INFORMATION TO USERS

NOTE TO USERS

The original manuscript received by UMI contains pages with indistinct and/or slanted print.   Pages were microfilmed as received.

This reproduction is the best copy available

UMI

.

Random dynamic fonts

Bernard Desruisseaux

School of Computer Science
McGill University, Montreal

October 1996

A thesis submitted to the Faculty of Graduate
Studies and Research in partial fulfilment of the
requirements of the degree of Master of Science

Canada

Random dynamic fonts

À ma mère,
qui aurait sûrement
trouvé cela amusant

À mon père,
sur qui je peux
toujours compter

Contents

Abstract

This thesis presents a general method and structure for storing, representing and reproducing random dynamic fonts, i.e., fonts whose each rendition of each letterform differs. Such fonts allow, e.g., to come closer to simulating true handwriting, by rendering its freedom, spontaneity, and unpredictability.

This method allows the generation of random letterforms with different overall shapes, derived from single letterform descriptions, according to specified parameters and constraints. Letterforms generated in this manner remain closely related—to a certain extent—to the original letterforms, and preserve the continuity and thickness of the strokes.

Several examples of a typeface family designed with this method, implemented as PostScript Type 3 font programs, are presented. A survey of the literature on random dynamic fonts is also proposed.

Résumé

Ce mémoire présente une méthode et structure générales pour mémoriser, représenter et reproduire des fontes dynamiques aléatoires, c'est-à-dire, des fontes dont chaque rendu de chaque caractère diffère. De telles fontes permettent, par exemple, une meilleure simulation de l'écriture manuscrite en traduisant sa liberté, sa spontanéité et son imprévisibilité.

Cette méthode permet la génération de caractères aléatoires de formes générales différentes, dérivés d'une seule description de caractère, selon des paramètres et contraintes spécifiés. Les caractères ainsi générés demeurent liés — dans une certaine mesure — aux caractères originaux et préservent la continuité et l'épaisseur des traits.

Plusieurs exemples d'une famille de fonte conçue avec cette méthode et implémentée sous forme de programmes de fontes PostScript Type 3 sont présentés. Un tour d'horizon de la littérature sur les fontes dynamiques aléatoires est aussi proposé.

Acknowledgments

First, I wholeheartedly thank my supervisor, Luc Devroye, for his invaluable comments, wisdom and enthusiasm for the subject. Thanks for keeping me on the right track, it has been a pleasure working with you.

I wish to thank Jacques André, at INRIA Rennes, who provided constructive criticism on the design of MetamorFont.

I also wish to thank the *Fonds pour la formation de chercheurs et l'aide à la recherche* (FCAR) for their financial support.

My deepest appreciation goes to my father, Gilles Desruisseaux, who kindly reviewed drafts of this thesis. Your editorial criticism was always to the point. Thank you for helping me.

Above all, I deeply thank my girlfriend, Sophie Gagné, for her love, understanding, and encouragement she has given me, especially during the writing of this thesis. Thanks for being who you are.

1 Introduction

In the early days of movable type, letter shapes were designed to seem handwritten, as all published material was then produced by hand. Johann Gutenberg (1398-1468), a pioneer* in movable type, worked 10 years to create a product that would reproduce nicely the Gothic handwriting of his day [43]. To reinforce the impression of original hand lettering as well as to blur the characteristics of the new technology, Gutenberg made several slightly different versions of the same letter.

Scribes, nevertheless, were trained to reproduce letters with identical shapes, so that the constant shapes of printed letters soon became accepted. The concept of characters as fixed geometrical objects was simply strengthened by the spreading of the movable type technology. As this concept is in accordance with most theories on legibility and readability, it comes as no surprise that typefaces, including script typefaces, are still used with fixed letterforms.

Nowadays type foundries offer extensive collections of script typefaces that simulate handwriting. Unfortunately, most of these typefaces lack the life and spirit of true handwriting. Letters with constant shape are simply inadequate to render the freedom, spontaneity, and unpredictability of handwriting.

The advent of computer technology brought new avenues to type design. The use of computers and digital devices led to new methods and

---

* Bi Sheng, a Chinese engineer, should be credited as the inventor of movable type, in preference to Gutenberg [15].

tools for the creation of typefaces. Generation of letterforms by mathematical means has become easier as computers carry out all computations.

Donald E. Knuth's font-design system, METAFONT [30], defines letterforms with sets of mathematical formulas from which their exact shapes are derived. Through the use of parameters, many shapes can be extracted from a single letterform description. If one of these parameters is a random variable, a random font is produced, i.e., every time the font is defined as a whole, a new set of letterform shapes is generated.

PostScript Type 3 font programs can define fonts in a parametric way and generate random fonts, but they can also vary these parameters dynamically. Therefore, each instantiation of each character differs from the last. Such fonts are called dynamic or random dynamic.

In general, script typefaces are used to bring elegance and simplicity to text. They are mainly used to draw the reader's attention and to create interesting contrast when used with conventional typefaces. Handwriting types can bring life to a page by suggesting the movement of the human hand. Random dynamic fonts reflect these movements more closely than do the conventional static fonts.

Although the use of random dynamic fonts may seem limited, these fonts are well fitted to a broad range of applications—announcements, brochures, greeting cards, restaurant menus, invitations, handwriting on posters, graffiti, comic strips (particularly for those published in many languages)—and wherever a touch of humor and warmth is desired.

Other fields might also benefit from such fonts: computer graphics, test samples for handwriting character recognition systems [46], and graphic design. Indeed, through experimentation with random dynamic fonts, interesting new letterforms may be discovered [25].

Random dynamic fonts could also be put to profit with the whole new class of typefaces that render the irregularities of printing tools such as

smudged typewriter machines, rubber stamps and hand-held label makers. Other effects, such as overprinting, inkspreading [6], and hand-sketched pressure brushstrokes [41] can also be achieved.

Finally, the economic advantage of random dynamic fonts over the services of lettering artists is obvious. The typeface has to be paid for only once, and it can be used as much and as long as the buyer wants, and at any time of the day!

Random dynamic fonts have already been explored by other computer scientists [11, 9, 18] and graphic designers [47, 48, 25]. This thesis presents another vision, while going much deeper at the programming and conceptual levels. The method presented here goes further than simply adding random perturbations to character outlines or interpolating character representatives. It allows alterations to the overall letterform shapes within predetermined limits.

While the technical and conceptual aspects are the primary concern of this research, the creation of beautiful, functional and inconspicuous random dynamic fonts is nevertheless its main intent. Random dynamic fonts should, however, be used sparingly as they are less legible and readable than conventional fonts. As Robin Williams [50, p. 89] puts it, "Scripts are like cheesecake—they should be eaten sparingly. I mean, *used* sparingly."

The versatility of the method described allows the design of conventional fonts as well as fonts that can be grouped in what has been called radical or grunge typography (see [51]) or part of the Broken Art movement, fonts mainly targeted at Generation X, or so-called experimental. It should be noted that the method is not limited to the creation of fonts but could also be applied to ornaments or any kind of drawing.

In the next chapter, the different digital font formats are discussed, and the mathematics of Bézier curves are introduced. Chapter 3 surveys several random dynamic fonts methods, and explains how the PostScript language

can be used to implement such fonts. The following chapter, the heart of this thesis, covers the proposed method in detail. Chapter 5 presents several technical examples of a typeface designed with the method. Finally, the interaction between PostScript random dynamic fonts, TeX and LaTeX, is considered. Complementary technical information can be found in the appendices.

2 Digital Fonts

This chapter gives a quick overview of the different representations of digital fonts, and introduces the mathematical foundations of Bézier curves, commonly used in such fonts. The whole area of digital fonts is covered at length in [26, 27, 43], and a comprehensive introduction can be found in [7].

2.1 Font Formats

Digital fonts are complex data structures used to store, represent, and reproduce sets of characters in a form suitable for digital output devices. Several coding methods are applied to represent such fonts: bitmaps, run lengths, vectors, circles, g-conics, splines and spirals [26]. They differ mostly in their storage usage, efficiency, accuracy, and convenience to scale up and down.

Bitmaps and run lengths simply specify which *picture elements* or *pixels* must be painted and therefore are tuned to a particular size and resolution. Although practically no computation is involved, a sizable amount of storage space is used. Other methods store the letterform shapes as outlines and provide collections of points used to describe different graphical constructs mathematically. The latter are more prone to scaling and occupy less storage space, but they require significant computations to generate the letterform shape at each size and resolution.

Nowadays, most digital fonts store the letterform shapes as outlines

(a) (b) (c)
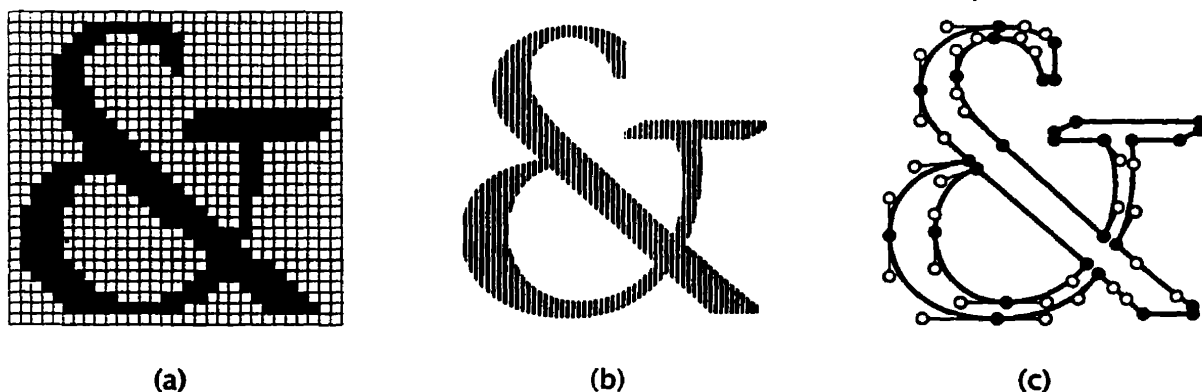
Figure 2.1. Various digital font formats: *(a)* bitmap, *(b)* run lengths, and *(c)* outline using cubic Bézier curves.

mathematically expressed by splines, and more precisely as collection of smoothly joined Bézier curves. All the curves drawn by METAFONT [30] and PostScript [3] are based on cubic Bézier curves, and those by TrueType [12] on quadratic Bézier curves.

2.2 Bézier Curves

Bézier curves were independently developed by two French automobile engineers, Paul de Casteljau of Citroën, in 1959, and Pierre Bézier of Renault, around 1962. Although de Casteljau preceded Bézier in his discovery, these curves have been named after Bézier, whose work was published first. Bézier curves are now widely used in computer-aided design (CAD) systems. A comprehensive treatment of the basic methods in curve design can be found in [19, 16, 37].

The de Casteljau Algorithm

Paul de Casteljau developed an algorithm to construct curves of arbitrary degree n from a sequence of points $b_0, \ldots, b_n$, the *control points*, that form a polygon, called the *characteristic polygon*. The algorithm can be formu-

lated recursively as follow:

$$b_i^r(t) = (1-t)b_i^{r-1}(t) + tb_{i+1}^{r-1}(t) \qquad \begin{cases} r = 1, \ldots, n \\ i = 0, \ldots, n-r \end{cases}$$

in which t varies from 0 to 1 and $b_i^0(t) = b_i$.
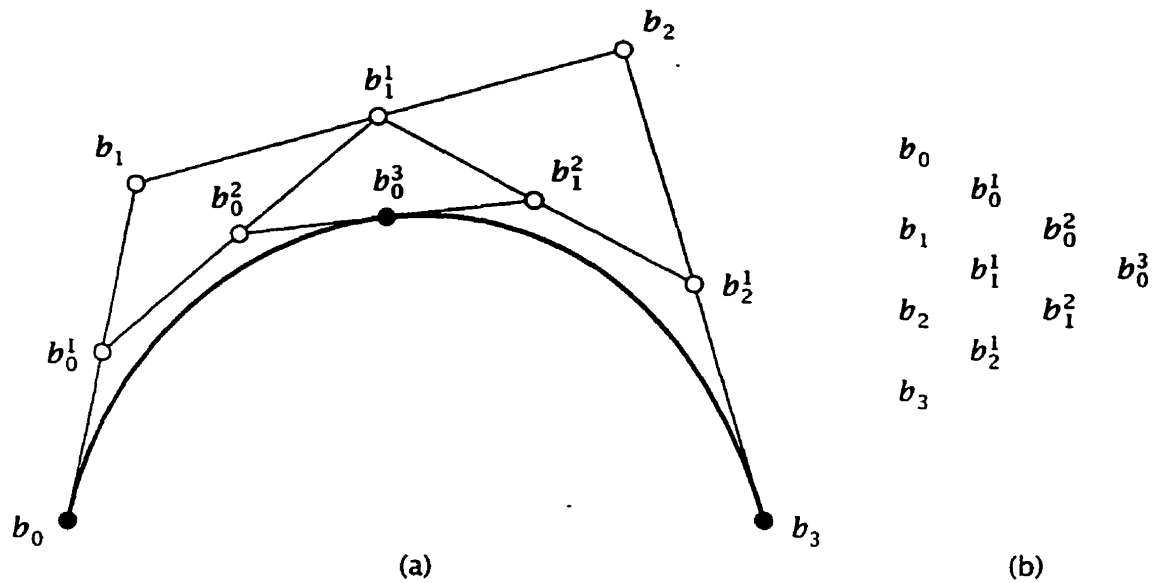
Figure 2.2. The de Casteljau algorithm: *(a)* cubic Bézier curve with $t = 0.5$, *(b)* scheme for cubic Bézier curve.

De Casteljau also discovered an interesting and simple way, somewhat related to his algorithm, to construct Bézier curves. By geometric subdivision of the control polygon of a curve, two new control polygons, as well as a new point on the curve, are obtained. The remaining points of the curve are found recursively with the same process being repeated ad infinitum on each new control polygons. In practice, this process converges quickly and is simply carried out until the curve is actually drawn, some execution efficiency being gained at the expense of accuracy.

Bézier Form

Pierre Bézier suggested that curves be defined as a linear combination of a certain class of functions f_i, referred to as *blending* or *basis* functions. A curve $\mathcal{B}$ of arbitrary degree n could be described by a parametric function of the form:

$$\mathcal{B}(t) = \sum_{i=0}^{n} b_i f_i(t), \qquad 0 \le t \le 1.$$

As blending functions, Bézier chose the density functions of the binomial distribution. It was later discovered, by A. R. Forrest, that Bézier curves could be written in terms of *Bernshtein polynomials*:

$$B_i^n(t) = \binom{n}{i} t^i (1 - t)^{n-i}, \qquad 0 \le i \le n.$$
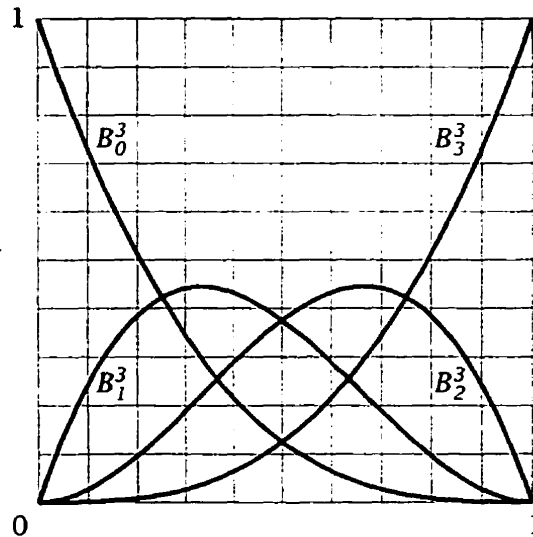
Figure 2.3. Bézier cubic blending functions: third degree Bernshteïn polynomials.

