ELECTRONIC PUBLISHING, VOL. 6(3), 183–194 (SEPTEMBER 1993)

Variable width splines: a possible font representation?

R. VICTOR KLASSEN

Xerox Webster Research Center
Building 128–27E, 800 Phillips Road,
Webster, NY 14580, USA.

email: `klassen.wbst128@xerox.com`

SUMMARY

Many fonts derive from stroke-based ancestry. Pressure applied to the pen or brush provided some variation in the stroke width, which defined a region on each side of a centreline. A simple representation of fonts as variable width strokes is presented in this paper. Advantages include a good first step toward typographic scaling (stroke width scales independently of overall scale factor), and preservation of topology at low resolutions (minimum stroke width can be enforced). A chief disadvantage is the lack of experience designing fonts in this paradigm, or building routines to convert from other paradigms.

KEY WORDS Strokes Scaling Typography Offset curves Variable width splines

1 INTRODUCTION

Many fonts originate in a form that was stroked by hand, either with brush or pen. Some were modified as technologies changed, and as computerized typesetting was introduced, further modifications ensued. Font representations have included bitmaps and contours. Both require that the original shapes be modified, and neither is fully satisfactory. METAFONT offers another alternative font representation [1], however it appears best suited to mathematicians and programmers, as it is more a constraint specification language than a convenient way to describe shapes as understood by a typical font designer. It is also a relatively slow format for rasterization, compared to outlines. The expressiveness of METAFONT may be sufficient to describe any font; the language may simply be too foreign for many of the better type designers to find out.

Traditionally, type was created letter by letter by the designer. The notion of scaleable fonts was meaningless, as naming a font implicitly specified its size. Some time before computerized typesetting, type was geometrically scaled. Updike (in 1922) deplored the use of the pantograph for mechanically making type of multiple sizes from a single master [2].

By 1964, the makers of the Linotron were unashamed to describe their system for selecting the size of type: a system of lenses optically scaled the characters in the master before they were scanned, selected and written on to the film or plate [3]. So it should have come as no surprise when the PostScript language provided the means to geometrically scale any font to any size, with no concern for the aesthetic pain that might cause. Truly

CCC 0894–3982/93/030183–12 *Received 1 September 1993*
Revised 3 December 1993

good typography requires that letters differ from size to size. At the very least, stroke widths and kerning tables should not scale in the same way as character height. This is what I shall call typographic scaling. It has been called optical scaling by some — an unfortunate choice of term: optical scaling is exactly what was performed in the Linotron machines! Of typographic scaling, altering stem widths is probably the most difficult part; variable width strokes may offer, at the very least, a good start.

There are many well-known problems with bitmap fonts: every required size must be hand designed to an excessive level of detail, they are tuned to a specific printer and cannot easily be adjusted for inter-printer variation, and, for large characters, the representation is far from compact. Outline fonts require extensive hinting at low resolutions, and it is difficult to come close to typographic scaling with outline fonts, leading again to the alternative of a complete redesign at each resolution. A nice compromise would be to specify the font at a relatively small set of resolutions, and interpolate control points for resolutions between, but it is difficult to guarantee the required consistency. And there are those who claim that outlines make a poor representation from a design point of view, as they distort the perceived shape of a character during the design process. (This is likely an avoidable user interface issue.)

Variable width splines may be an alternative to outlines and bitmaps. They are arguably a more natural representation, at least for many non-Latin fonts. Hinting is simplified significantly when the centreline and stroke width are known. And it may be that a good approximation to typographic scaling is possible based on relatively few sizes. Offsets of plane curves (of which variable width splines are a form), have been studied in the context of machine tool path design. It is known that the curve at a constant offset from a spline curve is neither a spline curve nor even a rational spline curve. Variable width offsets are even more complex mathematically. Many simple mathematical properties of spline curves are lacking, making some operations (such as finding the nearest point on the offset to a given point in space) much more complex. It is not clear that any of those more difficult operations are necessary for rasterizing type. There exist relatively simple algorithms for rasterizing variable width splines, based on well-known algorithms for rasterizing normal splines.

