RANDOM SEARCH IN AUTOMATIC FONT GENERATION

Luc Devroye and Sandro Mazzucato

School of Computer Science, McGill University, Montreal, Canada

Abstract. We present a method for creating a PostScript type one Bézier outline font from a scanned bitmap of all characters. The number and positions of the Bézier sections are found by a random search method in which the search parameters are learned on-line. The criterion we use in the minimization takes into account the curvature, an error for deviating from the original pixel bitmap, and a penalty for the number of Bézier sections. The output consists of a type one PostScript font file and corresponding afm and tfm files with full sets of kerning pairs. The process is virtually fully automatic.

Keywords and phrases. Automatic font generation, PostScript, random search, global optimization, Bézier curves, font design.

CR Categories: 3.74, 5.25, 5.5.

Research of the authors was sponsored by NSERC Grant A3456 and by FCAR Grant 90-ER-0291. Address: School of Computer Science, McGill University, 3480 University Street, Montreal, Canada H3A 2A7. Email addresses: luc@cs.mcgill.ca, pedro@cs.mcgill.ca.

1. Introduction.

Designing a typeface is a mammoth undertaking that has fascinated and absorbed some of the most creative minds over the past 500 years. From pioneers such as Garamond, Arrighi, Fournier and Bodoni to the twentieth-century designers such as Gill and Goudy, we find one invariant—most designs were based upon simple pen-and-paper drawings of the characters. Recent efforts in computer typography have attempted to smoothen the transition from such drawings to a given computer format such as PostScript or INTELLIFONT. The computer era has seen the creation of parametric fonts—descriptions of fonts as functions of parameters such that each selection of a set of parameters yields another font in the family. An example is the METAFONT system developed by Knuth (1986a) (see also Hobby (1986) and Haralambous (1993)). Attracted by the possibility of creating families of fonts instead of one font at a time, commercial companies have come up with their own solutions, such as Adobe's multiple master format.

In this paper, we are going back to the basic process of capturing a pen-drawn collection of characters and making just one font from it. Our first concern is with the user's time. The whole process from scanning to final output takes less than two (physical) hours and may thus be attractive when one wants a quick production of one's own handwriting. In fact, the design of fonts for handwritten text is what motivated us in this project. One aspect of this involves the creation of suitably randomized characters to simulate real handwriting. This won't be dealt with here—we refer to Devroye and McDougall (1996) for a theoretical development and some crude examples, and to André and Borghi (1989), Doojies (1989) and van Blokland and van Rossum (1991) for earlier attempts in this direction.

In a nutshell, our approach is as follows: first we scan the characters and obtain a bitmap. The bitmap is processed and a polygonal outline of the characters are found. The polygons are then replaced by a C^2 Bézier spline approximation based on Böhm's method. Thus far, this is all very standard. The number of initial Bézier sections is typically unacceptably large—often more than 3000—and must be pared down. After trimming, the resulting outlines lead to PostScript type one (`.pfa`) files of a size comparable to those found in the market, i.e., about 100k bytes per file.

Our main contribution here is the method used for trimming the outlines: the optimization of the number and the positions of the outlines is done by a random search algorithm based upon a carefully picked criterion. Random search has several advantages—it is robust, it seeks a global minimum, and it may be made beautifully adaptive by the introduction of parameters that tune themselves. The algorithm always produces reasonable results. As our main contribution is at the level of the optimization itself, we will keep the discussion of details of the other steps to a minimum. The interested reader can obtain more information by electronic mail or by consulting Mazzucato (1994). As a first reference on font design, we refer to Karow (1994a, 1994b) or André (1993).

2. Bézier curves

The generation of scalable fonts in the PostScript type one format requires the use of cubic Bézier curves in order to describe the contours of a character. Bézier curves were invented independently by de Casteljau around 1959 and by Bézier around 1962 and are described in the books by Farin (1993) and Su and Liu (1989).

Given the current point (x_0, y_0), the PostScript command `curveto` takes the three points $(x_1, y_1), (x_2, y_2), (x_3, y_3)$ as parameters. The four points are called the Bézier points. The Bézier curve $\mathcal{B}(t) = (x(t), y(t))$ can be written as

$$\begin{aligned} x(t) &= a_x t^3 + b_x t^2 + c_x t + x_0 \\ y(t) &= a_y t^3 + b_y t^2 + c_y t + y_0 \end{aligned}$$

where

$$\begin{aligned} a_x &= x_3 - 3(x_2 - x_1) - x_0 & a_y &= y_3 - 3(y_2 - y_1) - y_0 \\ b_x &= 3(x_2 - 2x_1 + x_0) & b_y &= 3(y_2 - 2y_1 + y_0) \\ c_x &= 3(x_1 - x_0) & c_y &= 3(y_1 - y_0) \quad . \end{aligned}$$

Equivalently,

$$\begin{aligned} x(t) &= x_3 t^3 + 3x_2 t^2(1-t) + 3x_1 t(1-t)^2 + x_0(1-t)^3 \\ y(t) &= y_3 t^3 + 3y_2 t^2(1-t) + 3y_1 t(1-t)^2 + y_0(1-t)^3 \,, \end{aligned}$$

in the Bernstein polynomial format. The monomial form of a Bézier curve allows the computations to be performed with Hörner's method. A Bézier curve of the third degree takes one of four possible shapes:

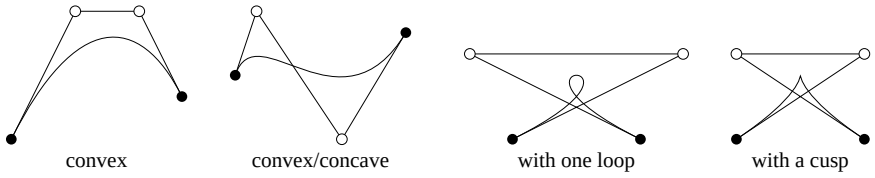

Figure 1. Third degree Bézier curves.

The junction point of the segments is known as a *knot* or a *breakpoint*. A spline $\mathcal{S}$, composed of two adjacent Bézier curves $\mathcal{B}_0$ and $\mathcal{B}_1$ may be created as follows. Each curve has its own *local parameter* t while $\mathcal{S}$ has a *global parameter* u, where $u \in R$. The knot sequence can be represented in terms of the parameter u, knot i having parameter value u_i. The correspondence between t and u depends on the actual length of each segment, $t = (u - u_i)/(u_{i+1} - u_i)$. We can think of $\mathcal{B}_0$ and $\mathcal{B}_1$ as two independent curves each having a local parameter t ranging from 0 to 1 or we may regard it as two segments of a composite curve with parameter u in the domain $[u_0, u_2]$. Æsthetically pleasing composite curves are obtained by introducing continuity restrictions and applying smoothness conditions to $\mathcal{S}$ (Manning, 1974).

Roughly speaking, continuity of the first and second order derivatives at knot points with respect to the global parameter u is called C^1 and C^2 continuity respectively.

To illustrate these notions of smoothness, take two adjacent Bézier sections $\mathcal{B}_0$ (with Bézier points $b_0, \ldots, b_3$) and $\mathcal{B}_1$ (with Bézier points $b_3, \ldots, b_6$). C^1 continuity at b_3, the knot, occurs if b_2, b_3 and b_4 are collinear, and if $||b_4 - b_3||/||b_3 - b_2|| = \Delta_1/\Delta_0$, where $\Delta_1 = u_2 - u_1$ and $\Delta_0 = u_1 - u_0$. Note that b_1 and b_5 do not appear in the condition. With C^2 continuity, the points b_1, b_2, b_3, b_4, b_5 influence the second derivative at the junction point. If the curve $\mathcal{S}$ is C^2 then there must a point d of a polygon b_1, d, b_5 that describes the same global quadratic polynomial as the five points mentioned above do. Hence assuming that the curve is already C^1, the following equations must be satisfied in order for d to exist:

$$b_2 = (1 - t_1)b_1 + t_1 d$$
$$b_4 = (1 - t_1)d + t_1 b_5 \ ,$$

where $t_1 = \Delta_0/(u_2 - u_0)$. The conditions for C^1 and C^2 curves are shown in the following figure.