Cubic Bézier Curves

Cubic Bézier curves are so widely used that they deserve expanded attention. Mathematically, a cubic Bézier curve is derived from a pair of parametric cubic equations:

$$x(t) = a_x t^3 + b_x t^2 + c_x t + x_0$$

$$y(t) = a_y t^3 + b_y t^2 + c_y t + y_0$$

where t ranges from 0 to 1, and the curve end points $(x_0, y_0), (x_3, y_3)$ and intermediate control points $(x_1, y_1), (x_2, y_2)$ are defined as:

$$x_1 = x_0 + c_x/3 \qquad\qquad y_1 = y_0 + c_y/3$$

$$x_2 = x_1 + (c_x + b_x)/3 \qquad\qquad y_2 = y_1 + (c_y + b_y)/3$$

$$x_3 = x_0 + c_x + b_x + a_x \qquad\qquad y_3 = y_0 + c_y + b_y + a_y$$

which can be converted into the equivalent Bernshteïn polynomial format:

$$x(t) = x_0(1 - t)^3 + 3x_1 t(1 - t)^2 + 3x_2 t^2(1 - t) + x_3 t^3$$

$$y(t) = y_0(1 - t)^3 + 3y_1 t(1 - t)^2 + 3y_2 t^2(1 - t) + y_3 t^3.$$

Cubic Bézier curves provide the necessary flexibility to obtain satisfactory approximations to a large number of curves. The figure below illustrates the various shapes cubic Bézier curves can take.

(a)   (b)   (c)   (d)
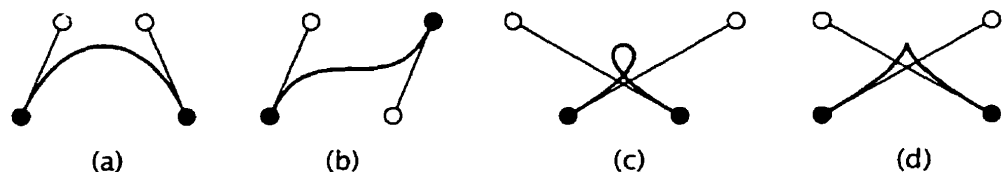
Figure 2.4. Various shapes of Bézier cubic curves: (a) convex, (b) with an inflection point, (c) with a loop, and (d) with a cusp.

Properties

Any Bézier curve share the following important geometrical properties:

- It begins at (x_0, y_0), heading in the direction from (x_0, y_0) to (x_1, y_1);

- It ends at (x_3, y_3), heading in the direction from (x_2, y_2) to (x_3, y_3);

- It lies entirely within the *convex hull* (see, e.g., [40]) defined by the characteristic polygon;

- It is invariant under affine maps, i.e., affine maps applied to the control points of the curve or to the computed points of the curve yield the same result;

- It is variation-diminishing, i.e., it never oscillates wildly away from its control points.

3 Random Dynamic Fonts

This chapter presents the two classes of digital fonts, and surveys different approaches to random dynamic fonts. Then, the PostScript language and its different font types are introduced, and the use of the PostScript language for the implementation of random dynamic fonts is explained and justified.

3.1 Font Classes

Digital fonts can be divided in two classes, static fonts and dynamic fonts [9]. Letterforms from static fonts can be viewed as fixed geometrical objects. First, they are designed by an artist, then digitized in some way, and finally used in a printing process. This results in letterforms of constant shape. Almost all fonts belong to this class.

Dynamic fonts are fonts whose letterform shape is defined every time the corresponding letterform is printed rather than when the font is defined as a whole.

3.2 Survey of the Field

Many font parameters can be controlled, randomized, and rendered dynamically. The main approaches worked out by others are presented according to an informal classification.

Random Perturbation of Control Points

The first steps towards random fonts involved random perturbations of the character shape descriptors. In *The* METAFONT*book* [30, chap. 21], Donald E. Knuth explains how to use METAFONT to produce random fonts by carefully adding noise to the characters' control points. Knuth basically used this approach to implement his Punk typeface [32], illustrated below.

$$\text{AAA} \quad \text{\textit{AAA}} \quad \text{AAA}$$

(a)   (b)   (c)
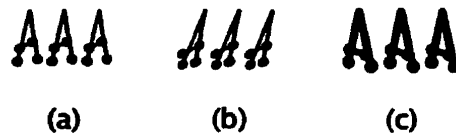
Figure 3.1. The Punk family: *(a)* roman, *(b)* slanted, and *(c)* bold.

Dynamic Fonts and PostScript

Being a batch font design system, METAFONT does not allow the creation of dynamic fonts. On the other hand, the PostScript font machinery allows on-line changes through the use of its Type 3 font programs (see, e.g., André and Borghi [9] or Sherman [44, p. 164]). André and Ostromoukhov [11] converted the Punk typeface to a PostScript Type 3 font program to make it dynamic. Packard [38] applied this technique to ransom fonts.

The use of the PostScript Type 3 font format does not imply the use of any specific method. On the contrary, it provides a general mechanism for the creation of random dynamic fonts. Furthermore, it is the format used in most cases for the implementation of the following methods. More details on the PostScript language [3] and its font machinery are given in section 3.3.

Outline Texture

Texture is an aspect of fonts that can be experimented with. Nowadays, most software applications can use fonts to define clipping paths (see McGilton and Campione [36]) that can be filled with special pattern defining textures, thus leaving texture handling separated from font design. However, the texture of the outlines used to define these clipping paths should be part of the font design.

Graphic designers Erik van Blokland and Just van Rossum developed several methods to produce rough and lively outlines [48]. Their first method is similar to the one used by Knuth for the Punk typeface, but the random perturbations are applied directly at the control points of the character outlines and in a carefree way.

Interrobang

Figure 3.2. Random perturbation of the control points in LucidaBright [13, 52].

The result of this method is closely related to the number of control points in the outline, in other words, to the length of each graphical component. Small perturbations can sometimes cause large differences in weight and yield uneven strokes. Thus, a second method was developed to achieve smoother roughness in the outlines. Basically, the original outlines are simply converted into a sequence of short straight lines prior to the random perturbations. Their Beowolf family [47] was implemented as PostScript Type 3 font programs using this method.

More complex algorithms can also be used to produce texture effect. Erik van Blokland and Just van Rossum present samples of simulated un-

Interrobang

Figure 3.3. Random perturbation of the control points—after conversion into sequence of short straight lines—in LucidaBright [13, 52].

der- and over-exposure of type in [48], and Jacques André developed a method for inkspreading simulation [6].

Multiple Caching

In order to speed up the rendition process, Erik van Blokland and Just van Rossum [48] proposed a method that trades memory for speed. The solution is to maintain multiple caches of randomized outlines, each cache containing a different instance of each character, and to randomly select one cache whenever a character is requested. Although limited, this method may be adequate under certain circumstances.

Geometrical Transformations

The appearance of characters can be controlled, as well as their position and orientation on a page. Random geometrical transformations can be applied to the whole characters to achieve interesting effects. Vertical translations produce a variable baseline [48], as observed in some handwriting. Rotation can be used, as shown by André in his Scrabble font [5], to move each tile by a randomly defined angle. Lastly, variations in the character size can also be achieved with scaling, provided a uniform character thickness is ensured.

Simulation of Handwriting

Luc Devroye and Michael McDougall presented three methods for the simulation of handwriting using random dynamic fonts [18]. The first one used the random interpolation and extrapolation of multiple representative characters. Easy to implement and robust, this method is also fast since little computation is required.

The drawback of this method is that all the representative characters must be designed a priori. Furthermore, every representative must have the same number of control points, and each of them should play the same role in the letterform shape than its counterparts in the other representatives. A simple solution used by Margo Johnson [25], and Luc Devroye (in unpublished work), is to generate the representative characters from a single master character using special transformations.
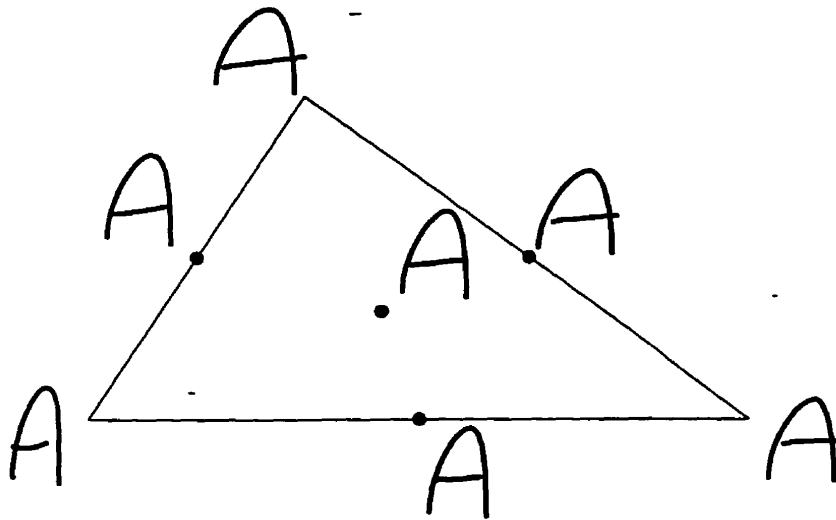
Figure 3.4. Random interpolation of multiple representative characters. (This figure is taken from [18], and is provided by Luc Devroye, McGill University.)

The second method, random selection of points from the minimal spanning tree, guarantees the production of points that are always uniformly

distributed inside the convex hull of the data points. This condition is not met in the first method when the number of representatives exceeds the dimension of the space by more than one. In a nutshell, once the minimal spanning tree of the data points has been determined (see, e.g., Cormen et al. [17, chap. 24]), an edge of the tree should be picked at random, a point should be generated uniformly at random on the edge, and then the interpolation should be performed to render the character.
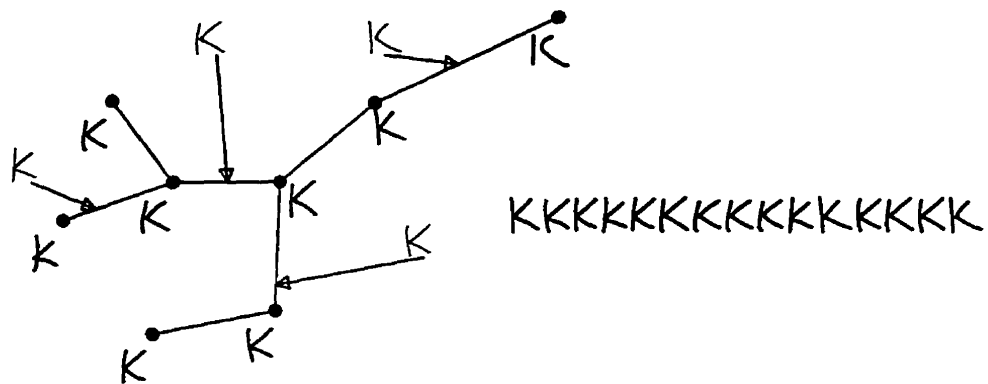
Figure 3.5. Random selection of points from the minimal spanning tree. (This figure is taken from [18], and is provided by Luc Devroye, McGill University.)

The last method presented, the kernel method, is the controlled random perturbation of Bézier control points in order to preserve C^1 continuity (see, e.g., Su and Liu [16] or Farin [19]) given a smoothing factor. The goal of this method is to generate new points with the same distribution as the original data points, as opposed to previous methods which exclusively use uniform distributions.

Contextual Fonts

Characters can also vary dynamically according to their context, e.g., the preceding and succeeding letters, beginning or end of words. Contextual

fonts, as they are called, are simple dynamic fonts where the context is used as a parameter. Randomness is not always involved in such fonts.

Kokula developed a method to smoothly link script font characters on-the-fly [34]. Great attention has been paid to the natural appearance of the curves joining characters. Details are given on how the proposed algorithm can be integrated into PostScript Type 3 font programs.

Signature Software [45] sells personalized contextual (cursive) hand-writing fonts based upon samples from a person's own hand. Although their products are mainly targeted toward the TrueType technology, contextual PostScript Type 3 font programs are also made available. Their solution is somewhat similar to the multiple caching method, although the selection process is deterministic and depends only on the context. Unlike Kokula, the context handling is not integrated into the font program, but PostScript primitives such as **show** are redefined, which may lead to unexpected results.

Portez ce vieux whisky au juge blond qui fume.   (a)

Portez ce vieux whisky au juge blond qui fume.   (b)

Portez ce vieux whisky au juge blond qui fume.   (c)

Portez ce vieux whisky au juge blond qui fume.   (d)

Portez ce vieux whisky au juge blond qui fume.   (e)
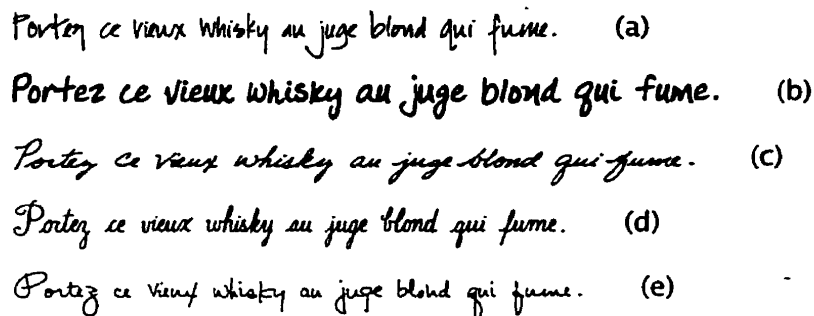
Figure 3.6. Signature Software SUPERscripts: *(a)* SigJocelyn, *(b)* SigLisa, *(c)* SigTsui, *(d)* SigVictoria, and *(e)* SigWilson.

Other interesting contextual fonts have also been developed. André and Borghi [9, Fig. 7] present a simple contextual font that adapts itself to its graphical environment. André and Delorme [10] and [7, pp. 87–95] developed the *Delorme* typeface, which offers interesting features for logo design.

3.3 PostScript and Random Dynamic Fonts

PostScript is a powerful graphical page description language, introduced by Adobe Systems in 1985. Although PostScript's primary application is to describe the appearance of text, graphics, and images on a printed page or display, it is, nevertheless, a full-featured, interpreted, programming language. Like most high-level programming languages, PostScript provides a conventional set of data types, control primitives, and general-purpose operators. However, PostScript differs from most languages by incorporating a postfix notation, in which operators are preceded by their operands, and by the extensive use of stacks and dictionaries, which form the heart of PostScript. The complete specification of the PostScript language appears in the *PostScript Language Reference Manual* [3], and comprehensive introductions can be found in [1, 36].

Font Dictionaries

PostScript unifies text and graphics by treating letter shapes as general graphic shapes that may be manipulated by any of the language's graphics operator. However, since letters are used so frequently, the PostScript language provides higher-level facilities to describe, select and render characters conveniently and efficiently.

In the PostScript language, sets of characters are organized into fonts, which in turn are nothing but PostScript dictionaries that conform to specific conventions and are registered with the font machinery. PostScript currently has three font types, each with its own conventions for organizing and representing font information.

- Type 0, known as *composite fonts*, are combinations of other fonts, base fonts or descendant fonts, that in turn may be any kind of font—Type 1, Type 3, or even another Type 0. This font type was created

to meet the needs of Asian languages that use large character sets—much larger than the 256 characters limit of the other font types. This format is described in [3].

- Type 1, sometimes referred to as *standard fonts*, define character shapes in a compact way using a subset and an extension of the Post-Script language, following a much stricter syntax, and a rigorously defined structure. Type 1 font programs can include special information, called *hints*, to improve the appearance of characters rendered at small sizes and low resolutions. The official specification for this format appears in *Adobe Type 1 Font Format* [2].

