FORMATTING FONT FORMATS

Luc Devroye
School of Computer Science
McGill University
Montreal, Canada H3A 2K6

September 15, 2003

ABSTRACT. Font formats are a tug of war between artists (designers and drawers), programmers (computer scientists), the business world, and users. Each of these four groups has had an influence on the path font formats have followed. We review the successes and failures, and present a wish list of properties that a good font format should have.

KEYWORDS AND PHRASES. History of typography, electronic font formats, PostScript, truetype, OpenType, metafont, drawing models.

CR CATEGORIES: 3.74, 5.25, 5.5.

1991 MATHEMATICS SUBJECT CLASSIFICATIONS: 60D05, 68U05.

Author's address: School of Computer Science, McGill University, 3480 University Street, Montreal, Canada H3A 2K6. The author's research was sponsored by NSERC Grant A3456 and FCAR Grant 90-ER-0291. FAX number: 1–514–398-3883. Email: `luc@cs.mcgill.ca`.

Introduction

Let us try to imagine what format fonts will be living in several decades from today. That question is very relevant in 2003, as the type world is ready for yet another overhaul. In this paper, we briefly comment on the present situation, in which the truetype and PostScript font formats are dominant, and the OpenType format, which was proposed about eight years ago, is being promoted. We then take a broader and long-term view and touch upon various issues related to the design of electronic font formats.

Before we embark on the more technical aspects of electronic font formats, it helps to identify the forces that are helping to shape these formats.

First and foremost, the users would like to see simple, useful formats, that are easy to manipulate and edit. They want to have access to the art created by great type artists and the technical refinement provided by digital font experts. In addition, professional users may demand a certain degree of flexibility in a font, in order to incorporate personal choices.

The artists and typographers have had a lot of input in pre-electronic font formats. The early typographers were nearly all craftsmen. In the twentieth century, various technological advances were made at companies like Linotype and Monotype, that were driven by the demands of the type designers, and we witnessed a shortening of the time between design on paper and actual glyph production. In the electronic era, the artists and typographers have been largely left out of the decisions on font formats, and this has led to an unfortunate split in the family of typographers: on the one hand, there are those who never adapted to the mouse and the screen, and continued designing typefaces using pen and ink. Perhaps the medium or perhaps the all too mathematical font formats and font editors worked as deterrents for them. On the other hand, we have seen the emergence of digital artists who design glyphs directly on the screen, and do so with extreme efficiency. In this category, we can place prolific artists such as Lucas De Groot, Jean-Franųis Porchez and David Berlow. A few even mastered the bitmap format, and became the ultimate digital technicians. Matthew Carter's Verdana, an outline font designed and tweaked for optimal screen output, is a prime example of the output of a master digital technician. For more on the designer's perspective, read Hermann Zapf's 1991 book. For both groups of designers, however, the font format came first, and they had to adapt to the technology. Perhaps, in the future, we should ask them for some input, and create a medium in which their freedom is unchallenged.

The engineers have a say in the matter as they report about the limitations of certain media. Screen renderers, printer specifications and other physical media limit the format in which fonts are presented to those media. There is a movable boundary defined by the partition of the responsibilities between computer and peripheral device. For example, a "lazy" computer may send a raw font to a printer, and the printer must do all the processing internally to put ink on a page [this is the strategy used in native PostScript printers, for example]. Other media expect a device-specific font format, often a bitmap or pixel font, adapted to the resolution and device specifications. The onus here is on the computer, not the device. In these cases, font formats are sometimes designed by engineers, who have very little typographic training.

Computer scientists and programmers (software artists) are increasingly important players. The creators of PostScript, Geschke and Warnock, who developed PostScript based on the page description language PDL by John Gaffney, 1976 (see Adobe, 1990a) and the type 1 and type 3 font formats (Adobe, 1990bc), were computer scientists with a graphical vision. PostScript succeeded thanks to its simplicity and flexibility. Today's influence of Adobe is in fact largely due to the invention of PostScript. In font software and format design, the computer scientists are largely preoccupied with logical organizations of files and with issues like standardization. This endeavour often carries them away, so, just as with the engineers, this group of people should remain dedicated to the users and the font designers, not the other way around.

