Extracting Vectors From Raster Images

J. R. Parker

Abstract - **It is often true that a user of computer images is interested in only a few kinds of features. In particular, in a class of pictures called line images, the major feature is a line or vector. Raster representations of line images, while common, are quite wasteful of storage and are difficult to manipulate. If the vectors could be extracted reliably from the raster image, then rotation, scaling, plotting, and many other operations could be speeded up enormously.**

Introduction

A line image or line drawing can be described as a picture that consists of a collection of lines against a contrasting background [1]. A great deal of human communication takes place using line images: printed and written text, maps, graphs, and tables of all kinds, numbers and mathematical symbols, and some kinds of human and computer drawn renditions of physical objects. Most often it is the placement of the lines, and not their structure, density, or color, that conveys meaning. In order that computers may be used to assist in the processing of pictorial information, an electronic representation of the pictures must be used. The two possibilities are vector representation or raster representation.

The usual representation of an image on a computer is the raster form, in which the image is stored as a two dimensional array of grey level values. Typical image acquisition devices produce this format; examples are video image digitizers and satellite imagers. A line drawing in raster form conveys the same meaning to a user as does the original image, but has been sampled to produce the raster elements (pixels). This is fine for some applications and are inconvenient for others. Raster pictures are difficult to rotate and scale, and it is also harder to label picture elements with objects. Raster pictures usually occupy a great deal of storage, and in the case of line images much of this storage is wasted. On the other hand there are a great number of algorithms for manipulation of raster images. Commonly performed operations are: thresholding, edge enhancement, filtering, segmenting, object recognition, and classification of features. These methods make use of the multiple grey levels, or even colors, available in raster images.

A vector representation stores only the positions and lengths of the lines in the picture. Different intensity levels are generally not encoded. For some kinds of image, this representation would result in the loss of a great deal of information (colors, intensities, regions). For line images, little information would be lost. However, vectors can, when needed, be converted into raster image form using a scan conversion algorithm. Then all of the usual raster processing methods may be applied. The reverse process, raster image to vector image, is more difficult.

Conversion to Vectors

The process of conversion from raster to vector representation consists of two major steps. First, the regions of interest must be identified and enhanced. If this were not done, an enormous number of vectors would be generated which would hide the relevant information and which would not properly convey the structure of the object. The second step is the actual identification and coding of the vectors that comprise the image. In the case of regions these vectors are often the edges that define the boundary of the area, though they could be linear features such as rivers or ridges on a map.

Enhancing regions of interest is done by a specialized thresholding program. The actual technique used would depend on the kind of raster image being used, but the object is to create a binary, or bi-level, raster image. This process is called *segmentation* and various methods for doing it can be found [2,4,5,6,7,8].

In the discussion below the assumption will be that the possible grey levels are 0 and 1 to begin with.

After segmentation, the region of interest is white and the background is black. Figure 1 shows the results of thresholding applied to a sample set of images. The result, in all cases, could be described as a line image.

The extraction of the lines as vectors is the next step, and this is done by searching the image for lines that satisfy certain criteria for being vectors. These criteria can be determined by examining the properties of short straight lines in a raster image. A line is a collection of pixels that will, within some error bound, satisfy the equation $Y = m*X+b$ for some m and b, where X and Y are the horizontal and vertical coordinates of a pixel. When attempting to extract lines from a raster image, the assumption will be made that all white pixels should belong to at least one line. Thus, the first step is to find a set pixel that has not yet been assigned to a line (is *unmarked*). Then the line will be grown from this pixel by looking for adjacent set pixels that are candidates. The question is: given a line of some length L, where may a pixel be added to continue the line?

