EXTRACTION OF LINES AND REGIONS FROM GREY TONE LINE DRAWING IMAGES

L. T. WATSON, K. ARVIND, R. W. EHRLICH
Department of Computer Science

and

R. M. HARALICK
Departments of Electrical Engineering and Computer Science,
Virginia Polytechnic Institute and State University,
Blacksburg, VA 24061, U.S.A.

(Received 14 September 1983; in revised form 22 November 1983; received for publication 19 December 1983)

Abstract—An algorithm is described for extracting lines from grey level digitizations of industrial drawings.
The algorithm is robust, noniterative, sequential and includes procedures for differentiating shaded areas
from lines. Examples are given for complex regions of a typical mechanical drawing.

Adaptive thresholding  Grey tone intensity surface  Line drawing  Line extraction
Line tracker  Region extraction  Region growing

1. INTRODUCTION

There are many applications, such as mapping, drafting, image compression and computer vision, that
require robust algorithms for extracting lines and boundaries from images. While a very substantial
effort has been made to solve that problem under the adverse image conditions typical in computer vision
applications, most popular techniques have many computational disadvantages when image conditions
are good. When there is no need to be concerned with serious line fragmentation due to noise, it is possible
to deal more directly with the problems of line drawing semantics, including curvature, endpoints, junctions,
intersections, variable width, smoothness, straightness and problems of approximation and encoding.

The particular application that motivated the work reported here is the problem of converting industrial
line drawings from hard copy into highly compressed graphical representations involving a small number
of primitives, such as lines, curves and regions. This problem vanishes when electronic drafting systems are
used to generate the original drawings. However, the reality is that many technical drawings are still created
and communicated on paper and relatively little use is being made of electronically stored representations. As
a consequence, revision and updating is difficult to do and field documentation of large, high-technology
systems, such as aircraft, space vehicles, buildings and computers, all too frequently consists of containers,
files or even entire rooms full of hard copy reproductions.

The algorithms described in this paper make use of grey level digitizations rather than binary digitizations
to maximize the amount of image information available to the interpretation procedures and to minimize the
intelligence required of the sensors themselves. The resolution is high and the images are assumed to be
large—at least 1024 × 1024 picture elements. No attempt is made to repair poorly executed drawings.
Because of the large data volume and in order to facilitate processing on inexpensive microcomputers,
the use of iterative algorithms, including grey level and binary thinning, is precluded and every possible
attempt is made to make processing sequential by scan line. It is also assumed that the line drawings to be
processed may contain small, solid, shaded (not textured) regions or regions of parallel adjacent lines too
fine to resolve individually. Before presenting specific algorithms, a review of some of the existing approaches
for extracting lines from high-contrast digitizations is given.

Currently, there are a number of systems for the extracting and encoding of line structured data. These
systems frequently contain dedicated hardware, such as fast optical scanners, that do fast local raster
scanning of image data. Black et al.¹ discuss a general purpose follower for line structured data which is table
driven using a PEPR flying spot scanner. Fulford² describes the FASTRAK system, which is an
interactive line following digitizer, scanning a reduction of a map with a laser beam. The system depends upon
human interaction and intervention for starting lines and guiding the tracker along noisy or ambiguous
lines. SysCan, a system described by Leberl and Olson³ features KartoScan, a raster scanner using
white light and a CCD array sensor. It is an oper-

* This work was supported by Control Data Corporation
Grant 82V101.
ational automated system which converts maps and drawings into digital format and the digital data is edited, stored, retrieved and generally manipulated.

Holdemann and Kazmierczak,(4) Peleg and Rosenfeld,(5) Ting and Prasada,(6) Wang et al.(7) and Weszka(8) describe general preprocessing techniques, such as thresholding and other forms of filtering applied to binary and gray tone images.

Complete systems for the extraction of line structured data from binary and gray scale images are described in the literature. A coding method for the vector representation of engineering drawings is discussed by Ramachandran.(9) This algorithm does not distinguish between lines and regions and, moreover, the final form of extracted information, which is an approximation, is not suitable for parametric representation (by splines, for example). Compression ratios of about 35:1 were achieved, which are not very satisfactory for line drawings. Woetzel(10) describes an automatic method for the scanning of cartographic maps and extracting the linework. The method only works on binary images and uses a fast thinning algorithm which may distort the line structure (this is not very critical for maps). Furthermore, regions cannot be extracted. Dudani(11) describes a contour following algorithm that can be adapted as a line following algorithm. However, the algorithm does not handle thick lines.

