Generalization and PostScript

Dezso Nagy, Geodetic Survey Division, Geomatics Canada
615 Booth Street, Ottawa, ON. Canada K1A E09

Dean D. Nagy chillipepper communications
8 Stonebriar Drive, Ottawa, ON. Canada K2G 5Y2

ABSTRACT

With the constantly increasing volume of graphical information in digital form, the automatic processing of this information for drawing vectors becomes more and more important. This usually means selecting vectors from the database, generalizing (data compression) and/or smoothing as needed. There are a number of application programs, which are helpful to perform this. However, occasionally one wants to have more specific control over the procedure. In this presentation our target is a simple vector drawing procedure. We propose to find a continuous smooth graphic representation of a polygon (open, or closed) given by *m* discrete points. We want to make it device independent. In order to do this, we shall make maximum use of the PostScript programming language. After a very brief introduction to the subject of generalization, the advantages of the parametric form, particularly the Bézier curve – which is the key to the procedure – is discussed in some detail. A few relevant operators of PostScript, the procedures to obtain our desired approximations, both interactively and automatically, will be presented. Some examples will be used to highlight the salient points.

Keywords: Generalization; Bézier curve; Interactive; Automatic; PostScript

1. Introduction

With the explosion of the collection of geospatial data in digital form, the organization of this information, the constant update and the selection of subsets of the data for various specific projects becomes a major problem. In addition, the lack of standards for exchanging this information and the presentation of it in different platforms requires major programming efforts. While there are a number of programs satisfying most queries, sometimes the user request is not met. In the following, as a complement to existing application programs, we target a very specific problem, namely vector generalization, and develop some procedures to satisfy this need. Our solution, whether interactive or automatic, relies on the use of Bézier curves. Because the aim is to produce device independent solutions, use of some operators of the PostScript programming language will be made, which fulfill such requirement.

While the application programs are extremely useful to carry out standard routine jobs, there are some consequences of their use. One of them is the periodic upgrade, which not only costly, but may interrupt the flow of processing, prone to occasional error with the associated pain and frustration of the user, requires retraining and sometimes drops one favorite procedure from the new version. In addition to this the user may lose insight of what goes on. Our proposal for vector generalization attempts to overcome some of the points mentioned above. Although it requires more work on the part of the implementer, the procedure can be targeted to make use of all available local facilities (database, distribution of output, programming skills, etc.), it provides a better understanding of the problems, and implementation is simple and very device independent.

Because of its major role in our proposed method, the properties of Bézier curves will be discussed briefly. Then after touching upon the problem of cartographic generalization, the method of obtaining interactive and automatic Bézier curve approximation is given. Next, because of its relevance to our proposed method of generalization, some pertinent operators of the PostScript programming language will be described. The paper concludes with some examples of the use of Bézier curves.

2. Bézier curves

Let us assume, that a vector is given by *m* pairs of points, in a Cartesian coordinate system. What is needed is a computational procedure either to weed out some points without introducing noticeable change when the vector is drawn and/or provide smooth representation. There are a large number of procedures for eliminating some points of the vector. However when a smooth representation is demanded, there is some problem with the Cartesian form: it cannot have loops.

For this reason, the parametric form of vector representation must be established. One suitable form is provided by the Bézier polynomials. Consider the useful properties of this form. Although higher than third degree polynomials could be used, in practice the cubic form is the one used mostly in the applications because it is simple, it is easy to use it to shape, it can easily be used to approximate more complex vectors by segmentation, and most importantly in PostScript, which provides us with true device independence, an operator is available to render it. Because all of these reasons in the rest of the paper when we refer to a Bézier curve we mean a cubic Bézier polynomial. Now we shall discuss some basic properties of this curve.

The technique of using Bézier curves (not coined yet at that time) was developed in the early 1960s, mostly by people involved in automobile manufacturing for the purpose of defining various shapes of automobiles, which were then machined by computer driven software. Because of secrecy, few details of the technique were available. Although Bézier may not have been the first to use the curves named after him, it was his contribution that was widely published and serves as the standard reference on the subject.

Before discussing the mathematical details, some basic properties of the Bézier curve will be summarized:

- requires only data points (no derivatives are needed),
- passes only through the endpoints,
- has tangency at the endpoints (discussed later),
- bounded by the polygon formed by the data points,
- the degree can be varied from segment to segment,
- the segments can be joined with various degree of continuity,
- it is suitable for interactive development,
- extension to higher dimensions is simple,
- implementation on PCs is easy.