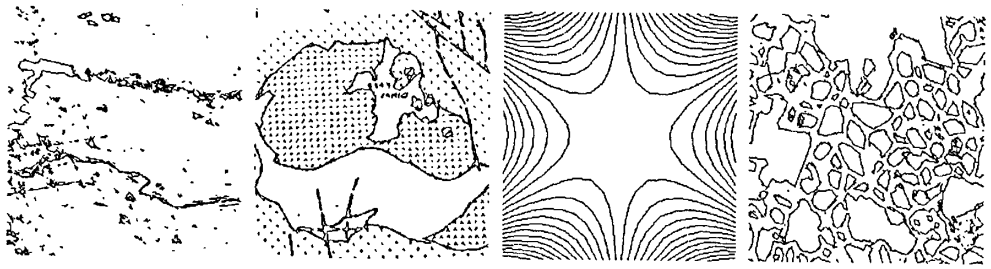
Use of the Chord Property

The *chord property* [3,11] can be used to select sets of pixels that belong to vectors. If S is a set of pixels belonging to some image, and p and q are any two pixels in S, then S has the chord property if the chord $\overline{pq}$ is *near* S, for all p and q. The segment $\overline{pq}$ is *near* S if for every real point (x,y) of $\overline{pq}$ there exists a lattice point (i,j) of S that satisfies $\max[|i-x|, |j-y|] < 1$. It has been shown that a digital arc S is the digitization of a straight line segment if and only if it has the chord property. It should therefore be possible to collect pixels that have this property, and use the endpoints to define the vector.

The problem is that the chord property can be difficult to check. For a set of N pixels there are $\binom{N}{2}$ line segments to check, and there will often be hundreds or thousands of vectors in the image. The time taken to convert an image could be prohibitive. What can be done is to use a computationally simpler method that gives similar results.

A method that appears to work quite well is the following. Pixels are added to the end of the line, one at a time. When a pixel is added, the equation of the new line between the endpoints is found, and a check is made to ensure that all pixels that currently belong to the line have a horizontal and vertical distance of less than one unit from the real line. Tests of this method, involving the extraction of all possible digital line segments from small (64 by 64) raster images, show that all vectors were perfectly extracted, and indicate no difference from using the chord property as defined.

Figure 2 illustrates the chord property and compares it with the new test, which will be referred to as the simplified chord test for the sake of a label. Figure 2a shows the chord property: the boxes around the pixels define the regions within which all real points of the line must reside. The dotted lines in figure 2b represent the distances that are computed by the simplified chord test: none of these may exceed or equal 1. As can be seen, the difference is that horizontal and vertical distances to the real line being constructed are computed only at lattice points. The maximum of the two distances is chosen as an error factor, and these are summed over all pixels that belong to the line to give a total error value for the pixel being added. If in



1a - Geological Micromodel

1b - Graph of $f(x,y)=x^3-3xy^2$

1c - Digitized Map

1d - Thresholded Landsat Image

Figure 1 - Sample Line Images To Be Vectorized.

any instance the distance is greater than or equal to one, the error is considered to be infinite, and the pixel currently under consideration will be rejected.

If there are many candidate pixels that may be added to a given vector, the error is computed for all of them, and the pixel that causes the least error is selected. It is possible, and even likely, that a number of vectors could be traced from a given pixel. In that case, it would be possible to trace all or most of them until one could be identified as the longest, and save that vector as being the "best". This is done in the method outlined below. Another definition of best would pick the vector, however long, that had the smallest total error. This leads to a large number of short, but accurate, vectors.

To help select candidates for the next pixel to be added, a *global slope constraint* was added to the algorithm. This constraint, simply expressed, says that once the direction of the vector is established, in terms of the sign on the Δx and Δy for each next pixel, it may not change, and pixels in certain directions may simply be ignored. Thus, for any pixel at the end of a vector, at most three candidates need to be examined further.

This method of assigning pixels to vectors was tested on a large set of sample images, of which those in figure 1 are a small subset.

Intersecting Lines and Pixel Reuse

When a pixel has been assigned to a vector, it will normally be removed from future consideration by setting that pixels to grey level 0. This works, but better results can be obtained if instead, the pixels are not set to 0 if they have more than 2 neighbors that are non-zero. This condition would be found at places where two lines intersect. Removing the pixels at the point of intersection results, in the case of two intersecting lines, in three vectors being produced. Instead of clearing these pixels, they should be marked as having been used before; later, these pixels may be used again. However, care should be taken to discard all extracted vectors that consist only of marked pixels, since these vectors are completely redundant.