The present work incorporates well known techniques for encoding, digital transmission and ridge detection. For completeness, a few relevant references are mentioned here. The paper by Freeman(12) is a good tutorial on line drawings and also explains the concept of chain codes. Graham(13) and Huang(14) discuss methods for digital transmission. Maxwell(15) attempts to evolve natural descriptors for line drawings for efficient human–computer communication in the domain of computer graphics. Watson et al.(16) and Haralick et al.(17) describe a method for the topographic labelling of gray scale image characteristics, such as peaks, ridges, valleys, etc., which can be applied to line drawings.

This paper describes a system which extracts the linework and solid regions from large gray tone images of line drawings using noniterative fast algorithms with minimal storage requirements.

2. ALGORITHMS

A large digitized image produced by a collimated white light scanner was initially blurred by a Gaussian filter and then resampled at every other pixel (2:1 sampling) to produce the input image that was used to develop the algorithms. This image (see Fig. 1) was of size 1024 × 1024 and consisted of thin and thick lines and a few regions with a uniform gray tone value except at the borders. The gray tone intensity surface of a line was ridge-like due to the Gaussian filtering. The gray tones in the image ranged from 0 to 255 in value and average line intensities varied (from 50 to 250) in different parts of the image. Line separation was as low as one pixel in some portions of the image and the valleys that separated these close lines had gray tone values in between those of the lines and the zero background.

Double adaptive thresholding

Because of the small differences in height between valleys and ridges (formally defined in Haralick et al.(17)) and variation in average intensities over the image, simple thresholding using a single cut-off value fails to resolve all the lines and suggests the use of local adaptive thresholding. A good local characteristic is the average of pixel values in a neighborhood centered around the pixel being tested. Thresholding can be described as determining which are the object pixels (typically represented by ones in the output image) and which are the background pixels (represented by zeros in the output image). A strategy whereby the threshold cut-off was computed by multiplying some constant by the average gray tone over a square \( n \times n \) window centered at the current pixel produced very good results. Decreasing the window size has the effect of increasing the resolution, making the thresholding more sensitive to local features. Since low background pixel values have the effect of depressing the computed average, thereby diluting large peaks, only those pixels in the neighborhood which are greater than a certain low threshold are used in the computation of the average. This lower threshold is chosen to be slightly higher than the background pixel values. Thus there are two thresholds, the upper threshold which is computed by multiplying the window average by a cutoff factor and the lower threshold—hence the name double adaptive thresholding. By choosing a value slightly greater than one for the cut-off factor, only the ridge-line pixels of the convex ridges have values more than the computed cut-off and therefore appear in the output as object pixels. Region pixels do not pass the thresholding condition because of the flat nature of the regions. Here we make a small modification of the algorithm. Pixels which do not pass the thresholding condition but which are greater than a certain region threshold (chosen to be consistent with region pixel values) are marked as region pixels in the output. Note that this double adaptive thresholding is a procedure to extract structure from a given image and not an intelligent program to restore interpretations that have been lost because of noise in some ideal image.

Thresholding that normally produces a binary image does not produce lines exactly one pixel thick, making line tracking rather difficult. One course of action would be to thin the binary image produced by the thresholding. A large variety of thinning algorithms are described in the literature and each has its own flavor from the point of view of the accuracy and aesthetic appeal of the skeletons they produce from the original binary image. Often the result is unappealing as for example when curved lines are reduced to thin
jagged lines. The problem is not with the thinning, but with the fact that once we create a binary image from thresholding, the information contained in gray tones of the original image is lost and no distinction can be made between weak and strong object pixels which are all ones. Hence, a gray tone skeleton is needed, as produced by, for example, the gray scale medial axis transformation of Wang and Rosenfeld. This algorithm was tried and found to decompose variable width lines, besides not dealing with irregular shaped regions. Hence, in the double adaptive thresholding algorithm, all pixels which are above their computed cut-off values (i.e. the object pixels) are represented in the output by their original gray tone values. Region pixels are represented by the negative of their original gray tone values. The reason for retaining the gray tone values for region pixels will be apparent later, when we describe region pruning.

