar AB, for example, lies within the area of bar FG, endpoint F of FG does not lie within the area of bar AB, so the bar merging condition is not met. The bars FG and GH cannot be part of the chain which starts with bar AB either, because the distances from F, G and H to any one of the points A, B, C, D, E, or F exceed the chain-endpoint distance threshold. Thus, two bar chains are formed from the same actual arc, leading to two detected arcs. These two arcs are merged in the arc merging routine, resulting the elimination of the smaller arc CDE.

Another artificial line drawing appears on the left hand side of Fig. 16. It is a square with four rounded, equal radii arcs of 90° each. All four arcs in this drawing were successfully segmented by PBT, as can be seen on the right hand side of Fig. 16. The edges and centers of the arcs are marked with circles and crosses. The interesting thing to note here is that even though the original radii are the same and the arcs were detected with the correct 90° angle, each pair of arcs on the diagonal of the square was found to have a slightly different radius. The reason for this is that the lines of the rounded rectangle are relatively thick, making the linear approximation obtained by OZZ different for each pair of arcs due to the sparse screening progression of OZZ from left to right and from top to bottom. This, in turn, results in a different location of the centers found by PBT for each pair of arcs.

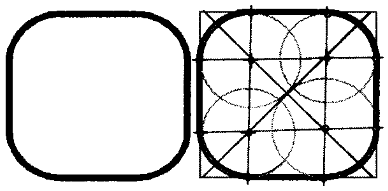


Fig. 16. A square with rounded corners of equal radii and the resulting detected arcs.

Fig. 17a is an artificial drawing containing eight pairs of arcs with different radii, orientations and line widths that are touching and curved in opposite directions. Fig. 17b shows the PBT results obtained without tuning any one of the algorithm parameters. Most of the arcs were detected, although some have a discontinuity around the location where they are supposed to meet. The reason for the discontinuity is that around the point of inclination, where the curvature direction is inverted, OZZ detected a relatively long bar, which exceeded the upper bound on the length of the bars that are candidates for arc chains. Other arcs, especially those with thick lines, were not completely discovered either. Here, OZZ failed to recover some bars because they exceeded the maximal line width parameter. It should be noted that a situation like this is very rare in real engineering drawings, because this feature is not useful and hard to manufacture. Thus, although the algorithm was not designed to handle such interactions among arcs, it still managed to segment most of the arcs arranged in this way.

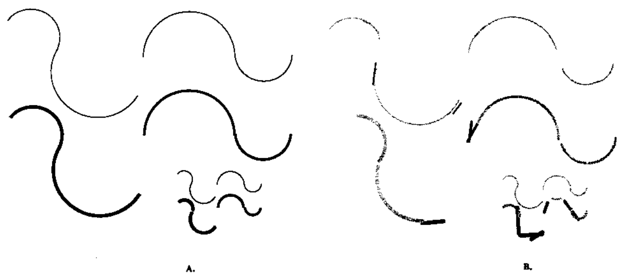


Fig. 17. Touching inverted arcs (a) and the result of the arc detection (b).

V. Complexity Considerations

The complexity of forming all candidate bar chains is $O(n^2)$, where n is the number of bars extracted by OZZ. Since n is much smaller than the number of pixels in the drawing, this is a reasonable complexity. Having found the bar chains, finding the arcs adds only a constant amount of time, because, as argued, the number of iterations in the recursive procedure that finds the arc center never exceeds four in practice. From a memory point of view, the algorithm has very modest requirements. It only needs to store a few intermediate traced points for computations. Since the algorithm is not related to any variant of the Hough transform, or any other transformation that requires massive pixel operations, it is efficient, as it has to examine only a small portion of the drawing pixel population. For example, the drawing "Horseshoe 1" in Fig. 3, whose size is about 1.35MB (1,500 × 900 pixels), required about 6 seconds to extract the arcs running on a DECSYSTEM 5400 RISC machine. This time includes the generation of bar chains, the determination for each bar chain of whether or not it approximates an arc, conversion of the appropriate bar chains into arcs, and updating the IGES file, such that the detected arcs are added and the bars that approximated those arcs are removed. Since the input is the IGES file, which already contains bars, there is no preprocessing involved. PBT does not operate directly on the entire bitmap, which, for real applications, may be in the order of 10^8 pixels. As noted, the complexity of forming all bar chains is $O(n^2)$, where n is the number of bars extracted by OZZ. As the size of the drawing grows, one may expect the number of bars extracted by OZZ to grow as well. However, the number of bars is not strictly directly proportional to the drawing size, because as the drawing area grows, there are more contiguous white areas between the orthogonal views.