The definition of variable width splines I shall use is similar to those used by Pham [4] and by Chua [5] and Knuth [6]. Pham uses an offset approximation in which each segment of the centreline curve is in one-to-one correspondence with a spline segment approximating each of its offset curves. The approximation is done at definition time, and does not improve with subdivision. Chua uses paired outlines, along the same lines as METAFONT 79's `ddraw` and METAFONT's `fill` [1]. Both Pham and Chua show a number of special effects — such as variation in colour with relative distance from the centreline — that are possible with a stroke-based representation. METAFONT's `draw` allows for strokes of variable width, but uses pens of fixed shape and orientation. With these exceptions, most prior work on offsets of plane curves appears to have been restricted to fixed width offsets. Farouki and Neff have explored at length the analytic and algebraic properties of plane offset curves [7,8], while Farin has considered computing offsets of conics [9] and Hoschek [10] and Klass [11] give methods for approximating offsets of planar and planar cubic (respectively) curves using splines.

This paper's main purpose is to explore the use of variable width splines as a representation of fonts. A working definition is given in the next section, along with several examples. The following section shows several characters designed using a modest number of control

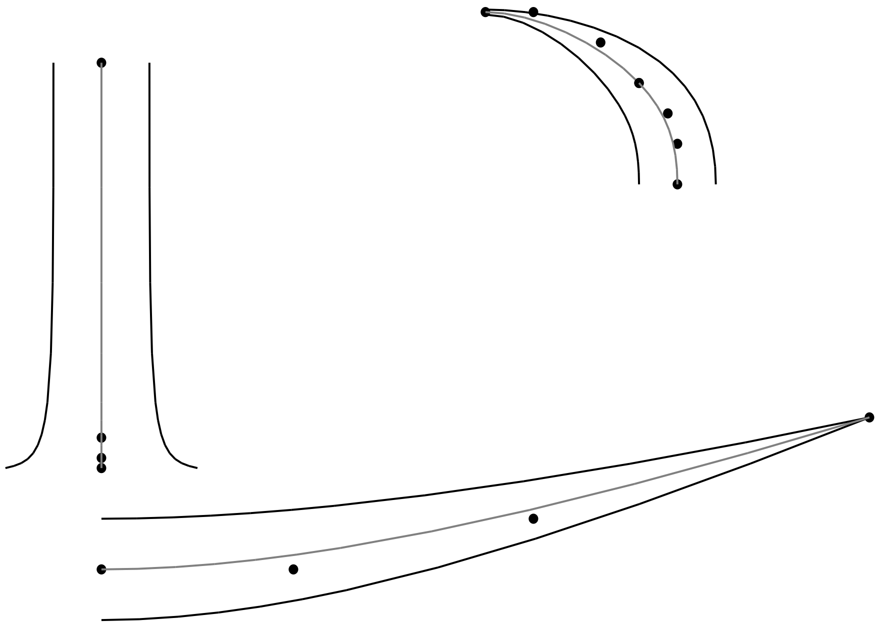

Figure 1. Two variable width strokes, each defined using a single Bézier segment, and a third, defined using two segments. The centreline curves are shown in grey

points in a variable width spline form. The example characters are to be taken not as works of art, but rather as proof that it is possible to make the necessary shapes. Only a skilled designer can make high quality characters. Section 4 is a discussion of the advantages and disadvantages of variable width strokes as a representation — initial impressions from the point of view of one very biased user with a limited user interface. Finally, some concluding remarks are given in Section 5.

2 DEFINITIONS

For the purpose of this paper, a variable width spline consists of a centreline curve, defined as a spline curve, and a width function, also a spline function. More specifically, the centreline curve is defined as $p(t) = [x(t),y(t)]$, where both of the coordinates are piecewise polynomial functions of the parameter t, (we shall concentrate on cubics), described as a series of control points and a (fixed) set of basis functions. Thus they might, for example, be represented in terms of Bézier control points, or B-spline control points. In addition to the usual two coordinates, control points carry with them a third coordinate w, used to control the width. The width is calculated from the w coordinates in exactly the same fashion as the position of the centreline is calculated from the other two coordinates in the control points related to the current segment. Figure 1 shows three sample variable width curves, defined using the Bézier basis.

To find the two offset points corresponding to a point $p(t) = [x(t),y(t)]$, on the centreline, compute the normal vector $\vec{n}(t) = [\dot{y}(t), -\dot{x}(t)]$, at that point, and compute the two points a distance w from p, in the $\pm\vec{n}(t)$ directions.