Going up the ladder, we find the vendors, foundries and companies, whose interests are often commercial, and who by definition are concerned with company reputation, sales volume, market share, proprietary formats, and software strategies. Fonts are often developed as part of larger software packages or in conjunction with certain operating systems. This world evolves also around patents, trademarks and copyrights, the various ways in which software and typefaces may be protected. The actual font format itself that is supported by this group is often the result of various market decisions, the prime example being the story of PostScript, truetype and OpenType that will be recalled a bit further on.

The final force at work in the creation of a font format is inertia, driven by tradition and historical models. An electronic font format is often the result of a modification of a previous format or technology. Backward compatibility is often cited as a requirement for a new format, but this been contradicted by the historic record, with dramatic incompatible quantum jumps in the technology.

The typeface repertoire is rich, with many typefaces existing in only one of several possible formats. Many historic faces only exist in print (in specimen books or old manuscripts), while hundreds if not thousands are only available in metal or wood. In the phototypesetting era—the 1950s to 1980s—, typefaces were stored in photographic format. And finally, in the 1980s, electronic font formats were introduced. Among these, the earliest are the bitmap formats such as "fon", "bdf", and "fnt". In 1982, Jim Warnock and Charles Geschke introduced PostScript (see Adobe, 1990a), and suggested storing glyphs by describing their outlines as Bezier curves. This led to the type 3 and type 1 font formats. Knuth also used Bezier curves for outlines, but had the idea of describing glyphs by programs in his METAFONT (1986a), which was introduced and perfected in the period from 1977-1985. In 1987-1989, Apple's Sampo Kaasila developed the truetype format, which was an economic decision to counteract the stranglehold Adobe had on the type technology market at the time with its proprietary PostScript. Finally, Microsoft and Adobe joined forces in the 1990s to create OpenType in the hope of reconciling truetype and PostScript. The discussion below will show that this is only a minor technological step. When we look into the future, we must take this varied historical record into account. The electronic era is the first one in which font formats were proprietary—they were designed and "belonged" to one or more companies. In taking the next step in formats, we should stear clear of this trap, and agree on a route that is open to everyone.

It is very likely that the present computer data model, in which the bits are the atoms, and in which bit storage is somehow achieved at the microscopic physical level, will survive for at least a few decades, so we will use words like files and bits in this paper, with the caveat that a future reader may find this vocabulary old-fashioned. Taking a long-term view, we will describe the hub model for font storage and manipulation. The details will be described in subsequent sections.

The hub model

A font is an implementation of a typeface: ideally, it contains the full description of that typeface. It is like a complete book—anyone can read it, nothing is missing, the author is clearly identified, and so on. Similarly, a font should thus be implemented in a human-readable "open book" format. None of the previous formats had this. In the metal days, valuable information about the creative process was missing, and only foundries actually owned metal type. Truetype, type 1 and OpenType fonts are only computer-readable. Metafont and type 3 can only be interpreted by programmers and computer scientists. In fact, because of the proliferation of formats, we have truetype, type 1 and OpenType versions for hundreds of typefaces, and each version is slightly different from the other one because of technical incompatibilities. In other words, at present, one typeface "lives on" in many fonts, and this is a chaotic situation.