The algorithm can be made more efficient by simple improvements, such as the application of more stringent upper and lower bar length thresholds to exclude unlikely bars from being members in arc chains. However, each such improvement must be carefully tested on a large sample of drawings to ensure that the increase in the algorithm efficiency (speed of execution) does not hamper its effectiveness (arc segmentation rate and accuracy). For large drawings, the $O(n^2)$ complexity can be reduced significantly by dividing the drawing area into squares, the size of which may be chosen contextually to optimize the computational load. On the average, the side of each such square will be something like 2.2D, where D is the expected diameter of the circle on which an arc with the maximal radius lies. There will be a 10% horizontal or vertical overlap between any two adjacent squares to prevent missing an arc which is exactly on the border. Each bar will be assigned into one of these squares if its two edges are within the square. If each edge of the bar is in a different square, then the bar will be listed in both squares. The complexity of assigning bars into squares is $O(n)$. The search for bar chains will now be restricted to bars that are listed in the same square. Assuming a homogeneous distribution of bars across squares and a ratio of 1:10 between 2D and the side of a large drawing, the number of squares is a drawing will be 100, so the average number of bars in each square will be $0.01n$, and the complexity will be reduced by a factor of 10,000 within each square and by 100 for the entire drawing. This surprisingly high speedup is obtained due to the fact that no time is wasted in checking whether any two bars, that are too far away from each other, may potentially generate a chain. This speedup factor of 100 assumes that the bars are evenly distributed throughout the drawing plane, which, as noted, is usually not the case. Assuming that all the n bars have a uniform distribution over just a quarter of the drawing plane (i.e., 25 of the 100 squares), the speedup factor will still be (0.04) – 2/25 = 25. There is, however, an $O(n)$ overhead in classifying the bars into the squares and in checking for multiply detected arcs, but it seems that for a ratio of at least 1:4 between 2D and the side of the drawing, this method should yield considerable speedup.

VI. Experimental Results

Limited by scanner size and memory limitations, we have tested about 25 real engineering drawings, or portions of drawings, of size A4. This section presents graphically arcs segmented from a sample of these engineering drawings. Fig. 18 shows a drawing "Horseshoe 2" and a magnified portion of the two 1800 arcs, as detected by the PBT algorithm. The larger arc is slightly larger than the one detected, causing its center to be shifted to the right. An improved version of PBT should remedy this by a post-processing step that checks whether the arc passes entirely through a black area of approximately constant width. If not, the post-processing might try to improve the result by slightly changing the radius and/or the center.

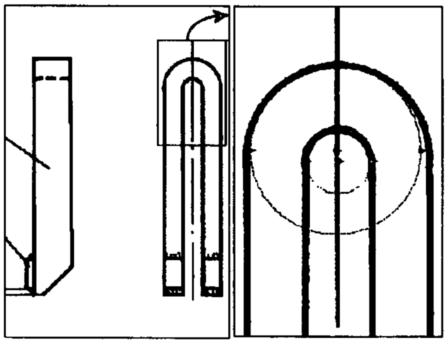


Fig. 18. "Horseshoe 2" (left) and a magnified portion showing the detected arcs with their centers and endpoints.

Fig. 19 shows the drawing "Horseshoe 1" and a magnified portion of the 45° dimension-set arc, accurately detected by the algorithm. Note that the presence of the two arrowheads at the arc edges did not have any adverse effect on the arc segmentation accuracy. In the drawing "Base-side view" (Fig. 20), an arc and a circle were correctly detected.

As can be seen, the arc is correctly defined to be somewhat larger than 90° due to the fact that the tangent bars are oriented with respect to each other in an angle which is less than 90°.

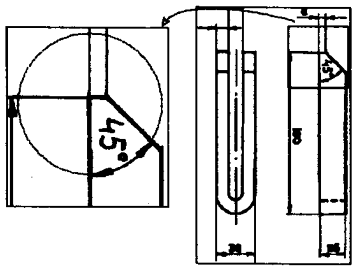


Fig. 19. "Horseshoe 1" (right) and a magnified portion showing the detected arc with its center endpoints.