More formally, the offset consists of the locus of points defined by

$$o(t) = p(t) \pm w(t)\hat{n}(t)$$

where $\hat{n}$ is the unit normal to p at t:

$$\hat{n} = \frac{\left(dy/dt, - dx/dt\right)}{\sqrt{\left(dy/dt\right)^2 + \left(dx/dt\right)^2}}.$$

The square root is computationally the least attractive part, (although it can be done efficiently by exploiting coherence), but without it the width depends on the parameterization (the rate at which the curve is traversed relative to t).

A single spline segment has two offset curves, and because the segment has associated with it a nondecreasing parameterization, it is natural to label one offset the left offset, and the other the right offset, treating the two together as an offset pair. The left (right) offset is the one on the left (right) as the centreline is traversed from 0 to 1. A segment may have associated with it a flag indicating that one offset or the other is to be omitted, as well as possibly a pointer to the segment containing the other offset (this has proven useful in constructing mitre joints such as occur at the top of an A or the base of a V). A character at a given size is represented as a collection of segments which, when drawn filled, collectively cover the pixels to be intensified when the character is rasterized.

The above definition of a variable width stroke is different from those used by Pham and Chua in that the offsets are not splines. It differs from Knuth's variants in that the infinitely thin 'pen' is always held perpendicular to the direction of travel. Like METAFONT, data values (in particular widths) can be parameterized as functions of point size.

A complete font can be represented by storing control points (including widths) at one representative size, along with width scale factors for several sizes. The scale factors are interpolated for intermediate widths, and multiplied by the widths in the control points to give adjusted stroke widths. This gives a compact representation for what may prove to be a very good approximation to true typographic scaling. Some care is required in the design to ensure that strokes continue to connect properly after widths are scaled, however making centrelines connect is generally sufficient to preserve the topology.

While there is no guarantee that strokes connect properly after widths are scaled, it is not hard to arrange. By contrast, interpolating control points of outlines corresponding to different sizes can fail badly if control points do not correspond to the same 'place' on the respective curves. Figure 2 shows an example of the difficulty of interpolating control points with outlines.

3 RENDERING

Offsets of constant width are not as well-behaved as are simple splines, and offsets of variable widths are even less so. Cusps and loops occur whenever the curvature of the centreline matches or exceeds the width. Monotonic centrelines with monotonic widths do not necessarily give rise to monotonic offsets. The two offsets of a single cubic segment can have as many as eight extrema (in either direction), and even more roots when intersected with a line. All of these considerations may be related to the fact that variable width strokes are not currently in common use as a means of representing fonts.

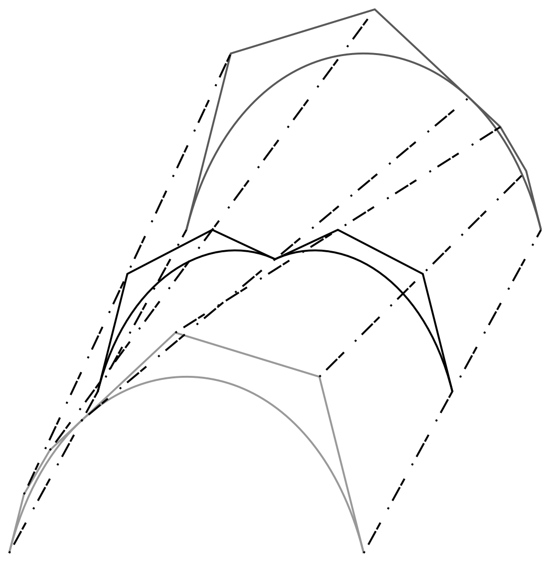

Figure 2. An example of catastrophic failure when trying to interpolate between two outlines. The light and mid-grey curves are in fact identical. Only their representations (control points) differ. The black curve results from linearly interpolating the control points of the two grey curves (dashed lines connect corresponding control points). The resulting curve is not even close to the average of two curves

Many of these mathematical difficulties are unlikely to occur in practice. For example, cusps and loops large enough to cause difficulty will not occur because they would not be designed into the font, or result from a faithful fitting of an existing font. The existence of many roots in the line/offset intersection problem is also unlikely to cause difficulty, since segments exhibiting this behaviour would not occur in actual fonts.

One method of rendering variable width strokes is adaptive subdivision. For this a stopping criterion is needed. In this paper the criterion used is based on straightness, and uses the combined linearity of the centreline and width to determine straightness. The deviation from straightness of the centreline is estimated from the area of its control polygon, while the maximum deviation from linearity of the width is calculated directly. These are summed to give an estimate of the maximum deviation from straightness. Alternatively, a looser tolerance could be used, such that Pham's (curved) offset approximation is sufficiently accurate [12]. No doubt better, more efficient schemes can be derived for determining the depth of subdivision required; this one is sufficiently fast and accurate for design. A similar variant of adaptive forward differencing is possible as well.