There will still be intersecting lines that cannot be extracted as such by methods described so far. For example, if lines meet at a small angle then there will be a set of shared pixels near the point of intersection that have only two neighbors, as seen in figure 3. Assigning these pixels to either vector results in the other vector being divided into two portions. This problem might be handled in a number of possible ways. First, the problem could be ignored. In many cases it will not matter that an extra 30 percent of vectors results. Second, the problem could be looked at as that of extending the line across a gap; this may turn out to be harder to solve [12]. A third solution is not to remove pixels from the image when they are assigned to a vector. Instead, leave them marked, and never begin a new vector with a marked pixel. Marked pixels can, however, be re-used in the middle of a vector; the result is that most intersecting lines are dealt with

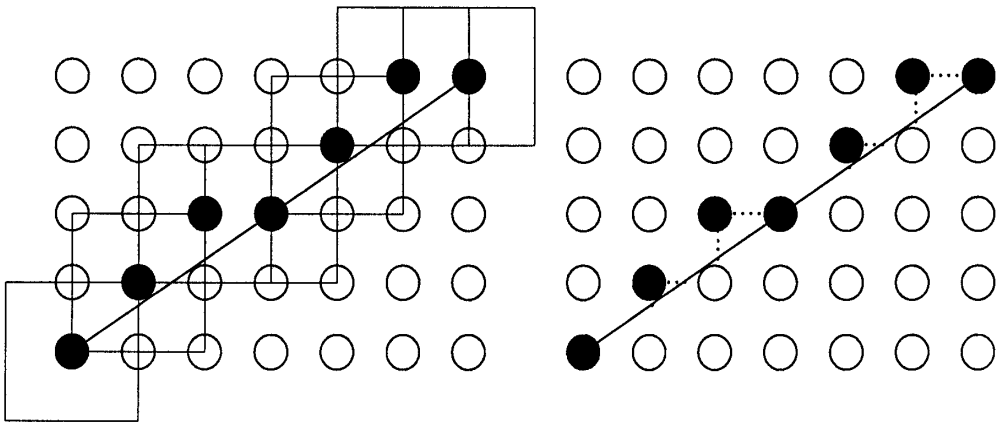


Figure 2a - The Chord Property

2b - The Simplified Chord Test

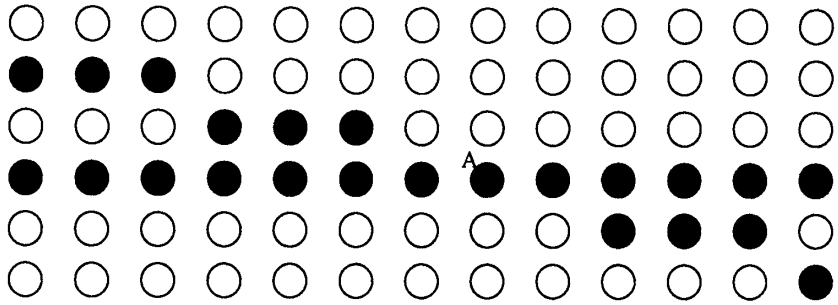


Figure 3 - Lines Meeting at a Small Angle

properly, but a great deal more time is required due to the larger number of pixels that need to be examined.

It may also be that a pixel belongs to many lines. To reduce the time taken to extract vectors from an image, all of our test programs consider all possible vectors that can begin at the selected starting pixel, of which there will be at most eight. For other pixels in the line being considered, only the best fit pixel is selected; other possible pixels will be unmarked, and will be examined later. When all vectors starting at a pixels are found, the longest one is kept. The others, which may or may not actually begin at the selected pixel, are unmarked.

Line Endpoint Constraints

If a vector trace ends because a pixel is found with only one neighbor then the vector can simply be saved. If a trace ends because no neighbors of the last pixel satisfy the linearity constraints, then the feature being traced is not a line. What can happen in this case is the inclusion of the first pixel in the new feature as the last pixel of the old. This results in larger, but still acceptable, errors in the line. If the last pixel is removed, the error decreases appreciably, and the resulting line is better.