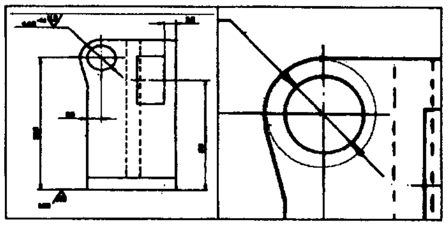


Fig. 20. "Base-side view" (left) and a magnified portion showing the two detected arcs with their centers and endpoints.

VII. Discussion and Conclusion

An algorithm for segmenting arcs from mechanical engineering drawings—perpendicular bisector tracing (PBT)—has been presented and demonstrated. Suited to the environment of the Machine Drawing Understanding System, it is capable of detecting arcs that are interleaved with other graphic objects, such as center (dash-dotted) lines and arrowheads. In addition to the arc endpoints and center, the algorithm finds the arc width—an important parameter for higher level understanding of engineering drawings. The complexity of the algorithm is not directly related to the size of the input image. It is only affected by the number of arcs, their width and curvature, and the number of bars detected by the previous bar extraction step. The ultimate goal of the Machine Drawing Understanding System is to obtain a level of engineering drawing understanding that approaches that of trained humans. This poses stringent demands on detection accuracy and efficiency of primitives in general and arcs in particular. The performance of PBT in terms of accuracy and efficiency and the way it is integrated within the MDUS system makes it a suitable algorithm for this purpose.

Acknowledgments

This research was supported by the Fund for the Promotion of Research at the Technion. The author wishes to thank the anonymous referees for their patient reading and valuable comments.

References

[1] 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. Pattern Analysis and Machine Intelligence*, vol. 12, no. 10. Oct. 1990.

[2] D. Dori, Y. Liang, J. Dowel, and I. Chai, "Sparse pixel recognition of primitives in engineering drawings," *Machine Vision and Applications*, vol. 6, pp. 69-82, 1993.

[3] D. Dori, "A syntactic/geometric approach to recognition of dimensions in engineering machine drawings," *Computer Vision, Graphics, and Image Processing*, vol. 47, pp. 1-21, 1989.

[4] I. Chai and D. Dori, "Orthogonal zig-zag: An efficient method for extracting bars in engineering drawings," *Visual Form*, C. Arcelli, L.P. Cordella, and G. Sanniti di Baja, eds. New York and London: Plenum Press, pp. 127-136, 1992.

[5] D. Dori, "Object-process analysis: Maintaining the balance between system structure and behavior," *J. Logic and Computation*, vol. 5, no. 2, pp. 227-249, 1995.

[6] D. Dori, "Representing pattern recognition-embedded systems through object-process diagrams: The case of the Machine Drawing Understanding System," *Pattern Recognition Letters*, vol. 16, no. 4, pp. 377-384, 1995.

[7] B. Smith and J. Wellington, "Initial graphics exchange specification," Nat'l Inst. of Standards, Gaithersburg, Md., 1986.

[8] I. Chai and D. Dori, "Extraction of text boxes from engineering drawings," *Proc. Soc. Photo-Optical Instrumentation Engineers—Int'l Soc. Optical Eng. (SPIE)/Soc. Imaging Science and Technology (IS&T) Symp. Electronic Imaging Science and Technology, Conf. Character Recognition and Digitizer Technologies*, SPIE vol. 1,661, pp. 38-49, San Jose, Calif., Feb. 10-12, 1992.

[9] D. Dori and A. Pnueli, "The grammar of dimensions in machine drawings," *Computer Vision, Graphics, and Image Processing*, vol. 42, pp. 1-18, 1988.

[10] D. Dori, "Dimensioning analysis: A step towards automatic high level understanding of engineering drawings," *Comm. ACM*, vol. 35, no. 10, pp. 92- 103, Oct. 1992.

[11] D.H. Ballard, "Generalizing the Hough transform to detect arbitrary shapes," *Pattern Recognition*, vol. 13, no. 2, pp. 111-122, 1981.

[12] D.J. Hunt and L.W. Nolte, "Performance of the Hough transform and its relationship to statistical signal detection theory," *Computer Vision, Graphics, and Image Processing*, vol. 43, pp. 221-238, 1988.

