Smoothing and Compression of Lines Obtained by Raster-to-Vector Conversion

Eugene Bodansky, Alexander Gribov, and Morakot Pilouk

Environmental Systems Research Institute, Inc – ESRI
380 New York St., Redlands, CA 92373-8100 USA
(ebodansky,agribov,mpilouk}@esri.com

Abstract. This paper presents analyses of different methods of post-processing lines that have resulted from the raster-to-vector conversion of black and white line drawing. Special attention was paid to the borders of connected components of maps. These methods are implemented with compression and smoothing algorithms. Smoothing algorithms can enhance accuracy, so using both smoothing and compression algorithms in succession gives a more accurate result than using only a compression algorithm. The paper also shows that a map in vector format may require more memory than a map in raster format. The Appendix contains a detailed description of the new smoothing method (continuous local weighted averaging) suggested by the authors.

1 Introduction

The result of raster-to-vector conversion of a black and white line drawing can be represented as borders of connected components and as centerlines. Some definitions for the term centerline can be found in [1]. The simplest vectorization consists of building piecewise linear curves or polylines that are borders of connected components. It has a unique solution, which can be obtained without any control parameters.

There are two standard methods of building borders of connected components. The first method [1] results in borders as orthogonal polylines, lines that consist only of horizontal and vertical segments. The second method [2] results in a set of digital lines that consist of border pixels. In this case, borders in the form of polylines can be obtained with special approximation algorithms.

Post-processing borders should increase accuracy, smooth and generalize contours, defragment straight lines, arcs, and circles, and compress data. Very often, it is impossible to separate these tasks. For example, increasing accuracy while suppressing a high frequency noise gives the effect of data compression. The same effect can be achieved by defragmentation or generalization, and data compression can be used for defragmentation and generalization.

D. Blostein and Y.-B. Kwon (Eds.): GREC 2002, LNCS 2390, pp. 256-265, 2002.

It is necessary to explain what we mean by increasing accuracy: Figure 1 shows a piece of a contour map that was converted into a raster format with a resolution of 300 dpi. Any differences between contours on a source map and corresponding contours of the raster image of this map will be called errors of digital description: errors that appear because of scanning, binarization, and discretization. For error correction, we can use some *a priori* information about depicted objects.

It is known that isolines (in our case contours) cannot intersect. Hence any intersections on Figure 1 are errors. These are most likely caused by scanning and binarization. Only an operator can correct these errors. To ascertain this, it is enough to see Figure 2a. Should it be two contours (Figure 2b) or only one (Figure 2c)?

Fig. 1. A piece of a contour map scanned with a resolution 300 dpi

a) b) c)

Fig. 2. A fragment of Fig.1. a) An error; b) Two contours; c) One contour

Another type of error can be corrected automatically. It is known that contours are smooth. The main cause of failure to carry out this condition is an error in discretization. This error can be corrected with a smoothing algorithm after choosing a value of a control parameter. In this paper we will analyze only the second type of error.

The importance of the pre-processing goals listed above change as time goes by. Not long ago it was very important to compress data. We can read in the literature that this was one of the main goals of raster-to-vector conversion [3, chapter 8], but now the situation has changed. Progress in computer technology has resulted in huge

increases of available computer memory and processing speed. This is why data compression is not as critical as it used to be. Progress in computer technology has also led to another very important result: we can now use very sophisticated algorithms for smoothing and generalization that were considered unrealizable not long ago.

2 Post-processing Algorithms

All existing algorithms that can be used for post-processing can be divided into three groups:

a) Deletion of some specific features of curves (for example, arms and narrow peninsulas from a coast line).
b) Deletion of some vertices of polylines.
c) Approximation of lines with some mathematical functions.