The human-readable mother font for a typeface should exist once, and ideally be frozen forever, just as with a "version" of a piece of software. Additions and modifications of it then yield new fonts. One can swim downstream from the mother font to popular implementations (truetype, type 1, etcetera) by filters, but horizontal swimming between OpenType and type 1, for example, is not recommended, and upstream conversions are to be avoided by all means. Font editors at present include Fontographer (owned by Macromedia, described by Moye, 1995, FontLab (by Yuri Yarmola), Font Studio (by Letraset), Ikarus (by Peter Karow at URW), Pfaedit (by George Williams), FontCreator (by Erwin Denissen), Softy (by Dave Emmett), Manutius (by A. Gebert) and Noah (by Yeah Noah). Each operates on one or more formats on one or more computer platforms. New editors should be designed to create or manipulate that mother font, thus leading to a more logical situation. Artists too should be able to directly access that mother font. Printers, screens, applications, and handheld devices can operate on compact electronic formats obtained downstream from the mother font. It should be noted that most serious editors store fonts in an internal human-readable format, and have in fact created models for mother fonts. Most of these do not go beyond a one-to-one translation of the corresponding binary format though. For surveys on font technology, we refer to the books by André (1993), Karow (1994ab) and Knuth (1999) and the articles by Gonczarowski (1993) and André and Hersch (1993).

Each of the sections below treats one of the aspects of the mother font in more detail.

Outline and pre-outline

One of the main contributions to computational geometry and computer-aided geometric design was the development of the Bézier curve by James Ferguson, an airplane designer, Pierre Bézier, an engineer with Renault, and de Casteljau, an engineer at the competing French automobile company, Citroen. Two and three-dimensional objects could be described and approximated rather simply by concatenating sections of curves. This is, in fact, a way of transforming a physical object into a number of bits, and thus, a way of compaction. One can take a 1MB high-detail photograph or scan of a letter, which after compaction by standard methods such as "zip" (which uses a mix of Huffman and Lempel-Ziv coding) may be reduced to 200 kilobytes or so. Yet, by just storing the collection of Bézier curves, the same letter can be locked in memory using under a kilobyte, as the formula for a n-th order Bézier curve just requires the knowledge of $n+1$ control points $x_0, x_1, \ldots, x_n$ in the plane:

$$x(t) = \sum_{i=0}^{n} \binom{n}{i} t^i (1-t)^{n-i} \cdot x_i \ , \ 0 \le t \le 1.$$

Here $x(t)$ is a parametric curve, a continuous convex combination of the control points (hence, the Bézier curve stays within the polygon formed by the control points), starting at x_0 and ending at x_n. The mathematical properties of Bézier curves and splines in general are described by Farin (1993), Su and Liu (1989) and Bu-Qing and Ding-Yuan (1989).

It was only natural that PostScript and METAFONT adopted the Bézier curve: their creators settled on the cubic Bézier curve ($n = 3$). Truetype uses quadratic Bézier curves ($n = 2$), which was an unfortunate decision, as a quadratic Bézier curve can without loss be transformed into a cubic one (given x_0, x_1, x_2, set $y_0 = x_0, y_1 = (2x_1 + x_0)/3, y_2 = (2x_1 + x_2)/3, y_3 = x_2$, to obtain the cubic control points y_i), but not vice versa. So, type 1 is downstream from truetype, yet, cubic approximations are usually heralded as being more compact than quadratic approximations. Artists report that cubic curves have a richer palette than quadratic curves.

Bézier curves cannot represent circles without error, no matter how large n is [for the mathematically inclined, this is an excellent exercise]. For example, a 90-degree circle arc is best approximated by a cubic Bézier if we take the control points $(0,1), (a,1), (1,a), (1,0)$, and $a = (4/3)(\sqrt{2}-1) = 0.5522847498\ldots$. This omission could have been rectified if Bézier had allowed parametric descriptions involving either a square root or a trigonometric function.

Type designers who design type on screen are in fact Bézier point placement artists. Their instrument is the mouse. This is very hard, as many control points are not on the curves, and continuity of derivatives between adjacent Bézier sections is difficult to achive by the naked eye. Some designers still use pen and paper, and rely on scanners for computer input. Yet others, used to software for artists, are good at placing points that are related to Bézier curves indirectly, such as demonstrated by Böhm splines (1977), where smooth continuous derivative Bézier sections are implied.

Hobby (1986) and Knuth (1986a) developed an algorithm for constructing a sequence of Bézier curves that is forced to visit the designer's set of points. This algorithm is built in in METAFONT (Knuth, 1986, p. 131), and can be a great on-line tool for some. We call such ways of describing outlines "pre-outlines".

