Dreamboat

Floating point numbers and METAFONT, METAPOST, TEX, and PostScript Type 1 fonts

Claudio Beccari

Abstract

Some features related to METAFONT, METAPOST, and TEX, concerning the ability of such programs to perform calculations, are discussed and the opportunity, not to mention the necessity, of extending these programs to handle real numbers with floating points calculations is strongly (and hopefully) suggested.

1 Introduction

Every TEXie knows, or should know, that Donald Knuth, the Grand Wizard of TEX and METAFONT, designed these wonderful programs using only integer arithmetic. There are and there were at the beginning of the eighties several reasons to choose this solution. The many calculations TEX and METAFONT have to do in order to compose a page or, respectively, render the pixels of a character, are, or were, certainly faster if integer arithmetic is used.

But perhaps the most stringent argument that convinced Knuth to follow this strategy was not speed, but the fact that by the beginning of the eighties the various platforms where TEX and METAFONT were supposed to run had different real word

lengths, different real encodings, and, most substantially, different precisions, so that the same source program could not be guaranteed to have the same results on different platforms, which was of paramount importance for Knuth's goal of portability.

I must admit that even today that choice is strategically perfect and very few people have experienced any drawback caused by this "limitation".

Nevertheless, there are some fields where floating point arithmetic is important, especially if we consider that nowadays there has been a great deal of standardization in the internal representation of floating point numbers, and compilers are much more uniform than twenty years ago. Certainly there is no guarantee that the same input data processed by the same input program on different platforms yields the same results, but the probability of getting the same results is much higher.

2 Fixed radix vs. floating point numbers

Both in TEX and in METAFONT (and later on, in J. Hobby's METAPOST) Knuth needed fractional quantities and decided to use fixed radix notation.

2.1 Real numbers in TEX

Lengths, glues, muglues and the like are all intrinsically real quantities consisting of a measure and a unit of measure. To implement fixed radix real numbers in practice, Knuth converted all these quantities into "scaled points"; a scaled point is the fraction $1/2^{16}$ of a printer's point. Since this is the smallest unit of measure used in TEX, and is such a small quantity (it equals approximately 200 billionths of an inch and 5 millionths of a millimeter, more precisely 53.6 Å, a small fraction of the visible wavelengths) this unit was correctly taken as an indivisible part. Thus, any other unit, any other length, glue or muglue could be thought of as an integer number of scaled points. This in substance is the idea of a fixed radix number, a number that has a fixed number of fractional bits and is expressed by an integer number which is the multiple of the least significant bit. And this is precisely the reason why a 32-bit word can hold limited quantities which amount to, in our case, a maximum[1] of $(2^{14} - 1)\text{pt} = 16\,383\text{pt}$. Since this huge number of points approximately equals 19 feet or 6 meters, it appears that the fixed radix notation is more than adequate for representing every useful piece of typographic information.

The transformation between input data (in decimal notation with a variable number of decimal digits) into a fixed radix number is done in the mouth of TEX, I suppose; that is when input data is being tokenized for further processing in TEX's stomach.

The allowed operations that TEX can perform on numerical data are addition, subtraction, multiplication and division, but with strong limitations; counters can be assigned integer values, including the contents of length registers, where the lengths have already been transformed into fixed radix numbers, that is into integers. Lengths, glues and muglues may be added, subtracted, and may be multiplied or divided by integers; in the input stream they can be "multiplied", or should I say "scaled", by a real number; this means that if you have a length variable named, say, `\foo`, containing some value, you can write something like this in your input stream

```
\foo=0.67\foo
```

or in LATEX

```
\setlength{\foo}{0.67\foo}
```

by which the contents of `\foo` is substituted with its previous contents multiplied by `0.67`.

All of the graphic extension packages to TEX and LATEX, such as PICTEX, have to produce some sort of calculation between real numbers; TEX was not built for that, but by means of some "dirty tricks", such as the ones described in Appendix D of *The TEXbook* [1], the task becomes possible. For example a multiplication is performed as such: the first factor "scales" the fixed length of 1 pt and is assigned to a length variable; this length variable is scaled by the other factor, and finally the result is extracted by means of the primitive `\the` and stripped of the ending `pt` unit of measure. Divisions are more complicated because of the possibility of dividing by zero and because they might give rise to oversized results (this can happen as well with multiplications). The extension package `calc.sty` helps the user in specifying the calculations to be performed on the fly, but does not extend the possibilities offered by TEX primitive commands. Close examination of the various other extension packages available in the CTAN archives clearly shows the many efforts spent by several TEXies in order to help making calculations a little more comfortable and reliable.