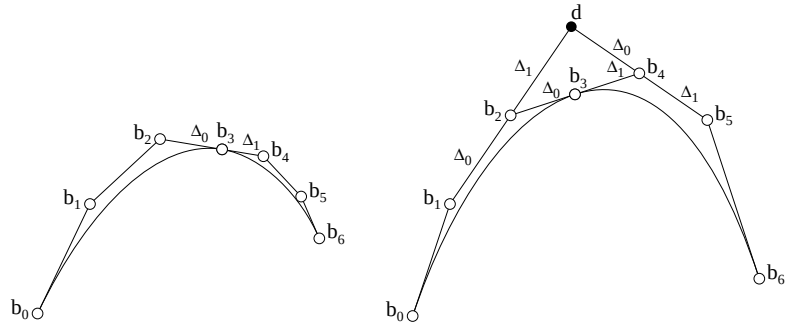

Figure 2. The different segment ratios for C^1 and C^2 Bézier curves.

3. From bitmap to polygonal outline.

In the development of a new typeface, typographers are concerned with legibility, uniformity among the characters and æsthetics. For handwritten characters, however, the major concern is with the accurate reproduction of the contours. The various steps are shown in the simplified figure 3.

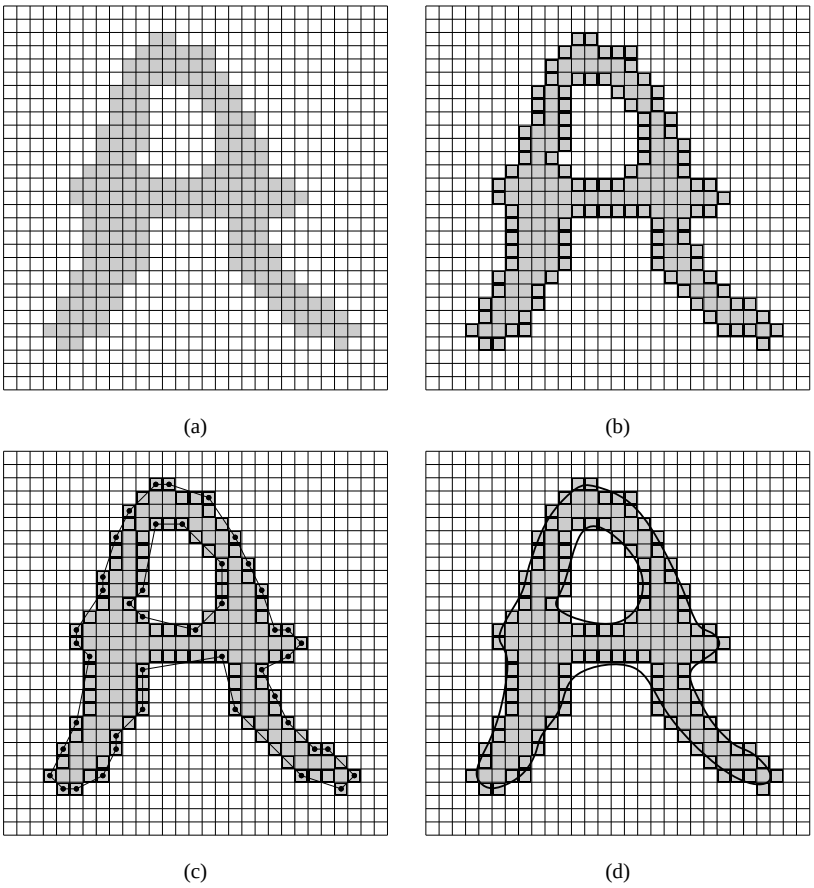

Figure 3. (a): The initial bitmap. (b): The contour pixels of a character. (c): The approximation by polygons. (d): Böhm sections derived from the polygon.

```
The algorithm.

1 Create characters with a pen and paper.
2 Scan the artwork, leaving a tiff bitmap image.
3 Remove impurities from the bitmap.
4 Find all contour pixels (see remark below).
5 Determine the starting polygonal outline (see remarks).
6 Trim the starting polygonal outline (see remarks).
7 Determine the starting Bézier outline (see remarks).
```

REMARK. BITMAP FORMATS. A multitude of bitmap formats, such as TIFF, GIF, EPS, BDF and PPM, are available. The different bitmap formats are, de facto, equivalent. Many programs for converting from one format to another are also available. See for example the PBMPLUS package by Poskanzer (1989).

REMARK. REMOVING IMPURITIES. The bitmap is cleaned by removing all specks and impurities, small connected islands of black pixels in a white sea and small connected islands of white pixels in a black sea. This is done by finding all connected components by depth first search of the bitmap and removing those that are too small.

REMARK. FINDING ALL CONTOUR PIXELS. As a character is represented by a bitmap, many black pixels are required to form the character. Some pixels are "interior" and others constitute the "contour" of the characters. The latter are extracted. A contour pixel can be defined as having at least one *white* pixel as a neighbor in a 4-connected representation. Neighbors are sometimes referred to by their relative position, north, east, south, or west. A pixel can be part of more than one contour and more than one contour may be present in a character. The contour pixels in a bitmap can be identified in time proportional to the number of pixels. We mark all black pixels that have at least one white pixel as a row neighbor or column neighbor. It is convenient to have a representation in which contour pixels are linked together in a chain or chains. We call the skin of a character the set of white pixels that have a contour pixel as one of its neighbors. The use of contour pixels in conjunction with the skin of a character permit one very simply to build the desired ordering. Some earlier contour-following algorithms (Duda and Hart, 1973) do not create the correct ordering for some 8-connected images, so one has to be a bit careful. Nevertheless, it is rather straightforward to find ordered contours in time linear in the number of contour pixels. To guarantee that all outlines are found, the visited pixels are marked and the search for another outline can be started by considering unvisited pixels. The search may simply be done by scanning the bitmap in an up-down, left-right fashion. The algorithm resembles in some respect depth first search (Cormen, Leiserson and Rivest, 1990).

REMARK. OTHER CONTOUR-FINDING ALGORITHMS. Some contour-finding algorithms are established according to the type of bitmap. For grey-level images, Avrahami and Pratt (1991) developed a contour extraction algorithm. This algorithm was modified and used in Itoh and Ohno (1993). A different contour-tracing algorithm derived from algorithms designed to verify connectedness of components (Minsky and Papert, 1969) has been employed by Gonczarowski's algorithm (Gonczarowski, 1991). Algorithms performing contour extraction are commonly used in the area of pattern analysis and recognition. For example, Moore's tracing algorithm (Pavlidis, 1982) works for all bitmap images.

REMARK. POLYGONAL OUTLINE. A simple polygon is a polygon with non-crossing edges. A polygonal outline is a finite collection of non-crossing simple polygons. A point is inside a polygonal outline if a ray emanating from it crosses an odd number of edges (this is called the even-odd rule, see Foley et al, 1992). The starting polygonal outline of our character has two properties: (i) its vertices are the centers of some black pixels; (ii) all centers of pixels in the original bitmap are correctly colored (black pixels have centers that are inside the polygonal outline). We refer to figure 3(c).

REMARK. TRIMMED POLYGONAL OUTLINE. The starting polygonal outline consisted of 800 to 3000 linear segments in our experiments. It may be trimmed by walking around the contour and identifying the longest edges that would induce no error on the bitmap. That is, we have a current polygon vertex v_i and consider the polygons in which the chains $(v_i, v_{i+1}, \ldots, v_j)$ are replaced by (v_i, v_j). The largest $j > i$ for which the resulting polygonal outline correctly colors all pixel centers is kept, and the attention moves to v_j. This is repeated until the entire contour is processed. There is no absolute guarantee that the resulting polygon is minimal, but it is considerably less complex. The time is roughly linear in the number of contour pixels.

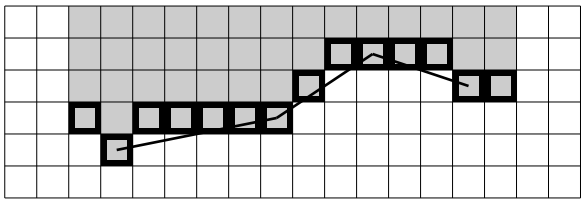

Figure 4. A portion of the contour is partitioned into longest possible contiguous line segments.