To summarize, the mother font should be flexible and permit choices between any of a number of outline and pre-outline formats, as long as each format defines a mathematical curve in a unique manner. Concatenations of Bézier curves of any degree (with n being a parameter) should be allowed, as well as several pre-outline formats to accommodate the typographers at large. At least one spline model should be included that stores circle arcs without any error, so that we can finally have exact approximations of those fantastic geometric ruler and compass creations of masters like Philippe Grandjean, the designer of the Romain du roi (1693-1745).

Ink models

The two main formats, truetype and type 1, and their derivative, OpenType, are all based on a primitive ink model, based on the principle that a character is defined by a number of closed outlines, which are then filled with ink according to the non-zero winding rule. This means that if a point or pixel is in a given region, then its color, black or white, can be determined by drawing a ray from that point to infinity (in any direction!) and keeping a weighted count of the outlines crossed. A weight of one is given to a clockwise turning contour and minus one to a counter-clockwise contour at each crossing point. But this is clearly not how we place ink on paper at home, where overwriting and erasing are two primary operations. Also, one should be able to work with many black/white images, perhaps levels of images, and define a final image as a logical operation on component images, using operators like "or", "and", "exclusive or", and "not". One should be able to mark regions black or white by pointing to it—in other words, the region containing "x" should always be black.

Stroke fonts are distinguished from outline fonts by their ink model: a stroke is defined, perhaps by a collection of splines of Bézier curves, and ink is placed by following the stroke with a brush or nibbed pen, perhaps tilted at an angle or suitably shaped. Japanese and Chinese seem like prime territory for such fonts. But closer to home, we should not forget about the characters that are created by the interaction between a pen and a tablet, as on palm-held devices, or signatures made with a magnetic pen, or input from a computer tablet. A person's handwriting is often better captured by letting the person write on a tablet (so that we obtain the stroke points in chronological order, with dynamic information), as opposed to scanning the individual's handwriting. Tablet input is naturally translated into strokes.

The recommendation to allow many ink models sounds like an extension of the PostScript graphical model, but it can be organized by rasterizers and printers without too much trouble as all can be internally reduced to outlines (out of sight of the font designer!) and to the classical non-zero winding rule. The extension is suggested, once again, to make type design easier, more universal, more current and more accessible.

Path complexities

Outlines and curve data are not unrestricted in our present electronic formats. For example, paths in PostScript and thus type 1 are limited to about 750 control points. Such limitations make it impossible to store certain complex characters as are found in ornaments, decorative initial caps, and outlines based on high resolution scans. Truetype has higher limits, but the mother font should in principle have no limit. Limits could be introduced by various formats downstream, and by various viewing media even further downstream, but it should not be introduced at the mother font level.

Accuracy

Outlines in any form require mathematical input. As points need to be represented in a unique manner across all platforms, it is imperative that all mathematical descriptions be in terms of integers. For example, a point can simply be (x, y), where x and y are integers, but it can also be $(x/x', y/y')$ where x, x', y, y' are integer, so that we can attain all rational numbers. At present, assuming that a character occupies the square $[0, 1]^2$, points in that square can be addressed as $(x/1000, y/1000)$ with x, y integer, as is common in type 1. Type 1 permits higher values than 1000, but not all interpreters of type 1 fonts are happy with that. In truetype, the 1000x1000 box is replaced by 2048x2048. The different box sizes shows that there is no lossless horizontal conversion between truetype and type 1, as $x/1000 = y/2048$ implies that x must be a multiple of 125 and y a multiple of 256, and any other values imply a loss in accuracy. OpenType inherits the type 1 restriction for its CFF style implementation, and the truetype restriction otherwise.