Notice that LATEX's New Font Selection Scheme frequently performs this kind of calculation, including stripping the `pt` part from `\the` output.

[1] Since 16 bits represent the fractional part, there remain 16 bits available for the integer part; one bit is used for the sign (or equivalent) and the other is for special purposes; there remain 14 bits available for the integer part of a fractional number, hence the maximum value is $2^{14} - 1$.

Another extension that makes heavy use of real numbers is the `graphics` package, which needs to determine the transformation matrix coefficients for rotations by means of the trigonometric functions. By reading the code, I discovered marvels of numerical computation; even considering that the TEX community includes many people with varying competence and skill, I must admit that these sorts of numerical procedures are difficult to find even in the specialized literature.

For this reason, writing any extension package that requires any kind of calculation becomes very difficult and error prone.

2.2 Real numbers in METAFONT

METAFONT works with integer and real numbers in a very efficient way; actually all numbers may be thought of as real numbers. It performs any kind of operation and allows the user to specify also some ordinary irrational and transcendental functions such as the trigonometric ones, logarithms and exponentials; it actually deals with complex numbers, whose real and imaginary components are real numbers, but users not skilled in complex analysis need not worry about them. Whoever has done any work with METAFONT cannot but be surprised in discovering its endless resources.

Nevertheless, even METAFONT uses fixed radix real numbers and the largest one, according to *The METAFONTbook*, is 4095.99998, which, taking into account the fractional part, equals $4096-2^{-16}$. Now since 4096 equals 2^{12} Knuth reserved four bits of the integer part for the sign and other internal METAFONT requirements. If that number is considered as a number of points, that is almost 5 feet or one meter and a half, certainly more than sufficient for drawing any character. The same limitations hold true for METAPOST.

If the instructions given to METAFONT, or to METAPOST, are correct, very rarely are they forced to issue overflow or underflow messages; nevertheless sometimes it happens. One such instance happened when trying to convert the METAFONT description of very large math delimiters into Type 1 PostScript format, since the program that does this conversion runs METAFONT with a pixel density large enough to fit the 1000×1000 PostScript coordinate space.

3 Floating point numbers

A floating point number, for those who are not familiar with this terminology, is a real number written down in a special way and may be thought of as being made up of two parts: the mantissa and the exponent. For example the decimal number 105.234 may be written in the form $0.105234 \cdot 10^3$; 0.105234 has a null integer part, as any other floating point number, so that the significant information is contained in the mantissa "105234" and in the exponent of 10, "3", and these two numbers are all that is needed to identify a floating point number. Since the exponent may be very large in absolute value and the mantissa can contain many digits, the floating point representation can deal with an extremely wide range of numbers.

So, what is the difference between fixed radix and floating point numbers, neglecting their range and coding system? The main and fundamental difference is the precision: fixed radix numbers have a fixed number of fractional digits, leading to a fixed *absolute* precision, while floating point numbers have a fixed number of *significant* digits, leading to an approximately fixed *relative* precision. The minimum distance between two consecutive fixed radix numbers is 2^{-16}, while the minimum distance between two consecutive floating point numbers[2] is 2^{-24} times the base 2 raised to the exponent of the floating point number.

Floating point numbers are essential for engineers, physicists, technologists, etc., whose achievements wouldn't even exist (or at least would be much less impressive) if they had to work with fixed radix numbers. Engineers in particular are the masters of approximation, the "artists" of approximation; engineering theories are based on reasonable simplifications and approximations so as to render the more exact physical theories applicable and useful. As an engineer — did you have any doubt? — I like floating point numbers, even if I appreciate the efforts made by the mathematicians and by Knuth in particular for producing such wonderful masterpieces of the "art of computer programming" given by TEX and METAFONT.

4 New typesetting systems

As every reader of *TUGboat* knows well, there are several teams that are working on "upgrades" of

[2] A word length of 32 bits, the same as with fixed radix numbers, is divided in two parts: 24 bits for the mantissa and 6 bits for the exponent (let me skip the technicalities) plus two bits, one for the sign of the mantissa and one for the sign of the exponent. This means that exponents of the binary base 2 can range from -63 to $+63$. High level languages such as Pascal or C (the two languages that TEX is commonly translated to from its original WEB literate programming language) also support double precision floating point numbers (64 bits) and sometimes even quadruple precision (128 bits); such representations allow for increased precision mantissas and an even larger range for the exponents, but I think they are excessive for our purposes with TEX and METAFONT.