REMARK. FROM POLYGONAL OUTLINE TO BÉZIER OUTLINE. In the next step, we make the breakpoints of the polygons the defining points of a Bézier spline that we shall call a Böhm spline (Böhm, 1977). Figure 3(d) shows the Böhm spline for our simple example. It has the same number of sections as the polygon of figure 3(c). Denote by d_i the vertices of the trimmed polygonal outline. Partition each edge (d_i, d_{i+1}) into three equal parts and denote the two cutpoints by u_i and v_{i+1} respectively. Let z_i be the midpoint of the segment joining v_i and u_i: the z_i's are thus the knots of the spline. The Bézier spline (a C^2 cubic B-spline in Farin's notation, Farin, 1993) consists of Bézier sections $(z_i, u_i, v_{i+1}, z_{i+1})$. Taken together,

by the odd-even rule of ray intersections, the Bézier splines (one for each polygon) define an inside and an outside. It is of course no longer true that each center of the original bitmap is correctly colored.

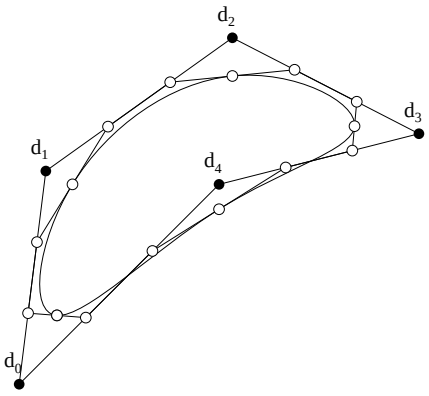

Figure 5. Böhm's C^2 construction algorithm. The white points on the curve are the z_i's.

Our parametrization is called uniform or equidistant. Other parametrizatioons lead to different distributions of the cutpoints (Schneider (1990), Itoh and Ohno (1993) and Plass and Stone (1983)). Duplicating some of the points d_i creates some sections of zero length. For example, the figure below shows the Bézier splines for $d_0, d_0, d_1, d_1, d_2, d_2, \ldots$ and $d_0, d_0, d_0, d_1, d_1, d_1, \ldots$ respectively. Unfortunately, this procedure destroys the C^2 continuity. Our method overcomes the necessity of corner detection and produces a flexible spline that is handy to manipulate.

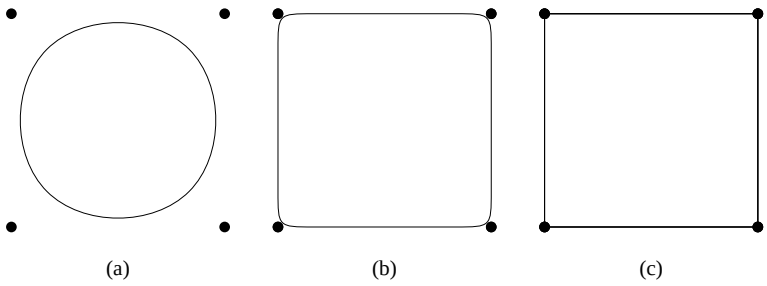

Figure 6. C^2 construction with knot multiplicities of 1, 2 and 3 respectively.

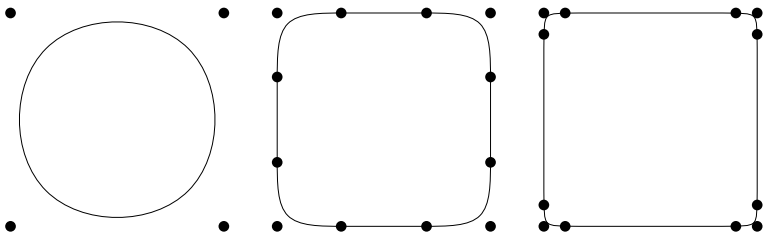

Figure 7. By cutting Bézier sections into parts, we may obtain new interesting splines. Each section of the Böhm spline is divided into three parts, in the proportions 0.3, 0.4 and 0.3 for figure (a). In figure (b), the ratios are 0.1, 0.8, 0.1.

Remark. Other methods of determining Bézier outlines. As knots define the endpoints of curves, dynamic programming methods may be used to find a good knot partitioning as in Plass and Stone (1983). A modified version of it, presented in Schneider (1990), consists of replacing the heuristic by a subdivision process that breaks the curve where the maximal error occurs. Other approaches perform first corner detection to define an initial set of knots (see, e.g., Lejun, Hao and Wah, 1994). Corner detection consists of interpreting the bitmap to find locations where the contour changes direction abruptly. Between two consecutive corners, a certain number of knots may be defined. An iterative approach, used in Gonczarowski (1991), consists of finding the longest curve from a given point such that it approximates the desired section of the bitmap with a user-specified threshold. As mentioned in Itoh and Ohno (1993), the precise detection of contour points is a very hard problem. The algorithm of Itoh and Ohno uses the estimated corner points for defining segments. The approximation of contour pixels or polygonal outlines by curves is sometimes referred to as *auto-tracing*. Some commercial packages perform such an operation.

4. Optimization: the quality function.

Call the Böhm spline $\mathcal{S}$. To further reduce the number of sections and to make smooth outlines that remain close to the original bitmap, one must define a quality function Φ and an optimization algorithm. The choice of both differs from what we have found in the literature. Our choices are developed in the next two sections.

The most used quality function Φ is based upon the least-squares criterion (see Plass and Stone (1983), Itoh and Ohno (1993), Gonczarowski (1991) and Schneider (1990)). It evaluates the distance between the contour pixels and their corresponding locations on the interpolating curve. It requires the computation of a mapping between the pixels and the local parameter t. Different methods are used to perform the approximation mapping. Our method does not require any such mapping. We simply take the following quality function $\Phi(\mathcal{S})$ of a spline $\mathcal{S}$:

$$\Phi(\mathcal{S}) = \alpha \text{ pixel error}(\mathcal{S}) + \beta \text{ curvature}(\mathcal{S}) + \gamma(\#\text{ of sections})(\mathcal{S}) \ .$$

Here the weights α, β and γ are nonnegative and sum to one. The pixel error penalizes big differences with the original bitmap. The curvature penalizes curves that arte not smooth. The last term in Φ places a penalty on the description length or complexity of the solution. The first two errors are described in more detail.

PIXEL ERROR. The pixel error criterion looks at the centers of all pixels in the bitmap. The original bitmap colors each pixel. Each center of a pixel gets colored again by filling the Bézier spline ensemble $\mathcal{S}$ according to the odd-even rule explained earlier. The pixel errors merely counts the number of pixels for which the colors are flipped. Note that it does not attempt to compute the area of the points that are incorrectly colored.

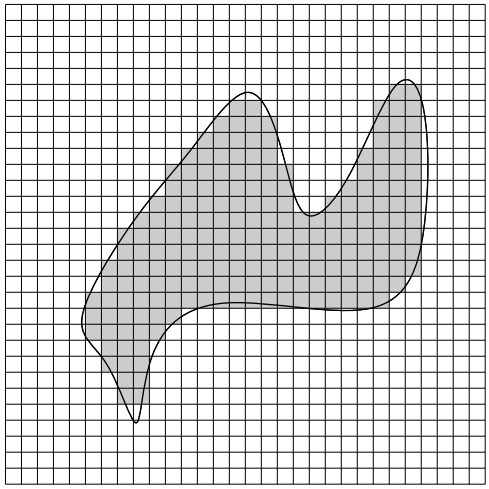

Figure 8. The figure shows an orginal bitmap and a Bézier spline with zero pixel error. Colors of points are determined by the even-odd algorithm.