The double adaptive thresholding algorithm (DAT) works well for lines when their gray tone intensity surfaces have a convex ridge shape, which can be artificially produced by Gaussian filtering, local averaging or other known blurring techniques. The DAT algorithm is given below.

\[ n := 3 \quad (* \text{window size} *) \]
\[ \text{factor} := 1.063 \]
\[ \text{region\_threshold} := 200 \]
\[ \text{low\_threshold} := 6 \]
\[ \text{avg} := \text{average of all pixel values} \]
\[ > \text{low\_threshold} \text{in an } n \times n \]
\[ \text{neighborhood of the current pixel} \]
\[ \text{cur\_pixel} := \text{current input pixel value} \]
\[ \text{out\_pixel} := \text{current output pixel value} \]

\[ \text{IF cur\_pixel} < = (\text{avg} \times \text{factor}) \]
\[ \text{THEN IF cur\_pixel} > \text{region\_threshold} \]
\[ \text{THEN out\_pixel} := (-\text{cur\_pixel}) \]
\[ \text{ELSE out\_pixel} := 0 \]
\[ \text{ELSE out\_pixel} := \text{cur\_pixel}. \]

**Region determination**

The DAT, though it handles regions well, has the effect of marking some stray pixels as region pixels. In addition, there may also be holes at the boundaries of the regions. In the next two steps, called “growing” and “shrinking”, stray region pixels are eliminated and small holes in the regions are filled. Both these steps are based on the connectivity of region pixels. In the “growing” operation each non-zero non-region pixel is marked as a region pixel if more than \( k = 2 \) of its eight neighbors are region pixels. This has the effect of filling up small holes and growing the regions. Stray region pixels are not affected because of their low region connectivity. In the “shrinking” operation, which is complementary to and follows the “growing” operation, a region pixel is changed to a non-region pixel if less than \( k = 4 \) of its eight neighbors are region pixels. Stray region pixels are converted to non-region (i.e. line) pixels in this operation. The two steps described above which constitute region determination are shown below.

\[ n := 3 \quad (* \text{window size} *) \]
\[ \text{cur\_pixel} := \text{current input pixel value} \]
\[ \text{out\_pixel} := \text{current output pixel value} \]
\[ \text{num\_neg} := \text{number of negative neighbors} \]
\[ \text{in an } n \times n \text{ neighborhood of} \]
\[ \text{the current pixel not counting} \]
\[ \text{the current pixel}. \]
\[ \text{const1} := 3 \]
\[ \text{const2} := 4 \]

**Grow:**

\[ \text{IF num\_neg} \geq \text{const1 AND cur\_pixel} > 0 \]
\[ \text{THEN out\_pixel} := (-\text{cur\_pixel}) (* \text{region} *) \]
\[ \text{ELSE out\_pixel} := (\text{cur\_pixel}). \]

**Shrink:**

\[ (* \text{with output of Grow as input} *) \]
\[ \text{IF cur\_pixel} < 0 \text{AND num\_neg} \geq \text{const2} \]
\[ \text{THEN out\_pixel} := \text{cur\_pixel} (* \text{region} *) \]
\[ \text{ELSE out\_pixel} := \text{abs} (\text{cur\_pixel}). \]

**Region extraction**

The region determination step is followed by region extraction and line extraction. The regions can be represented by following and encoding their contours. Duda\textsuperscript{11} describes a method for region extraction using boundary following. Alternatively, simple run length coding or one of its more complex variations could be used. Such schemes work well because of good correlation of pixel runs between adjacent scan lines in the regions.

**Line extraction**

Using a simple thinning algorithm to reduce the line width to one pixel, followed by a simple line tracker, is unsatisfactory for the reasons given above. A more complicated line tracker which tracks the ridges of the gray tone intensity surfaces of the lines is described below in two steps.