Here only two-dimensional cubic Bézier curves defined over four points will be treated. To use higher degree curves or to join the piecewise approximations, the reader is referred to the references. The equation for the cubic Bézier curve in parametric form is given below:

$$x = (1-t)^3 x_0 + 3t(1-t)^2 x_1 + 3t^2(1-t)x_2 + t^3 x_3,$$
$$y = (1-t)^3 y_0 + 3t(1-t)^2 y_1 + 3t^2(1-t)y_2 + t^3 y_3.$$

Four points define it: the two endpoints (x_0, y_0, x_3, y_3) and the two control points (x_1, y_1, x_2, y_2). The points on the curve can be calculated by specifying the parameter `t`, which runs from `0` to `1`. By substituting `0` and `1` for t into the above equation, it can be seen that the curve passes through the first and the last points. It can easily be verified that the tangent at the endpoints passes through the two control points (take the first derivative, then substitute `t=0` and `t=1` respectively). This property can be used when the joining of Bézier curves with continuity (for smooth joining) is required.

3. Generalization

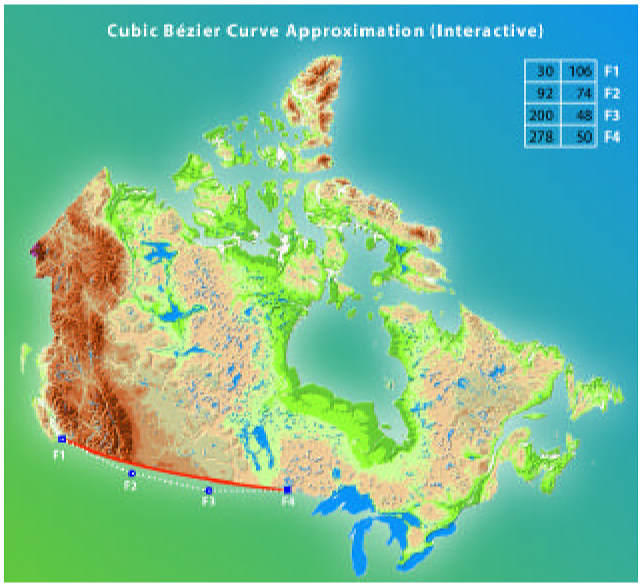
'Generalization: art clarified by science' [Eckert 1908]

This quote may be the first reference to generalization and defines the process completely. It states the interaction between science (various computer algorithms) and art (cartographic knowledge) as it applies to generalization in cartography. In the last 100 years various procedures have been developed to carry out generalization. With the increase of availability of inexpensive computing facilities, the interest focused on automatic generalization. However many believe that the generalization of a complex map can never be done automatically. Probably a large part of the work can be carried out by automatic procedures involving easily defined mathematical algorithms, and the artistic (cartographic) element can be added by some interactive procedure. Generalization encompasses such a variety of topics that even a brief survey of it is beyond the scope of this short note. Instead we provide some references for the reader. In addition to the references listed in the paper, the reader should consult the recent issues of cartographic journals. Here we only give a few lines about our purpose to provide some guidelines to carry out both an interactive and an automatic vector approximation (generalization) process using Bézier curves.

Generalization may consist of at least two steps: first the weeding out of points, which do not contribute to the visual representation. In this step, based upon some criteria, some points are eliminated from the data but no change is made to the remaining points. This step may be followed by the second process, in which the input is either interpolated by some function (i.e. from the function all input points can be reproduced), or approximated (i.e. the input points usually cannot be reproduced). In case of using Bézier curves, the process is one of approximation: apart from the two endpoints, the original data are not on the curve. When developing some approximation to get to a mathematical form, the measure of the goodness of the fit must be established. In most cases this is a function of the distances measured between the input points and the approximating function. So the process involves a given tolerance. Because, as stated earlier, a cubic Bézier is used for approximation, it is not expected that a single approximation with an arbitrary tolerance can be obtained. This means that the number of iteration is another preset parameter in the process: if the tolerance cannot be achieved within a given number of iterations the input must be segmented and the process must be carried out for each segment. The last step, when segmentation is required, is how to join the segments with some continuity. In conclusion, the procedures outlined above not require preprocessing. The use of the parameters: tolerance, number of iterations and segmentation are sufficient to achieve a smooth approximation for any input data.