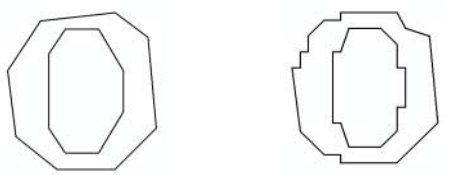
For a rigorous solution of generalization tasks, the best algorithms are those of the first group. Because these algorithms are essentially recognizing algorithms, they are very complex. In [4], the development of such algorithms is characterized as a "fantastically complex problem." Moreover, they have a very high computational complexity. The generalization of different objects may require different algorithms, as it is unlikely that the algorithm used for cost line generalization can be used to generalize roads, contours or building silhouettes. There is some progress in developing such algorithms, but at the beginning of the sixties, the difficulties of development and implementing such algorithms seemed so insuperable that the algorithms of the second group were used not only for compression but also for generalization.

Obviously, algorithms of the second group have to be used for data compression and straight-line defragmentation. But until recently, they were also used for generalization (in cartography) and smoothing. Many algorithms of this group [4-7] were developed. These algorithms result in a new polyline with vertices (critical points) that are a subset of vertices of the source polyline. Critical points are the points that determine the shape of the processed polyline. The distances of other vertices (noncritical points) from the resulting polyline have to be less than some threshold *h,* chosen by an operator. The algorithms of this group differ from each other in inessentials.

The author of [8] wrote that the Douglas-Peucker method is "mathematically superior" and the author of [9] suggests that "the generalizations produced by the Douglas algorithm were overwhelmingly – by 86 percent of all sample subjects – deemed the best perceptual representations of the original lines." But it is necessary to note that all these algorithms, including the Douglas-Peucker algorithm, have a common shortcoming: errors of vertices of resulting polylines are equal to errors of critical points of the source polylines, and therefore they cannot be reduced by decreasing the control parameter h. Because of this, resulting polylines are uneven, orientation of segments of resulting polylines depends on errors of the critical points' coordinates, and these algorithms cannot be used to filter random noise.

The lower the resolution of the image, the more noticeable the errors are. These shortcomings cause undesirable consequences when these algorithms are used for map generalization, defragmentation of straight lines, or noise suppression. Figure 3 shows a connected component and Figure 4 shows the borders of this connected component after processing with the Douglas-Peucker algorithm, with values of the threshold h = 1 and h = 0.75.

Fig. 3. Raster connected component



h = 1.0 *h* = 0.75

Fig. 4. Border of the connected component after compression with different values of a threshold (*h)*

We have already said that contours have to be smooth. The source image (Figure 1) contains a large amount of binarization noise. Comparing borders before and after compression, it is easy to conclude that it is easier to smooth source borders manually than to smooth compressed borders manually. The same conclusion may be drawn for straight lines. To define the correct direction of straight segments after compression is more difficult than to define direction of these segments before compression. So we can conclude that compression algorithms at least sometimes result in irreversible loss of information about a noise.

3 Smoothing and Accuracy

The most appropriate algorithms for suppressing a discretization noise (at least for contours) are those of the third group, which use mathematical functions to approximate the original polylines. They include algorithms that define straight lines and polynomials with the least square method and algorithms of local smoothing, which are especially important for cartography. An approximation can be done for a whole polyline (global approximation), or sequentially, for small pieces of polyline (the local approximation). In [4], several such methods are described and their shortcomings are listed:

a) The resulting curves are smoother than real objects.
b) These methods look for an average behavior of polylines and consequently corrupt the critical points that define a polyline shape.
c) Processed polylines contain more points than source polylines.

We have objections to all these conclusions:

a) Methods of this group use a control parameter, which defines a power of smoothing. An operator can choose a value for this parameter that gives the required power of smoothing. Figure 5 shows polylines obtained with the same algorithm of smoothing but with different values of the smoothing parameter (d = 3 and d = 5). Developed by A.Gribov, this algorithm is described in the Appendix.
b) There are different methods for looking for critical points [9], based on evaluation and analysis of a curvature and the first derivation of a curvature. If critical points are found, it is possible to split a curve at critical points before smoothing. Each of the new curves can be smoothed with fixed end points.
c) Let us try to understand why this algorithm can produce too many points. These algorithms can calculate smoothed curve points with any given step. Usually this step is constant, i.e. distances between any two consecutive points are almost equal. The appropriate number of points required for a good approximation of the curve with a polyline depends on the nature of the processing curve and is defined by the operator. Extra points can appear if the chosen step was too small or if curves contain long segments with a very small curvature. It is possible to delete extra points by increasing a step or by using compression algorithms. We explain in the next section why using smoothing and compression algorithms consecutively gives a better result than simply using the same compression algorithm.