(1) **Finding the starting pixel for a new line.** The image is scanned from left to right and top to bottom. Thus, line by line each pixel is examined to check whether it continues a line or is a candidate for starting a new line. The following conditions must be satisfied by an unmarked pixel (called the candidate pixel) in order to be a starting point of a line.

(a) Its value is more than a certain threshold to indicate which pixel values are background and which are not.

(b) Its value is more than a factor (a good value is 0.7) times the average of its non-zero marked or unmarked eight neighbors. (When a pixel is tracked it is marked.)

(c) Its value is more than a factor (0.9) times the average of its marked eight neighbors. This condition ensures that the pixel is strong compared to nearby tracked vectors.

(d) Let pixel \( P_1 \) be the unmarked, untested eight neighbor of the candidate pixel with maximum gray tone value. This selection implies a probable
direction for a new line. Directions of the eight neighbors with respect to the candidate pixel are labeled as follows. Let \(d\) be the direction of \(P_1\) with respect to the candidate pixel. \(P_1\) must also satisfy conditions (a) through (c).

(e) Let \(P_{11}, P_{12}, P_{13}\) be the neighbors in the directions \(d-1, d, d+1\) (modulo 8), respectively, from \(P_1\). At least one of these pixels \(P_j\) must not be marked, must not have a marked eight neighbor in the directions \(s-1\) and \(s+1\) from \(P_1\) (\(s\) is the direction of \(P_j\) from \(P_1\)) and must satisfy conditions (a) through (c). Otherwise, consider \(P_1\) tested and attempt to satisfy (d) and (e) with a different \(P_1\), until all possibilities have been tested.

If conditions (a) through (e) are satisfied by the candidate pixel, then it is marked along with \(P_1\) and the line tracker is invoked to continue tracking from pixel \(P_1\).

2) Tracking a line. Let \(P_1\) represent the previously marked pixel, \(P_2\) the current marked pixel and \(P_3\) the next pixel sought by the line tracker. Let \(d_1\) be the direction to \(P_1\) from the pixel marked prior to \(P_1\), \(d_2\) the direction from \(P_1\) to \(P_2\) and \(d_3\) the direction from \(P_2\) to \(P_3\). From among the unmarked neighbors of \(P_2\) in the directions \(d_2 - 2\) (mod 8), \(d_2 - 1\) (mod 8), \(d_2\), \(d_2 + 1\) (mod 8), \(d_2 + 2\) (mod 8), \(P_3\) is chosen as the pixel (if it exists) with maximum gray tone value. \(P_3\) is marked if it satisfies the conditions:

(a) \(P_3\)’s gray tone value is greater than a certain threshold, which is slightly more than typical background values;
(b) \(P_3\) has no marked eight neighbors in the direction \(d_3 - 1\) (mod 8) and \(d_3 + 1\) (mod 8) from \(P_2\);
(c) \(\min\{\text{abs}(d_3 - d_1), 8 - \text{abs}(d_3 - d_1)\} = 2\).

If \(P_3\) does not exist or does not satisfy conditions (a) through (c), then the current vector is terminated and the next line continuation pixel or line starting pixel is sought. If no such pixel is found, the line tracker moves down to the next scan line.

The algorithm for the line tracker is shown below.

STEP 1. Determination of the starting point.
\[
\begin{align*}
\text{cur\_pix} & := \text{current pixel} \\
\text{VALUE}(\text{cur\_pix}) & := \text{value of current pixel} \\
\text{cur\_val} & := \text{VALUE}(\text{cur\_pix}) \\
n & := 3 \quad (\text{\# window size}) \\
\text{avg\_marked} & := \text{average of all the marked pixel values in an } n \times n \text{ neighborhood of the current pixel}
\end{align*}
\]

\[
\begin{align*}
\text{avg\_nonzero} & := \text{average of all the non-zero pixel values in an } n \times n \text{ neighborhood of the current pixel} \\
c\text{utoff1} & := 0.9 \times \text{avg\_marked} \\
c\text{utoff2} & := 0.7 \times \text{avg\_nonzero} \\
\text{CUTOFF}(\text{cur\_pix}) & := \text{MAX}(50, \text{cutoff1}, \text{cutoff2}) \quad \text{for } \text{cur\_pix} \neq \text{unmarked pixel}
\end{align*}
\]