- Type 3, or *user-defined fonts*, define character shapes as ordinary PostScript language procedures. Unlike Type 1 fonts, Type 3 fonts do not provide a *hinting* mechanism, although one could be implemented. There are very few restrictions on this format, as the font developer is free to use whatever method and structure to supply the character descriptions. Not as efficient and convenient as Type 1, Type 3 fonts come in handy whenever complex graphic constructions, color setting operators, image operators, or font cache control operators are involved. This format is described in [3].

Font Cache

Since letters are used repeatedly, the PostScript interpreter's font machinery includes a font cache to optimize the character rendering process. The font cache stores the results of a character *scan conversions* (see, e.g., [22]) in an internal data structure. Thus, when a character is requested again the font cache provides the character bitmap without any computation.

The font cache does not retain color information and disallows the execution of the **image** operator. PostScript provides two operators to con-

trol the behavior of the font cache mechanism within Type 3 fonts. The **setcachedevice** operator is used to declare the character metrics and requests the font machinery to store the computed bitmap into the font cache, and **setcharwidth** is simply used to declare the character metrics, bypassing the font cache mechanism. The use of **setcharwidth** is mandatory when dynamic fonts are called for, otherwise the font cache mechanism would prevent the differentiation of characters.

A Sensible Choice

As a device- and resolution-independent page description language, Post-Script provides a common software interface to deal with the general class of raster output devices. Since its introduction, many PostScript interpreters have been developed to control a wide variety of output devices. Through the years, the PostScript language has become the industry standard for imaging high-quality graphics and text.

Since the PostScript language is currently supported by many software applications and hardware platforms, and as it provides a flexible development environment as a full-featured programming language, it seems a sensible choice for the implementation of random dynamic fonts, which is the topic of the following chapter.

4 Method Proposed

This chapter presents a general method and structure for storing, representing and reproducing random dynamic fonts. The first section is a general description of the method—the letterform description and the randomization process. Subsequent sections discuss the implementation of the method using the PostScript Type 3 font format, the organization of font dictionaries, and the parametrization of fonts.

4.1 Making Random Letterforms

The method developed here allows the generation of random letterforms with different overall shapes, derived from single letterform descriptions, according to specified parameters and constraints. Letterforms generated in this manner remain closely related—to a certain extent—to the original letterforms, and preserve the continuity and thickness of the strokes.

Letterform Description

Letterform description is the heart of any font program. Letterforms described directly as collections of connected and unconnected Bézier curves are inadequate, as many letterform shapes are to be derived from single descriptions. Preserving the continuity of smoothly connected curves imposes constraints too severe to yield any interesting results. Indeed, a greater abstraction is required in the letterform description if pleasing letterforms are called for.
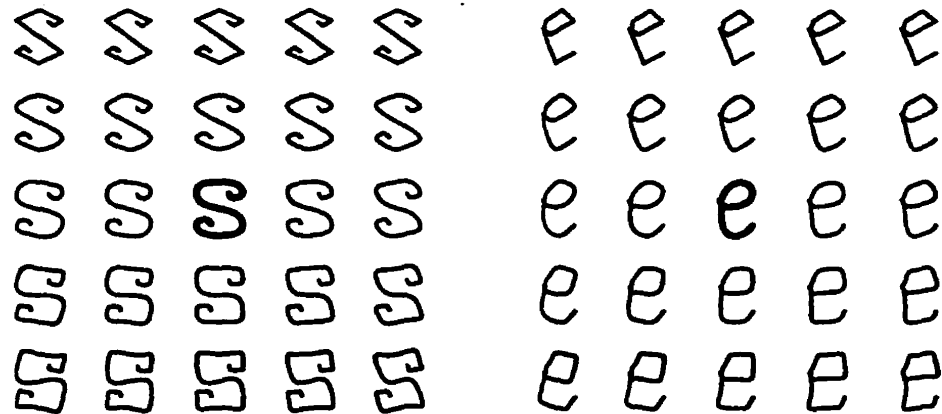
Figure 4.1. Simple continuity-preserving transformations: tension variations on control points, in the spirit of the **tension** parameter in METAFONT [30, pp. 15-16] *(increasing from top to bottom)*; and rotations of control points around junction points *(angles varying counterclockwise from left to right)*. Original letterforms, shown in bold, are from the Birke typeface designed by Luc Devroye, McGill University.

Complete letterforms can be described at a higher level with spline curves, i.e., continuous curves composed of several polynomial segments. Splines provide more flexibility and can easily yield continuous curves of complex shapes. Spline curves are described by polygons, referred to as spline characteristic polygons.
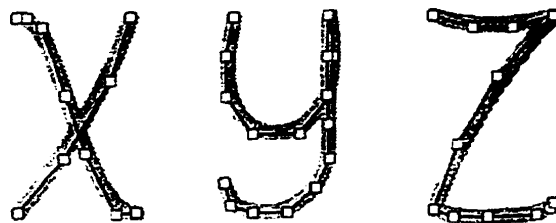
Figure 4.2. Letterforms described by spline characteristic polygons.

METAFONT provides an algorithm, due to Hobby [23], for smooth interpolating splines, i.e., splines that smoothly pass through the data points. On the other hand, PostScript does not support splines directly, thus, a spline interpolation or approximation algorithm has to be implemented.

The algorithm selected to convert spline characteristic polygons into sequences of continuous cubic Bézier curves is described in section 4.2.

To ensure a uniform thickness of the strokes, letterforms are defined inline, i.e., they are created by inking along a path, instead of filling outlines (see Fig. 2.1c). The method provides a calligraphic pen effect, controlled by several parameters, to create more interesting shapes. Otherwise, all letterforms would look as if they were drawn by a felt-tip pen with a perfectly round nib.

$$e \; e$$

Figure 4.3. Calligraphic pen (on the right) creates more interesting shapes.

Randomization Process

Working with spline characteristic polygons greatly simplifies the randomization process. Indeed, random letterforms can be obtained by simply applying perturbations to each vertex of the polygons in a carefree way. Unfortunately, only small perturbations yield interesting letterforms, as large perturbations simply destroy the nature of the original letterforms, even though continuity is preserved.

In order to preserve the shapes of certain parts of the letterforms, for the sake of decipherability, polygons can be divided into sections at design time. Sections are lists of consecutive vertices that shall preserve their relative position among themselves through the randomization process, and thus preserving the shape they describe in an affine manner.

For better control over the randomization process, constraints, such as
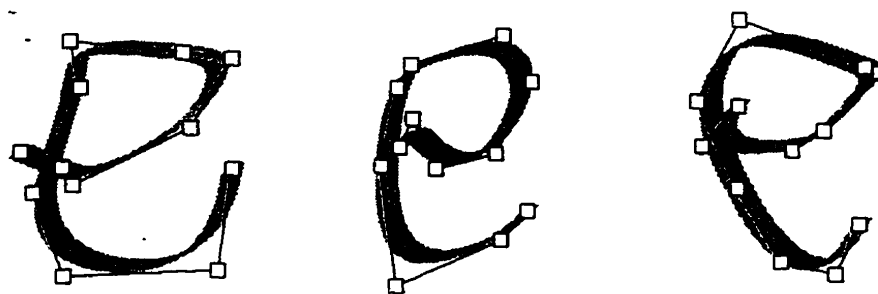
Figure 4.4. Large perturbations simply destroy the nature of the original letterforms.
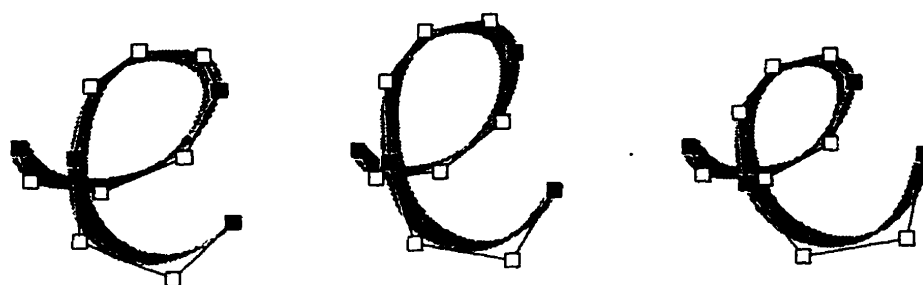
Figure 4.5. Sections, shown delimited by black squares, preserve the shape they describe.

the maximum perturbations allowed, can added to each vertex of the spline characteristic polygon.

After the randomization process it is possible to restore the original height or width or both of the spline characteristic polygon, by applying simple scaling and translation operations to the polygon. This allows to preserve a uniform letterform height, as well as the original letterform width. Preserving the letterform width is particularly important when typesetting text with systems, such as TEX, that rely on static metric information.

Lastly, small random vertical translations can be applied to the whole spline characteristic polygon to break the monotonous baseline that would be produced otherwise.
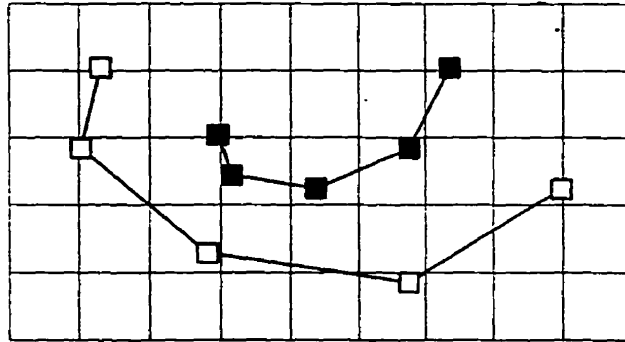
Figure 4.6. Distortion of a single section: all vertices are transformed in an affine manner.

4.2 Cubic Spline Curves

Cubic Bézier curves cannot model every possible curve. Curves with complex shapes could be best approximated by Bézier curves of higher degree but with significant computational complexity increase. Such complex curves can, however, be modeled using *composite Bézier curves*, also known as *piecewise polynomial curves*, or simply called *spline curves*. Spline curves can be represented in terms of B-spline functions, in which case they are called B-spline curves. Bézier curves of degree n are special cases of B-spline curves of degree n. This section focuses on cubic spline curves.

Continuity of Connected Curves

Connected curve segments are characterized by their order of differentiability, or say continuity. A spline curve is a continuous map of a collection of intervals $u_0 < \ldots < u_L$, where each interval $[u_i, u_{i+1}]$ is mapped onto a polynomial curve segment. In the following, the length of an interval shall be noted as $\Delta_i = u_{i+1} - u_i$.

Two Bézier curve segments with polygons $b_0, \ldots, b_n$ and $b_n, \ldots, b_{2n}$, i.e., sharing a common end point, are said to have at least C^0 continuity. Furthermore, if the curve segments also share a common tangent line at their
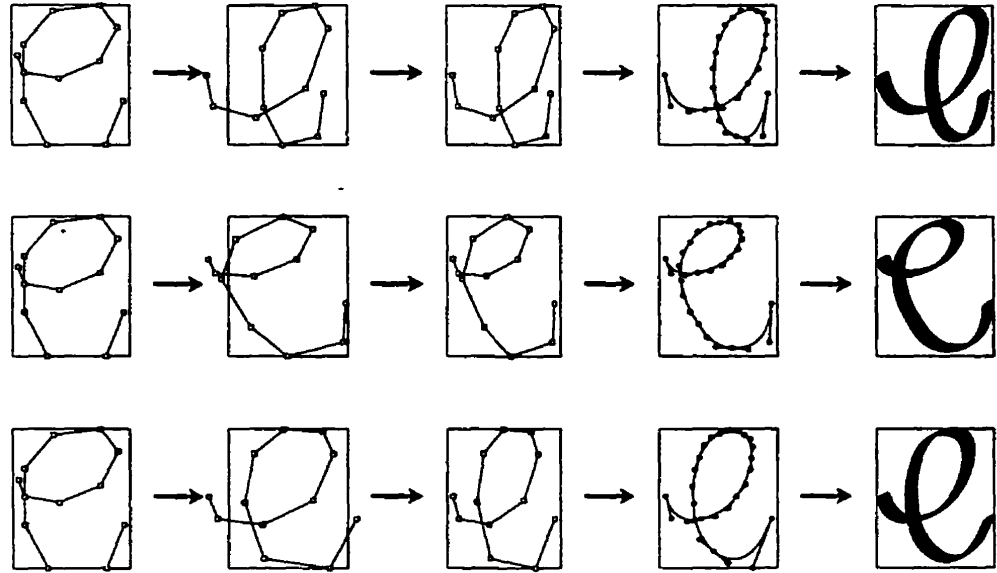
Figure 4.7. Randomization process: original letterform, randomization of letterform, restoration of height and width, generation of Bézier control points, stroke with calligraphic pen.

junction point, i.e., b_{n-1}, b_n, b_{n+1} are collinear, and moreover are in the ratio $(u_1 - u_0) : (u_2 - u_1) = \Delta_0 : \Delta_1$, that is

$$b_n = \frac{\Delta_1}{\Delta_0 + \Delta_1} b_{n-1} + \frac{\Delta_0}{\Delta_0 + \Delta_1} b_{n+1},$$

then C^1 continuity is obtained. Lastly, if the two curves possess equal curvature at their joint, or on the geometrical view, an auxiliary point d exists such that the points b_{n-2}, b_{n-1}, d and d, b_{n+1}, b_{n+2} are in the same ratio, that is

$$b_{n-1} = \frac{\Delta_1}{\Delta_0 + \Delta_1} b_{n-2} + \frac{\Delta_0}{\Delta_0 + \Delta_1} d,$$

$$b_{n+1} = \frac{\Delta_1}{\Delta_0 + \Delta_1} d + \frac{\Delta_0}{\Delta_0 + \Delta_1} b_{n+2}.$$

then C^2 continuity is obtained.
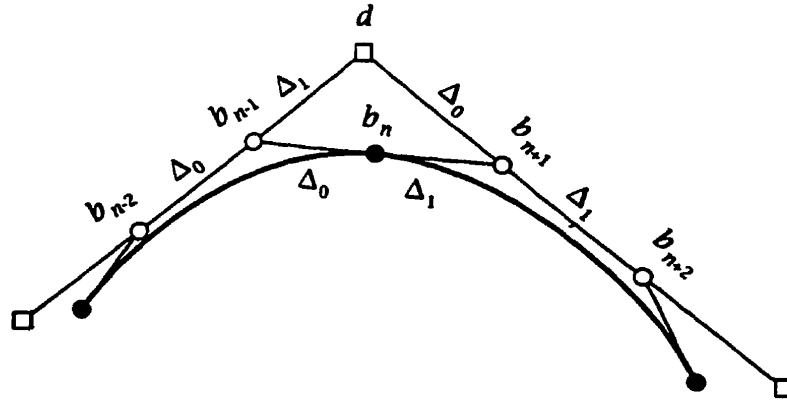
Figure 4.8. Geometric conditions required for C^2 continuity.

Böhm Construction Algorithm

Wolfgang Böhm [14] derived a simple algorithm, based on the continuity conditions stated above, to determine the location of the Bézier control points of a spline curve such that the C^2 constraints are satisfied. Thus, the well known representation of curves in Bézier form is carried over to splines.

Given a spline characteristic polygon with vertices d_j; $0 \leq j \leq L$, the Bézier control points for a C^2 continuous spline are calculated as follows:

Divide $\quad d_{j-i}, d_j \quad$ by $\quad b_{3j-2}, b_{3j-1} \quad$ in the ratio $\quad \Delta_{j-1} : \Delta_j : \Delta_{j+1}$,

divide $\quad b_{3i-1}, b_{3i+1} \quad$ by $\quad b_{3i} \quad$ in the ratio $\quad \Delta_i : \Delta_{i+1}$.