In relatively straight regions offset pairs are faster to render than outlines, as both sides are generated at once. In curved regions the inside curve can be moved forward a single step and then ignored while the outside catches up. Again, both sides are generated at once. Somewhat more work is performed per step, but each step supplies points on both sides. By far the vast majority of characters rasterized on typical pages today have outlines consisting of very many short segments. The start-up overhead is therefore quite significant. An offset representation, if it can use half as many segments, may be faster to render, once fully optimized rasterization algorithms have been implemented.

4 EXAMPLES

Six example characters are shown in this section. Three are from the Japanese katakana alphabet, one is from the hiragana, and the remaining two are the roman V and O. The katakana include enough stroke forms to provide a reasonable argument that the remaining 43 could be designed to the same level of quality. Given the origin of katakana as adaptations of kan'ji radicals, it seems reasonable to believe that a system that can describe katakana shapes is equally capable of describing kan'ji. Hiragana introduces a greater variety of strokes, being more curved, so a sample hiragana character is included as well. The roman V is particularly difficult for variable width strokes. It contains a mitre joint at the base, and horizontal serifs connected to diagonal strokes. In addition to the character designs, scaling, including minimum width constraints are shown. The letter O illustrates the use of minimum width constraints.

The examples in this section are *not* intended as samples of good type design. Rather they are intended to lend credibility to the claim that it is possible to design type using the variable width stroke paradigm. They were designed using very limited tools (a text editor was used to modify the control points), by one with no training in type design. At design resolution they come within several pixels of matching the shapes of actual scanned characters. The data quantities are also given. These should serve as a guideline only. More data, or perhaps only more care in specifying the values of the data, would give better looking type.

Figure 3 shows an uppercase Times Roman V, scaled up to allow inspection of the control points. (The design point size is unknown: it was based on a PostScript font). Most of the control points occur in the serifs, which contain several single-sided strokes each. The mitre joint consists of four pieces: two are the left and right strokes with one offset turned off when appropriate; the other two are isosceles triangles used to fill the gap. One triangle is has its base at the bottom of the V, the other's base connects the bottom right corner to the corner formed by the insides of the two strokes. The triangular patch-up would not be necessary in a more complete implementation; at present it is not possible to identify a left offset with a right offset of a different stroke. The data used to generate the strokes in Figure 3 appear in the appendix.

Figure 4 shows a letter O at three resolutions. At low resolutions, a minimum width constraint is invoked, preventing the strokes from disappearing. Some modified characters have been observed to look blocky in regions where the width constraint is active, but such regions have sizes in the single pixel range, and so the distortion is not visible at device resolution. Each is drawn from the same data: eight stroke segments, twenty-four control points.

Figure 5 shows a hiragana *su*. The stroking paradigm works well here (18 segments, 56 control points). The round ends of the strokes require more control points than pointed ones would, but an outline representation would require a higher number in the vicinity of curved ends as well. This character was relatively easy to design. The character scales gracefully to different sizes with the stroke widths decoupled from the character size. Any lack of smoothness in the curve is in the design: a more carefully executed design would

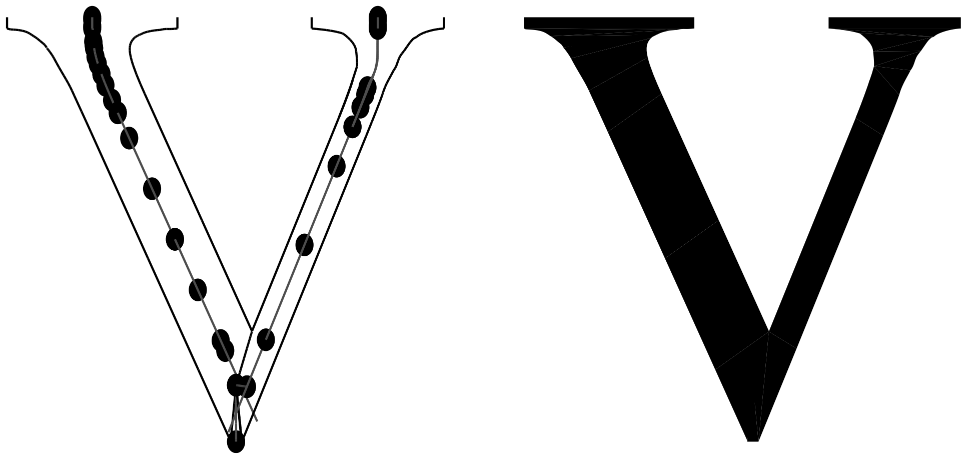

Figure 3. A Times Roman V. Control points are shown as filled circles, with the centreline drawn in grey. On the right is the character that results from filling the stroke. Most of the effort is expended in making the serifs behave correctly