IF \text{cur\_pix} is not marked AND \text{cur\_val} > \text{CUTOFF}(\text{cur\_pix})

THEN WHILE (some unmarked neighbor of \text{cur\_pix} not tested) DO

BEGIN
\[
\begin{align*}
\text{temp\_pix} & := \text{unmarked, untested neighbor of } \\
\text{cur\_pix} \text{ with maximum value} \\
\text{temp\_val} & := \text{VALUE}(\text{temp\_pix}) \\
\text{IF} \text{temp\_val} > \text{CUTOFF}(\text{temp\_pix}) \text{ THEN}
\end{align*}
\]

BEGIN
\[
\begin{align*}
\text{cur\_dir} & := \text{direction from } \text{cur\_pix} \text{ to } \\
\text{temp\_pix} \\
\text{pix1} & := \text{pixel in } \text{dir. } \text{cur\_dir} - 1 \text{ from } \text{temp\_pix} \\
\text{pix2} & := \text{pixel in } \text{dir. } \text{cur\_dir} \text{ from } \text{temp\_pix} \\
\text{pix3} & := \text{pixel in } \text{dir. } \text{cur\_dir} + 1 \text{ from } \text{temp\_pix}
\end{align*}
\]

\[
\begin{align*}
\text{IF} \text{VALUE}(\text{pix1}) > \text{CUTOFF}(\text{pix1}) \text{ AND pix1} \text{ not marked AND pix1 has no adjacent marked neighbors } \text{(*) adjacent means in the direction from temp\_pix + 1 or - 1 *)} \text{ OR VALUE}(\text{pix2}) > \text{CUTOFF}(\text{pix2}) \text{ AND pix2} \text{ not marked AND pix2 has no adjacent marked neighbors} \text{ OR VALUE}(\text{pix3}) > \text{CUTOFF}(\text{pix3}) \text{ AND pix3} \text{ not marked AND pix3 has no adjacent marked neighbors}
\end{align*}
\]

THEN mark \text{cur\_pix}, \text{temp\_pix}, and call tracker
END
ELSE designate \text{temp\_pix} as tested
END (* WHILE *)

STEP 2. Tracking the vector.
\[
\begin{align*}
\text{cur\_dir} & := \text{current direction} \\
\text{prev\_dir} & := \text{previous direction} \\
\text{threshold} & := 50 \\
\text{next\_pix} & := \text{unmarked neighbor in direction } \text{d from } \\
& \text{cur\_pix, abs } (d - \text{cur\_dir} \text{ mod 8}) < = 2, \text{ with maximum value.}
\end{align*}
\]

IF \text{next\_pix} exists

THEN
BEGIN
\[
\begin{align*}
\text{next\_dir} & := \text{direction from } \text{cur\_pix} \text{ to } \text{next\_pix} \\
\text{next\_val} & := \text{VALUE}(\text{next\_pix})
\end{align*}
\]
END
ELSE \text{next\_val} := 0

IF \text{next\_val} > \text{threshold AND}
\[
\begin{align*}
\text{abs}(\text{next\_dir} - \text{prev\_dir} \text{ mod 8}) = = 2 \text{ AND neighbors of next\_pix in directions } \\
\text{next\_dir} - 1, \text{ next\_dir} + 1 \text{ from cur\_pix are not marked}
\end{align*}
\]

THEN
BEGIN
mark \text{next\_pix} \\
\text{cur\_pix} := \text{next\_pix} \\
\text{prev\_dir} := \text{cur\_dir} \\
\text{cur\_dir} := \text{next\_dir} \\
\text{continue tracking the current vector}
END
ELSE
BEGIN
\text{terminate tracking the current vector}
\text{determine the starting point of next vector}
END.

The line tracker, in addition to marking the pixels in
the image, also outputs the new chain code or absolute coordinates of the pixels. The output format depends on the user's requirements. Skiansky and Gonzales (18) discuss a method for fast polygonal approximation of the pixels as the pixels are being tracked. With their scheme a straight line would be represented by its two end points.