If we define the error to be the number of incorrectly colored pixels, then each pixel has the same weight in the criterion. Experiments show that it is preferable to give more weight to pixels that are far away from the contour. This, in effect, creates better-fitting splines. Therefore, we propose various definitions of pixel error and let the user make a selection. The distance from the outline is the length of the shortest adjacent-pixel-path (in which only north, south, east, west moves are allowed) starting at the pixel to reach a pixel of the appropriate color. These distances may be computed by breadth first search in time proportional to the number of pixels by starting with a queue of contour pixels and working away from the contour (Cormen, Leiserson and Rivest, 1990). We will call this the bushfire algorithm. More details may be found on page 254 of Preparata and Shamos (1985) and in Mazzucato (1994). The pixel error is in general defined by

$$\sum_{\text{all pixels } p} W(D_p) \ ,$$

where D_p is the distance between p and the nearest pixel for which the original bitmap color matches the assigned color of p. Thus, if p is correctly colored, $D_p = 0$. In the expression above, W is an increasing

penalty function such as

$$W(U) = \begin{cases} I_{U>0} & \text{(ordinary pixel error)} \\ U & \text{(linear penalty)} \\ U^2 & \text{(quadratic penalty)} \\ U^3 & \text{(cubic penalty)} \\ UI_{U\leq\delta} + \infty I_{U>\delta} & \text{(linear penalty, } \infty \text{ beyond } \delta\text{)} \\ U^2 I_{U\leq\delta} + \infty I_{U>\delta} & \text{(quadratic penalty, } \infty \text{ beyond } \delta\text{)} \\ U^3 I_{U\leq\delta} + \infty I_{U>\delta} & \text{(cubic penalty, } \infty \text{ beyond } \delta\text{)} \end{cases},$$

and $\delta > 0$ is a design parameter. As mentioned above, many existing algorithms use the least-squares method. Roughly speaking, these correspond to picking $W(U) = U$ as the sum of penalties $1, 2, 3, \ldots, k$ for a pixel at distance k is $k(k+1)/2$.

Remark: Updating the pixel error. We would like to point out that updating the pixel error can be done efficiently. The convex hull property of Bézier curves ensures that all modifications to pixel coloring are relatively local. If a modification θ is applied to a Bézier curve $\mathcal{B}_1$ to produce another Bézier curve $\mathcal{B}_2$, the region of the bitmap for which pixels might change color is delimited by the convex hulls of $\mathcal{B}_1$ and $\mathcal{B}_2$. Note that the two cannot be disjoint since $\mathcal{B}_2$ must be attached to the portion of the spline that $\mathcal{B}_1$ was connected to initially. For simplicity, a bounding box BB_θ can be used to enclose the two convex hulls. With a spline that gets modified at each step of the generation process, the computations are thus kept to a minimum. The even-odd rule suggests that we should store and keep track of the intersections between the curves and the horizontal and vertical lines of the pixel grid. Given a Bézier curve $\mathcal{B}$, with control points b_0, b_1, b_2, b_3 and a bounding box BB, the intersections of $\mathcal{B}$ with the rows and columns of BB need to be computed. An intersection for $\mathcal{B}$ is calculated by solving the cubic roots of one of the two polynomials of the Bézier monomial form

$$a_x t^3 + b_x t^2 + c_x t + x_0 = x_l$$
$$a_y t^3 + b_y t^2 + c_y t + y_0 = y_l$$

depending on whether a vertical line at $x = x_l$ or a horizontal line at $y = y_l$ is considered. Note that x_l and y_l are contained in the bounding box BB. Without loss of generality, let us consider the case of a vertical line at $x = x_l$. We call a root t_0, t_1, t_2 valid, if it is real and falls in the range $[0, 1]$. Let t_j be such a valid root. Then the curve intersects the line at point $(x_l, \mathcal{B}(t_j))$. Let $y_b = \lfloor \mathcal{B}(t_j) \rfloor$. If the total number of curves passing between the points (x_l, y_b) and $(x_l + 1, y_b)$ is odd then the color of the two points (x_l, y_b) and $(x_l + 1, y_b)$ is different. If the considered bitmap is of size $r \times c$, then knowing the color of pixel center (x, y), $0 \leq x < r, 0 \leq y < c$, as well as the number of times the line (x, y) $(x + 1, y)$ gets intersected by curves is sufficient to determine the color of the pixel $(x + 1, y)$. The only information that must be kept for a pixel center (x, y) is thus the number of intersections between (x, y) and $(x + 1, y)$. The information that must be retained is the set of intersection points with the horizontal and vertical lines.

Curvature. The curvature is a good indicator of the wiggliness of a curve. For a line, the curvature is zero, and for a circle the curvature is constant and is inversely proportional to the radius of the circle. If the curvature at a point z of a curve C is κ, the curve locally behaves like a circle with radius $1/\kappa$

(Swokowski, 1975). In the case of parametrically defined curves $(x(t), y(t))$, the curvature at t is defined as

$$\kappa = \frac{|x^{'}(t)y^{''}(t) - y^{'}(t)x^{''}(t)|}{[(x^{'}(t))^2 + (y^{'}(t))^2]^{\frac{3}{2}}},$$

and the total curvature is $\int_0^1 \kappa(t)dt$ or $\sqrt{\int_0^1 \kappa^2(t)\,dt}$. For our splines, κ varies continuously across sections, so that the integrations may routinely be performed by by Simpson's rule (Davis and Rabinowitz, 1984).

5. Optimization

The Böhm spline introduces a possibly substantial error on the original bitmap and has too many sections. We will optimize both the number of sections and the locations of the Böhm control points (called d_i above). We recall that these points are not endpoints of Bézier splines, but rather define a closely related Bézier spline, the Böhm spline. In the main part of our algorithm, the number of sections is reduced and the quality of the approximation is enhanced by minimizing Φ. As Φ depends in a complicated and multimodal manner on its parameters, and the number of parameters varies as well, we use a rather robust and general optimization method such as random search. For general descriptions of random search, simulated annealing, genetic algorithms or evolutionary methods, see Törn and Žilinskas (1989), Zhigljavsky (1991), Männer and Schwefel (1991), Devroye (1994), Rechenberg (1973), Schwefel (1977, 1981), or Rinnooy Kan and Timmer (1987a, 1987b). Roughly, we have:

```
start from S
perform the following n times:
        T is obtained from S by modifying S
        if Φ(T) < Φ(S) then S ← T
return S
```

Random search has the advantage that it converges in all circumstances to a global optimum, and that it finds acceptable solutions relatively quickly. The "modification" defined in the algorithm must be based upon operations defined on Böhm splines. We took two such operations but realize that there are endless other possibly better operations one might consider as well. Our operations are described below.

THE MERGE OPERATION. A *merge* operation consists of replacing two adjacent Bézier curve segments by a single one. Since the C^2 spline constraint is always present, the merge is executed by replacing two adjacent sections s_j and s_{j+1} with one, simply by defining the new section with the endpoints of the polysegment (s_j, s_{j+1}). The total number of sections in the contour thus decreases by one. This process is also called knot removal.

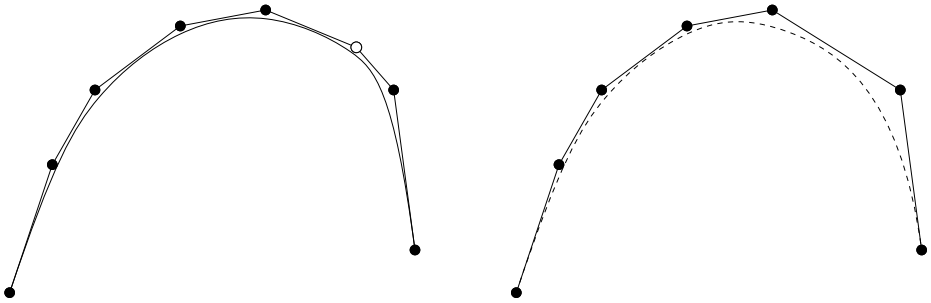

Figure 9. Knot removal: the white point on the left (a d_i point) is deleted, resulting in an updated spline on the right.

THE MOVE OPERATION. A *move* operation moves a section endpoint a certain (random) distance away from its current location, causing two sections to be modified.