TEX. I am not aware of any work going on for upgrading METAFONT, but maybe I am wrong.

As every reader knows well, Knuth declared some years ago that TEX and METAFONT are frozen; any further modification will simply correct existing errors, but the programs will not be modified so as to add new functionality. But Knuth himself encourages downward compatible new programs that can do things that the actual TEX and METAFONT cannot do.

The very popular pdfTEX offers the opportunity of producing portable document formatted documents directly from a `tex` source; this is not the only new functionality, because pdfTEX improves the already excellent TEX paragraph line breaking algorithm by implementing a hanging punctuation facility and variable width font justification. *TUGboat* readers may have enjoyed the transcript of Hàn Thế́ Thành's PhD thesis on this subject [4].

ε-TEX [6] is another approach to extending TEX: it removes the limitation of 256 counters, lengths, glues, ..., it allows bidirectional typesetting, this feature being very useful for mixing languages such as Latin/Cyrillic/Greek scripts with Hebrew and/or Arabic ones, plus many other improvements, too many to go into a detailed description. I directly asked Phil Taylor, one of the authors of ε-TEX within the $\mathcal{NTS}$ working group, to examine the possibility of extending the program functionality to floating point numbers, but he very humorously refused in a unquestionable way.

Another ongoing project is Ω, by Plaice and Haralambous, together with its companion Λ; the former extends TEX while the latter is the Ω extension of LATEX. Besides removing the limitation of 256 counters, lengths, etc., Ω can deal with Unicode fonts because it can address glyphs in a set of 2^{15} instead of being limited to the 256 characters per font we are all used to. The many other enhancements offered by Ω are described in [7]. A shorter but extremely good description of Ω and Λ is in [8, chap. 10].

Maybe one day all these project extensions of TEX will be merged in the successor of TEX; but as far as I know none of them deals with floating point numbers. Very nasty situation for a fond appreciator of floating point numbers such as myself.

But what I am asking is nothing special: it sums up to introducing the notion of floating point numbers, with the associated `fpcounter`s, and a set of rules for operating on them: (a) the usual arithmetic operations, (b) the trigonometric functions,[3] logarithm, exponential, square root (as they are provided by Pascal and C), (c) a set of rules for mixed mode calculations, and (d) extension of the allowable TEX expressions as they are defined in ε-TEX, although ε-TEX now accepts all numerical quantities except real numbers.

I propose the usual operations on numbers with different operand types, such as: any arithmetic operation between floating point and integer numbers, the result being floating point; truncation of a floating point number to an integer; scaling of lengths by floating point numbers, the result being a length; the assignment of a length to a floating point variable, where the implied fractional separator is taken into account; the same operations on glues (with the same limitations); and so on. Most of these operations are already hardwired in the high level languages, so that their implementation is quite simple; the remaining ones imply operations (multiplication and division) between floating point numbers and fixed radix ones, which should not be a problem at all.

Nothing should be changed in the way this suggested extension to TEX formats paragraphs and pages for at least one obvious reason: it should yield exactly the same results as TEX, in order to be a downward compatible extension producing the same results for existing documents. But the extension to real numbers could produce simpler and more efficient extension packages, especially those that have to deal with graphics, and with PostScript and pdf output.

5 Real numbers and METAFONT

As I explained above, METAFONT already uses real numbers, although they are internally represented with fixed radix ones. I believe that the modification of METAFONT into a new program that uses floating point real numbers could avoid some of the actual limitations on the size of the quantities dealt with, with negligible impact on the shape of the characters being produced.

When Knuth decided to use fixed radix real numbers in METAFONT, computers used to behave in different ways and by the start of the '80s were rather slow compared to modern computers. Back in 1985 my 286 AT computer with an 8 MHz clock needed a few minutes to produce any one of the 128 character fonts of the Computer Modern family.

[3] The default angle unit should be the nonagesimal degree, as in METAFONT, not the radian.

Now my Pentium based laptop requires a few seconds to produce any one of the 256 character fonts of the EC family, although my laptop is rather "old" and runs with a clock of 233 MHz. Back in the '80s integer arithmetic (as the fixed radix representation substantially is) was much faster than floating point arithmetic; modern CPUs have special facilities for floating point arithmetic so that the number of floating point operations per second is not dramatically different from the corresponding number of integer operations as it was in the past.