[13] J. Illingworth and J. Kittler, "The adaptive Hough transform," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 9, no. 5, pp. 690-697, 1987.

[14] R.M. Haralick and L. Shapiro, *Computer and Robot Vision*. Reading, Mass.: Addison Wesley, 1992.

[15] R.S. Conker, "A dual plane variation of the Hough transform for detecting nonconcentric circles of different radii," *Computer Vision, Graphics, and Image Processing*, vol. 43, pp. 115-132, 1988.

[16] C. Kimme, D.H. Ballard, and J. Sklansky, "Finding circles by an array of accumulators," *CACM*, vol. 18, pp. 120-122, 1975.

[17] L. O'Gorman and A.C. Sanderson, "The converging squares algorithm: An efficient method for locating peaks in multidimensions," *IEEE Trans. Pattern Analysis and Machine Intelligence*, pp. 280-288, 1984.

[18] H. Asada and M. Brady, "The curvature primal sketch," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 8, no. 1, pp. 1-14, Jan. 1986.

[19] P.L. Rosin and G.A.W. West, "Segmentation of edges into lines and arcs," *Image and Vision Computing*, vol. 7, no. 2, pp. 109-114, May 1989.

[20] L. O'Gorman, "Curvilinear feature detection from curvature estimation," *Proc. Ninth ICPR*, pp. 116-119, Rome, Nov. 1988.

[21] L. O'Gorman, "An analysis of feature detectability from curvature estimation," *Proc. CVPR*, Ann Arbor, Mich., June 1988.

[22] D.G. Lowe, "Three dimensional object recognition from single two dimensional images," *Artificial Intelligence*, vol. 31, pp. 355-395, 1987.

APPENDIX A

```
Procedure 1: Find trace direction
  determine_trace_direction (current_bar_chain, trace_direction)
  {          /* decide trace_direction by tracing input bar chain */
             current_vector = get_next_vector (current_bar_chain);
                                                     /* get first bar */
     a = 0;                                /* inital value of sum of q */
     while (a > 0) or (! is_empty (current_bar_chain))
     {
  (DX0 , DY0) <- current_vector;
  next_vector = get_next_vector (current_bar_chain);
                                                /* get next bar */
  (DX1 , DY1) <- next_vector;
  q = arctan2(DY1DX0 - DY0DX1 , DX0DX1 + DY1DY0);
  if (q > 95) break;
     /* if angle change is too drastic, no more bars on arc */
  if ((q * a) < 0) break;
             /* if the curvature is in the wrong direction, */
                                           /* no more bars on arc. */
  a = a + q ;                                        /* sum up q's */
  current_vector = next_vector;
     }
                     /* summing up all change of angles */
     if (a > 0)                                      /* arc goes clockwise */
  trace_direction = counter_clockwise;
     else if (a < 0)                       /* arc goes counter_clockwise */
  trace_direction = clockwise;
  }
```

APPENDIX B

```
Procedure 2: Perpendicular Bisector Tracing (PBT)
 PBT_trace (starting_point, trace_direction, detected_arc_point)
  {          /* trace following input trace direction to find an arc point */
             trace_direction = trace_direction / magnitude;
                                                     /* get unit vector */
             search_step = 0;              /* search has not started */
             while (! check_on_arc (current_search_point))
                     and (search_step < limit)
                     {
                     current_search_point = search_step *
             trace_direction + starting_point;
                     search_step ++;        /*search one more step */
                     }          /* keep tracing until edge of arc is hit */
                     starting_arc_point = current_search_point;
                     search_step = 0                 /* reset search step */
                     do                    /* trace the width of arc */
                          {
                     search_step ++;
                     ending_arc_point = search_step*trace_direction +
             starting_arc_point;
                          }
                     while (check_on_arc(ending_arc_point))
                                     and (search_step < limit);
                     search_step -- ;               /* retreat one step */
                     ending_arc_point = search_step*trace_direction + starting_arc_point;
                     detected_arc_point = middle_point (starting_arc_point, ending_arc_point);
  }
```