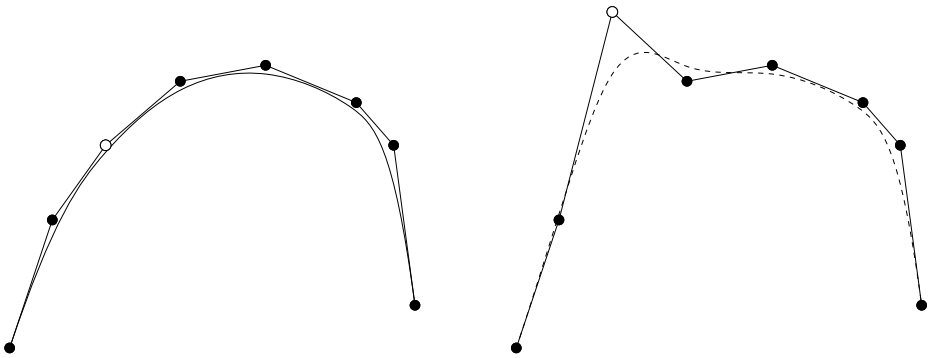

Figure 10. When a point d_i is moved, only four Bézier sections of the spline are affected.

Consider the Bézier curve segments $\mathcal{B}_0$, $\mathcal{B}_1$, $\mathcal{B}_2$, $\mathcal{B}_3$, $\mathcal{B}_4$ of a spline $\mathcal{S}$. If $\mathcal{B}_2$ and $\mathcal{B}_3$ are merged into $\mathcal{B}$, only the curves associated with $\mathcal{B}_1$, $\mathcal{B}_4$ and $\mathcal{B}$ need to be recomputed. Similarly if a move is performed on the junction point of sections $\mathcal{B}_2$ and $\mathcal{B}_3$, the affected curve segments are $\mathcal{B}_1$, $\mathcal{B}_2$, $\mathcal{B}_3$, and $\mathcal{B}_4$. These operations ensure the locality of the modifications on a spline $\mathcal{S}$. A merge operation can perturb the curve significantly. As it lowers the number of sections in a character, the sections become longer. It allows the removal of small imperfections introduced during the input process. The merge is most efficient on consecutive sections that do have more or less the same orientation.

6. Details of our random search algorithm.

We use the following notation. $\mathcal{S}$ is a Böhm-type Bézier spline. A section of a Bézier spline $\mathcal{S}$ is denoted by $\mathcal{B}$. Its curvature is $\kappa_{\mathcal{B}}$. Each section also has a priority value $\pi_{\mathcal{B}}$ and the sections are organized into a heap $\mathcal{H}$ with the smallest priority value on top. Furthermore, we need

A. δ_1, an initial step size (in terms of number of pixels).

B. $\xi \in \{1, 2, 3\}$, a user parameter for changing the step size.

C. $\{p_n\}$, a sequence of adjustable probabilities.

D. N, a limit on the number of iterations.

```
δ_1 ← initial step (initial step size for random search)

for n = 1 to N do
with probability p_n do:
  B ← top(H)
  B' ← shortest section neighboring B on S
  T ← S with B and B' merged (and adjacent sections modified)

otherwise do:
  select d uniformly and at random from
    the junction points of S
  set d' ← d + δ_n U, where U is uniformly distributed
    on the unit circle (so ||d' − d|| = δ_n)
  T ← S with d replaced by d' (and adjacent sections modified)

if Φ(T) < Φ(S)
  then (a success):
    δ_{n+1} ← δ_n + ξ
    S ← T
    update H by removing obsolete sections
      and/or altering the priorities of updated sections
      (Note: maximally 4 sections are involved,
      and for each one do π_B ← κ_B)
  else (a failure):
    δ_{n+1} ← max{1, δ_n − ξ}
    π_B ← 1.1 max{π_B'} for all neighbors B' of B
adjust p_n as described below
```

The algorithm above dsiffers from ordinary algorithms in two fundamental ways.

A. IT USES AN ADAPTIVE STEP SIZE. The above algorithm differs from fixed step size random search, in which random perturbations are always of constant magnitude. Small step sizes yield small improvements,

while large step sizes reduce the probability of a successful trial. As noted by Schumer and Steiglitz (1968), the optimum step size is between those two extremes. Since the optimum step size is unknown, *adaptive step size random search* algorithms were created. The magnitude δ_n of a step in a random direction varies according to the past experience. The basic principle behind these adaptive algorithms is to try bigger steps as an improvement occurs and to reduce the step on unsuccessful trials (Matyas, 1965). Each algorithm uses a different variant. For example, the adaptive step size random search algorithm (ASSRS; see Schumer and Steiglitz, 1968) tries two step sizes (δ_n and $\delta_n(1+a)$) in the same random direction, where $0 < a < 1$, and waits a certain number of consecutive unsuccessful trials before reducing the step size δ_n. If on the other hand the attempt succeeds, δ_{n+1} is set to δ_n or $\delta_n(1+a)$, depending upon which corresponded to the best improvement. A rule of thumb that may be found in several publications (see Devroye, 1972, and more recently, Bäck, Hoffmeister and Schwefel, 1991), is that the step size should increase after a successful step and decrease after a failure, and that the parameters should be adjusted to keep the probability of success around 1/5. Schumer and Steiglitz (1968) and others investigate the optimality of similar strategies for local hill-climbing. Alternately, the optimal step size may be found by a one-dimensional search along a random direction (Bremermann, 1968, Gaviano, 1975). Another adaptive procedure (Devroye, 1972) combines random search with non-random direct search. The *compound random search algorithm* (CRSA) basically inspects a deterministic modification to the approximation as well as a controlled random variation. The interesting feature here is the control of the step size. The step size δ_n is updated as follows:

$$\delta_{n+1} = \begin{cases} \delta_n(1+A) & \text{if the trial is successful} \\ \delta_n(1-B) & \text{otherwise,} \end{cases}$$

where $A > 0$ and $0 < B < 1$. The probability of a successful trial stabilizes roughly around $B/(A+B)$, and this level must be picked strictly in $(0, 1/2)$ (so that $A > B$). For example, if we choose $\mathbb{P}\{\text{success}\} = 0.2$, then $A = 4B$. It is recommended though that P_{success} be kept between 0.15 and 0.35. Unfortunately, this method cannot be employed naïvely. The PostScript type one format requires that the different instruction parameters be integer values (see Adobe, 1990b). Thus, junction points are always truncated to a sufficiently small integer grid. Step sizes less than one just do not make sense. Thus, in our algorithm, step sizes are integer-valued and are updated by the rule

$$\delta_{n+1} = \begin{cases} \delta_n + \xi & \text{if the trial is successful} \\ \delta_n - \xi & \text{if the trial is not successful and } \delta_n - \xi \geq 1 \\ \delta_n & \text{otherwise,} \end{cases}$$

where $\xi \in \{1, 2, 3\}$.

B. IT PICKS THE MOST EFFECTIVE STRATEGY ON THE FLY. In the algorithm, we have a parameter p_n which controls the probability of trying a merge operation. In a sense, the merge operation "competes" against the move operation. After starting with a fixed p_n for a warm-up, the algorithm measures the average decrease in Φ observed over the last $N' = 50$ similar operations (this includes the failed attempts, in which the change in Φ is zero). If the absolute value of the average was Δ_m for a merge and Δ_μ for a move, then we set $p_n = \Delta_m/(\Delta_m + \Delta_\mu)$ to insure that the more successful strategy receives preferential treatment. The case 0/0 is elegantly avoided by stopping altogether when both Δ's are zero; this only occurs if no improvement is seen in Φ in the last N' attempts with either strategy. We resorted to such a rule as it was very difficult to estimate the number of required iterations beforehand in view of the unknown nature of the characters—simple outlines require far less work for example. Attempts at

adjusting random search parameters on-line go back to the early seventies, where learning automata are used to select best search strategies on the fly—see Poznyak (1972), Volynskii and Filatov (1974), and Ripa (1970, 1971). In 1975, Jarvis introduced competing local random searches. The n-th trial is spent on the i-th search strategy with probability $p_{n,i}$, where $p_{n,i}$ is adapted as $n \to \infty$. See also Ermakov and Zhigljavskii (1983), Hill (1969), McMurtry and Fu (1966), and Shapiro and Narendra (1969).