If the points d_j describe a closed polygon, these operations simply need to be performed in modulo L. On the other hand, if the points d_j describe an open polygon, the process gets a little more complicated near the ends. The spline polygon is defined to have vertices $d_{-1}, d_0, \ldots, d_L, d_{L+1}$ and then the extreme Bézier control points are sets as follows:

$$b_0 = d_{-1},$$

$$b_1 = d_0,$$

$$b_2 = \frac{\Delta_1}{\Delta_0 + \Delta_1} d_0 + \frac{\Delta_0}{\Delta_0 + \Delta_1} d_1,$$

$$b_{3L-2} = \frac{\Delta_{L-1}}{\Delta_{L-2} + \Delta_{L-1}} d_L + \frac{\Delta_{L-1}}{\Delta_{L-2} + \Delta_{L-1}} d_0 +$$

$$b_{3L-1} = d_L,$$

$$b_{3L} = d_{L+1}.$$
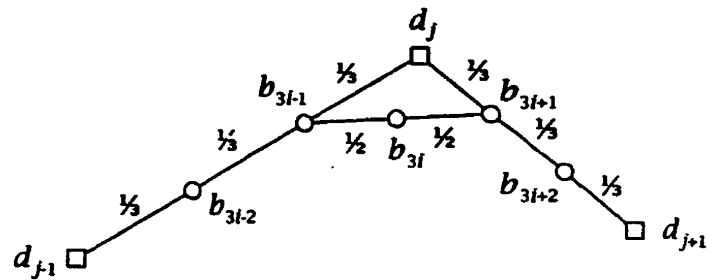
Figure 4.9. Special case of equidistant partition, as used in the implementation.
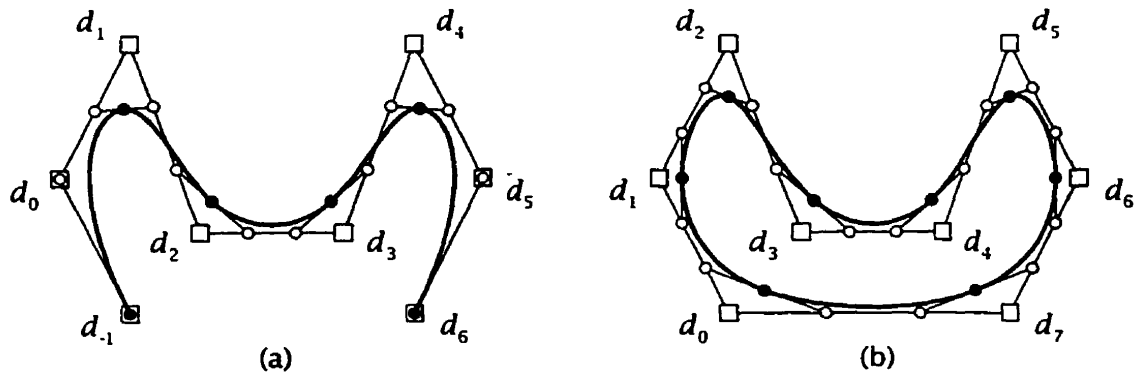
Figure 4.10. Böhm construction: *(a)* open-end polygon, *(b)* closed polygon.

Properties

Spline curves share most properties of Bézier curves (see Sect. 2.2). Spline curves provide a better local control than Bézier curves. Perturbing a single vertex of the characteristic polygon produces only a local perturbation of the curve in the vicinity of that vertex.

Yet another useful property in the design of letterforms is the multiplicity of vertices of spline characteristic polygons. It is possible to refine the shape of a curve by repeating a vertex once of more times in the sequence of vertices of a characteristic polygon. Repeating a vertex increase its multiplicity by one.
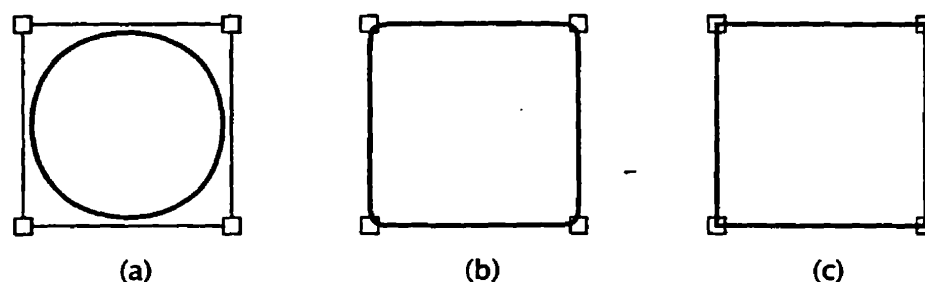
Figure 4.11. Polygons with all vertices having multiplicity: *(a)* 1, *(b)* 2, and *(c)* 3.

4.3 Font Program Organization

The method has been implemented into PostScript Type 3 font programs. A PostScript Type 3 font program is a collection of procedures describing letterform shapes, organized into a PostScript dictionary. Unlike Type 1 font programs, there is no such thing as a "typical Type 3 font program" as very few restrictions are put on their structure and format. This section describes the organization of the font programs used to implement the method. To facilitate support with most software applications, and under-

standing by Type 1 font designers/programmers, conventions pertaining to Type 1 font programs have been followed whenever possible.

Font Dictionnaries

The following table describes the entries of a font dictionary implememented with the method.

Key	Type	Semantics
FontType	integer	Indicates where the information for the character descriptions is to be found and how it is represented.
FontMatrix	array	Transforms the *character coordinate system* into the user coordinate system.
FontName	string	The name of the font.
FontInfo	dictionary	Provides information about the font, for the benefit of programs using the font. Entries are described in a separate table.
LanguageLevel	integer	Minimum language level required for correct behavior of the font.
Encoding	array	Array of names that maps character codes to character names.
FontBBox	array	Array of four numbers in the character coordinate system giving the overall font bounding box.
UniqueID	integer	Integer in the range 0 to $2^{24} - 1$ that uniquely identifies the font.
(FID)	fontID	Font identifier generated by the **definefont** operator for internal purposes in the font machinery.

Key	Type	Semantics
CharProcs	dictionary	Provides a procedure for each letterform defined in the font. Letterform procedure can also call subroutines.
FontMetrics	dictionary	Contains the computed left sidebearing and width of each letterform.
BBox	dictionary	Contains the computed bounding box of each character in the font, taking into account the specified left sidebearings.
RealBBox	dictionary	Contains the bounding box of each character in the font.
FontParams	dictionary	Contains global font parameters. Entries are described in the following section.
CharParams	dictionary	Contains individual characters parameters. Entries are described in the following section.
SideBearings	dictionary	Contains the left and right sidebearings of each letterform.
KerningPairs	dictionary	Contains all kerning pairs value. See Fig. 4.15 for example of kerning.
Ligatures	dictionary	Contains all ligatures. See Fig. 4.16 for example of ligatures.

The **CharProcs** dictionary provides the description of each letterform.

```
/e {
  [
    [
      [
        [ 42 313 << 25 50 25 25 box >> ]
        [ 63 250 ]
        [ 188 229 ]
        [ 333 292 ]
        [ 396 417 << 25 75 50 25 box >> ]
      ] [
        [ 333 500 ]
        [ 167 479 ]
        [ 63 354 ]
        [ 63 146 << 50 75 25 75 box >> ]
      ] [
        [ 146 -17 ]
        [ 354 -17 ]
        [ 417 146 << 50 100 50 50 box >> ]
      ]
    ]
  ]
} def
```
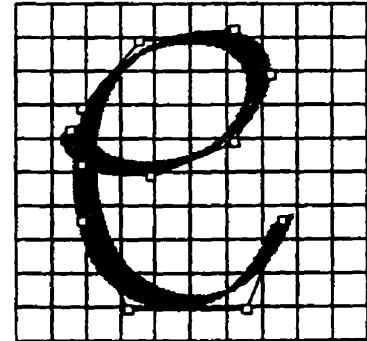
Figure 4.12. Example of a typical letterform description. The << and >> operators are the PostScript Level 2 short-cut dictionary constructors. **box** is a short-hand operator to define the four parameters **-dx**, **-dy**, **+dx** and **+dy** according to its four operands.

$$+dy$$
$$\uparrow$$
$$-dx \longleftarrow \bullet \longrightarrow +dx$$
$$\downarrow$$
$$-dy$$
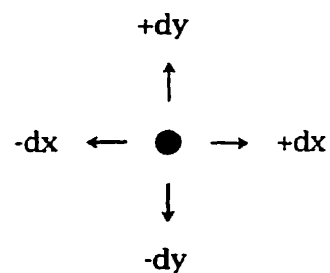
Figure 4.13. Perturbation of the vertices of spline characteristic polygons are done according to the four parameters **-dx**, **-dy**, **+dx** and **+dy**.
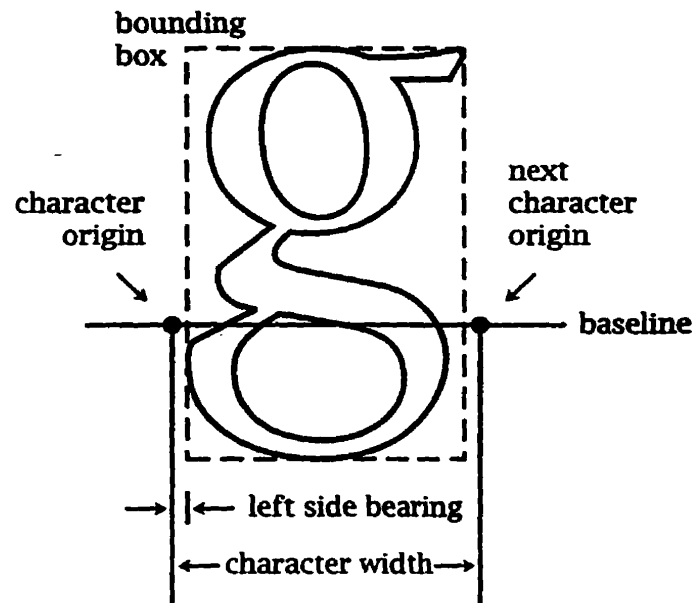
Figure 4.14. Character metrics.

AVATAR   AVATAR

(a)   (b)

Figure 4.15. Comparaison between *(a)* kerned text and *(b)* unkerned text, using the Computer Modern Roman typeface designed by Donald E. Knuth [31].

ff fi- fl ffi ffl   ff fi fl ffi ffl

(a)   (b)

Figure 4.16. Comparaison between *(a)* the ligatures and *(b)* simple characters in the Computer Modern Roman typeface designed by Donald E. Knuth [31].

The **FontInfo** dictionary provides font information to programs. The format documented in the *PostScript Language Reference Manual* [3] has been preserved and is described here for the sake of completeness.

Key	Type	Semantics
FamilyName	string	Specifies the font family to which the font belongs.
FullName	string	Unique name for an individual font.
Notice	string	Trademark or copyright notice.
Weight	string	Name for the weight, or *boldness*, attribute of the font.
version	string	Version number of the font program.
ItalicAngle	number	Angle in degrees counterclockwise from the vertical of the dominant vertical strokes of the font.
isFixedPitch	boolean	Indicates if the font is a fixed-pitch (monospaced) font.
UnderlinePosition	number	Recommended distance from the baseline for positioning underlining stroke.
UnderlineThickness	number	Recommended stroke width for underlining.

4.4 Parametrization

It is only through parametrization that one really get some control over random dynamic fonts. Some parameters influence every characters of a font, while some are specified for each characters. Furthermore, some parameters can greatly influence the metric of characters, in which case the **UseMetrics** parameters should be set to **false**.

Font Parameters

Font parameters are specified in two dictionaries: **FontParams** for global parameters, and **CharParams** for individual character parameters. The following table summarize the font and character parameters currently supported by the method.

Key	Type	Semantics
DynamicFont	boolean	Indicates if the font cache should be disabled, i.e., use **setcharwidth** rather than **setcachedevice**.
RandomFont	boolean	Indicates if random perturbations should be applied to letterforms.
UseSections	boolean	Indicates if sections should be used.
UseConstraints	boolean	Indicates if constraints dictionaries should be used.
UseMetrics	boolean	Indicates if provided metrics should be used or computed on-the-fly.
JumpyFont	boolean	Indicates if vertical translations should be applied to letterforms.
JumpyFactor	number	Specifies the degree of vertical haphazard variation.
JumpsRelative	boolean	Indicates if jumps should be relative.
LastJump	number	Contains value of last jump.
PaintType	integer	Painting method (0 = outline filled, 1 = path stroked, 2 = outline stroked, 3 = no painting).
StrokeWidth	number	Line width.
LineCap	integer	Shape of line ends for stroke (0 = butt, 1 = round, 2 = square).
LineJoin	integer	Shape of corners for stroke (0 = miter, 1 = round, 2 = bevel)

Key	Type	Semantics
MiterLimit	number	Miter length limit.
UsePen	boolean	Indicates if the calligraphic pen should be used.
ThickThinRatio	number	Thick thin ratio for the calligraphic pen.
StressAngle	number	Stress angle for the calligraphic pen.
-dx	number	Degree of horizontal haphazard variation (downward).
+dx	number	Degree of horizontal haphazard variation (upward).
-dy	number	Degree of vertical haphazard variation (towards the left).
+dy	number	Degree of vertical haphazard variation (towards the right).
Craziness	number	Degree of haphazard variation in all directions.
RestoreWidth	boolean	Indicates if the width of the original letterform should be restored.
RestoreHeight	boolean	Indicates if the height of the original letterform should be restored.
UsePTM	boolean	Indicates if the PTM transformation matrix should be applied to the polygon.
PTM	array	Transformation matrix for the spline characteristic polygon. Usually a procedure that returns an array defined in terms of the five following parameters.
XScale	number	Horizontal scaling factor in the PTM.
YScale	number	Vertical scaling factor in the PTM.
XSquashAngle	number	Horizontal squash angle factor in the PTM.
XShearAngle	number	Horizontal shear angle factor in the PTM.

Key	Type	Semantics
YShearAngle	number	Vertical shear angle factor in the PTM.
ShowBBox	boolean	Draws the letterform bounding box.
ShowPolygon	boolean	Draws the spline characteristic polygon.
ShowBezier	boolean	Draws the Bézier control points.
ShowBounds	boolean	Draws the sections' end points' constraints.
ShowSections	boolean	Draws the sections' end points in spline characteristic polygon.

Modifications to Font Parameters

Although the PostScript font machinery has no intented mechanism to modify or pass parameters to existing fonts, i.e., fonts registered in the font machinery, it is nevertheless possible to do so. The most common modification to existing fonts is installing a different encoding vector. As font dictionaries are made *read-only* once registered to the font machinery, it is not possible to modify this value directly. The usual way is to make a copy of the font dictionary, install the new encoding vector, and register the modified font under another name.

Although registered font dictionaries are made read-only, their sub-dictionaries are not. Thus, entries in sub-dictionaries, such as **FontParams** and **CharParams**, can be modified directly. To simplify the process of modifying these dictionaries two procedures are provided: **SetFontParams**, merges the entries of a dictionary supplied as operand with the ones in the **FontParams** dictionary of current font; and **SetCharParams** which merges the entries of a dictionary supplied as operand with the ones of a specified character in the **CharParams** dictionary of current font. Changes to the **FontParams** and **CharParams** dictionaries are cumulative, i.e., the procedures **SetCharParams** and **CharParams** simply add new entries to the

ones already present, or override their previous value. Source code for these new procedures is given in Appendix C.

5 Font Samples

Typeface design is an art. As the American type designer Frederic Goudy [21, p. 155] warned, "Letters should be designed by an artist and not by an engineer." This chapter presents, nevertheless, a typeface designed exclusively by a computer scientist, with the sole purpose of better illustrating the applicability of the method presented in Chapter 4. This presentation is limited to technical examples to show the capabilities of the method through parameters variations. Readers are referred to Appendix E for sample usage of MetamorFont.