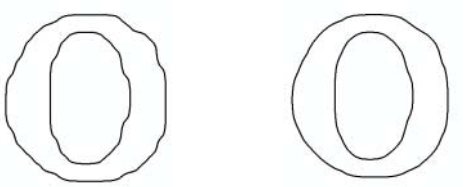


Fig. 5. Border of the connected component after smoothing with different values of a smoothing parameter (d)

4 Using Smoothing and Compression Consecutively

Consider one essential difference between the algorithms of the second (compression) and the third (approximation) groups. For the second group, we have already noticed that all vertices of the processed polyline are a subset of the vertices of the source polyline. That means that along with the vertices we inherit their errors so that these algorithms are unsuitable for noise suppression. Methods of the third group produce

polylines with vertices that do not coincide with vertices of the source polyline; they can, therefore, be used for accuracy enhancement.

Assume that a smoothing algorithm produced points located on the original smooth curve without any errors. Let us plot a polyline using these points as vertices. If the vertices were calculated with a small step, the polyline would be a good approximation of the smooth curve. If we process this polyline with a compression algorithm, the vertices of the compressed polyline would be free of error. The maximum distance between the source and resulting polylines would be less than the threshold h, i.e. we could build the approximation with any given accuracy. Of course, smoothing algorithms produce the result with some error. But if the control parameter of smoothing is selected correctly, errors of the smoothed polylines will be much less than errors of the original polyline. Processing this smoothed polyline using a compression algorithm with small enough threshold h will not increase the error enough to matter.

Figure 6 shows the border of the connected component after smoothing with the algorithm described in the Appendix and with the parameter of smoothing $d = 5$, followed by processing with the Douglas-Peucker algorithm with the threshold h equal to 0.25 and 0.5, respectively. Comparing the polylines shown in Figures 6 and 4 visually, it is easy to arrive at the conclusion that accuracy increases if you use smoothing before compression. The polylines ($h = 0.5$) shown on Figure 6 (smoothing with compression) have only 5% more vertices than the polylines ($h = 0.75$) shown on Figure 4 (only compression).

d = 5, h = 0.25 d = 5, h = 0.50

Fig. 6. Border of the connected component after smoothing with the same value of a smoothing parameter (d) and compression with different thresholds (h)

5 Different Formats and Requirement for Memory

Fairly often, you can read that vector representation of maps, engineering drawings, and other line drawings is much more economical than raster representation, and that this is one of the main incentives for raster-to-vector conversion [3]. Maybe this is true for drawings consisting mainly of straight lines and circles, such as engineering drawings and electrical schematics, but it is not true for maps and other documents consisting mainly of irregular curved lines. To show this, we evaluated the memory required for storing the piece of a contour map shown in Figure 1 in both raster and vector formats.

The raster image of this fragment consists of 293 rows and 879 columns of 257547 pixels. Because raster-to-vector conversion systems usually process black and white images, only 1 bit is required for each pixel. So for storing the image in uncompressed raster format, approximately 32 Kbytes will be required. If the image will be converted into the run-length encoding (RLE) format, the required memory will decrease to 24 Kbytes.

The exact borders of all connected components shown on Figure 1 include 14169 points. To store information about each point will require 16 bytes (2 coordinates in the double format). So for the exact borders in the vector format, more than 220 Kbytes will be required, or almost 7 times more than for the raster format. A vector database saves not only coordinates but also other information about the geometry and topology of image features: area of polygons, length of perimeters of the polygons, number of holes, and so on. After smoothing (for the smoothing algorithm, see the Appendix) with the power of smoothing d = 5 and d = 3, the number of points increased to 26921 and 48174, respectively.