7. Results

The code is written in C and consists of filters that take TIFF input (from the scanner), transform it to XBM (by using `tifftopnm` and `pbmtoxbm`), and create a type three PostScript bitmap font. The last part is called `scrpt2s` and was written by Luc Mikiszko at McGill University during the summer of 1993, and used ideas from Leisher's program `bdftops` (Leisher, 1990). The algorithm described above grabs the last file and creates in one execution a type one PostScript font with corresponding `.afm` file and kerning pairs. There are 35 options related to the choice of quality function, the parameters of the optimization process, the generated font (weight, italic angle, sidebearing, name, height, monospacing, unique ID number, stroking versus filling) and statistics to be collected during the generation. For example, bold versions are easily obtained by coating the original bitmap with a fixed number of layers of pixels. Different quality function settings lead to different outputs as each resulting font is indeed a best possible compromise. The following figures highlight some of the technical difficulties we overcame:

A. Characters drawn with a thin pen are often just a few pixels wide, and are very sensitive to variations in thickness. Examples of successful conversions include the font Bunsbeek in figure 15 and the font Isa-LightItalic in figure 11.

B. Characters with highly irregular contours, such as from old typewriters require more Bézier sections for faithful reproductions. See Gete in figure 12.

C. Characters with many intersections require crisp rendering where strokes cross. As an extreme example, we created the font Oplinter, in which each letter was drawn twice. Observe the quality of the curves near intersections in figure 13.

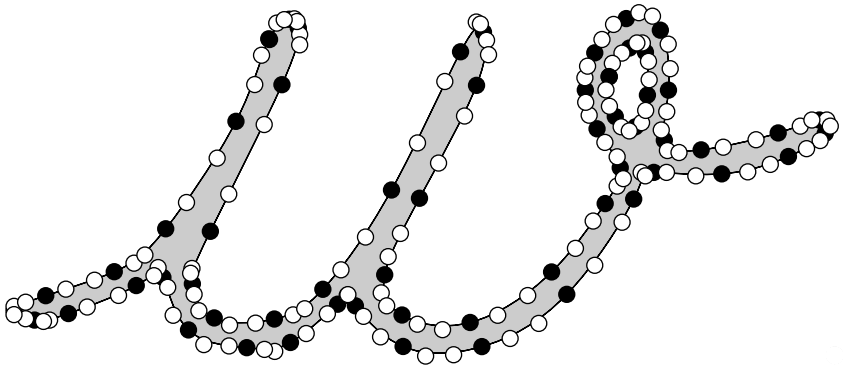

Figure 11. The letter w in the font Isa-LightItalic, based upon the handwriting of Isabelle Massarelli. Black dots denote Bézier endpoints, and white dots represent control points.

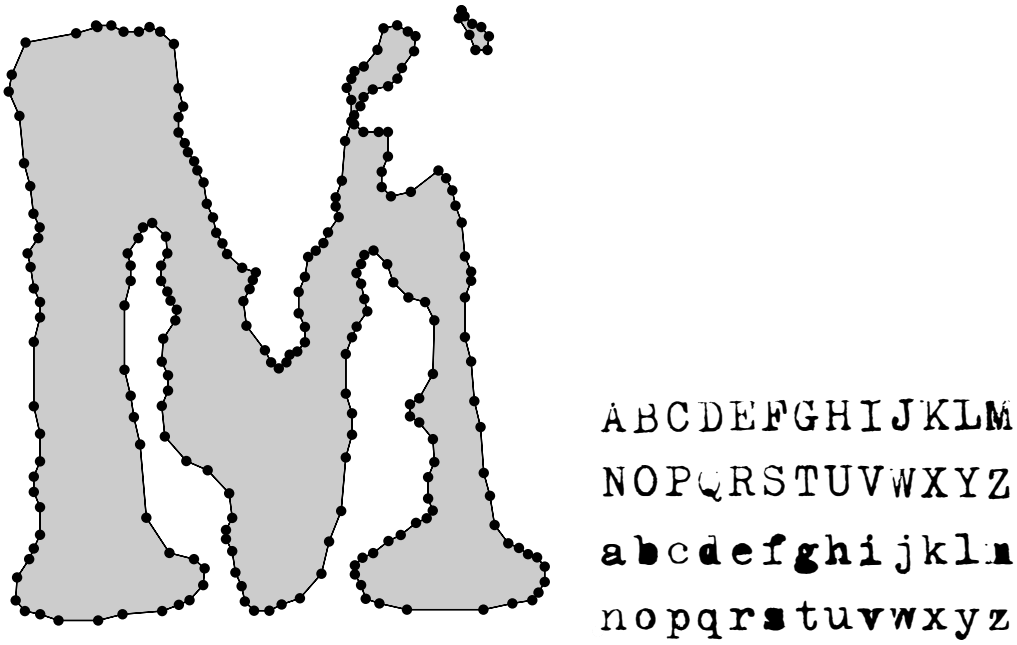

Figure 12. A font called Gete derived from a sample from an old typewriter. For such irregular characters, the trimmed polygonal outline often suffices if characters will not be used at large point sizes.

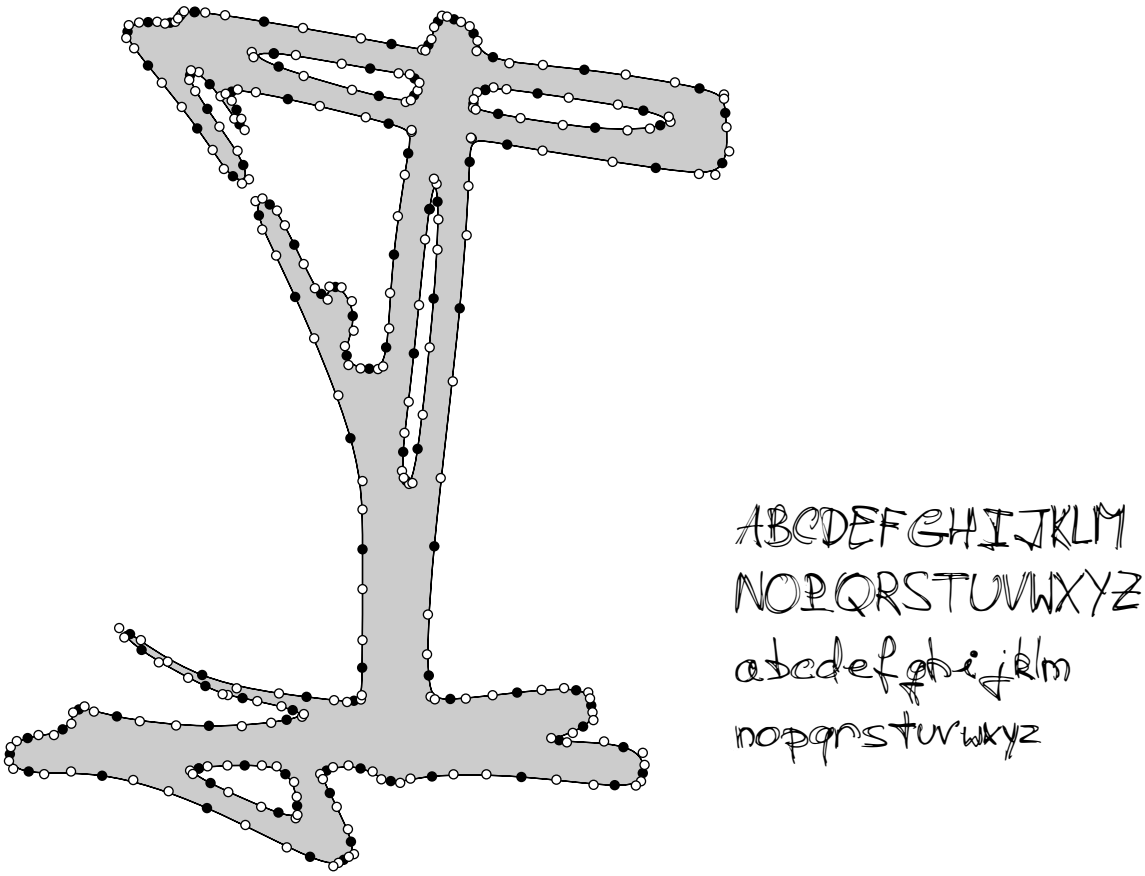

Figure 13. A font called Oplinter was derived from a sample in which each character was drawn twice, to test the algorithm's performance in the presence of many intersections of strokes. Note that the number of Bézier sections increases with the curvature of the contour.