In order to handle this situation, the last pixel of a vector is tested to see if it contributes over a given percentage of the total error. For the simplified chord test, error is the sum of the horizontal and vertical distances to the real line, over all pixels in the line. If the last pixel exceeds the threshold, it is discarded and unmarked. The threshold is a parameter to the algorithm; a value of 30% works well in many cases, but, in general, the smaller is the average length of vectors in an image the larger should be this factor.

Another detail that will save on the reuse of pixels is to discard re-used pixels from both ends of the vector. In fact, since a starting pixel can never have been used before, only the pixels at the other end of the line need to be checked.

Scanning For A Starting Pixel

When looking for a pixel at which to begin tracing a vector, the usual way is to start at the (0,0) pixel and look down a row or column, then the next row (column), until an unmarked set pixel is found. This can cause problems in a small number of cases.

Consider the almost horizontal digitized line in figure 4. If the raster image is scanned for a starting pixel in row major order, then the point labeled A is the first pixel seen and a vector will be traced from there to the image boundary. The next starting pixel will be B, and another vector will be traced from there, in spite of the fact that there is clearly only one line in this image. In order to avoid this, the start pixel is re-examined after the vector is traced to see if the vector could be extended in the opposite direction. This can be done very simply, since all possible vectors from the start pixel are kept.

The Detailed Vectorization Method

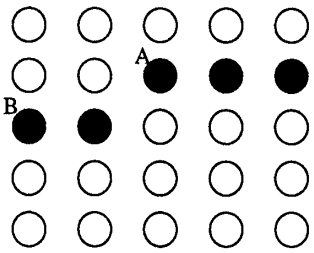


Figure 4 - A Nearly Horizontal Line

Let A be a raster image that has been thresholded and boundary enhanced by some unspecified methods. The following method will extract vectors from the image A, if A contains only grey levels 0 (BLACK) and 1 (WHITE). Let S_v be a set of vectors, initially = ϕ. Let P be a *pixel*, which has an X and Y coordinate P_x and P_y. Let Q and array R[8] be pixels also. Initialize Δx and Δy to 0. Let a pixel be *marked* if bit 5 of that pixel is set, and *unmarked* if that bit is clear.

Procedure vectorize:

[1] Find an unmarked white (1) pixel P in A. This can be done by scanning the image in row or column major order, like a two dimensional array. This pixel will be the start of a vector; mark P. If there are no such pixels, go to Step 10.

[2] Find all unmarked white pixels adjacent to P and place them in a set S_p. Adjacency is based on 8 neighbors.

[3] If $S_p = \phi$ then go to step 9.

[4] Choose a pixel Q from S_p . Remove Q from S_p and mark Q.

[5] Establish the direction of the vector in both the X and Y directions, based on pixels P and Q. The change in one of these directions will be non-zero, and once established may not change. That is : if $\Delta x = 0$ then compute $\Delta x = P_x - Q_x$; also, if $\Delta y = 0$ then compute $\Delta y = P_y - Q_y$.

[6] For all unmarked white neighbors R_i of Q which **satisfy the global slope constraint,** compute the maximum of the horizontal and vertical distances between the line $\overline{PR_i}$ and every marked pixel in the current vector. Reject any R_i for which any single distance >= 1. Let R be the remaining pixel that has the minimum *total* distance. If there is no such pixel then go to step 8. Otherwise, if either Δx or Δy are 0, then recompute them as in step 5 above and continue to the next step.

[7] Add R to the vector being constructed, let Q = R, and mark pixel R. Go to Step 6.

[8] Save the vector $\overline{PQ}$ in the set S_v. i Let $\Delta x = \Delta y = 0$. Go to Step 3.

[9] Let V be the longest vector in the set S_v. **Save** V and **unmark** S_v. Go to Step 1. OPTION: Reverse the order of the pixels in V, and try to extend it in the other direction by continuing from step 6. First, swap the values of P and Q.

[10] Done. No more black pixels remain.

The vectors are written to a file as start and end pixels. Each pixel has an X and Y index, so a vector consists of four numbers. The coordinates of the ends of the vectors can easily be converted into meaningful image-dependent units, such as meters, feet, or latitude and longitude, if this is desired.