At the same time the implicit rounding involved in any fixed radix multiplication (and division) implies that results are "never" precise, but always approximate; if you ask METAFONT to show the result of the expression `(1/3)*3` it will display `0.99998`,[4] instead of `1.00000` as shown on page 62 of *The METAFONTbook* [2], together with many other calculations that exhibit the same sort of "error". With floating point numbers the situation would not be any worse, because with 24 bits available for the mantissa, floating point real numbers have from 6 to 7 "precise" *significant* decimal digits (the relative error ranging approximately from $6 \cdot 10^{-8}$ to 10^{-6}).

But in my opinion the opportunity to change METAFONT to floating point arithmetic arises from the strong arguments connected to section 7.

6 Real numbers and METAPOST

METAPOST also works with fixed radix arithmetic (although nothing is said about in the user manual [3]), but since its purpose is to output graphics of different kinds in PostScript format, and since PostScript uses floating point arithmetic, the arguments discussed in section 7 become even more compelling for extending METAPOST to floating point arithmetic.

7 PostScript and floating point arithmetic

The PostScript language uses both integer and real numbers, the latter being operated upon as floating point numbers.

When Knuth wrote TEX and METAFONT, PostScript, and in particular PostScript fonts, were in their infancy. Printers that could interpret the PostScript language were expensive and relatively slow, to the point that when Tom Rokicki wrote his excellent translator from `dvi` to PostScript, he stated that "pk fonts produce better results and run faster on PostScript printers".[5] When Rokicki originally wrote his translator, this was true, in the sense that processors and language interpreters mounted on the PostScript printers and phototypesetters of that age were much slower that modern printers and phototypesetters.

At that time there was no Portable Document Format (pdf), so PostScript output was originally used only for printing. Previewing was made possible by Ghostscript (by Peter Deutsch) and GhostView (by Tim Theisen), respectively a PostScript interpreter intended to be resident on the computer instead of the printer and capable of driving a variety of output devices, and a graphic interface for displaying the PostScript output on the screen with the ability of moving within the virtual document.

Nowadays, pdf has become a standard for moving typeset documents from one computer to another along the infinite routes of the Internet, with the assurance that each document will be viewed, and possibly printed, in the same way on any platform, running under any operating system. The TEX suite has been complemented with the pdfTEX typesetting program [4], with its pdfLATEX variant, with the `hyperref.sty` extension package, so as to make available internal and external cross-linking, and so on.

On the other side the PostScript output produced by `dvips` may also be converted to pdf format with Adobe Distiller, by Ghostscript itself (by specifying the suitable output driver and the other necessary pieces of information), and other tools.

But this is the real point: viewers for pdf documents display bitmapped fonts very poorly. At the same time, pdf documents are to be read (primarily) on the screen, irrespective of the screen pixel density and size. On large screens the reader may want to keep more than one window open, so that resizing windows implies the simultaneous resizing of what is contained in the window itself. Bitmaps are very poor suited to this task, and bitmapped fonts may become almost unreadable even if originally they were produced with high pixel density settings. Printing the documents does not suffer so much from the poor quality of bitmapped fonts on the screen, because the physical page cannot be resized at will.

In other words TEX, LATEX, and other TEX dialects, *have to drop the bitmapped fonts*, as well as pdfTEX, pdfLATEX, etc. All these programs must be capable of using Type 1 PostScript fonts.

[4] With floating point numbers having 24 bits for the mantissa the same calculation would yield `0.99999994` with a relative error of $6 \cdot 10^{-8}$; the fixed radix calculation has a relative error of $2 \cdot 10^{-5}$.

[5] I am quoting by heart, because the last documentation on his translator `dvips`, [9], does not contain this statement any more.