If, after smoothing with $d = 5$, the resulting image is processed with the Douglas-Peucker algorithm with parameter $h = 0.25$, then the number of points will be reduced to 5872, but all the same a vector representation of the map fragment will require much more memory (about 92 Kbytes) than the raster representation.

If the image consists of linear elements (as Figure 1), it can be represented in a vector format as a set of centerlines with their width. That will require almost two times less memory in comparison with borders. If the float or short format is used instead of the double format, the requested memory will be even less, but it will still be comparable with the memory required for the raster image.

The analyzed example allows one to conclude that memory compression cannot be the main reason for raster-to-vector conversion of line drawings (at least for maps). Obviously this transformation is performed for other, more important goals. That is why the importance of this problem was not reduced by the enhancement of computers and the tremendous increase in available computer memory.

6 Conclusion

The raw results of raster-to-vector conversion of linear drawings often require post-processing, usually by generalization, smoothing, and compression. Smoothing can be used to increase accuracy by depressing discretization noise. Increasing accuracy is especially important for processing images with a low resolution. Smoothing can be implemented with the algorithm of local averaging. The continuous local weighted averaging algorithm used in this paper is given in the Appendix. This algorithm has some advantages; particularly that it does not require preliminary densification and that it attains a worst-case running time of $O(n)$, where n is a number of vertices of a source polyline.

Compression algorithms do not increase the accuracy of the vectorization, but they can be used after smoothing for reducing the required memory with little diminution of accuracy.

A vector format of map representation may require more memory than a raster one, but raster-to-vector conversion is performed not for saving memory but for other

more important goals. That is why the significance of vectorization was not reduced by the enhancement of computers and the tremendous increase of available computer memory.

References

1. Bodansky, E., Pilouk, M.: Using Local Deviations of Vectorization to Enhance the Performance of Raster-to-Vector Conversion Systems. IJDAR, Vol. 3, #2 (2000) 67-72.
2. Quek, F.: An algorithm for the rapid computation of boundaries of run-length encoded regions. Pattern Recognition, 33 (2000) 1637-1649.
3. Ablameyko, S., Pridmore, T.: Machine Interpretation of Line Drawing Images. (Technical Drawings, Maps, and Diagrams.) Springer (2000).
4. Douglas, D., Peucker, Th.: Algorithm for the reduction of the number of points required to represent a digitized line or its caricature. The Canadian Cartographer, Vol.10, #2 (1973) 112-122.
5. Ramer, U.: Extraction of Line Structures from Photographs of Curved Objects. Computer Graphics and Image Processing. Vol.4 (1975) 81 – 103.
6. Sklansky, J., Gonzalez, V.: Fast Polygonal Approximation of Digitized Curves. Pattern Recognition, Vol.12 (1980) 327-331.
7. Curozumi, Y., Davis, W.: Polygonal approximation by minimax method." Computer Graphics and Image Processing, Vol.19 (1982) 248-264.
8. R.B.McMaster Automated line generalization. Cartographica, Vol. 24, #2, 1987, pp. 74-111.
9. E.R.White. "Assessment of line-generalization algorithms using characteristic points." American Cartographer, Vol. 12, #1, 1985, pp. 17-27.
10. H.Asada, M.Brady. "The curvature primal sketch." IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol.8, #1, 1986, pp. 2-14.

Appendix: Smoothing with Continuous Local Weighted Averaging

We analyze curves that are approximated with piecewise linear polylines. The smoothing algorithm is based on an approximation of a source curve with a mathematical function. A local averaging algorithm calculates points of a smoothed curve, taking into account only a few neighboring points of the source curve. In [10], coordinates of points of the smoothed polyline are calculated as the convolution of the coordinates of points of the source polyline and the Gaussian.