There are three procedures that are mentioned in the above method but which have not been described. The first is *satisfy the global slope constraint,* appearing in step 6. The details of this procedure are:

Procedure satisfy_global_slope_constraint:

Assume that P, Q, R_i, Δx and Δy are as defined in procedure *vectorize* above.

[1] If $\Delta x = 0$ then go to step 4.

[2] Compute $DD = Q_x - (R_i)_x$. If DD=0 then skip to step 4.

[3] If the sign of DD is NOT the same as the sign of Δx the skip to step 8.

[4] If $\Delta y = 0$ then go to step 7.

[5] Compute $DD = Q_y - (R_i)_y$. If DD=0 then skip to step 7.

[6] If the sign of DD is NOT the same as the sign of Δy then skip to step 8.

[7] It is true that pixel R_i satisfies the global slope constraint relative to the line segment $\overline{PQ}$. Return.

[8] The pixel R_i DOES NOT satisfy the global slope constraint. Return.

The second procedure that requires description is the **unmark** operation. This is more complex than it appears at first glance, because it is at this stage that a pixel re-use method is applied to ensure that intersecting lines are recognized properly.

Procedure UNMARK:

A pixel is *marked* if bit 5 of that pixel is set, and is unmarked if bit 5 is clear. A marked pixel has been used in a vector.
Let S_v and V be as defined in procedure vectorize above.

[1] For all pixels T in each vector V_i in the set S_v, clear bit 5. This unmarks that pixel.

[2] For each pixel T in the vector V, do steps 3 through 4 inclusive.

[3] First, mark pixel T. It may have been unmarked in step 1. Then let N be the number of set pixels adjacent to pixel T. There will be a maximum of eight.

[4] if N > 2 then set the mark bit of T to 0, and set bit 1 ot T to 1. Bit 1 being set will mean that this pixel can be re-used.

[5] For each pixel T in V, if bit 1 of T is 0 then set T=0.

The UNMARK procedure will unmark all pixels not belonging to V, the selected pixel. It will also unmark, and give a special value to, those pixels in V that have more than two neighbors. All other pixels in V get cleared.

The final procedure to be specified is the SAVE procedure.

Procedure SAVE:

Let V be the vector to be saved, as in procedure VECTORIZE step 9.

[1] Let B be the first pixel in vector V that has not been previously used (in another vector). Let C be the last pixel in V that has not been previously used. Such pixels will have bit 2 set. Return without any output if B or C do not exist.

[2] For all pixels T in the vector V do steps 3 through 5 inclusive:

[3] If $A[T_x, T_y]=4$ then set $A[T_x, T_y] = 0$. This pixel has been used before, but cannot be used again.

[4] If $A[T_x, T_y] = 3$ then let $A[T_x, T_y] = 4$. This pixel can be re-used, but has not been used previously.

[5] If $A[T_x, T_y] = 6$ then let $A[T_x, T_y] = 4$. This pixel has been used before, and may be used again.

[6] If not all of the pixels in V have been previously used, then output pixels B and C as the endpoints of a new vector. Return.

The SAVE procedure shortens the vector V by removing duplicate pixels from the ends. It then deletes re-used pixels that are no longer needed, and flags reusable pixels using bit 2.

There are many possible extensions to this method, including:

1. Ignoring all vectors smaller than a specified length.
2. Looking at ALL possible R_i in step 6 of **vectorize** instead of just the best fit. This should produce better results, at the expense of a great deal of time and memory.
3. Omit all pixel reuse. This produces more and shorter vectors.
4. Allow the maximum distance value, currently 1, to be a parameter.

Results

The line images of figure 1 have been subjected to this vectorization procedure, and the resulting images are shown in figure 5. The relevant features of the image have been preserved, and are still available in graphical form. It is still a simple matter to find features in a specified geographic area, as might be needed by a mapping data base, but the storage needed is a great deal less than for raster images. The average reduction in storage for the images of Figure 5 is 94%. These images are 256x256 pixels, or 64K bytes each. The average size of the corresponding vector file is 4206 bytes. Further details appear in table 1. The price to be paid is, of course, processing time. For the 64K images used the conversion to vectors took an average of 23.0 seconds on a VAX 11/780 running UNIX. The time taken for a conversion is roughly proportional to the product of the average vector length and the number of vectors. The software is written in both FORTRAN and C, and the tests had to be run while other users were also using the system.