This explains why recently the CTAN archives have seen a variety of package extensions that make available *complete* PostScript fonts to TEX and, especially, to LATEX; I am referring myself to the excellent `txfonts` (Times extended) and `pxfonts` (Palatino extended), that are *complete* in the sense that they contain all the glyphs, ligatures, symbols that are used by LATEX with the `amsmath` extension, plus many more; if the `T1` encoding is specified the font glyphs corresponding to the 256 character TEX encoding are all present, and if the `textcomp` extension package is invoked, the Text Companion glyphs become available. All families, series and shapes are available and are designed in such a way that blend together without the difficulties experienced in the past when the standard PostScript fonts were adapted via several patches (and several omissions) for use with TEX. Another precious set of Type 1 fonts is the collection of the CM-super fonts; these are Type 1 fonts including the Latin and the Cyrillic alphabets, all the math fonts and the Cork encoded fonts. In a near future it should include also my Greek fonts, at least the author is working on it. The Latin Modern font collection is an even more recent development, not to mention the CM-LGC font collection that contains the PostScript versions of the CM Latin, Greek and Cyrillic fonts. Both available in the CTAN archives.

But this trend is going to cut off METAFONT from the TEX community if it does not upgrade to output PostScript format.

This is the real point I would like to stress in order to support my request for floating point arithmetics in TEX, METAFONT, and METAPOST.

8 METAFONT, METAPOST, and Type 1 PostScript fonts

The CTAN archives contain many different fonts designed with METAFONT; such fonts are generally of a very high quality, unless they were just filling the role of patches for supplying certain glyphs not available elsewhere.

It is possible to convert the METAFONT generated fonts to PostScript format in a variety of ways, most of which are also applicable to converting bitmapped fonts obtained by optically scanning original specimens. Some information is available in the paper [18]; more recently Karl Berry has written a detailed paper which appeared in *TUGboat* [10], showing some of what can be done with the software tools available today.

Here I summarize some of these tools and add some of the information I collected either by direct experience or from the documentation. In any case I must warn the reader that most of these programs are native on UNIX or Linux platforms. However, by means of the environment cygwin [11] such programs can be operated also on Win32 platforms (sometimes with minor limitations); therefore, it is not a problem to load and use these programs even if one doesn't have a Unix or Linux platform.

8.1 METAPOST and roex

METAPOST can directly read METAFONT programs and produce PostScript files — but they have to be edited quite a bit in order to make PostScript fonts, and then the result is not of Type 1, but of Type 3. The main difference is that every connected part of each glyph in Type 1 fonts is described by just two contours, the external one and the internal one, while METAFONT, and thus METAPOST, produce the glyphs as a superimposition of several strokes that intersect or join superimposing some of their parts, so that they may only be classified as Type 3. But this limitation can (sometimes) be overcome, so read on:

CTAN holds an extension to METAFONT that allows for redefining the **filldraw** and **stroke** METAFONT commands at **shipout** time, so as to join several strokes into a single stroke with one external and one internal contour. This makes the output suitable for further processing by METAPOST to produce PostScript files that conform with Type 1 format, as described above. The resulting files still have to be edited and merged in order to have a single file describing the whole font. The extension package for removing overlaps and expanding strokes in METAFONT fonts is contained in the file `roex.mf` available from CTAN within a zipped file `roex.zip` that contains other valuable information and examples [19]. The drawback is that this procedure is not suitable for processing existing METAFONT fonts because `roex` is not infallible and some editing of the source code may be required.

8.2 Autotrace

The Internet offers a mighty program, `autotrace` [12] by Martin Weber, together with a graphical front end `frontline` (available from the same home page as `autotrace`) that may be used for determining the contours of arbitrary stroked lines, and therefore of character glyphs. You may refer to Karl Berry's paper [10] for a short description of the possibilities of this software.

The essential point is that `autotrace` can determine the third order Bézier splines that make up the internal and the external contours of every connected part of a glyph, so as to output the node and

control points of such splines, which in turn are the only information PostScript requires for drawing the glyph. The input to `autotrace` must be a bitmap in certain specialized formats; the most likely for our purposes is the 'portable network bitmap' (`.pnm`).

8.3 Pfaedit

The superb interactive graphical font editor `pfaedit` by George Williams [13] supports creating and editing PostScript and TrueType fonts. It is capable of importing bitmaps, including the `gf` format that METAFONT outputs, and of invoking `autotrace` so as to trace them.

By the end of the tracing task the designer already has the contours of each and every glyph already set in its graphical window, so that she can optionally edit the contours and/or simplify the set of nodes; add kerning and ligature information, write the initial PostScript preamble information; and eventually output the font file in `pfa` (printer font ASCII), `pfb` (printer font binary), TrueType, or other formats.

8.4 textrace and mftrace

In spite of the apparent simplicity of the operation with `autotrace` through `frontline` or `pfaedit`, it is even easier and faster to operate through one of two other front-end scripts that make the necessary preliminaries, launch the programs, and output the desired font files.