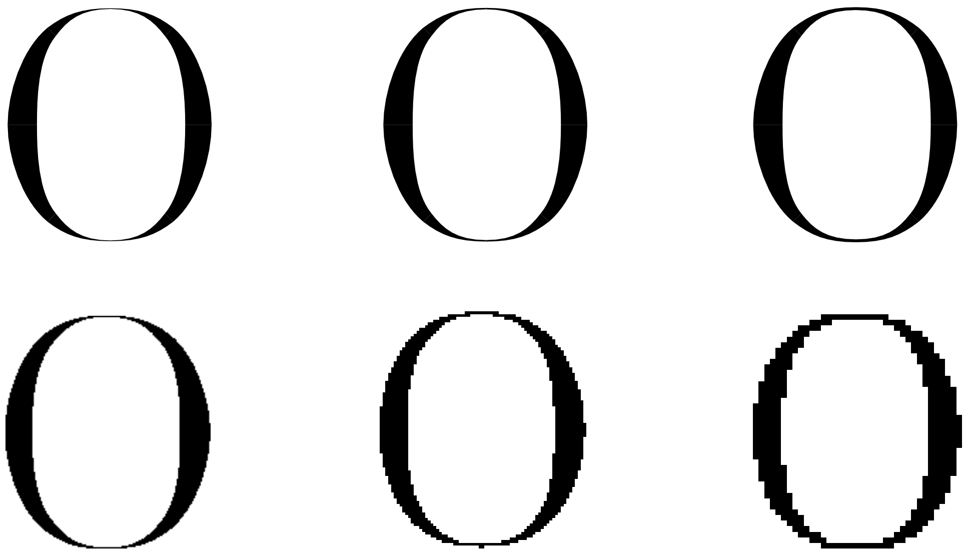

Figure 4. Minimum width constraint. At top are the shapes, and below are rasterizations (lowest resolution at the right). Because the stroke width is explicitly stored, minimal width constraints can prevent the stroke from disappearing entirely at low resolutions. Without hinting, the thin strokes are made to remain. Hinting could be used to position the centrelines so that the rasterizations are symmetric

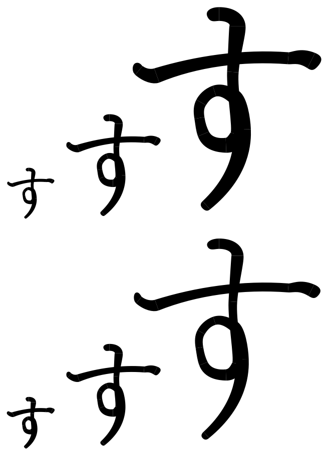
Figure 5. Typographic scaling of the hiragana su. *At the top is normal (PostScript style) scaling; at the bottom the widths are scaled independently of the control point locations. The left character is 25% thicker than a strictly scaled version would be, while the right character is 25% thinner than if it had been scaled from the one in the middle*

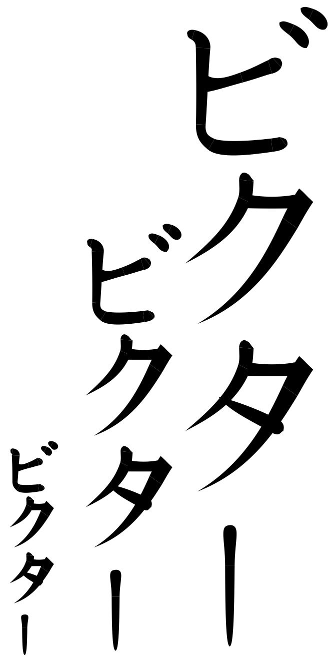
Figure 6. Katakana bvi-ku-ta — *(Victor) with independent width scaling*

be smooth. Figure 6 shows three katakana. The more angular nature of the katakana is somewhat more difficult to mimic, requiring extra control points near the ends of most strokes. In all, there were 15, 9 and 11 segments, with 49, 30, and 37 points, in the three characters from top to bottom. The vowel-lengthening stroke below the *ta* required two segments, seven points.