It is incomprehensible that no one has even attempted to increase these limits of accuracy. Picture a complex character consisting of 50 rows and 50 columns of circles that touch other. In a 1000x1000 integer box, this would force the radius of each circle to be 5. In a cubic Bézier implementation of a quarter circle, we need to place the control points at $(0, 5), (a, 5), (5, a), (5, 0)$, and must select the values $1, 2, 3$ or 4 for a, recalling that the ideal value is about 2.75 (see above). By picking $a = 3$, the circles will be far from perfect!

There is an even more compelling reason why the accuracy must be increased: the historical record. As we scan historical designs, in our quest to store everything in some electronic format for the future, we must ensure that as little as possible is lost in the process. Just as the noise in old LPs was due to mechanical limitations, so is the noise introduced by storing valuable designs using less-than-ideal accuracy. Reconstruction and de-noising will be difficult once the damage is done.

In a 1000x1000 box, storing a point (x, y) requires about 20 bits. In a 1 million by 1 million box, the storage increases to about 40 bits, and for an unimaginable 1 billion by 1 billion box, the storage increases to about 60 bits. Thus, by doubling the storage requirements, we can in fact increase the number of point positions by a factor of one million! By tripling, that multiplication factor becomes one trillion. In other words, this is a change that comes relatively cheaply. Furthermore, since the mother font is upstream of everything else, one can always drop down to lower accuracies when moving downstream. For storing points, perhaps the best method is to work with (x, y, n), where n is the accuracy, and x and y are integers in or near the range $[0 \ldots n]$. The triple then represents $(x/n, y/n)$. The value of n should not a priori be restricted. Accuracy should be a variable parameter, perhaps different from font to font.

It must be mentioned that accuracy is not an issue in a pure PostScript type format such as type 3, and that theoretically, in a type 1 font, it can be controlled by the Font Matrix, although, in practice, many applications expect a 1000x1000 matrix.

Programming and fonts

The current crop of electronic font formats are just tables. Just as with their metal counterparts, they are dead objects that require manipulation by an external master or computer program. Even though some companies claim that their fonts are programs, this is false, with the exception of METAFONT and Type 3, which were both major steps forward in font technology. In addition, some truetype fonts have some bits of code in their hinting sections, but it is debatable whether this too should not be considered as a table rather than a program.

The type 3 format allows the use of the full PostScript language: there are parameters, variables, conditional instructions and loops. It is possible to make randomized fonts, e.g., for the simulation of handwriting, and to create connected context-sensitive glyphs. Characters can be programmed in terms of tunable parameters. Perhaps the simplest tunable fonts are the multiple master fonts that Adobe proposed in the 1990s, in which one can vary one or more parameters to interpolate between extremal fonts. Of course, this can be emulated in type 3 fonts. METAFONT has similar capabilities, and, in fact, Knuth demonstrated with his Computer Modern family (1986c) that one program per glyph suffices to create a family of 72 component fonts, ranging from typewriter type to serif and sans serif (see also Haralambous, 1993). Other attempts at parametrization, such as Infinifont (McQueen and Beausoleil, 1993) and LiveType (Shamir and Rappoport, 1996, 1997) were short-lived.

The disadvantage with such programmable fonts is the necessity to have at one's fingertips, in printers, and in applications, powerful interpreters or on-the-fly converters to other formats. Furthermore, the danger of a virus lurks in every piece of code—indeed, executing a type 3 "font" can have as side effect the creation or deletion of one or more files. Finally, interpreters for powerful languages are often legally protected and can only be licensed at enormous fees. With language features wisely restricted to purely mathematical and graphical operations, one should be able to flag mother fonts that contain active code, analogously to the present flagging of multiple master fonts.

Reviving the idea of programmable fonts will have enormous benefits for mathematical typesetting. Knuth's model (Metafont + TeX, 1986b) is now over 20 years old, and has a few shortcomings that require an update. There should be a continuum of optically adjusted symbols like brackets and parentheses, with line thickness and size adapted to the surrounding text. At present, the symbols are selected from a finite set, which often leads to aesthetic mismatches. Improvements should be made in optical size matching of subscripts and superscripts.