3. RESULTS
The line drawing in Fig. 1a was digitized by a collimated white light scanner producing a first generation image of size 2048 × 2048 pixels. The lines are sparse in most of the image except in the spool of wire. The 1024 × 1024 test image (to which the algorithms were applied) was obtained by blurring the first generation image using a Gaussian filter of standard deviation 1.039 and then resampling the filtered image at every other pixel. Compression ratios higher than 2:1 cause Moire patterns in the spool of wire, attributed to the classical problem of aliasing. Figure 1b
shows a close up view of the spool of wire in the test image. This is the most crucial part of the image, since the line separation here is very low and the lines merge at the edge of the coil to form textured regions. The test image corresponded very closely with what would have been produced by some industrial imaging hardware already in place. The larger image was taken as the starting point simply for the purposes of comparison and testing.

Figure 2a displays the gray tone values of a portion
of the spool of wire, where the upper portion resembles a "bobbin". The ridge pixels of some lines have been highlighted. Note the presence of a considerable amount of noise, some of which was present in the original image and the rest was added by the filtering.

A well known iterative sharpening algorithm EHHNUM (9) (an iterative algorithm which replaces each pixel's gray tone value by the nearest of the max or min of its eight neighbors' gray tone values) was applied to this noisy image. The result is illustrated in Fig. 2b.
Fig. 3. Topographic labelling of bobbin, where 1 = flat, 2 = convex hillside, 3 = concave hillside, 4 = saddle hillside, 5 = slope, 6 = ridge, 7 = peak, 8 = ravine, 9 = pit, 10 = saddle, 11 = inflection point.

Although the image in Fig. 2b does have a cleaner, sharper appearance, the lines are now rather jagged because the gray tone values of many insignificant pixels have now been raised. There is a more serious defect, namely, although there are only two vertical lines at the top in the input image Fig. 2a, the sharpening algorithm EHNUM has found three (highlighted in Fig. 2b).
Figure 3 shows the result of applying the topographic labeling algorithm\(^{(4)}\) to the image in Fig. 2a. The algorithm fits a two-dimensional cubic polynomial to the gray tone values in a square 5 \times 5 window (this window size can be changed) centered around each pixel. This polynomial represents a surface which best fits, in a discrete least squares sense, the pixel data in the window. The topography of the surface at the position of the central pixel is now determined using partial derivatives of the polynomial.
Fig. 4b. Topographic labelling of coil, where 1 = flat, 2 = convex hillside, 3 = concave hillside, 4 = saddle hillside, 5 = slope, 6 = ridge, 7 = peak, 8 = ravine, 9 = pit, 10 = saddle, 11 = inflection point.

By tuning threshold parameters in the algorithm, a good representation of the ridges (flagged by asterisks in Fig. 3) was obtained.

Figure 4a shows a dense part of the coil with a typical line highlighted. The lines are not well resolved and there is a lot of noise between the lines. The topographic labeling algorithm was applied to this image with the same parameter values that were used.
to obtain the image in Fig. 3 (see Fig. 4b). The detection of the gray tone surface ridges, which clearly exist in Fig. 4a, is obviously extremely poor in Fig. 4b, where what should have been a line is highlighted. This illustrates the sensitive nature of the algorithm to the threshold parameter values, making it unsuitable for the present application.

The DAT was used with very good results on the image in Fig. 4a. Figure 5 shows this result, in which the lines have been accentuated very well. The same
line is highlighted in Figs 4a, 4b and 5. The negative pixels represent the regions on which the GROW and SHRINK algorithms were applied to finally get the regions shown in Fig. 6. These extracted regions were quite consistent with the perceived regions in the original unprocessed image.

The line tracker was now applied to the image in Fig. 5 after the regions shown in Fig. 6 had been removed. Pixels which the line tracker marked are shown with negative values in Fig. 7. For further illustration, the
same tracked image is shown in Fig. 8, but as a binary image indicating the pixels marked by the tracker. Observe that the tracked lines very faithfully represent the lines in the original unprocessed image.

Fig. 7. Output of line tracker on coil (shown with regions removed).

4. CONCLUSION

The difficulties associated with real digitized line drawing images, as opposed to artificially generated
images, are significant. Many papers on line drawings have used binary images as their starting point, but the present work shows that producing a good binary image from a raw digitized gray tone image is highly nontrivial. Any practical production algorithm must clearly begin with noisy gray tone images. Standard thinning, sharpening and medial axis transformations were tried and (despite occasional exemplary performances) none were found to be uniformly good.