5 INITIAL IMPRESSIONS

On the whole the representation appears capable of representing characters of Latin or non-Latin fonts, although non-Latin fonts are easier to represent. Serifs are difficult. The (low) quality of the curves is probably deceptive, as the control points were manipulated with the crudest possible user interface (typing numbers). It really is not at all clear whether there is more or less data required to store a typical character.

Manipulation of variable width strokes appears more difficult than manipulating outlines, at least when the manipulation is done with the text-editor user interface. Of course there are half as many objects to fit when fitting strokes as there are when fitting independent outlines, and that may outweigh the difficulty of fitting individual strokes. Guaranteeing that two edges are parallel is trivial, even in an interactive environment.

This paper represents only a little experience with this representation; not enough to indicate whether the representation is useful in a design environment. Independent of its applicability in a design environment is its utility as an internal representation. The control over minimum stroke width and knowledge of stroke centreline are sure to simplify hinting. Because of the large body of outline fonts in existence, it is important to have the capability of converting from outline to variable width stroke form. Doing so in a manner faithful to the topology remains an open problem (it is hoped that this is as a result of lack of effort, rather than fundamental difficulties). It is not even clear whether direct autotracing from rasters or conversion from outlines is the better approach (given a solution to either problem the other can be reduced to it). Typical skeletonization approaches — such as the medial axis transform [13] — find a different centreline from the one used here: they generally send a line into each corner of a stroke. For outlines, the challenge is finding the corresponding points on pairs of outlines, and the centreline that generates them. Before these points can be found, corresponding outlines must be identified.

While the representation brings back an important element of typographic scaling, it loses certain features of outline fonts (purists might claim those features should never be there). Most notable among them is the ability to automatically create an oblique font by skewing the control points from a non-oblique font. Automatic oblique is still possible, but much more expensive. Rather than simply skewing the control points, it is necessary to de-skew the normal vectors at each step in the rendering. In addition, drawing a character as its outline is no longer trivial. On the other hand, it does permit other special effects, provided the strokes are set up with them in mind: varying the colour or grey shade over the width of the stroke, and textures that appear to follow the natural stroking pattern.

6 CONCLUSIONS

Variable width strokes have much to offer as a paradigm for representing characters. The mathematical anomalies they encompass are unlikely to occur in practice, so their rendering will typically be no slower than filled outlines. Because of the ability to control stroke width independently of overall scaling, they offer an inexpensive approach to typographic-like scaling. Controlling minimum width can prevent strokes from disappearing entirely at small sizes.

There is little experience with designing characters based on variable width strokes, and none at all with autotracing to create variable width representations from masters created in other forms. While rendering is relatively straightforward, the degree of difficulty associated with creation is relatively unknown.

Some conveniences do not exist with variable width strokes, including the easy generation of outlines and obliques. This may be considered a mixed blessing and curse.

The total data space requirement for a variable width stroked font may be greater or less than that for a traditional (outline) font. One would expect that — for some letters at least — a very good representation would only require three quarters as much data (half as many control points each of one and a half times the data size).

REFERENCES

1. Donald E. Knuth, *The* METAFONT *book*, Addison Wesley, 1986.
2. Daniel B. Updike, *Printing Types: Their History, Forms and Use*, Harvard University Press, 1922.
3. V. M. Corrado, 'The Linotron system', in *Proc. Computer Typesetting Conference*, (July 1964).
4. Binh Pham, 'Expressive brush strokes', *CGVIP: Graphical Models and Image Processing*, **53**(1), 1–6, (January 1991).
5. Yap Siong Chua, 'Bézier brushstrokes', *CAD*, **22**(9), 550–555, (November 1990).
6. Donald E. Knuth, 'Metafont', in *Tex and Metafont: New Directions in Typesetting.* Digital Press, 1979.
7. R. T. Farouki and C. A. Neff, 'Analytic properties of plane offset curves', *CAGD*, **7**(1-4), 83–99, (1990).
8. R. T. Farouki and C. A. Neff, 'Algebraic properties of plane offset curves', *CAGD*, **7**(1-4), 101–127, (1990).
9. Gerald Farin, 'Curvature continuity and offsets for piecewise conics', *TOG*, **8**(2), 89–99, (April 1989).
10. Josef Hoschek, 'Spline approximation of offset curves', *CAGD*, **5**, 33–40, (1988).
11. Reinhold Klass, 'An offset spline approximation for plane curves', *CAD*, **15**(5), 297–299, (September 1983).
12. Binh Pham, 'Offset approximation of uniform b-splines', *Computer-Aided Design*, **20**(8), 471–474, (October 1988).
13. Blum and Nagel, 'The medial axis transform', *Pattern Recognition*, **10**, 167–178, (1978).