3.1 Interactive Bézier curve approximation

The basic idea is to be able to draw a Bézier curve using simple tools. As seen earlier, four points define the curve. The function keys **F1, F2, F3** and **F4** are used to get the input to the program. As soon as the points are defined (move the cursor to the desired position and left click), the curve is drawn. Move the cursor to any of the four points, then drag: the new curve is drawn. By moving different points, one can easily learn what approximation is possible with a cubic Bézier. One can display the values of the four points, which, can be output if requird. It is of course easy to extend the program and read in a vector, which can be displayed. Then picking up the two points as *endpoints* from the vector plot (the two endpoints can be *any two points of the vector*), and two *control points*, (the control points naturally will never be *points of the vector*), one can try to interactively move the control points until acceptable approximation is found. If this is not possible, then segmentation may be the solution: move one of the endpoints to shorten the vector size, and try again. Instead of a vector, a graphic file may be displayed. The great advantage of this interactive Bézier curve generation is, that it *converts graphic information into a device independent digital form*. Figure 1. shows the generation of such a transformation: the 49th parallel (graphic) is turned into a device independent form (Bézier curve).



30	106	F1
92	74	F2
200	48	F3
278	50	F4

Figure 1. The 49th parallel showing the International Boundary between the USA and Canada

3.2 Automatic Bézier curve approximation

As mentioned earlier, in order to judge the goodness of the approximation the distance from the points of the vector to the curve must be calculated. In order to do this the connection between the Cartesian and the parametric form must be established. As seen earlier in the parametric form, `t` is the parameter that runs from `0` to `1`. So the problem is to relate the x,y coordinates to `t`. This can only be done by an iterative process. Once the distances from each point of the vector to the curve are calculated, the maximum is compared to a desired tolerance. If this test is satisfied, then the sought Bézier approximation is found and the process is terminated. If the test fails, then better values of the parameters corresponding to the points of the vector are calculated (this involves recalculating the two tangents at the endpoints and refining the position of the control point along the tangent). This should produce a better approximation. Distances are calculated again and tested against the tolerance. If the test is satisfactory, the job is finished; otherwise another iteration is done. Practice shows that convergence is fast and if the vector can be approximated by a single Bézier curve then usually less than 5 iterations are sufficient. If the process does not converge, then the vector is split at the maximum residual, and the procedure described above is repeated for each segment. If several segments are required for proper smooth approximation, the program can include provision for continuity by picking control points between segments *on the tangent vector* of the appropriate endpoints. Note that the process involves least squares fitting and the use of the Newton-Raphson method for iteration. Obviously the detailed discussion of the process is beyond the available space. However in Figure 2, we show how the tolerance, the number of Bézier curves and the number of iterations behave on an 11 point vector data calculated at `0.1` increment of `t` from a Bézier curve. The tolerance is given in point units (1"=72pt). As can be seen with the tolerance set to 10, after 6 iterations, a single Bézier curve (shown under **B**) is found. Tolerance 5 from the first iteration provides the approximation consisting of 3 Bézier curves. The result is the same even after 99 iterations. Finally with tolerance of 2, no less than 7 Bézier curves must be used. As can be noticed on the most detailed approximation, the 7 segments are joined well using the above-mentioned geometric continuity between segments.