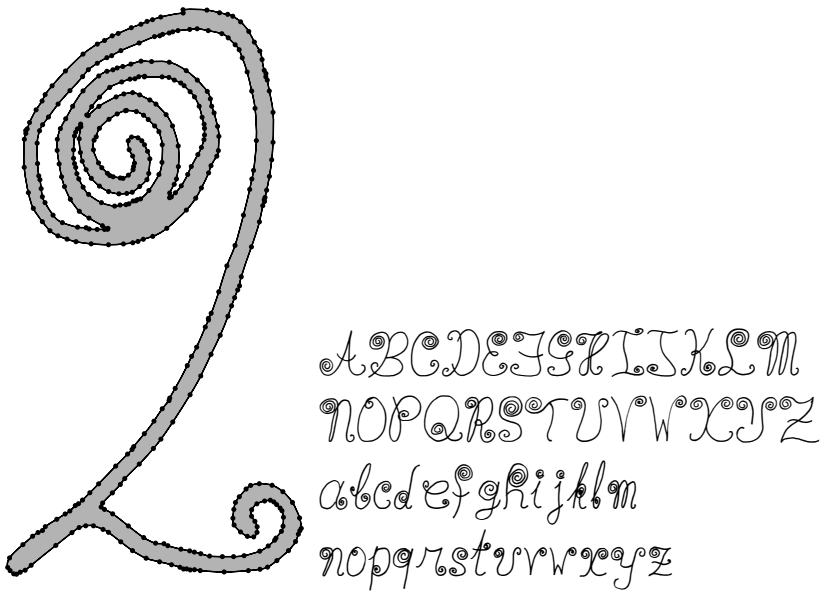

Figure 14. In this baroque font called Waaiberg, we applied ultra-thin strokes that ran into each other in several spirals. These inkruns were caught by the algorithm and faithfully reproduced. Only the Bézier endpoints are shown.

We wrote a filter for generating kerning pairs. To illustrate this, look at text samples of Bost-Bold and Houtem:

The quick brown fox jumps over the lazy dog.

The quick brown fox jumps over the lazy dog.

A few examples of some of the generated fonts are shown below in a format adapted from Wallis's book (1990).

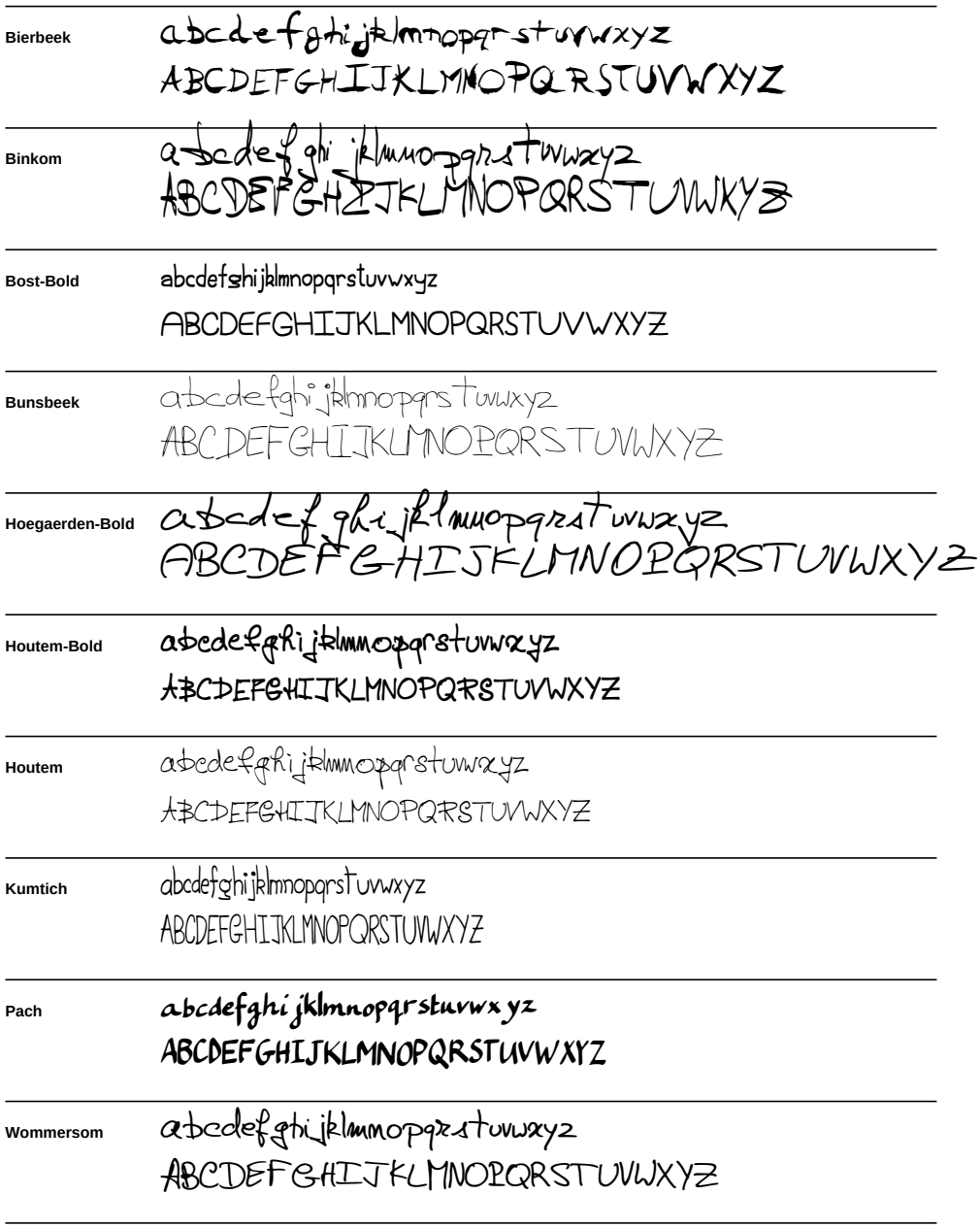

Figure 15. Fonts based upon simple pen-and-paper drawings.

8. References

Adobe, *PostScript Language Reference Manual*, Addison-Wesley, Reading, MA, 1990a.

Adobe, *Adobe Font Metric Files Specification Version 3.0*, Adobe, 1990c.

Adobe, *Adobe Type 1 Font Format*, Addison-Wesley, Reading, MA, 1990b.

J. André, "Création de fontes et typographie numérique," IRISA, Campus de Beaulieu, Rennes, 1993.

J. André and B. Borghi, "Dynamic fonts," in: *Raster Imaging and Digital Typography*, ed. J. André and R. D. Hersch, pp. 198–204, Cambridge University Press, Cambridge, 1989.

G. Avrahami and V. Pratt, "Sub-pixel edge detection in character digitization," in: *Raster Imaging and Digital Typography II*, ed. R. A. Morris and J. André, pp. 54–64, Cambridge University Press, Cambridge, 1991.

H. J. Bremermann, "Numerical optimization procedures derived from biological evolution preocesses," in: *Cybernetic Problems in Bionics*, ed. H. L. Oestreicher and D. R. Moore, pp. 597–616, Gordon and Breach Science Publishers, New York, 1968.

T. Bäck, F. Hoffmeister, and H.-P. Schwefel, "A survey of evolution strategies," in: *Proceedings of the Fourth International Conference on Genetic Algorithms*, ed. R. K. Belew and L. B. Booker, pp. 2–9, Morgan Kaufmann Publishers, San Mateo, CA, 1991.

W. Böhm, "Cubic B-Spline curves and surfaces in computer-aided geometric design," *Computing*, vol. 19, pp. 29–34, 1977.

W. Böhm, G. Farin, and J. Kahmann, "A survey of curve and surface methods in CAGD," *Computer-Aided Geometric Design*, vol. 1, pp. 1–60, 1984.

T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, MIT Press, Cambridge, MA, 1990.