Our smoothing method differs from them by the convolution kernel. Instead of the Gaussian we used function $e^{-\frac{|t-\tau|}{d}}, d > 0$, where ($\tau$) is the length of a path along source polyline. It gives us the possibility to build the algorithm that attains a worst-case running time of *O(n),* where *n* is a number of points of the smoothed polyline. Besides, a local approximation is performed with the second order polynomial

$$u_\tau(t) = \alpha_0^u(\tau) + \alpha_1^u(\tau)\cdot(t-\tau) + \alpha_2^u(\tau)\cdot(t-\tau)^2$$

This equation represents two equations, which can be obtained by substitution x or y for u.

Values of coordinates $x(\tau)$ and $y(\tau)$ are calculated as

$$x(\tau) = \alpha_0^x(\tau),$$
$$y(\tau) = \alpha_0^y(\tau),$$

where $\alpha_0^u(\tau)$ is the solution of the next equation

$$\sum_{i=0}^{2} c_{i+j}(\tau)\,\alpha_i^u(\tau) = b_j^u(\tau),\ j=\overline{0,N},\ \text{where } c_k(\tau) = \int_0^L e^{-\frac{|t-\tau|}{d}}(t-\tau)^k\,dt,$$

and

$$b_k^u(\tau) = \int_0^L e^{-\frac{|t-\tau|}{d}}(t-\tau)^k\,u(t)\,dt$$

A calculated curve has smooth first derivatives at calculated points.

To simplify the expressions, we assign $d=1$, and the influence of this coefficient on the power of smoothing takes into account the value τ.

$$I_i(\tau) = \int_0^L e^{-|t-\tau|}(t-\tau)^i\,u(t)\,dt = I_i^L(\tau) + I_i^R(\tau),$$

where

$$I_i^L(\tau) = \int_0^\tau e^{t-\tau}(t-\tau)^i\,u(t)\,dt,\ I_i^R(\tau) = \int_\tau^L e^{\tau-t}(t-\tau)^i\,u(t)\,dt$$

$$I_i^L(\tau_0) = I_i^L(0) = 0;\ I_i^R(\tau_n) = I_i^R(L) = 0;$$

$$I_i^L(\tau_{k+1}) = \int_{\tau_{k+1}}^L e^{t-\tau_{k+1}}(t-\tau_{k+1})^i\,u(t)\,dt =$$

$$= e^{\tau_k-\tau_{k+1}}\sum_{j=0}^{i}\left\{C_i^j(\tau_k-\tau_{k+1})^{i-j}\,I_j^L(\tau_k)\right\} + \int_{\tau_k}^{\tau_{k+1}} e^{t-\tau_{k+1}}(t-\tau_{k+1})^i\,u(t)\,dt \qquad (1)$$

$$I_i^R(\tau_{k-1}) = \int_{\tau_{k-1}}^{L} e^{\tau_{k-1}-t}(t-\tau_{k-1})^i u(t)dt =$$
$$= \int_{\tau_{k-1}}^{\tau_k} e^{\tau_{k-1}-t}(t-\tau_{k-1})^i u(t)dt + \quad (2)$$
$$e^{\tau_{k-1}-\tau_k}\sum_{j=0}^{i}\left\{C_i^j(\tau_k-\tau_{k-1})^{i-j} I_j^R(\tau_k)\right\}$$

If at first you calculate the integrals $\int_{\tau_k}^{\tau_{k+1}} e^{t-\tau_{k+1}}(t-\tau_{k+1})^i u(t)dt$ and $\int_{\tau_{k-1}}^{\tau_k} e^{\tau_{k-1}-t}(t-\tau_{k-1})^i u(t)dt$, the integrals $I_i^L(\tau_k)$ and $I_i^R(\tau_k)$, where τ_k is a parameter corresponding to the vertex of the source polyline, can be calculated with the recursive equations (1) and (2). Omitting u(t) we receive formulas for calculating $c_k(\tau)$. To calculate $u_\tau(t), \tau \neq \tau_k \forall k$ it is possible to use the formulas written above, substituting τ_{k+1} and τ_{k-1} to τ in (1) and (2), respectively. This algorithm also gives good results for smoothing centerlines.