Of course, optically and continuously adjusted symbols are only part of my mathematical typesetting wishlist. There should ideally be a symbiosis of figures, formulas and text, all playing and interacting on the page, a bit as with blackboard mathematics in the hands of a master mathematician. This requires a paradigm that transcends TeX.

In the area of randomized fonts for the simulation of handwriting, we refer to Devroye and McDougall (1995) for a theoretical development and some crude examples, to Desruisseaux (1996) for a thoroughly researched font called MetamorFont, and to André and Borghi (1989), Dooijes (1989) and van

Blokland and van Rossum (1991) for earlier attempts in this direction. All these developments used the programming power of type 3 to create random-looking characters that are either based on a sample of one's handwriting (as in the first reference above) or that are constructed artificially by programming the randomness in the outlines (as in MetamorFont). It would be a shame not to include a random number generator in the specification of the mother font. Of course, one should make sure that the random sequence generated can be "replayed" for debugging purposes.

Ligatures and context sensitivity

Ligatures are combinations of two or more characters. Context sensitive characters are single characters that change shape as a function of their context or neighborhood. The activation of a context sensitive change should always be the responsibility of the application—the font should only contain the various shapes without getting involved in questions related to context.

This separation of form and application should also apply to ligatures. Fonts provide the shapes only. This division has been rigorously supported in the METAFONT+TeX model, with TeX taking care of the actual activation of ligatures. In OpenType, a GSUB table was introduced that in combination with software such as InDesign will activate ligatures. However, which letters react in what manner is stored in the GSUB table, so that the separation is less clear, forcing the font designers to worry about non-artistic issues, and thus making the design process too hard. Artists can hardly be expected to design GSUB tables!

Arabic requires a large number of ligatures for proper typesetting (see, e.g., Smitshuijzen AbiFarès, 2001). However, a large number may also be required for Latin handwriting. The author has experimented with ligatures in an interesting way, creating glyphs in type 3 with a tablet for about 1600 ligatures. These consisted of the most popular pairs of letters, with a distinction between starting pairs in words, ending pairs, and mid-word pairs. In addition, triples were added, again by popularity as measured in a large body of text. Finally, single letters came in three forms, starting letters, sentinels, and mid-word letters. Combinations of capitals with one or two trailing lower case characters were also thrown into the collection. Given a text, a small program decided on the optimal composition of a word using these ligatures thanks to a formula based on rewards and penalties. Others can improve the typesetting by changing that parsing program, without touching the font file, keeping the activation of the ligatures away from the fonts.

Bitmaps and images

The preservation and restoration of old typefaces if done in outline format requires an increased accuracy. Nevertheless, at some point, a crucial transformation from bitmap or image to outline is necessary. This process is often called tracing or auto-tracing. Algorithms for this abound (see Avrahami and Pratt (1991), Plass and Stone (1983), Itoh and Ohno (1993), Gonczarowski (1991), Schneider (1990), Lejun, Hao and Wah (1994), or Mazzucato (1994)). However, the perfectionists may wish to keep the original image, rather than the possibly polluted outline. The storage may be prohibitive, but one might want to compress the images by clever lossless (or reversible) compression methods that are designed to look for straight edges and smooth outlines. Such dedicated or "smart" compression methods may yield high compression ratios. The mother font should allow for the storage of bitmaps of extremely fine grain.

It is not far-fetched to project that one day, all fonts will be stored in a compressed bitmap format, with storage capabilities expanding at an enormous pace, and with smart compression an active area of research in information theory. The benefit of such a format is that the design of a font editor will be much easier, while the editing process itself will feel more natural to the typographers. In fact, paper and electronic format will converge again.

Standardization and coding

The effort to standardize the naming of symbols and the positioning (or: coding) of symbols by attaching permanent numbers to each of them, should continue. Unicode has changed the typographic scene in this respect, but it is unrealistic to expect each font to be "complete", using whatever definition of "complete" one wants. For one thing, new symbols are invented daily, so that the standardizers will never be able to keep up. Furthermore, special unique and innovative glyphs add to the value of a typeface, especially if no other typeface offers them. It is in the human nature to create and invent, and thus, the mother font should not be tied to one particular coding scheme. It could be flagged as Unicode-compliant or Unicode-subset-compliant, but in the matter of encoding and naming, we cannot the predict the future—who could have predicted the Euro symbol in 1970—, and will therefore have to recommend that mother fonts be unrestricted.