Relaxation of the distance constraint to allow pixels to be further than one unit from the line makes some sense for some kinds of sampled image. A thinning process is often applied to the image, and the result is often less than ideal. In such an instance, accepting slightly greater distances can have a positive impact on the quality of the vectors produced.

References

[1] Freeman, H., *Computer Processing of Line Drawing Images,* Computing Surveys, Vol. 6 No. 1, March 1974.

Data Obtained From Vector Conversion					
Image	# Vectors	Time	Average Length	New Size(bytes)	Storage Savings
Figure 1a	1464	29.09	5.33	5856	92%
Figure 1b	395	21.69	14.9	1580	98%
Figure 1c	1479	25.01	3.97	5916	91%
Figure 1d	868	16.32	4.12	3472	95%

Table 1

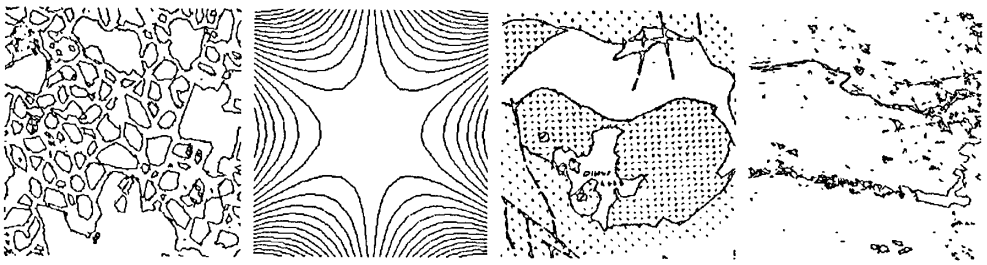


Figure 5 - Vectorized Sample Images

[2] Pavlidis, T., *Segmentation of Plane Curves,* IEEE Transactions on Computers, Vol. C-23 No. 8, August 1974.

[3] Rosenfeld, A., *Digital Straight Line Segments,* IEEE Transactions on Computers, Vol C-23 No. 12, December 1974.

[4] Davis, L.S., Rosenfeld, A., Weszka, J.S., *Region Extracting By Averaging And Thresholding,* IEEE Trans. on Systems, Man, And Cybernetics, No 5 1975, pp 383-388.

[5] Ohlander, R., Price, K., Reddy, D.R., *Picture Segmentation Using A Recursive Region Splitting Method,* Comp. Graphics and Image Processing, 8 1978, pp 313-333.

[6] Schachter, B.J., Davis, L.S., Rosenfeld, A., *Some Experiments In Image Segmentation By Clustering Of Local Feature Values,* Pattern Recognition, No 11, 1978. pp 19-28.

[7] Coleman, G.B., Andrews, H.C., *Image Segmentation By Clustering,* Proc. IEEE, No 67 1979, pp 773-785.

[8] Chow, C.K., Kaneko, T., *Automatic Boundary Detection of the Left Ventricle From Cineangiograms,* Comput. Biomed. Res., 5, 388-410.

[9] Groch, W., *Extraction of Line Shaped Objects from Aerial Images Using a Special Operator to Analyze the Profiles of Functions,* Computer Graphics and Image Processing 18, 1982.

[10] Jimenez, J., Navalon, J.L., *Some Experiments in Image Vectorization,* IBM Journal of Research and Development, Vol. 26 No. 6, November 1982.

[11] Dorst, L., Smeulders, A.W.M., *Discrete Representation of Straight Lines,* IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. PAMI-6, No. 4, July 1984.

[12] Pavlidis, T., Van Wyk, C., *An Automatic Beautifier for Drawings and Illustrations,* SIGGRAPH Conference Proceedings, July 22-26 1985, San Francisco, CA.

[13] Dorst, L., Smeulders, A., *Best Linear Unbiased Estimators for Properties of Digitized Straight Lines,* IEEE PAMI Vol. PAMI-8 No. 2, March 1986.