5.1 MetamorFont

MetamorFont is a random dynamic font family implemented using the method presented in Chapter 4. Its name comes from the fusion of the word *metamorphose*, which means to change into a different form, and of the word *font*, for obvious reasons. It is also a wink at Donald E. Knuth font-design system METAFONT, as some *metaness* is also involved in the fonts.

5.2 Methodology

This typeface design was developed directly on-screen, painstakingly translated into digital form manually from hand drawn sketches. Despite its hand-drawn appearance, MetamorFont has been carefully worked to en-

sure easy reading and yet an informal appearance. Furthermore, some attention was paid to letterspacing as well as to kerning.

5.3 Character Sets

MetamorFont is available under two variations, Regular and BoldExtended, which contain all the necessary glyphs to fully support the following encodings: CorkEncoding, TEX Text encoding, TEXBase1Encoding, StandardEncoding, ISOLatin1Encoding, ISOLatin2Encoding, and WinLatin1Encoding.

	0x	1x	2x	3x	4x	5x	6x	7x	8x	9x	Ax	Bx	Cx	Dx	Ex	Fx
0	`	˝	␣	0	@	P	'	p	Ă	Ř	ă	ř	À	Đ	à	ð
1	´	˝	!	1	A	Q	a	q	Ą	Ś	ą	ś	Á	Ñ	á	ñ
2	^	.	"	2	B	R	b	r	Ć	Š	ć	š	Â	Ò	â	ò
3	~	«	#	3	C	S	c	s	Č	Ş	č	ş	Ã	Ó	ã	ó
4	¯	»	$	4	D	T	d	t	Ď	Ť	ď	ť	Ä	Ô	ä	ô
5	˙	–	%	5	E	U	e	u	Ě	Ţ	ě	ţ	Å	Õ	å	ō
6	˙	—	&	6	F	V	f	v	Ę	Ů	ę	ů	Æ	Ö	œ	ö
7	ˇ		'	7	G	W	g	w	Ğ	Û	ğ	ů̇	Ç	Œ	ç	œ
8	˘	˳	(	8	H	X	h	x	Ĺ	Ÿ	ĺ	ÿ	È	Ø	è	ø
9	¯	ı	)	9	I	Y	i	y	Ľ	Ź	ľ	ź	É	Ù	é	ù
A	˙	J	*	:	J	Z	j	z	Ŀ	Ž	ŀ	ž	Ê	Ú	ê	ú
B	¸	ff	+	;	K	[	k	{	Ń	Ż	ń	ż	Ë	Û	ë	û
C	˛	fi	,	<	L	\	l	|	Ň	Ĳ	ň	ĳ	Ì	Ü	ì	ū
D	˛	fl	-	=	M	]	m	}	Ŋ	İ	ŋ	ı	Í	Ý	í	ý
E	‹	ffi	.	>	N	^	n	~	Ő	đ	ő	¿	Î	Þ	î	þ
F	›	ffl	/	?	O	_	o	-	Ŕ	§	ŕ	£	Ï	SS	ï	ß

Table 5.1. MetamorFont-Regular following the Cork encoding vector.

	0x	1x	2x	3x	4x	5x	6x	7x	8x	9x	Ax	Bx	Cx	Dx	Ex	Fx
0		˘		0	@	P	`	p				°	À	Ð	à	ð
1	˙	ı	!	1	A	Q	a	q			¡	±	Á	Ñ	á	ñ
2	fi	ȷ	"	2	B	R	b	r	,		¢	²	Â	Ò	â	ò
3	fl	ff	#	3	C	S	c	s	ƒ	˝	£	³	Ã	Ó	ã	ó
4	/	ffi	$	4	D	T	d	t	„	”	¤	´	Ä	Ô	ä	ô
5	˝	ffl	%	5	E	U	e	u	_	•	¥	µ	Å	Õ	å	õ
6	ŀ		&	6	F	V	f	v	†	–	¦	¶	Æ	Ö	œ	ö
-7	ł		'	7	G	W	g	w	‡	—	§	·	Ç	×	ç	÷
8	¸		(	8	H	X	h	x	^	˜	¨	¸	È	Ø	è	ø
9	˙		)	9	I	Y	i	y	‰	™	©	¹	É	Ù	é	ù
A			*	:	J	Z	j	z	Š	š	ª	º	Ê	Ú	ê	ú
B	˘		+	;	K	[	k	{	‹	›	«	»	Ē	Û	ē	û
C	–		,	<	L	\	l	|	Œ	œ	¬	¼	Ì	Ü	ì	ū
D		¯	-	=	M	]	m	}				½	Í	Ý	í	ý
E	Ž	`	.	>	N	^	n	~			®	¾	Î	Þ	î	þ
F	ž	'	/	?	O	_	o			ḡ	¯	¿	Ï	ß	ï	ÿ

Table 5.2. MetamorFont-Regular following the TEXBase1 encoding vector.

	Ox	1x	2x	3x	4x	5x	6x	7x	8x	9x	Ax	Bx	Cx	Dx	Ex	Fx
0	`	´	␣	0	@	P	'	p	Ă	Ř	ă	ř	À	Đ	à	ð
1	´	"	!	1	A	Q	a	q	Ą	Ś	ą	ś	Á	Ñ	á	ñ
2	^	„	•	2	B	R	b	r	Ć	Š	ć	š	Â	Ò	â	ò
3	~	«	#	3	C	S	c	s	Č	Ş	č	ş	Ã	Ó	ã	ó
4	¯	»	$	4	D	T	d	t	Ď	Ť	ď	ť	Ä	Ô	ä	ô
5	˝	–	%	5	E	U	e	u	Ě	Ţ	ě	ţ	Å	Õ	å	õ
6	·	—	&	6	F	V	f	v	Ę	Ů	ę	ů	Æ	Ö	æ	ö
7	ˇ		'	7	G	W	g	w	Ğ	Ű	ğ	ű	Ç	Œ	ç	œ
8	˘	°	(	8	H	X	h	x	Ĺ	Ÿ	ĺ	ÿ	È	Ø	è	ø
9	¯	ı	)	9	I	Y	i	y	Ľ	Ź	ľ	ź	É	Ù	é	ù
A	˙	J	*	:	J	Z	j	z	Ł	Ž	ł	ž	Ê	Ú	ê	ú
B	¸	ff	+	;	K	[	k	{	Ń	Ż	ń	ż	Ë	Û	ë	û
C	˛	fi	,	<	L	\	l	|	Ň	IJ	ň	ÿ	Ì	Ü	ì	ü
D	˝	fl	-	=	M	]	m	}	Ŋ	İ	ŋ	i̇	Í	Ý	í	ý
E	‹	ffi	.	>	N	^	n	~	Ő	đ	ő	¿	Î	Þ	î	þ
F	›	ffl	/	?	O	_	o	-	Ŕ	§	ŕ	£	Ï	SS	ï	ß

Table 5.3. MetamorFont-BoldExtended following the Cork encoding vector.

	Ox	1x	2x	3x	4x	5x	6x	7x	8x	9x	Ax	Bx	Cx	Dx	Ex	Fx
0		˘		0	@	P	'	p				°	À	Ð	à	ð
1	˙	ι	!	1	A	Q	a	q			¡	±	Á	Ñ	á	ñ
2	ﬁ	J	¨	2	B	R	b	r	,		¢	²	Â	Ò	â	ò
3	ﬂ	ff	#	3	C	S	c	s	ƒ	˝	£	³	Ã	Ó	ã	ó
4	/	ﬃ	$	4	D	T	d	t	▪	˜	¤	´	Ä	Ô	ä	ô
5	˝	ﬄ	%	5	E	U	e	u	_	•	¥	µ	Å	Õ	å	õ
6	ļ		&	6	F	V	f	v	†	–	¦	¶	Æ	Ö	æ	ö
7	ŧ		'	7	G	W	g	w	‡	—	§	·	Ç	×	ç	÷
8	.		(	8	H	X	h	x	ˆ	˜	¨	.	È	Ø	è	ø
9	˙		)	9	I	Y	i	y	‰	™	©	¹	É	Ù	é	ù
A			*	:	J	Z	j	z	Š	š	ª	º	Ê	Ú	ê	ú
B	˘		+	;	K	[	k	{	‹	›	«	»	Ë	Û	ë	û
C	¯		,	<	L	\	l	|	Œ	œ	¬	¼	Ì	Ü	ì	ü
D			-	=	M	]	m	}			-	½	Í	Ý	í	ý
E	Ž	`	.	>	N	^	n	~			®	¾	Î	Þ	î	þ
F	ž	´	/	?	O	_	o			ÿ	¯	¿	Ï	ß	ï	ÿ

Table 5.4. MetamorFont-BoldExtended following the TEXBase1 encoding vector.

5.4 Technical Samples

The goal of this section is to show concretely the impact of parameters on the typeface appearance.

Fifi was stiffly sniffing the flowery fragrance of her fine coffee.
Fifi was stiffly sniffing the flowery fragrance of her fine coffee.
Fifi was stiffly sniffing the flowery fragrance of her fine coffee.
Fifi was stiffly sniffing the flowery fragrance of her fine coffee.

Figure 5.1. Variation on the **Craziness** parameter with **UseConstraints** sets to **true.**

Fifi was stiffy griffing the flowory fragrance of her fine coffee.
Fifi was stiffy sniffing the flowery fragrance of her fine coffee.
Fifi was stiffly sniffing the flowery fragrance of her fine coffee.
Fifi was stiffly sniffing the flowery fragrance of her fine coffee.

Figure 5.2. Variation on the **Craziness** parameter with **UseConstraints** sets to **false.**

Fifi was stiffly sniffing the flowery fragrance of her fine coffee.
Fifi was stiffly sniffing the flowery fragrance of her fine coffee.
Fifi was stiffly sniffing the flowery fragrance of her fine coffee.
Fifi was stiffly sniffing the flowery fragrance of her fine coffee.

Figure 5.3. Variation on the **Craziness** parameter with **RestoreHeight** and **RestoreWidth** set to **false.**

Fif အၢၥ ၁fိ้ုy ၾ๑ုၸ์n၌ lha ฿๑ၯ๑ry ไrxgนance ๑ၯ ืစr bne c๘ฟ๑ฝ.
Pี์฿ ฿ฺလၵ๑ ๑lฺ)ฝ๋y ၁hฺฺิ฿฿ืๅg lฝๆ ฿ๅ๑ฺvฺ๑r၌ ไr๑yฺๅฺๅ๑ฺ๑ ๑ไ ฿๑ำ lnฺ๑ ๑๑ไ๑๑๑.
Fิฺิ wฺๅๅ๑ ๑lฺิฺฝ์ๅy ๑ฺ๑ฺiไฺ๑nฺ๑ rฺๆฺ๑ ฿ฺๅำๅ๑ry ไr๑๑rฺๅ๑ฺ๑๑ ๑ไ hฺๅ๑r ฿n๑ ๑๑ไฺๅ๑฿.
Fิฺิ wฺๅๅ๑ ๑lฺิฺ฿฿์ำy ๑ฺ๑ิฺฺ่ำฺ่ำฺ๑ lhฺ๑ ไๅๅ๑wฺ๑ry ไๅฺๅๅ๑gr๑ๅ๑ฺ๑๑ ๑ไ hฺๅ๑r ฿ิฺ่๑ cๅฺๅฺๅ๑ffฺ๑ฺ๑.

Figure 5.4. Variation on the **Craziness** parameter with **UseSections** and **UseConstraints** set to **false**.

Fifi was stiffly sniffing the flowery fragrance of her fine coffee.
Fifi was stiffly sniffing the flowery fragrance of her fine coffee.
Fifi was stiffly sniffing the flowery fragrance of her fine coffee.
Fifi was stiffly sniffing the flowery fragrance of her fine coffee.

Figure 5.5. Variation on the **JumpyFactor** parameter with **JumpsRelative** sets to **true**.

Fifi was stiffly sniffing the flowery fragrance of her fine coffee.
Fifi was stiffly sniffing the flowery fragrance of her fine coffee.
Fifi was stiffly sniffing the flowery fragrance of her fine coffee.
Fifi was stiffly sniffing the flowery fragrance of her fine coffee.

Figure 5.6. Variation on the **JumpyFactor** parameter with **JumpsRelative** sets to **false**.

se se se se se se se

Figure 5.7. Variation on the **XShearAngle** parameter within the **PTM**.

se se se se se se se

Figure 5.8. Variation on the **YShearAngle** parameter within the **PTM**.

se se se se se se se

Figure 5.9. Variation on the **ThickThinRatio** parameter.
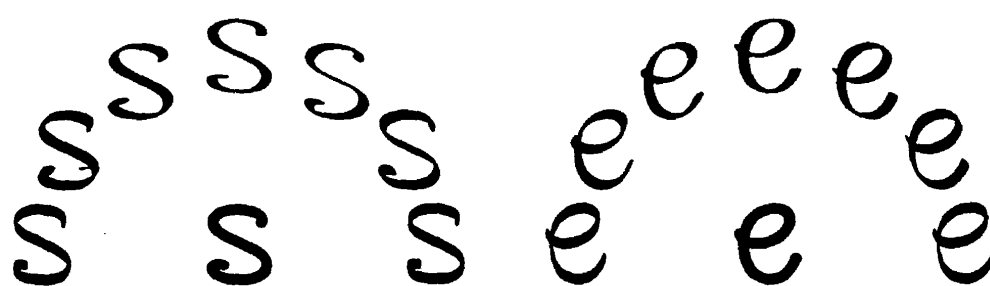
Figure 5.10. Variation on the calligraphic pen stress angle.

6 PostScript Fonts with TeX and LaTeX

This chapter provides an introduction to the origin and use of the TeX and LaTeX typesetting systems. The following sections deal with the set up and use of PostScript parametric fonts, as well as the interfacing with these systems. More details on the interaction between PostScript fonts, TeX and LaTeX can be found in [49, 20].

6.1 TeX and LaTeX

TeX is a powerful text-processing system for creating professional quality typeset text—and especially text containing mathematics. Developed in the late 1970s and early 1980s by Donald E. Knuth, TeX still ranks among the best typesetting systems, notably for its careful line and page breaking, skilfulness for setting mathematics, high-quality hyphenation, and multilingual capabilities—not to mention its portability across a wide range of computer platforms. The definitive guide to the use of TeX is The TeXbook [29].

LaTeX is document preparation system [35] developed by Leslie Lamport. Roughly speaking, LaTeX is a collection of TeX commands designed to simplify the typesetting of a document by allowing the user to concentrate on the content and structure of the document rather than on the exact appearance of the finished product. LaTeX2ε is the newly revised LaTeX standard, and is described in [20].

6.2   Font Metrics

As a typesetting system, TeX exhibits a primitive knowledge of fonts. TeX
regards fonts as ordered sets of rectangular boxes; it does not consider
the actual shapes of the characters. In order to set the characters into
their proper position, TeX only requires the font metrics information—the
widths, heights, and depths of characters—as well as extra information
such as ligatures, kerning pairs, and italic correction. This information
must be stored in external files called TeX font metric (.tfm) files.

Adobe Systems adopted a similar solution to specify the font metrics
information of their PostScript font programs. To each PostScript font
corresponds an Adobe Font Metric (.afm) file [4] that describes both global
metrics for the font and the metrics for each character. The .afm files are
practically equivalent to TeX's .tfm.

To use PostScript fonts with TeX, .tfm files containing the same infor-
mation as the .afm must be provided. Fortunately, many programs can
convert .afm into .tfm files. More details on such programs are given in
section 6.5.

6.3   Font Encodings

The encoding vector describes the order and position of characters within
a font. It can be viewed as a one-dimensional array indexed by character
code, usually an integer in the range 0 to 255, to which corresponds a
character glyph.