The number of glyphs in one font should not a priori be limited. Each glyph should have a name and an integer-valued position, but the maximal value among those integers should have no obvious bound, not even the seemingly large bound (65,536) that comes with Unicode.

Font information

One of the key components of the mother font relates to the information and history of the typeface and the font. Each font or typeface has a genealogical history. There is an ancestral tree or dag (directed acyclic graph) that explains the present. The tree should be shown, and each node and link in it explained and if possible, dated. It is a pipedream to think that one can have a permanent font information depository somewhere. The best we can hope for is to make the font information an essential part of the mother font. In many cases, the ancestors can and should be traced back to the days of metal type.

Font names should be unique, perhaps by introducing foundry letters and short version numbers in the font name. In no case should information about the font be separated into a "readme" file, another invention of the eager computer scientists. The font information should explain the absolute and earliest origins of the typeface. It should then report on the changes, revivals, additions, and extensions that have transformed the original typeface into this font. Clearly, this information can be erased and fraudulently altered, but no practical system will prevent this. At present, many foundries such as Adobe and Linotype do not mention the typographer who created the typeface anywhere in or near the font. They offer biographies of their designers on web pages that may one day disappear while their information-starved fonts survive. Thus, the information field should be used to pay a permanent tribute to the creators, typographers and artistic forefathers.

Human-readable format

The mother font has to exist in a simple human-readable form. For truetype and openType, the TTX tool by Van Blokland and Van Rossum (2002) permits a one-to-one transformation between the binary font file and a human-readable XML file. Other examples of such mapping programs exist for other formats. Non-commercial formats such as METAFONT essentially exist only in text format.

Motivated by the simple requirement that anyone, even a person without appropriate software, or without software attached to a certain decade, can read and interpret the instructions, all font information and all outlines must be readily accessible. Adding an accent or dieresis to a character should be a trivial operation. And importantly, even moderately capable programmers should be able to write simple code to act upon the mother font to achieve a certain effect. For all these reasons, a binary model should be excluded. Those who argue that the storage may be prohibitive should be reminded that fonts can be compressed on the fly when sent over a network or to a device.

Automated operations

In any typeface, metric and kerning information is essential. Kindersley (1976) has provided nice ideas on how letters should be spaced. At URW in 1991, an attempt was made at automating character spacing through internal programs cryptically called hz and Kq. The choice of spacing around each character requires a certain amount of expertise, and a well-kerned font is out of reach of most typographers. Therefore, the mother font should have flags that indicate the automation of the process of determining the sidebearings of the characters and the kerning between all pairs of glyphs. And if set, another parameter could be used to select an algorithm from a collection of possible algorithms, with further parameters left to the user's choice. The kerning algorithms should unambiguously defined, but not in a programming language. In this manner, automation and hand-kerning can coexist, and one can override the other.

Hinting is uniquely tied to electronic fonts, as earlier formats were not concerned with discretized media. It too can be dealt with in the way suggested above for kerning, via the setting of a parameter which selects one of the built-in hand-kerned data sets, or one of the automated algorithms. Examples of the latter are described by Karow (1989), Andler (1990), Hersch and Bitrisey (1991) and Herz and Hersch (1994). An argument could be made to exclude hinting altogether from a font, and insist that it is the responsibility of the printing or screening device.

References

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

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

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

S. F. Andler, "Automatic generation of gridfitting hints for rasterization of outline fonts," in: *Proceedings of the International Conference on Electronic Publishing, Document Manipulation & Typography, Gaithersburg, Maryland, September 1990*, (edited by R. Furuta), pp. 221–234, New York, 1990.