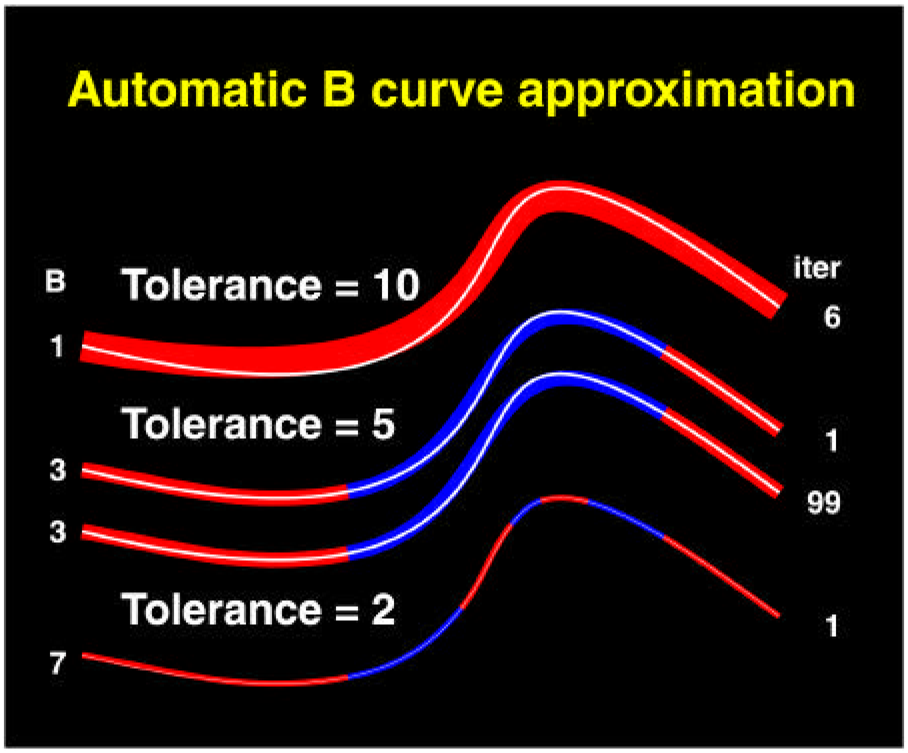


Figure 2. The relationship amongst the tolerance, the number of Bézier segments and the number of iterations

4. PostScript

PostScript is a programming language specifically designed to make marks on a page, be it font or line or the combination of both. The language has a large number of instructions, including operators for evaluation of mathematical functions, testing, looping, and handling fonts of practically any size and a large number of different families. Paths can be constructed from the outline of fonts, or can be built by using different path elements, such as lines, arcs and Bézier curves. Any closed path can be used for filling, or clipping. PostScript provides all the operators needed for carrying out the above tasks.

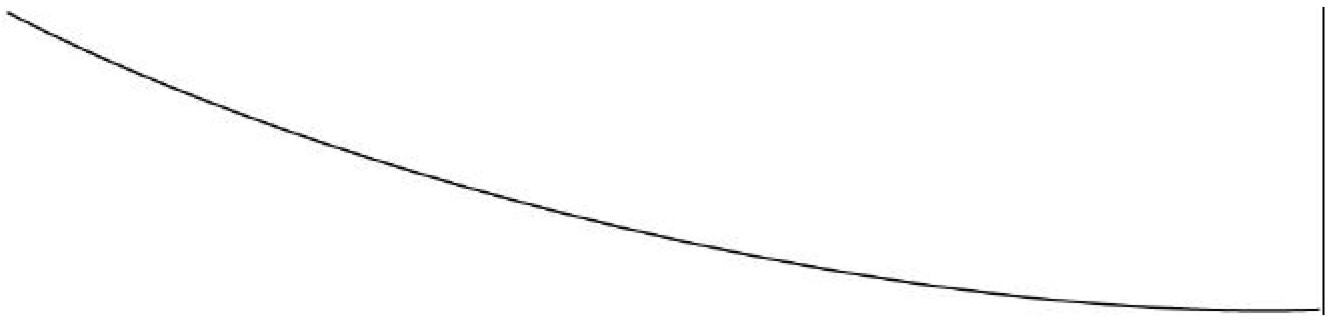
One of the important features of the language is that it is device independent, as far as the user is concerned. That is, the same program, without any change, can be sent to various devices of resolutions from 300 dots per inch to 4000 dpi. This means that development can be proofed on inexpensive low-resolution devices, and one must pay the higher cost of high resolution imaging only for the final product.

In many cases in cartography, there is a need for smooth representation of experimental data, such as digital coastlines, roads, boundary lines, etc. One of the very useful PostScript operator for this purpose is the curveto operator, which draws a Bézier cubic section. In the following, the Bézier curve (cubic only) will be discussed briefly. Then, a few examples will be presented showing the use of Bézier curves.

In Figure 1 a Bézier curve was shown as a representation for the 49th parallel. This segment of the International Boundary between the USA and Canada consists of 1,020 Monuments. It is about 2009 km in length. This curve is shown as Figure 3, and to give an idea about the power of PostScript, the code is presented, which draws the curve in a device independent way. The program can be printed as shown on any PostScript printer, or viewed with appropriate programs. Here is the code:

```
%!PS
60 612 moveto  183 548 400 495  555 500 curveto stroke
showpage
```

Using default settings (for line width, color, background, etc.) the output is shown in Figure 2 below:



3. The 49th parallel (consisting of 1,020 Monuments), drawn from a single Bézier curve

The following PostScript operators may be used to *dress-up* the curve:

```
setrgbcolor
setlinewidth
fill
moveto
lineto
```

for drawing frame, fill background, specify color and line thickness.