P. J. Davis and P. Rabinowitz, *Methods of Numerical Integration*, Academic Press, Orlando, FL, 1984.

L. Devroye, "The compound random search algorithm," *Proceedings of the International Symposium on Systems Engineering*, pp. 105–110, Lafayette, IN, 1972.

L. Devroye, "Random optimization methods," in: *New Directions in Simulation for Manufacturing and Communications*, ed. S. Morito, H. Sakasegawa, K. Yoneda, M. Fushimi and K. Nakano, pp. 20–31, Operations Research Society of Japan, Tokyo, 1994.

L. Devroye and M. McDougall, "Random fonts for the simulation of handwriting," *Electronic Publishing*, vol. 0, pp. 0–0, 1996.

E. H. Doojies, "Rendition of quasi-calligraphic script defined by pen trajectory," *Raster Imaging and Digital Typography*, in: *Raster Imaging and Digital Typography: Proceedings of the International Conferences, Ecole Polytechnique Fédérale, Lausanne, Switzerland, October 1989*, ed. J. André and R. D. Hersch, pp. 251–260, Cambridge University Press, Cambridge, 1989.

R. O. Duda and P. E. Hart, *Pattern Classification and Scene Analysis*, John Wiley & Sons, New York, 1973.

S. M. Ermakov and A. A. Zhiglyavskii, "On random search for a global extremum," *Theory of Probability and its Applications*, vol. 28, pp. 136–141, 1983.

G. Farin, *Curves and Surfaces for CAGD, A Practical Guide*, Academic Press, New York, 1993 .

J. D. Foley, A. van Dam, S. Feiner, and J. Hughes, *Fundamentals of Interactive Computer Graphics*, Addison-Wesley, Reading, MA, 1992.

M. Gaviano, "Some general results on the convergence of random search algorithms in minimisation problems," in: *Towards Global Optimisation*, ed. L. C. W. Dixon and G. P. Szegö, pp. 149–157, North Holland, New York, 1975.

J. Gonczarowski, "A fast approach to auto-tracing (with parametric cubics)," in: *Raster Imaging and Digital Typography*, ed. R. A. Morris and J. André, vol. 2, pp. 1–15, Cambridge University Press, Cambridge, 1991.

Y. Haralambous, "Parametrization of PostScript fonts through METAFONT–alternative to Adobe multiple master fonts," *Electronic Publishing*, vol. 6, pp. 145–157, 1993.

J. D. Hill, "A search technique for multimodal surfaces," *IEEE Transactions on Systems, Science and Cybernetics*, vol. SSC-5, pp. 2–8, 1969.

J. D. Hobby, "Smooth, easy to compute interpolating splines," *Discrete Computational Geometry*, vol. 1, pp. 123–140, 1986.

K. Itoh and Y. Ohno, "A curve fitting algorithm for character fonts," *Electronic Publishing*, vol. 6, pp. 195–205, 1993.

R. A. Jarvis, "Adaptive global search by the process of competitive evolution," *IEEE Transactions on Systems, Man and Cybernetics*, vol. SMC-5, pp. 297–311, 1975.

P. Karow, *Digital Typefaces*, Springer-Verlag, Berlin, 1994a.

P. Karow, *Font Technology*, Springer-Verlag, Berlin, 1994b.

D. E. Knuth, *The* METAFONT *book*, Addison-Wesley, Reading, MA, 1986a.

D. E. Knuth, *The TEXbook*, Addison-Wesley, Reading, Mass, 1986b.

D. E. Knuth, *Computer Modern Typefaces*, Addison-Wesley, Reading, Mass, 1986c.

M. Leisher, "`bdftops`: a program to transform a BDF font into a PostScript font," New Mexico State University, 1990.

S. Lejun, Z. Hao, and C. K. Wah, "FontSript—A Chinese font generation system," in: *Proceedings of the International Conference on Chinese Computing (ICC94)*, pp. 1–9, 1994.

J. R. Manning, "Continuity conditions for spline curves," *The Computer Journal*, vol. 17, pp. 181–186, 1974.

J. Matyas, "Random optimization," *Automation and Remote Control*, vol. 26, pp. 244–251, 1965 .

S. Mazzucato, "Optimization of Bézier outlines and automatic font generation," M.Sc. thesis, School of Computer Science, McGill University, Montreal, 1994.

G. J. McMurtry and K. S. Fu, "A variable structure automaton used as a multimodal searching technique," *IEEE Transactions on Automatic Control*, vol. AC-11, pp. 379–387, 1966.

M. Minsky and S. Papert, *Perceptrons, an Introduction to Computational Geometry*, MIT Press, Harvard, MA, 1969.

R. Männer and H.-P. Schwefel, *Parallel Problem Solving from Nature*, Lecture Notes in Computer Science, vol. 496, Springer-Verlag, Berlin, 1991.

T. Pavlidis, *Algorithms for Graphics & Image Processing*, Computer Science Press, Rockville, MD, 1982.

M. Plass and M. Stone, "Curve-fitting with piecewise parametric cubics," *Computer Graphics*, vol. 17, pp. 229–239, 1983 .

J. Poskanzer, `pbmplus`*: a program to transform between various bitmap formats*, 1989.

A. S. Poznyak, "Use of learning automata for the control of random search," *Automation and Remote Control*, vol. 33, pp. 1992–2000, 1972.

F. P. Preparata and M. I. Shamos, *Computational Geometry, an Introduction*, Springer-Verlag, New York, NY, 1985.

I. Rechenberg, *Evolutionsstrategie—Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*, Frommann-Holzboog, Stuttgart, 1973.

A. H. G. Rinnooy Kan and G. T. Timmer, "Stochastic global optimization methods part I: clustering methods," *Mathematical Programming*, vol. 39, pp. 27–56, 1987a.

A. H. G. Rinnooy Kan and G. T. Timmer, "Stochastic global optimization methods part II: multi level methods," *Mathematical Programming*, vol. 39, pp. 57–78, 1987b.

K. K. Ripa, "Some statistical properties of optimizing automata and random search," *Automatic Control*, vol. 4(3), pp. 28–32, 1970.

K. K. Ripa, "Random search of the extremum of a multidimensional object as a stochastic automaton," in: *Problems of Statistical Optimization*, ed. L. A. Rastrigin, Zinatne, Riga, 1971.

P. J. Schneider, "An algorithm for automatically fitting digitized curves," in: *Graphics Gems*, ed. A. S. Glassner, pp. 612–626, Academic Press, San Diego, CA, 1990.

M. A. Schumer and K. Steiglitz, "Adaptive step size random search," *IEEE Transactions on Automatic Control*, vol. 13, pp. 270–276, 1968.

H.-P. Schwefel, *Modellen mittels der Evolutionsstrategie*, Birkhäuser Verlag, Basel, 1977.

H.-P. Schwefel, *Numerical Optimization of Computer Models*, John Wiley, Chichester, 1981.

I. J. Shapiro and K. S. Narendra, "Use of stochastic automata for parameter self-optimization with multimodal performance criteria," *IEEE Transactions on Systems, Science and Cybernetics*, vol. SSC-5, pp. 352–360, 1969.

B.-Q. Su and D.-Y. Liu, *Computational Geometry–Curve and Surface Modeling*, Academic Press, Boston, 1989.

E. W. Swokowski, *Calculus with Analytic Geometry*, Prendle Webes & Schmidt, Boston, MA, 1975.

A. Törn and A. Žilinskas, *Global Optimization*, Lecture Notes in Computer Science, vol. 350, Springer-Verlag, Berlin, 1989.

E. van Blokland and J. van Rossum, "Different approaches to lively outlines," in: *Raster Imaging and Digital Typography II*, ed. R. A. Morris and J. André, pp. 28–33, Cambridge University Press, Cambridge, 1991.

E. I. Volynskii and G. V. Filatov, "On step adaptation in random-search algorithms," *Automatic Control and Computer Sciences*, vol. 8(4), pp. 58–62, 1974.

L. W. Wallis, *Modern Encyclopedia of Typefaces, 1960-90*, Van Nostrand Reinhold, New York, NY, 1990.

A. A. Zhigljavsky, *Theory of Global Random Search*, Kluwer Academic Publishers, Hingham, MA, 1991.