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

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

J. André, "An introduction to digital type," in: *Visual and Technical Aspects of Types*, (edited by R. D. Hersch), pp. 56–63, Cambridge University Press, Cambridge, UK, 1993.

J. André, "Ligatures & informatique," *Cahiers GUTenberg*, vol. 22, pp. 61–86, 1995.

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

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

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

S. Bu-Qing and L. Ding-Yuan, *Computational Geometry: Curve and Surface Modeling*, Academic Press, 1989.

Unicode Consortium, "Unicode," `www.unicode.org`, 2003.

B. Desruisseaux, "Random dynamic fonts," M.Sc. thesis, School of Computer Science, McGill University, Montreal, Canada, October 1996 .

L. Devroye and M. McDougall, "Random fonts for the simulation of handwriting," *Electronic Publishing*, vol. 8, pp. 281–294, 1995.

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

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

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

J. Gonczarowski, “Industry standard outline font formats,” in: *Visual and Technical Aspects of Types*, (edited by R. D. Hersch), pp. 110–125, Cambridge University Press, Cambridge, UK, 1993.

J. Gonczarowski, “Curve techniques by autotracing,” in: *Visual and Technical Aspects of Types*, (edited by R. D. Hersch), pp. 126–147, Cambridge University Press, Cambridge, UK, 1993.

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

Y. Haralambous, “Tour du monde des ligatures,” *Cahiers GUTenberg*, vol. 22, pp. 87–100, 1995.

R. D. Hersch and C. Bitrisey, “Model-based Matching and Hinting of Fonts,” *ACM Computer Graphics*, vol. 25, pp. 71–80, 1991.

J. Herz and R. D. Hersch, “Towards a Universal Auto-Hinting System for Typographic Shapes,” *Electronic Publishing (EP-odd)*, vol. 7, pp. 251–260, Special issue on Typography, John Wiley, 1994.

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

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

P. Karow, “Automatic hinting for intelligent font scaling,” in: *Raster Imaging and Digital Typography: Proceedings of the International Conferences, Ecole Polytechnique Fédérale, Lausanne, Switzerland, October 1989*, (edited by J. André and R. D. Hersch), pp. 232–241, New York, 1989.

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

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

D. Kindersley, *Optical Letter Spacing for New Printing Systems*, Wynkyn de Worde Society, distributed by Lund Humphries Publishers Ltd., 26 Litchfield St. London WC2, 1976.

D. Kindersley and N. Wiseman, “Computer-Aided Letter Design,” *Printing World*, pp. 12–17, 1979.

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

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

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

D. E. Knuth, *Digital Typography* , Cambridge University Press, 1999.

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

C. W. Liao and J. S. Huang, “Font generation by beta-spline curve,” *Computers and Graphics*, vol. 15, pp. 527–534, 1991.

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

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

C. D. McQueen III and R. G. Beausoleil, "Infinifont: a parametric font generation system," *Electronic Publishing*, vol. 6, pp. 117–132, 1993.

S. Moye, *Fontographer: Type by Design*, MIS Press, 1995.

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

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

A. Shamir and A. Rappoport, "Extraction of typographic elements from outline representations of fonts," *Computer Graphics Forum*, vol. 15(3), pp. 259–268, 1996.

A. Shamir and A. Rappoport, "LiveType: a Parametric Font Model Based on Features and Constraints," Technical Report TR-97-11, Institute of Computer Science, The Hebrew University, 1997.

H. Smitshuijzen AbiFarès *Arabic Typography*, Saqi Books, London, 2001.

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

URW, "Kerning on the Fly," Technical Report, URW, 1991.

URW, "Phototypesetting with the URW hz-program," Technical Report, URW, 1991.

URW, "Phototypesetting with the URW Kq-program," Technical Report, URW, 1991.

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

E. van Blokland and J. van Rossum, "TTX," `www.letterror.com/code/ttx`, 2002.

H. Zapf, *Classical Typography in the Computer Age*, Oak Knoll Books, 1991.