5. Examples

Figure 4 is a part of a diagram done in PostScript showing the length of the day in Ottawa, Canada (not showing the daylight saving time shift). The sunrise and sunset is calculated to the nearest minute for each day, requiring 730 points for plotting. Using PostScript, to plot it needs ony14 points (2 Bézier curves joined with first order continuity. Of course the additional advantage of the Bézier curve representation is not only that it requires much less data, but that even on a wall-size chart the dividing line between sunrise and sunset will be smooth at any plotting resolution.

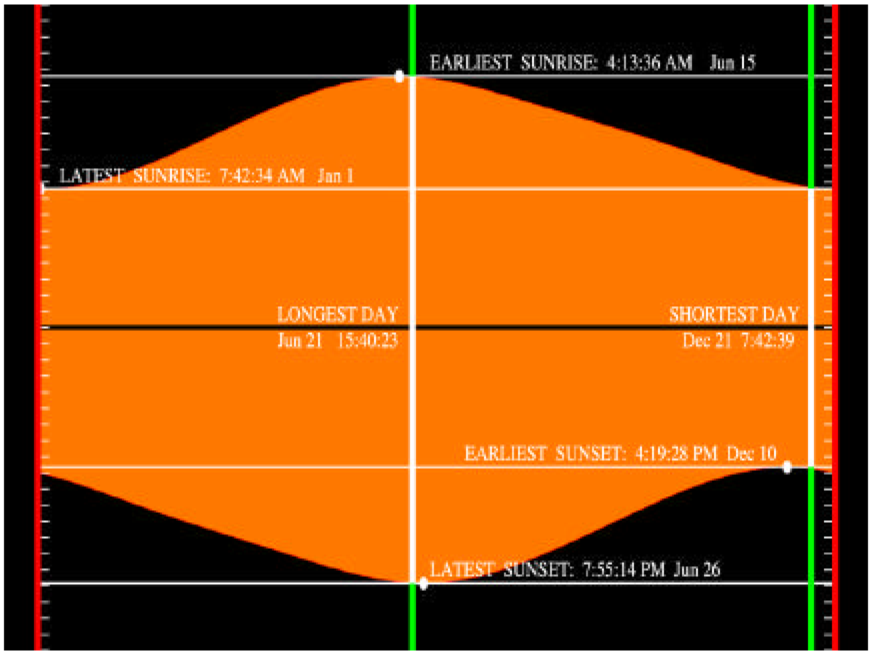


Figure 4. SUNRISE - SUNSET in Ottawa

Figure 5 demonstrates the use of automatic fitting of vector data with a given tolerance. The input is a vector obtained by digitizing a character, and it consists of 51 points. For the approximation a 10 pt. tolerance level was set. The procedure described above for automatic approximation found seven Bézier segments. The process joined the segments with first order continuity.

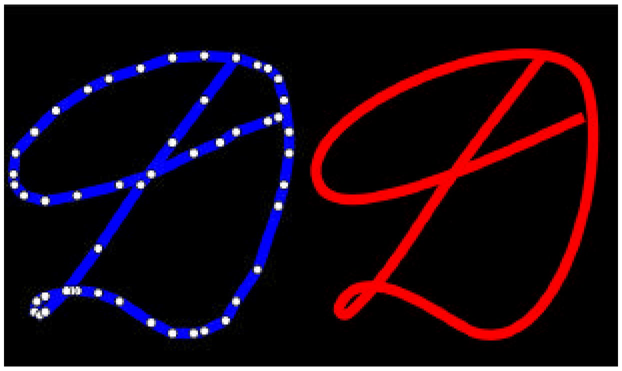


Figure 5. Vector and automatically approximated Bézier representation of a vector

6. Summary

In this short note methods to obtain Bézier curve representation from various forms (graphics, digital data in Cartesian form, free hand), both interactively and automatically, are outlined. Their applications are shown on a few examples. Because the Bézier curve is most advantageous when used in connection with the `curveto` operator of the PostScript programming language, an introduction to this language is also included. Although there is no space to discuss it here, a very major use of the Bézier curves is in font generation. Most of the fonts are designed using only straight lines and Bézier curves: arcs are really not needed at all, because a Bézier curve can reproduce a circle with no noticeable error. Investing the necessary time to learn the very few operators, many of the projects needed in preparation of research papers, including diagrams and maps can be done with reasonably simple tools. One may use another programming language (for example Fortran or C++), in which computations (such as preprocessing of data if needed, map projections, etc.) can be carried out. In addition it is very easy to generate in these programs the necessary simple PostScript code. Then the output can go directly either to a PostScript printer, or to a viewing program.