Operating systems and software applications make use of several differ-
ent encoding vectors. TeX Text, Cork, and Adobe Standard are all examples
of encoding vectors. In practice, the encoding vector is usually chosen
according to the hardware or software used.

Unless specified, T_EX assumes that fonts use the T_EX Text (OT1) encoding, a primitive 7-bit encoding, thus with space for only 128 characters. In 1990, the T_EX User's Group adopted a new encoding scheme, the Cork Encoding (T1), that contains all the characters required by more than 20 languages using the Latin alphabet. Among other things, this new encoding enables the use of true accented characters instead of relying on T_EX's accenting mechanism. Font encodings are discussed at length in [8]

6.4 Virtual Fonts

The virtual font mechanism provides a general interface to change the encoding vector of a font. Virtual fonts are defined in terms of characters from one or more fonts, and possibly from other virtual fonts, including themselves. They can map a character to another character in a different font (composite font), a different character in the same font, a character to multiple characters (composite character), or even an arbitrary sequence of DVI commands.

Although virtual fonts can serve many purposes, they are commonly used to interface PostScript fonts to T_EX. They can change the encoding vector, gather glyphs missing in the base font from so-called expert sets, as well as construct composite characters. A discussion on the virtual font mechanism can be found in [33].

6.5 Adobe File Metrics to T_EX File Metrics

While some .afm to .tfm conversion programs only perform a crude translation of the original .afm file, others can specify different encodings, or carry out special manipulations. The afm2tfm program, distributed with the dvips driver, provides re-encoding facilities as well as special effects

to construct synthetic fonts—faked small caps, obliqued, expanded, and condensed variants—through the use of virtual fonts.

Alan Jeffrey's fontinst package [24]—a font installation software for TeX—provides a better support for LaTeX2ε users. While providing all the facilities supported by afm2tfm, it also allows the generation of the files required by the New Font Selection Scheme (NFSS), extension of LaTeX2ε, such as the font definitions (.fd) files.

6.6 Device Independent Drivers

As output, TeX produces a DeVice Independent (.dvi) file that is not directly printable. The .dvi file describes the typeset document in a simple stack language that can be rendered on any device. The task of translating the .dvi file into printable or viewable form is left to a DVI driver. A DVI driver is a special program that translates the DVI commands into a form suitable for a particular device, and it also provides all the required fonts.

Thus, TeX can be used for almost any kind of output device, if an appropriate DVI driver is available. Moreover, DVI drivers free TeX of any dependency on printing technologies. Only DVI drivers will require updates as the technology evolves.

6.7 Interfacing PostScript Parametric Fonts

Tomas Rokicki's DVI driver for PostScript, dvips [42], allows for the inclusion of native PostScript code within a TeX document via the \special primitive. TeX's \special command is provided to transmit special instructions directly to the DVI driver, usually to take advantage of special features offered by a particular output device.

In the context of parametric fonts, this feature can be put to profit as

the user can select fonts and also set parameters of selected fonts. As there are currently no standards for parametric fonts, each parametric font must provide an interface to set the parameters. An interface written for the MetamorFont family is provided in the package metamorfont.sty, shown in Appendix D.

7 Conclusion

This thesis presents a general method and structure for storing, representing and reproducing random dynamic fonts. The generated letterform shapes are guaranteed to be C^2 continuous and to preserve a uniform thickness.

The intent of this thesis is to create beautiful, functional and inconspicuous random dynamic fonts. In order to illustrate the versatility of the method proposed, a new typeface, MetamorFont, is designed and implemented. The result, more than a simple typeface, is nothing less than a software workbench dedicated to the development of random dynamic font programs.

Indeed, new random dynamic fonts could easily be designed with the tools developed for the creation of MetamorFont. Although, on the practical side, a graphical user interface would be more than welcome to automate the input process.

The current implementation offers many parameters to achieve a wide range of special effects. Furthermore, the structure proposed is developed in an extensible way so that new parameters and random transformations may easily be added in the future.

There is still room for improvements. As an alternative to outlines, a calligraphic pen effect is proposed. Currently, the implementation only supports constants stroke width, stress angle and thick-thin ratio within each glyph. Variation of these parameters within a glyph, as can be done in METAFONT, would certainly allow the generation of more interesting

shapes. The use of variable width splines [28] might also be considered.

Connected random dynamic fonts, i.e., fonts in which all characters are smoothly linked might be a good direction to pursue research. Although context handling could be done within fonts, it is clear that this is more the responsibility of the typesetting system.

Random dynamic fonts derived from true handwritten characters pose a serious challenge. A careful attempt in this direction is given in Devroye and McDougall [18]. It would be interesting to look at the mathematical models of human handwriting used in the field of optical character recognition [46, 39].

While random dynamic fonts allow to come closer to simulating true handwriting, it is clear that the fonts alone aren't sufficient to simulate true handwriting, randomness should also be involved in the typesetting process as the position of the characters on the page also play an important role in the final result.

From the creative point of view, random dynamic fonts present a new challenge to artists and typographers. Indeed, the design of random dynamic fonts is much more difficult than that of static fonts. One not only has to think in terms of letterform shapes but rather in terms of sets of letterform shapes that depend on random parameters.

A · PostScript Type 3 Font

This appendix exhibits the structure of a PostScript Type 3 font program, MetamorFont, designed with the method proposed. Many dictionaries and procedures have been left out to save space.

```
%!PS-Adobe-3.0 Resource-Font
%%Title: ($ RCSfile: MetamorFont-Regular.ps,v $)
%%Creator: ($ Author: bernard $)
%%CreationDate: ($ Date: 1996/10/22 15:02:40 $)
%%Version: ($ Revision: 1.1 $)
%%Copyright: (c) 1995, 1996 by Bernard Desruisseaux.  All rights reserved.
%%For: Bernard Desruisseaux (bernard@cs.mcgill.ca)
%%EndComments
%%BeginProlog                        _
%%BeginResource: font MetamorFont-Regular


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Type 3 Font        :   MetamorFont-Regular
%-----------------------------------------------------------------------
% Description        :   MetamorFont is a random dynamic font.
%-----------------------------------------------------------------------
% Copyright          :   (c) 1995, 1996 by Bernard Desruisseaux.
%                        All rights reserved.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
100 dict begin

/FontName /MetamorFont-Regular def
/FontType 3 def                                              _
/FontMatrix [ 0.001 0 0 0.001 0 0 ] readonly def
/FontBBox [ 0 0 0 0 ] readonly def % (atend)
/Encoding StandardEncoding def
/UniqueID 1314159 def
/XUID [ 1000000 1314159 ] readonly def
/LanguageLevel 2 def


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Dictionary    :   FontInfo
%-----------------------------------------------------------------------
% Description   :   Font Information conforming to Type 1 format.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
/FontInfo 9 dict dup begin
    /version (001.000) readonly def
    /Notice (Copyright (c) 1995, 1996 by Bernard Desruisseaux) readonly def
    /FamilyName (MetamorFont) readonly def
    /FullName (MetamorFont Regular) readonly def
```

```
/Weight (Regular) readonly def
/ItalicAngle 0.0 def
/isFixedPitch false def
/UnderlinePosition -150 def
/UnderlineThickness 50 def % [Bold: 80]
end readonly def


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Dictionary    :   FontParams
%-----------------------------------------------------------------
% Description    :   Global font parameters.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
/FontParams 100 dict dup begin
    %--[ Generals ]--------------------------------------------
    /DynamicFont true def              % setcharwidth or setcachedevice
    /RandomFont true def               % Use DistortCharPolygon?
    /UseConstraints true def           % Use constraints dictionary?
    /UseSections true def              % Use sections?
    /UseMetrics dup where              % Use FontMetrics dictionary?
        { pop UseMetrics }             %
        { true } ifelse def            %
    /NullGlyph false def               % Most glyphs are non-null


    %--[ JumpyFont ]-------------------------------------------
    /JumpyFont false def               % Characters go up and down
    /JumpyFactor 50 def                %   by +- JumpyFactor
    /JumpsRelative true def            % Relative jumps?
    /LastJump 0 def                    % Last jump value


    %--[ Painting Parameters ]---------------------------------
    /PaintType 1 def                   % 0:fill,1:stroke,2:stroke,3:null
    /StrokeWidth 50 def                % Stroke width [Bold: 80]
    /LineJoin 1 def                    % Line join
    /LineCap 1 def                     % Line cap
    /MiterLimit 2.5 def                % Miter limit


    %--[ Pen Parameters ]--------------------------------------
    /UsePen true def                   % Use calligraphic pen effect?
    /ThickThinRatio 5 def              % Pen Ratio of thick to thin
    /StressAngle 120 def               % Pen Stress Angle


    %--[ Randomization Parameters ]----------------------------
    /Craziness 0 def                   % Haphazard variation
    { /-dx /-dy /+dx /+dy }            % Independant haphazard var.
        { Craziness def } forall       %
    /RestoreWidth true def             % Restore Width after trans.
    /RestoreHeight true def            % Restore Height after trans.


    %--[ Polygon Transformation ]------------------------------
    /UsePTM false def                  % Use the PTM ? [Bold: true]
    /XShearAngle 0 def                 % X Shear (xtilt) the polygon
    /YShearAngle 0 def                 % Y Shear (ytilt) the polygon
    /XSquashAngle 0 def                % Squash the polygon
    /XScale 1 def                      % [Bold: 1.05]
    /YScale 1 def                      % [Bold: 0.95]
    /PTM { [                           % Polygon Transformation Matrix
        XSquashAngle cos XScale mul
        YShearAngle sin
        XShearAngle sin YScale mul
        YScale 0 0
```

```
     ] } def

     %--[ Debugging Parameters ]------------------------------------------
     /ShowBBox false def              % Show Character Bounding Box?
     /ShowBounds false def            % Show Coordinate Bounds?
     /ShowSections false def          % Show Sections?
     /ShowPolygon false def           % Show Character Polygon?
     /ShowBezier false def            % Show Bezier Polygons?
end def

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Dictionary    :    CharParams
%------------------------------------------------------------------------
% Description   :    Character parameters.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
/CharParams 400 dict dup begin
     /.notdef         << /NullGlyph true >> def
     /acute           << /StressAngle 60 /RestoreWidth false >> def
     /asciicircum     << /StressAngle 90 /RestoreWidth false >> def
     /at              << /StressAngle 90 >> def
     /backslash       << /StressAngle 90 >> def
     /bar             << /StressAngle 60 /RestoreWidth false >> def
     /breve           << /StressAngle 90 >> def
     /brokenbar       << /StressAngle 60 /RestoreWidth false >> def
     /bullet          << /StressAngle 90 /StrokeWidth 150
                         /ThickThinRatio 1 >> def
     /caron           << /StressAngle 90 >> def
     /cedilla         << /StressAngle 110 >> def
     /circumflex      << /StressAngle 90 /RestoreWidth false >> def
     /currency        << /StressAngle 90 >> def
     /cwm             << /NullGlyph true >> def
     /compwordmark    cwm def
     /compworkmark    cwm def
     /divide          << /StressAngle 60 >> def
     /emdash          << /StressAngle 0 /RestoreHeight false >> def
     /endash          << /StressAngle 0 /RestoreHeight false >> def
     /equal           << /StressAngle 60 >> def
     /exclam          << /RestoreWidth false >> def
     /exclamdown      << /RestoreWidth false >> def
     /grave           << /RestoreWidth false >> def
     /greater         << /StressAngle 90 /RestoreWidth false >> def
     /guillemotleft   << /StressAngle 90 >> def
     /guillemotright  << /StressAngle 90 >> def
     /guilsinglleft   << /StressAngle 90 >> def
     /guilsinglright  << /StressAngle 90 >> def
     /hyphen          << /StressAngle 60 /RestoreHeight false >> def
     /hungarumlaut    << /StressAngle 80 >> def
     /less            << /StressAngle 90 /RestoreWidth false >> def
     /logicalnot      << /StressAngle 135 >> def
     /macron          << /StressAngle 60 /RestoreWidth false >> def
     /minus           << /StressAngle 60 /RestoreHeight false >> def
     /multiply        << /StressAngle 90 >> def
     /numbersign      << /StressAngle 135 >> def
     /ogonek          << /RestoreWidth false /RestoreHeight false >> def
     /parenleft       << /StressAngle 90 >> def
     /parenright      << /StressAngle 90 >> def
     /percent         << /StressAngle 90 >> def
     /perthousand     << /StressAngle 90 >> def
     /plus            << /StressAngle 60 >> def
     /plusminus       << /StressAngle 60 >> def
```

```
/quotedbl          << /StressAngle 110 >> def
/quotesingle       << /StressAngle 110 >> def
/ring              << /RestoreWidth false /RestoreHeight false >> def
/slash             << /StressAngle 90 /RestoreWidth false >> def
/space             << /NullGlyph true >> def
/three             << /StressAngle 140 >> def
/tilde             << /RestoreWidth false >> def
/underscore        << /StressAngle 60 /RestoreHeight false >> def
/yen               << /StressAngle 90 >> def
/z                 << /StressAngle 110 >> def
/Z                 << /StressAngle 110 >> def
end def


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% FontMetrics dict -- Horizontal Left Sidebearing and Width [ LSBx Wx ]
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
/FontMetrics 400 dict def


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% BBox dict -- Bounding boxes [ llx lly urx ury ]
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
/BBox 400 dict def


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% RealBBox dict -- Defined for computation if we don't UseMetrics.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
FontParams /UseMetrics get not { /RealBBox 400 dict def } if


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Procedure       :   BuildGlyph
%-------------------------------------------------------------------------
% Description      :   Interfaces the font machinery to the font's
%                      PostScript procedures for drawing the shapes.
%                      Called by the PostScript interpreter.
%-------------------------------------------------------------------------
% Caveats          :   This procedure is enclosed between a save/restore
%                      pair since memory is allocated during the
%                      construction of glyphs.
%-------------------------------------------------------------------------
% Operand Stack  :   font name => _
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
/BuildGlyph {                                              % font name
    save 3 1 roll exch                                     % name font
    begin FontParams begin                                 % name
    dup CharProcs exch known not{ pop /.notdef } if % name
    dup GetMetrics 0                                       % name LSBx Wx 0
    DynamicFont { setcharwidth }{ 3 index GetBBox setcachedevice } ifelse
    ShowBBox {
        1 index                                            % name LSBx name
        DynamicFont { currentgray 0 setgray exch } if
        GetBBox DrawBBox
        DynamicFont { setgray } if
    } if                                                   % name LSBx
    JumpyFont {
        JumpyFactor dup neg exch Uni
        JumpsRelative { LastJump add dup /LastJump exch def } if
    } { 0 } ifelse translate                               % name
    mark exch dup CharProcs begin load exec end   % mark name [...]
    {
        counttomark 0 eq { exit } if
```

```
                    exch CharParams exch 2 copy known
                    dup 5 1 roll { get begin } { pop pop } ifelse
                    dup xcheck { CharProcs begin exec end } if
                    ShowBounds { dup { DrawBounds } false MarkCharPolygonSection } if
                    RandomFont {
                        RestoreWidth RestoreHeight or
                            { dup GetCharPolygonBBox 5 -1 roll } if
                        UseSections { DistortCharPolygon }
                            { CharPolygon2Polygon DistortPolygon } ifelse
                        RestoreWidth RestoreHeight or {
                            dup GetPolygonBBox 5 -1 roll 9 1 roll
                            RestoreWidth RestoreHeight GetRestoreMatrix
                            TransformPolygon
                        } if
                    } { CharPolygon2Polygon } ifelse
                    UsePTM { PTM TransformPolygon } if
                    ShowPolygon { dup } if
                    PaintType 3 ne ShowBezier or {
                        BuildBohmConstruction DrawBohmConstruction
                        ShowBezier { gsave } if
                        //PaintProcs PaintType get exec
                        ShowBezier { grestore false LabelPath } if
                    } { pop } ifelse
                    ShowPolygon {
                        DynamicFont { 1 setgray } if
                        { DrawSquare } true MarkPolygon
                    } if
                    { end } if
                } loop pop
            end end
            restore
    } bind def


    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Procedure      :   BuildChar
    %---------------------------------------------------------------------------
    % Description    :   BuildGlyph ancestor.  Required for PS Level I.
    %                    Called by the PostScript interpreter.
    %---------------------------------------------------------------------------
    % Operand Stack :  ·dict code => _
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    /BuildChar {
        1 index /Encoding get exch get
        1 index /BuildGlyph get exec
    } bind def

    currentdict
    end
    dup /FontName get exch definefont pop
    %%EndResource
    %%EndProlog
```