For certain industrial applications it may not be practical or economic to connect the imaging equipment to mainframe computers with high speed transmission lines. Thus the low level processing (extraction of the lines and regions) must be done with limited local computer power and storage. The DAT algorithm presented here meets these requirements by being reasonably cheap and noniterative, although several sequential passes through the image are required. Numerous experiments on different types of line drawings also strongly suggest that a good algorithm must be adaptive, and the adaptive nature of the DAT is crucial to its success.

The overall problem being addressed here for real digitized gray tone images of line drawings consists of: (1) recognition and extraction of lines and solid regions (textured regions are not considered); (2) the compression and transmission of the line and region data; (3) the high level representation (e.g. as graphics primitives) of the line drawing (lines, curves, regions). This paper represents a solution to (1) requiring only limited local computer power and storage. The next goals are an economic solution to (2) using slow transmission speeds (300 baud) and a powerful and sophisticated high level encoding of the line drawing as graphical primitives in, for example, CDC's TUTOR system.

REFERENCES

About the Author—Layne T. Watson was born in Vanderburg County, Indiana, on 24 December 1947. He received the B.A. degree (magna cum laude) in Psychology and Mathematics from the University of Evansville, Evansville, Indiana, in 1969 and the Ph.D. degree in Mathematics from the University of Michigan, Ann Arbor, in 1974. While at the University of Michigan, he studied numerical analysis under C. Molter, C. de Boor and G. Fix and worked with K. Murti on complementary problems. He served on the faculties of the University of Michigan and Michigan State University before coming to Virginia Polytechnic Institute and State
University, Blacksburg. His current interests are fluid dynamics, structural mechanics, homotopy algorithms and image processing.

Dr. Watson is a member of Phi Kappa Phi, Blue Key, Psi Chi, Kappa Mu Epsilon, Phi Beta Chi and SIAM.

About the Author—Krishna Arvind was born in Rajahmundry, Andhra Pradesh, India, on 30 June 1956. He received the B.Sc. degree in Mathematics, Physics and Chemistry from Osmania University in 1976. He received the B.Tech. degree in Electronics and Telecommunications from the Madras Institute of Technology, Madras, India, in 1979. He then joined the Research and Development Wing of the Computer Maintenance Corporation, Hyderabad, India, and was involved for about two years in the design and development of digital computer circuits.

Since September 1981 he has been working on his Masters degree in Computer Science at Virginia Polytechnic Institute and State University, Blacksburg. His interests include image processing, database systems and computer networks.

About the Author—Roger W. Ehrlich was born in Rochester, New York, and received the B.S.E.E. degree from the University of Rochester. In 1969 he received the Ph.D. degree in Electrical Engineering from Northwestern University and joined the Electrical Engineering Department at the University of Massachusetts.

Since 1975 he has been at the Virginia Polytechnic Institute and State University, where he is currently Professor of Computer Science. At Virginia Tech he is Director of the Computer Science Laboratory and Associate Director of the Spatial Data Analysis Laboratory. His principal research interests are in computer graphics, computer vision and in the human engineering of large software systems.

About the Author—Robert M. Haralick received a Ph.D. degree from the University of Kansas, Lawrence, in 1969.

He is presently a Professor in the Departments of Electrical Engineering and Computer Science and Director of the Spatial Data Analysis Laboratory, Virginia Polytechnic Institute and State University, Blacksburg. He has done research in pattern recognition, multimage processing, remote sensing, texture analysis, image data compression, clustering, artificial intelligence and general system theory.

Dr. Haralick is an Associate Editor of the IEEE Transactions on Systems, Man and Cybernetics, Pattern Recognition and Computer Graphics and Image Processing. He is on the editorial board of the IEEE Transactions on Pattern Analysis and Machine Intelligence and is the Chairman of the IEEE Computer Society Technical Committee on Pattern Analysis and Machine Intelligence. He is also the Computer Vision and Image Processing Editor of Communications of the ACM. He is also a member of the Association for Computing Machinery, the Pattern Recognition Society and the Society for General System Research.