The Perl script `textrace` [14] can analyze a bitmapped drawing and build up the contours comprising it. It produces the bitmap file in `pnm` format with the necessary scaling for the 1000×1000 PostScript coordinate space, and launches `autotrace` with suitable parameters. It eventually assembles the PostScript font and outputs its file in the desired format. As far as I know, most of the METAFONT fonts now available on the CTAN in PostScript format have been produced by using `textrace` or similar programs. Also the EC fonts, as produced with `textrace`, have been available on the CTAN for some time (but I can't give credit to the author because his/her name is encrypted in his/her e-mail address).

The Python script `mftrace` by Han-Wen Nienhuys [15] was formerly known as `pktrace`; the name was changed to the present one after some interaction with its author who was so kind to introduce some specific extensions so as to allow me to use the program in a cygwin environment without resorting to the teTEX distribution (available with cygwin), but by using the MikTEX distribution that I had on my laptop. `mftrace` is a Python script explicitly designed for translating METAFONT fonts to PostScript format. It starts by generating the `tfm` file, if it's not already available (this operation must be done by hand under cygwin with MikTEX); from this file it computes the magnification to feed to METAFONT in order to scale the generated bitmap in `gf` format to suit the 1000×1000 PostScript coordinate space. It then transforms the `gf` bitmaps to the `pnm` format and feeds such new bitmaps to `autotrace` with suitable parameters. Next, it feeds the generated code to `pfaedit` with suitable parameters and optionally with the specification of an encoding, so as to optionally simplify the contours and to fill up all the necessary information in the font preamble; the output is finally written to a `pfa` or a `pfb` file. Nothing else should be necessary for using the font, because the TEX metric file was available or was explicitly generated during the process, except for adding the font name to a font map that `dvips`, or `pdftex`, or `dvipdfm` can access and read.

8.5 Metafog

Finally there is `metafog`, a program for doing exactly the direct passage from METAFONT to PostScript. It was written by Richard Kinch, who also wrote a paper on the subject [16]. From what I know, it is not in the public domain, in that it is only available by buying Kinch's TrueTEX product.

- * -

I have used `mftrace` and obtained good results in very little time; since I have available a cygwin environment, I can draw fonts with `pfaedit`, or I can do the work with METAFONT and then pass the result to `mftrace`. Being accustomed to METAFONT, I find it more comfortable to use the the latter method, but I always verify the result with `pfaedit` and possibly make some little corrections.

9 Proposal for a PostScript-enabled METAFONT

But if `metafog`, `mftrace`, and the like already do the conversion, why am I pleading in favor of the introduction of floating point arithmetics in TEX and especially in METAFONT and METAPOST? Well, on one side `metafog` is not publicly and freely available as are all the other pieces of software that belong to the TEX distributions. On the other, it's a philosophical position: I find it stupid to start with a beautiful piece of software such as METAFONT, that has so many facilities for finding the contour points and for describing the third order Bézier arcs in order to draw the glyph strokes, to proceed with rasterization that loses all this information even if it

smoothly handles the arbitrary (convex) pen shape, and to try to recover it from the rasterized drawing when you had all the information from the very beginning.

Unfortunately I can't program in a decent way either in Pascal or in C; my knowledge of METAFONT is greater than the average user's, but it is far from perfect. My geometrical competence is nowhere near that of John Hobby, just to mention the author of METAPOST and the contributor of the algorithm for finding the control points of the third order Bézier splines as described by Knuth on pages 130–131 of *The METAFONTbook*. Therefore I cannot give any suggestion to someone who might want to try to work on a New METAFONT.

Such a New METAFONT should be downward compatible with the "old" METAFONT, of course, just as ε-TEX and pdfTEX are downward compatible with TEX. But one of the modes should be a `ps` mode, by which a font `pfb` file is produced together with the usual TEX font metric file. The files produced this way should be quite a bit smaller and the scalable fonts should be drawn in a simpler way, since the splines are generated directly by METAFONT with the minimum number of points; anybody who has designed a font knows well that rarely does a particular glyph require more than a dozen triplets of z points; some simple strokes require just a couple of triplets, because the control points are determined by METAFONT itself. Altogether, most glyphs may be drawn with less than two dozen points for the inner and respectively the outer contour, which is much less than what you get when using `mftrace` or `textrace` or any other interface to `autotrace`.