The hope of the writers is that this paper may prompt the reader to investigate applications to solve projects, which are not treated well by the available application programs.

References

Adobe Systems Incorporated (1999) *PostScript Language Reference, Third Edition.* Addison-Wesley

Bartels HR, Beatty J C, Barsky BA (1987) *An Introduction to Splines for use in Computer Graphics and Geometric Modeling.* Morgan Kaufmann Publishers, Inc. Los Altos, California 94022

Bézier P (1966) *Definition numerique des courbes et surfaces I.* Automatisme, 11, pp 625-632

Bézier P (1967) *Definition numerique des courbes et surfaces II.* Automatisme, 12, pp 17-21

Bézier P (1972) *Numerical Control; Mathematics and Application.* Translated by R. Forrest, Wiley, New York

Bézier P (1986) *The Mathematical Basis of the UNISURF CAD System.* Butterworths, London

Braswell FM (1989) *Inside PostScript.* Peachpit Press, Inc. Berkeley California

Coons S (1964*) Surfaces for computer aided design.* Technical Report, M.I.T.

de Boor C (1978) *A Practical Guide to Splines.* Springer

de Casteljau P (1959*) Outillages methodes calcul.* Technical Report, A. Citroen, Paris

de Casteljau P (1963) *Courbes et surfaces a poles.* Technical Report, A. Citroen, Paris

Douglas DH, Peucker TK (1973) *Algorithms for the Reduction of the Number of Points Required to Represent a Digitized Line or Its Caricature.* The Canadian Cartographer, Vol. 10, No. 2, pp 112-122

Foley J, Van Dam A (1982) *Fundamentals of Interactive Computer Graphics.* Addison-Wesley

Forrest AR (1972) *Interactive interpolation and approximation by Bézier polynomials.* Computer Journal, 15, pp 71-79

Kunkel G (1990) *Graphic Design with PostScript.* Scott, Foresman and Company, Glenview, Illinois

McMaster RB (1987) *Automated Line generalization.* Cartographica, Vol. 24. No. 2, pp 74-111

McMaster RB (1989) (ed) *Numerical generalization in cartography.* Cartographica, Vol. 26, No. 1, 121 pages.

McMaster RB, Shea KS (1992) *Generalization in Digital Cartography.* Association of American Geographers

Nagy D (1979) *Computer memory expansion and its uses in large datafile handling.* Presentation. AGU Spring Annual Meeting, Washington, D.C. USA.

Nagy D (1991) *Introduction to Bézier curves.* Acta Geod. Geoph. Mont. Hung. **26**, pp 19-28

Nagy D (1992a) *A brief presentation on the use of PostScript in Cartography.* SORSA '92, University of Ottawa.

Nagy D (1992b) *Application of PostScript in Cartography.* 27th International Geographical Congress, July, Washington, D.C. USA

Nagy D, Nagy DD (2002) *Vector generalization in GIS*. Appearing in the 2002 GEOTec Event Proceedings Volume, April 8-11, 2002, Toronto, Ontario, Canada.

Plass M, Stone M (1983*) Curve Fitting with Piecewise Parametric Cubics.* Computer Graphics, Vol. 17, No. 3, July.

PostScript Language (1985) *Tutorial and Cookbook.* Addison-Wesley Publishing Company, Inc.

PostScript Language (1988) *Program Design.* Addison-Wesley Publishing Company, Inc.

Reid GC (1990) *Thinking in PostScript.* Addison-Wesley Publishing Company, Inc.

Roth SF ed. (1988) *Real World PostScript.* Addison-Wesley Publishing Company, Inc.

Schneider PJ (1990) *An algorithm for automatically fitting digitized curves.* In Graphics Gems ed. Andrew S. Glassmer. Academic Press, Inc. Boston. pp 612-626