B   Adobe Font Metrics

This is a generalized example for a typical Adobe Font Metrics (AFM) file. It is derived from the AFM file generated for MetamorFont-Regular. Because many parts of an AFM are repetitive, much of the repetition in the following example has been omitted. The omitted portions are documented with comments. The technical specification of the AFM format is covered in [4].

```
StartFontMetrics 4.1
Comment This file was automatically generated by:
Comment $ Id: t3gen.ps,v 1.4 1996/10/16 11:27:38 bernard Exp $
Comment Bernard Desruisseaux (bernard@cs.mcgill.ca)
FontName MetamorFont-Regular               -
Weight Regular
Notice Copyright (c) 1995, 1996 by Bernard Desruisseaux
UnderlinePosition -150
Version 001.000
FamilyName MetamorFont
ItalicAngle 0.0
UnderlineThickness 50
IsFixedPitch false
FullName MetamorFont Regular
FontBBox -167 -298 970 1138
EncodingScheme AdobeStandardEncoding
CapHeight 739
XHeight 514
Ascender 840
Descender -261
StartCharMetrics 314
C 97 ; WX 548 ; N a ; B 40 -14 528 508 ;
C 102 ; WX 331 ; N f ; B 30 -14 341 846 ; L l fl ; L i fi ; L f ff ;
Comment *** many character metrics omitted ***
C -1 ; WX 535 ; N ff ; B 30 -14 545 846 ; L l ffl ; L i ffi ;
C 171 ; WX 423 ; N guillemotleft ; B 20 203 393 463 ;
EndCharMetrics
StartKernData
StartKernPairs 72
KPX T o -150
Comment *** many kerning pairs omitted ***
KPX r a -60
EndKernPairs
EndKernData
EndFontMetrics
```

C Modifications to Parameters

PostScript source code to two procedures used to modify the **FontParams**
and **CharParams** dictionaries of random dynamic fonts designed with the
proposed method.

```
%XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
% Procedure       :   SetFontParams
%---------------------------------------------------------------------
% Description     :   Merge the entries of the operand dictionary in the
%                     FontParams dictionary of the current font.
%---------------------------------------------------------------------
% Example         :   << /Craziness 100 >> SetFontParams
%---------------------------------------------------------------------
% Operand Stack :   paramsdict => _
%XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
/SetFontParams {
    currentfont begin FontParams begin
        {
            1 index /Craziness eq {
                def { /-dx /-dy /+dx /+dy } { Craziness def } forall
            } { def } ifelse
        } forall
    end end
} bind def


%XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
% Procedure       :   SetCharParams
%---------------------------------------------------------------------
% Description     :   Merge the entries of the operand dictionary in the
%                     'name' dictionary in the CharParams dictionary of
%                     the current font.
%---------------------------------------------------------------------
% Example         :   /guillemotright << /StressAngle 90 >> SetCharParams
%---------------------------------------------------------------------
% Operand Stack :   name paramsdict => _
%XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
/SetCharParams {
    currentfont begin CharParams dup begin
        2 index known {
            exch load begin
                { def } forall
            end
        } { def } ifelse
    end end
} bind def
```

D MetamorFont with LaTeX 2ε

The LaTeX 2ε MetamorFont package, `metamorfont.sty`, provides a simple front-end to the **SetFontParams** operator listed in Appendix C.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% $ Id: metamorfont.sty,v 1.1 1996/10/22 15:02:48 bernard Exp $
%-------------------------------------------------------------------------
% Description   :   Package to use the new family MetamorFont.
%                   Option 'rmdefault' sets MetamorFont as the
%                   default roman family.
%-------------------------------------------------------------------------
% Note          :   This package make use of the driver definition files
%                   part of the Standard LaTeX 'Graphics Bundle.'
%-------------------------------------------------------------------------
% Caveat        :   Parameter changes are only valid for the current
%                   page, as pages are embedded in a save/restore pair.
%-------------------------------------------------------------------------
% Author        :   Bernard Desruisseaux (bernard@cs.mcgill.ca)
%-------------------------------------------------------------------------
% Copyright     :   (c) 1996 by Bernard Desruisseaux.
%                   All rights reserved.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{metamorfont}[1996/10/01 v1.0 LaTeX2e MetamorFont package]
\providecommand\Gin@driver{}
\DeclareOption{rmdefault}{\renewcommand{\rmdefault}{fmf}}
\DeclareOption{dvips}{\def\Gin@driver{dvips.def}}
\DeclareOption{xdvi}{\ExecuteOptions{dvips}}
\DeclareOption{dvipsone}{\def\Gin@driver{dvipsone.def}}
\DeclareOption{dviwindo}{\ExecuteOptions{dvipsone}}
\DeclareOption{emtex}{\def\Gin@driver{emtex.def}}
\DeclareOption{dviwin}{\def\Gin@driver{dviwin.def}}
\DeclareOption{oztex}{\def\Gin@driver{oztex.def}}
\DeclareOption{textures}{\def\Gin@driver{textures.def}}
\DeclareOption{pctexps}{\def\Gin@driver{pctexps.def}}
\DeclareOption{pctexwin}{\def\Gin@driver{pctexwin.def}}
\DeclareOption{pctexhp}{\def\Gin@driver{pctexhp.def}}
\DeclareOption{dvi2ps}{\def\Gin@driver{dvi2ps.def}}
\DeclareOption{dvialw}{\def\Gin@driver{dvialw.def}}
\DeclareOption{dvilaser}{\def\Gin@driver{dvilaser.def}}
\DeclareOption{dvitops}{\def\Gin@driver{dvitops.def}}
\DeclareOption{psprint}{\def\Gin@driver{psprint.def}}
\DeclareOption{pubps}{\def\Gin@driver{pubps.def}}
\DeclareOption{ln}{\def\Gin@driver{ln.def}}
\InputIfFileExists{metamorfont.cfg}{}{}
\ProcessOptions
```

```
\if!\Gin@driver!
    \PackageError{metamorfont}
    {No driver specified}
    {You should make a default driver option in a file \MessageBreak
     metamorfont.cfg\MessageBreak
     eg: \protect\ExecuteOptions{dvips}%
    }
\else
    \PackageInfo{metamorfont}{Driver file: \Gin@driver}
    \@ifundefined{ver@\Gin@driver}{\input{\Gin@driver}}{}
\fi
\DeclareRobustCommand{\fmffamily}{\fontfamily{fmf}\selectfont}
\DeclareTextFontCommand{\textfmf}{\fmffamily}
\newcommand*{\fmfr}{\usefont{\encodingdefault}{fmf}{m}{n}}
\newcommand*{\fmfx}{\usefont{\encodingdefault}{fmf}{bx}{n}}
\newcommand*{\fmfrp}{\usefont{U}{fmf}{m}{n}}
\newcommand*{\fmfrparams}[1]{\Gin@PS@raw{/MetamorFont-Regular
                            findfont setfont <<#1>> SetFontParams}}
\newcommand*{\fmfxparams}[1]{\Gin@PS@raw{/MetamorFont-BoldExtended
                            findfont setfont <<#1>> SetFontParams}}
\newcommand*{\fmfrpparams}[1]{\Gin@PS@raw{/MetamorFont-Ornaments
                            findfont setfont <<#1>> SetFontParams}}
\endinput
%%
%% End of file 'metamorfont.sty'.
```

The MetamorFont LaTeX 2ε font definition .fd file, T1fmf.fd, for the Cork (T1) encoding. More information on font definition .fd file can be found in [49, 20].

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% $Id: back.tex,v 1.15 1996/10/24 12:25:52 bernard Exp $
%-----------------------------------------------------------------------
% Description : MetamorFont font definitions for Cork (T1) encoding.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
\ProvidesFile{T1fmf.fd}[1996/10/15 MetamorFont font definitions.]
\DeclareFontFamily{T1}{fmf}{}
\DeclareFontShape{T1}{fmf}{m}{n}{<-> fmfr8t}{}
\DeclareFontShape{T1}{fmf}{bx}{n}{<-> fmfx8t}{}
\DeclareFontShape{T1}{fmf}{l}{n}{<->ssub * fmf/m/n}{}
\DeclareFontShape{T1}{fmf}{db}{n}{<->ssub * fmf/bx/n}{}
\DeclareFontShape{T1}{fmf}{b}{n}{<->ssub * fmf/db/n}{}
\endinput
%%
%% End of file 'T1fmf.fd'.
```

E MetamorFont Gallery

The following pages present texts and arrangements specially selected to show off the MetamorFont family.
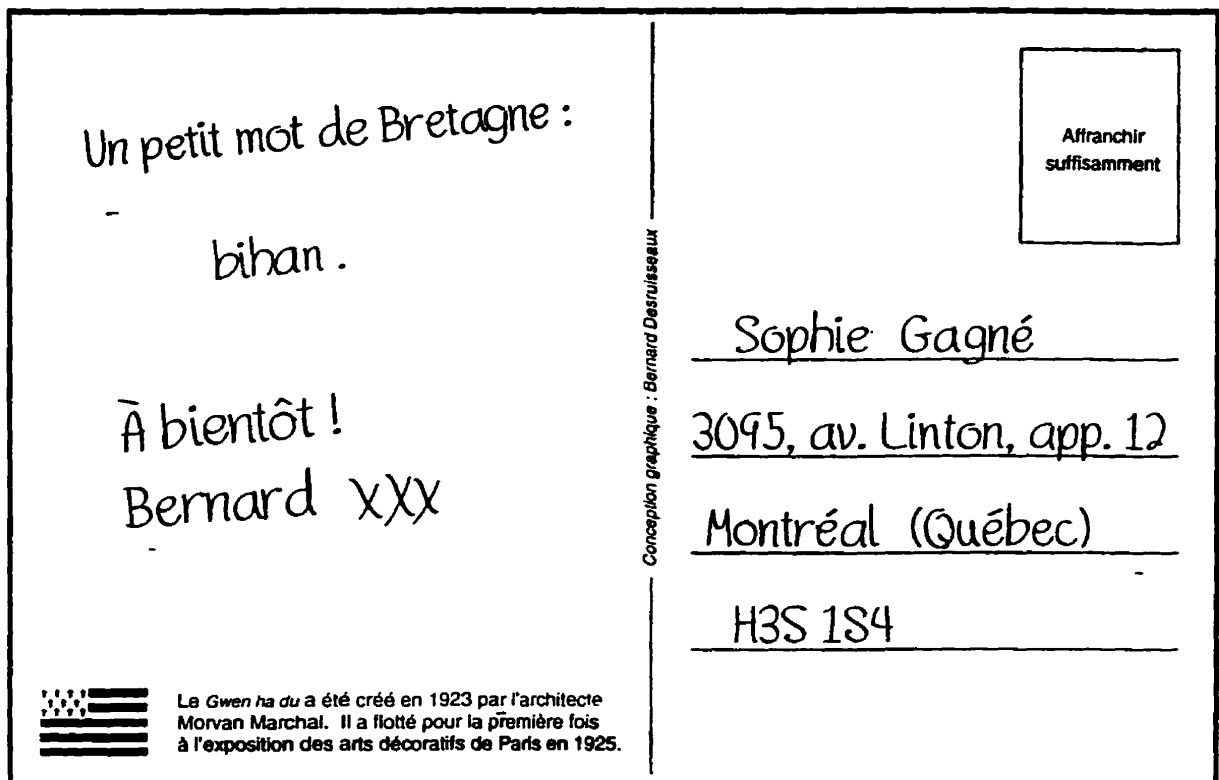

Un petit mot de Bretagne :

bihan .

À bientôt !
Bernard XXX

Conception graphique : Bernard Desruisseaux

Affranchir
suffisamment

Sophie Gagné

3095, av. Linton, app. 12

Montréal (Québec)

H3S 1S4

Le *Gwen ha du* a été créé en 1923 par l'architecte
Morvan Marchal. Il a flotté pour la première fois
à l'exposition des arts décoratifs de Paris en 1925.

Figure E.1. MetamorFont on a personal postcard.

MetamorFont

by Bernard Desruisseaux

Inspiring Typeface
Personal Look
Dynamic Feel
Eye Opening
Beyond Cool
Simply Lovely

aA bB cC dD eE fF gG hH iI jJ kK lL mM
nN oO pP qQ rR sS tT uU vV wW xX yY zZ

Figure E.2. MetamorFont specimen page.

MetamorFont

by Bernard Desruisseaux

ABCDEFGHIJ
KLMNOPQ
RSTUVWXYZ
ÃÆŒÇØŊ&§¶
abcdefghijklmn
opqrstuvwxyz
áâàãäåăąœœ
fiffflﬃﬄþß£ƒ¥$
čçďtðőøñŋż
(.,:;?)~@ ∗ „ " » © «
1234567890

Figure E.3. MetamorFont specimen page.

Señor Presid...
Alberto Fujim...
Palacio de Gobierno
Plaza de Armas
Lima 1 Pérou

Monsieur le Président,

J'ai... par Amnistie internationale que Myriam Guadalupe Galvez Vargas, accusée de « terrorisme », a été condamnée, en 1994, à vingt ans de réclusion.

Selon Amnistie internationale, les charges pesant contre cette étudiante de trente ans, mère de deux enfants, reposent sur des déclarations peu fiables faites par la police, selon laquelle Myriam Guadalupe Galvez Vargas aurait entretenu des liens avec le Sentier lumineux.

Aucune preuve de culpabilité n'ayant pu être établie contre madame Galvez Vargas, détenue actuellement à la prison de haute-sécurité de Chorillos, à Lima, je vous prie d'intervenir rapidement en sa faveur et de la faire libérer immédiatement et inconditionnellement.

Persuadé que vous saurez entendre mon appel, je vous prie de croire, Monsieur le Président, à l'expression de ma haute considération.

Bernard Desruisseaux

c.c. Son Excellence Hernan Couturier Mariategui, ambassadeur

Figure E.4. MetamorFont for personal letter.