APPENDIX

Below is the set of strokes used for the uppercase V shown in Figure 3. Shown are Bézier control points; at the start of each block of four is a single letter c, l or r. These indicate whether a normal stroke segment is to be drawn, or whether only the left or right offset is to be used, respectively. There are 68 unique points.

```
c  19.0500  99.5000  19.0000
   19.0500  99.0000  19.0000
   19.0500  98.0000  19.0000
   19.0500  97.5000  19.0000
r  19.2100  97.5000  19.1600
   19.2100  97.4950  15.0000
   19.2138  97.0500  14.5000
   19.2288  96.5000  13.2500
l  19.1500  97.5000  18.9000
   19.1500  97.0000  16.7500
   19.1625  96.7500  16.7500
   19.1750  96.5000  15.0000
r  19.2288  96.5000  13.2500
   19.2437  95.9500  12.0000
   19.2638  95.3750  11.6000
   19.2775  95.2500  10.4500
l  19.1750  96.5000  15.0000
   19.2000  96.0000  11.5000
   19.2500  95.5000  11.0000
   19.2775  95.2500  10.4500
c  19.2775  95.2500  10.4500
   19.3075  95.0000  10.2000
   19.3515  94.6000   9.5350
   19.4725  94.0000   9.1250
c  19.4725  94.0000   9.1250
   19.7750  92.5000   8.1000
   20.2900  91.2000   7.4250
   21.1000  89.4000   7.2000
c  21.1000  89.4000   7.2000
   22.0000  87.4000   6.9500
   23.4750  84.7500   6.9000
   24.7500  82.5000   6.9000
c  24.7500  82.5000   6.9000
   27.3000  78.0000   6.9000
   32.4000  69.0000   6.9000
   37.5000  60.0000   6.9000
c  37.5000  60.0000   6.9000
   42.6000  51.0000   6.9000
   47.7000  42.0000   6.9000
   48.7200  40.2000   6.9000
r  48.7200  40.2000   6.9000
   55.8600  27.6000   6.9000
   55.8600  27.6000   6.9000
   55.8600  27.6000   6.9000
c  51.1350  24.0000   1.2250
   51.1350  24.0000   1.2250
   51.1350  34.0500   0.0000
   51.1350  34.0500   0.0000
c  51.1350  34.0500   0.0000
   51.1350  34.0500   0.0000
   53.5550  33.7750   9.8750
   53.5550  33.7750   9.8750
l  57.7843  42.1300   3.4500
   49.4138  25.6625   3.4500
   49.4138  25.6625   3.4500
   49.4138  25.6625   3.4500
c  77.1000  80.0000   3.4500
   73.5500  73.0000   3.4500
   66.4000  59.0000   3.4500
   57.7843  42.1300   3.4500
c  80.4500  87.0000   3.5500
   79.9500  85.7500   3.5000
   78.8750  83.5000   3.4500
   77.1000  80.0000   3.4500
r  81.7000  89.0000   4.0000
   81.3250  88.2500   3.8000
   78.8750  83.5000   3.4500
   77.1000  80.0000   3.4500
l  82.5000  91.0000   3.9000
   82.1250  89.5000   2.8950
   80.9500  88.2500   3.6000
   80.4500  87.0000   3.5500
r  82.6800  93.5000   4.7500
   82.7050  92.0000   4.6500
   82.4500  90.5000   4.4000
   81.7000  89.0000   4.0000
l  82.6930  93.5000   5.9000
   82.6937  92.0000   4.4750
   82.7500  92.0000   4.5700
   82.5000  91.0000   3.9000
r  82.7250  96.2500   7.9000
   82.7125  96.0000   7.2500
   82.6550  95.0000   4.8500
   82.6800  93.5000   4.7500
l  82.7250  96.5000   8.7500
   82.6837  95.7500   7.6250
   82.6925  94.5000   6.8500
   82.6930  93.5000   5.9000
```