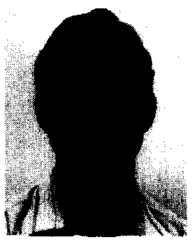
APPENDIX C

```
Procedure 3: arc endpoint refinement
             end_point_refinement (end_point_C, end_point_D, trace_direction)
     {/* input end points need to be further approximated to decide tracing direction */
                                                  /* get stretch direction */
     left_stretch_direction = (end_point_C - end_point_D)/magnitude;
     right_stretch_direction = - left_stretch_direction;
     left_anchor_point1 = trace_black_pixel (end_point_D, right_stretch_direction);
     right_anchor_point1 = trace_black_pixel (end_point_D, right_stretch_direction);
                                /* trace direction at left end of arc */
     left_trace_direction = choose_trace_direction (left_anchor_point1, left_stretch_direction);
                                /* trace direction at right end of arc */
     right_trace_direction = -
```

```
    choose_trace_direction
                (right_anchor_point1, right_stretch_direction);
    if (left_trace_direction == right_trace_direction)
                                                /* if identical directions */
            trace_direction = - left_trace_direction;
    else fail_in_detecting;
                /* PBT cannot have two different trace directions */
    left_anchor_point2 =
            trace_black_pixel (left_anchor_point1, left_trace_direction);
    right_anchor_point2 = trace_black_pixel
                (right_anchor_point1, right_trace_direction);
    end_point_C =
                middle_point (left_anchor_point1, left_anchor_point2);
    end_point_D =
                middle_point (right_anchor_point1, right_anchor_point2);
    }
```

APPENDIX D

```
Procedure 4: recursive arc center computation
detect_arc_center (left_end_point, right_end_point, arc_center)
/* left_end_point and right_end_point are two input arc end points */
  {
    start_trace_point = middle_point (left_end_point, right_end_point);
                                                /* PBT trace for arc point */
    PBT_trace (start_trace_point, tracing_direction, middle_arc_point);
  if on_straight_line (left_end_point, right_end_point, middle_arc_point)
            return 0;
                /* if three points are on the same line, no arc is found */
                                                /* first estimation of center */
  temp_center_1 =
    compute_center (left_end_point, right_end_point, middle_arc_point);
  start_trace_point =
    middle_point (left_end_point, middle_arc_point);
  PBT_trace (start_trace_point, tracing_direction, left_arc_point);
  if on_straight_line (left_end_point, middle_arc_point, left_arc_point)
            return 0;
                                                /* second estimation of center */
  temp-Center_2 =
    compute_center (left_end_point, middle_arc_point, left_arc_point);
  start_trace_point =
    middle_point (right_end_point, middle_arc_point);
  PBT_trace (start_trace_point, tracing_direction, right_arc_point);
  if on_straight_line (right_end_point, middle_arc_point, right_arc_point)
            return 0;
                                                /* third estimation of center */
temp_center_3 =
    compute_center (right_end_point, middle_arc_point, right_arc_point);
  diff = compute_difference (temp_center_1, temp_center_2, temp_center_3);
  if (diff <= LOWER_DIS_THRESHOLD)                  /* if accurate enough */
            {
    arc_center = average (temp_center_1, temp_center_2, temp_center_3);
    return 1;                                       /* center is found */
            }
  else if (diff >= UPPER_DIS_THRESHOLD)             /* if too far away */
    return 0;                                               /* fail */
            else
    if detect_arc_center (left_arc_point, right_arc_point, arc_center)
            /* if arc center is detected by deeper recursive detection */
    return 1;                                           /* find center */
            else return 0;                                  /* fail */
    }
```



Dov Dori 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 Weizmann Institute of Science, Rehovot, Israel, in 1988.

From 1975 to 1984, Dr. Dori served as an officer at Israel Defense Force and was chief industrial engineer of Merkava Tank Plant and head of a computer unit. Between 1988 and 1990 he was an assistant professor in the Department of Computer Science at the University of Kansas. He has been Alexander Goldberg Senior Lecturer with the Faculty of Industrial Engineering and Management, Technion, Israel Institute of Technology, since 1991, and has been head of the Information Systems Engineering Area there since 1995.

Dr. Dori's research interests include document analysis, understanding engineering drawings, machine vision, system analysis and design methodologies, development of the object-process paradigm, and computer integrated manufacturing. He is co-editor of the book *Shape, Structure, and Pattern Recognition*. He has served as a consultant for several Israeli software and manufacturing firms. He won first place in the Dashed Line Detection contest held at Pennsylvania State University in August of 1995. Dr. Dori is a member of the IEEE, the IEEE Computer Society, the ACM, and the International Association for Pattern Recognition (IAPR).