Basilic	Bâtons de cannelle	Poudre de curcuma	Graines de moutarde
Feuilles de laurier	Cerfeuil	Muscade moulue	Safran
Estragon	Ciboulette	Marjolaine	Sauge
Poivre de cayenne	Clous de girofle	Romarin	Cardamome moulue
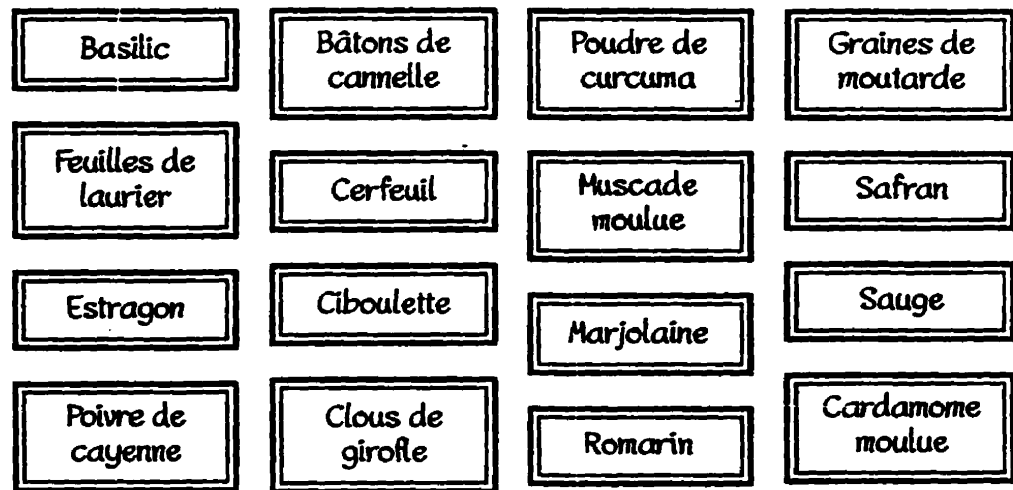
Figure E.5. MetamorFont for spice and herb bottle labels.

Coffee makes a bad man cheerful; a languourous man, active; a cold man warm; a warm man, glowing; a debilitated man, strong. It intoxicates, without inviting the police; it excites a flow of spirit, and awakens mental powers thought to be dead ... When coffee is bad, it is the wickedest thing in town; when good, the most glorious. When it has lost its aromatic flavor and appeals no more the eye, smell, or taste, it is fierie; but when left in a sick room, with the lid off, it fills the room with a fragrance only jacqueminots can rival. The very smell of coffee in a sick room terrorizes death.

—John Ernest McCann, in *Over the Black Coffee* (1902)

Figure E.6. MetamorFont used to typeset a quote.

Gâteau du diable

Préparation : 30 minutes
Cuisson : 25 minutes
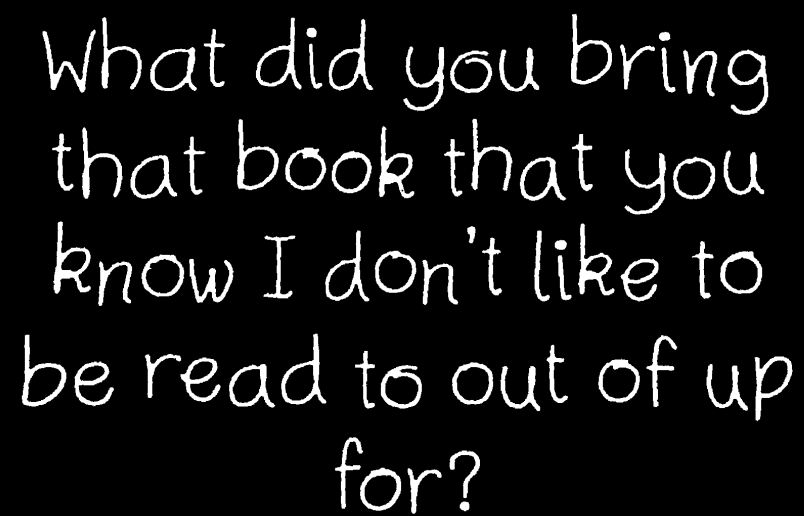
50 mL	Beurre
250 mL	Sucre
2	Œufs battus
300 mL	Farine tout usage
5 mL	Poudre à pâte
75 mL	Lait caillé
125 mL	Café noir bouillant
100 mL	Chocolat mi-sucré fondu
5 mL	Bicarbonate de soude
5 mL	Essence de vanille

1 – Battre le beurre en crème avec le sucre.
2 – Ajouter un à un les œufs sans cesser de battre
le mélange.
3 – Tamiser la farine et la poudre à pâte. Incorporer
au mélange en alternant avec le lait caillé.
Terminer avec la farine.
4 – Verser le café bouillant sur le chocolat fondu, et y
ajouter le bicarbonate de soude. Laisser refroidir
quelque peu, puis ajouter au premier mélange.
5 – Parfumer avec l'essence de vanille.
6 – Verser dans un moule de 25 × 15 × 5 cm.
7 – Faire cuire au four à 190°C, durant 25 minutes.

Figure E.7. Recipe taken from *Cuisine du Québec*, Institut de tourisme et
d'hôtellerie du Québec. Les Éditions TransMo, 1985, p. 54.

Figure E.8. Overprint effect used with MetamorFont. A general guiding principle of Design (and of Life) from graphic designer Robin Williams.

What did you bring
that book that you
know I don't like to
be read to out of up
for?

Figure E.9. Sentence allegedly produced by a young child ... probably af-
ter too much PostScript programming! The child's father had
brought a book upstairs, to read the child a bedtime story.
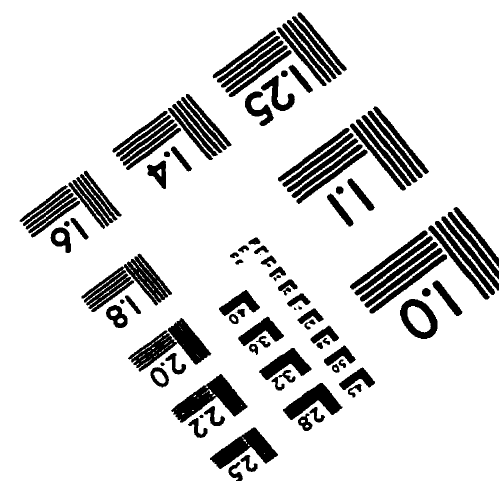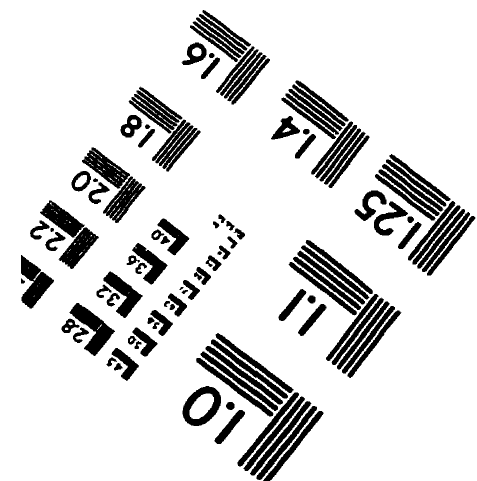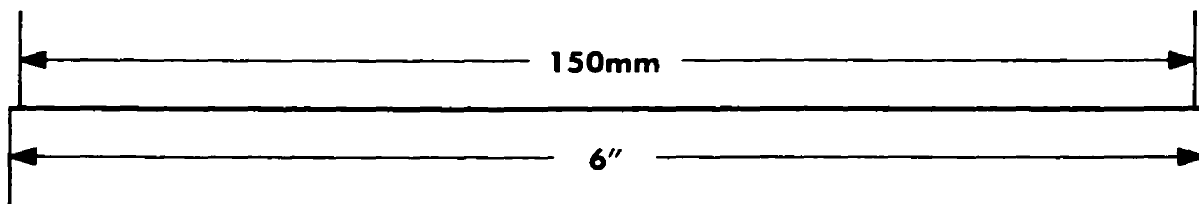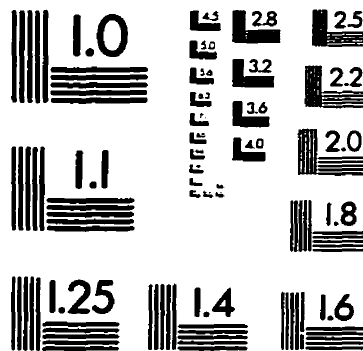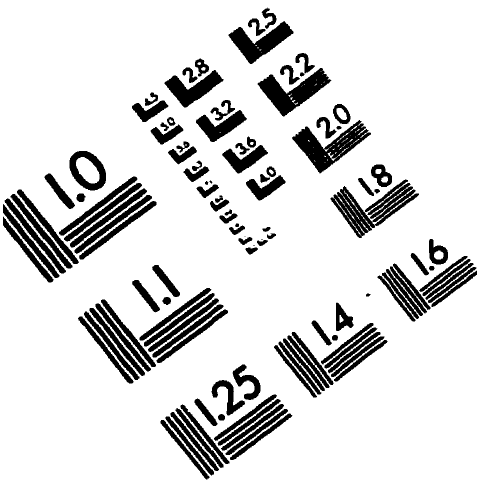However, it was a book the child did not like.

Bibliography

[1] Adobe System Incorporated. *PostScript Language Tutorial and Cookbook.* Addison–Wesley, Reading, Mass., 1985.

[2] Adobe System Incorporated. *Adobe Type 1 Font Format, version 1.1.* Addison–Wesley, Reading, Mass., 1990.

[3] Adobe System Incorporated. *PostScript Language Reference Manual.* Addison–Wesley, Reading, Mass., second edition, 1990.

[4] Adobe System Incorporated, Mountain View, Calif. *Adobe Font Metrics File Format Specification,* October 1995. Version 4.1, Technical Specification 5004.

[5] Jacques André. The Scrabble font. *The PostScript Journal,* 3(1):53–55, 1990.

[6] Jacques André. Random fonts and inkspreading simulation. Research note, INRIA, projet Opéra, Rennes, November 1992.

[7] Jacques André. Création de fontes en typographie numérique. Documents d'habilitation, IRISA + IFSIC, Campus de Beaulieu, Rennes, September 1993.

[8] Jacques André. Cahiers GUTenberg, Codage des caractères d'ASCII à UNICODE, 1995.

[9] Jacques André and Bruno Borghi. Dynamic fonts. In Jacques André and Roger Hersch, editors, *Raster Imaging and Digital Typography,* pages 198–204, Cambridge, England, October 1989. Cambridge University Press.

[10] Jacques André and Christian Delorme. Le Delorme : un caractère modulaire et dépendant du contexte. *Communication et langage,* 86:65–76, 1990.

[11] Jacques André and Victor Ostromoukhov. Punk : de METAFONT à PostScript. *Cahier GUTenberg,* 4:23–28, 1989.

[12] Apple Computer. *The TrueType Font Format Specification,* July 1990.

[13] Charles Bigelow and Kris Holmes. The design of Lucida®: an integrated family of types for electronic literacy. In J. C. van Vliet, editor,

Text Processing and·Documentation Manipulation: Proceeding of the International Conference, University of Nottingham, pages 3–17, New York, April 1986. Cambridge University Press.

[14] Wolfgang Böhm. Cubic B-Spline Curves and Surfaces in Computer Aided Geometric Design. *Computing,* 19:29–34, 1977.

[15] Robert Bringhurst. *The Elements of Typographic Style.* Hartley & Marks, Vancouver, 1992.

[16] Su Bu-qing and Liu Ding-yuan. *Computational Geometry—Curve and Surface Modeling.* Academic Press, San Diego, Calif., 1989.

[17] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms.* McGraw-Hill, Cambridge, Mass., 1989.

[18] Luc Devroye and Michael McDougall. Random fonts for the simulation of handwriting. *Electronic Publishing,* 1996. To appear.

[19] Gerald E. Farin. *Curves and Surfaces for Computer Aided Geometric Design: A Practical Guide.* Academic Press, Boston, third edition, 1993.

[20] Michel Goossens, Frank Mittelbach, and Alexander Samarin. *The LaTeX Companion.* Addison-Wesley, Reading, Mass., 1994.

[21] Frederic W. Goudy. *Typologia: Studies in Type Design & Type Making.* University of California, Berkeley, Calif., 1940.

[22] Roger D. Hersch, editor. *Visual and Technical Aspects of Types.* Cambridge University Press, Cambridge, England, 1993.

[23] John D. Hobby. Smooth, easy to compute interpolating splines. *Discrete and Computational Geometry,* 1:123–140, 1986.

[24] Alan Jeffrey. *fontinst: Font installation software for TEX,* September 1995. Version 1.500.

[25] Margo Johnson. Hybrid Digital Typefaces. *Emigre,* 38:47–58, 1996.

[26] Peter Karow. *Digital Typefaces: Description and Formats.* Springer-Verlag, Berlin, 1993.

[27] Peter Karow. *Font Technology: Methods and Tools.* Springer-Verlag, Berlin, 1994.

[28] R. Victor Klassen. Variable width splines: a possible font representation? *Electronic Publishing,* 6(3):183–194, 1993.

[29] Donald E. Knuth. *The TEXbook,* volume A of *Computers and Typesetting.* Addison-Wesley, Reading, Mass., 1984.

[30] Donald E. Knuth. *The METAFONTbook,* volume C of *Computers and Typesetting.* Addison-Wesley, Reading, Mass., 1986.

[31] Donald E. Knuth. *Computer Modern Typefaces*, volume E of *Computers and Typesetting*. Addison-Wesley, Reading, Mass., 1986.

[32] Donald E. Knuth. A Punk meta-font. *TUGboat*, 9(2):152-168, August 1988.

[33] Donald E. Knuth. Virtual Fonts: More Fun for Grand Wizards. *TUGboat*, 11(1):13-23, April 1990.

[34] Michael Kokula. Automatic generation of script font ligatures based on curve smoothness optimization. *Electronic Publishing*, 1994. To appear.

[35] Leslie Lamport. *LaTeX: A Document Preparation System—User's Guide and Reference Manual*. Addison-Wesley, Reading, Mass., 1986.

[36] Henry McGilton and Mary Campione. *PostScript by Example*. Addison-Wesley, Reading, Mass., 1992.

[37] Michael E. Mortenson. *Geometric modeling*. Wiley & Sons, New York, 1985.

[38] T. Packard. Ransom fonts. *The PostScript Journal*, 2:44-45, 1989.

[39] Réjean Plamondon, Ching Y. Suen, and Marvin L. Simner, editors. *Computer Recognition and Human Production of Handwriting*. World Scientific, Singapore, 1989.

[40] Franco P. Preparata and Michael Ian Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, New York, 1985.

[41] Thierry Pudet. Realtime fitting of pressure brushstrokes. Research Report 29, Digital, Paris Research Laboratory, March 1993.

[42] Tomas Rokicki. *Dvips: A DVI-to-PostScript Translator*, January 1995. Version 5.58f, Edited for Dvipsk by Karl Berry.

[43] Richard Rubinstein. *Digital Typography: An Introduction to Type and Composition for Computer System Design*. Addison-Wesley, Reading, Mass., 1988.

[44] John F. Sherman. *Taking Advantage of PostScript*. Wm. C. Brown Publishers, Dubuque, Iowa, 1992.

[45] Signature Software Incorporated. SUPERscripts: Cursive Handwriting Fonts. Windows Version 2, 1994. 489 North 8th Street, Suite 201, Hood River, Oregon 97031.

[46] Ching Y. Suen, Marc Berthod, and Shunki Mori. Automatic recognition of handprinted characters: the state of the art. *Proceedings of the IEEE*, 68(4):469-487, April 1980.

[47] Erik van Blokland and Just van Rossum. Random code—the Beowolf random font. *The PostScript Journal*, 3(1):8-11, 1990.

[48] Erik van Blokland and Just van Rossum. Different approaches to lively outlines. In Robert A. Morris and Jacques André, editors, *Raster Imaging and Digital Typography II*, pages 28–33, Cambridge, England, October 1991. Cambridge University Press.

[49] Norman Walsh. *Making TEX Work*. O'Reilly & Associates, Sebastopol, Calif., 1994.

[50] Robin Williams. *The Non-Designer's Design Book*. Peachpit Press, Berkeley, Calif., 1994.

[51] Robin Williams. *A Blip in the Continuum*. Peachpit Press, Berkeley, Calif., 1995.

[52] Y&Y. LucidaBright + LucidaNewMath, 1992. 106 Indian Hill, Carlisle, Mass. 01741.

IMAGE EVALUATION
TEST TARGET (QA-3)

150mm

6"