Not only that, but the New METAFONT might be capable of producing Multiple Master fonts with little or no effort; in fact the Computer Modern fonts by themselves are a collection of several master fonts, `cmr5, cmr6, cmr7, cmr8, cmr9, cmr10, cmr12, cmr14, cmr17`; every stroke is described by the same METAFONT statements, only the dimensional parameters are varied from one master to the other, giving rise to a set of fonts that are not obtained one from the other by simple scaling operations.

The EC fonts by Jörg Knappen do even better: the parameters are defined in a single file as vectors of data in univocal correspondence with the design size. EC fonts have master files that contain in their name the design size, so that `ecr1440.mf` is the master file for the serifed upright medium font and its design size of 14.40 pt is set by a statement contained in this master file. But nothing forbids copying and editing that file into another one, say `ecr3125.mf` where the design size assignment is corrected to 31.25 pt, run it and get the font designed at that very size, without any magnification.

The CB Greek fonts, in this respect, are even simpler, in the sense that the design size is directly determined by the font name, stored in the `jobname` variable, that contains in the last four characters the design size multiplied by 100. Therefore the CB Greek master files have all the same contents, because they need not be edited in order to redefine the design size.

Well, the New METAFONT might be able to read all the font master files starting with the same literal preamble, for example all the `cmr5.mf ... cmr17`, get all the parameters and produce a single Multiple Master Type 1 PostScript font, named, say, `cmr`; the same could be done with the EC fonts or the CB Greek ones.

On this subject there is an extension package, together with its documentation, that already sort of simulates Multiple Master fonts with METAFONT; see [17] for further details.

All these capabilities are already embryonically present in METAFONT and METAPOST: why not to try to develop them and bring them to light? METAFONT and METAPOST are powerful tools for font (and graphic) design, much more powerful than most other extant programs for font design. Why should we, the `texmf` community, give them up in favor of less sophisticated programs?

10 Conclusion

The line of thought of introducing floating point numbers in TEX, METAFONT, and METAPOST carried me further, into discussing the urgent need to think also to a New METAFONT program capable of directly producing Type 1 PostScript fonts, possibly even Multiple Master fonts. In this way it is possible to keep both TEX and METAFONT up to date with the modern necessities of document production, compatible with the Portable Document Format. In fact, by its very nature, it suggests that documents are directly read from the computer screen, a task that today's METAFONT fonts are not particularly good for.

References

[1] Knuth D.E., *The TEXbook*, Volume A of the series "Computers and Typesetting", Addison Wesley, Boston (1986).

[2] Knuth D.E., *The METAFONTbook*, Volume C of the series "Computers and Typesetting", Addison Wesley, Boston (1986).

[3] Hobby J.D., *User manual for METAPOST*, file

mpman.pdf, (1997), in any CTAN repository.

[4] Hàn Thế Thành, *Microtypographic extensions to the TEX typesetting system*,, *TUGboat* 21.4 (2000), pages 317–434.

[5] Hàn Thế Thành, *The pdfTEX user manual*, file pdftexman.pdf, (2000), in any CTAN repository.

[6] Breitenlohner P., *The ε-TEX manual*, file etex_man.pdf, (2000), in any CTAN repository.

[7] Plaice J. and Haralambous Y., *Draft documentation for the Ω system*, file omega-manual.pdf, (1999), in any CTAN repository.

[8] Syropoulos A., *Digital typography Using LATEX*, Springer Verlag, New York 2002.

[9] Rokicki T., *a DVI-to–PostScript translator*, file dvips.dvi, (1997), in any CTAN repository.

[10] Berry K., "Making outline fonts from bitmap images", in *TUGboat*, v. 22 (2001),No. 4, p. 281–285.

[11] cygwin, a Unix-style environment that allows operating a Unix program on a 32-bit Windows platform; available from http://sources.redhat.com/cygwin.

[12] Weber M., autotrace, available from http://autotrace.sourceforge.net.

[13] Wilson G., pfaedit, available from http://pfaedit.sourceforge.net.

[14] Szabó P., textrace, contained in textrace-latest.tar.gz to be found in any CTAN repository or available from http://textrace.sourceforge.net.

[15] Han-Wen Nienhuys, mftrace, available from http://www.cs.uu.nl/~hanwen/mftrace/index.html

[16] Kinch R., "MetaFog: Converting METAFONT shapes to contours", *TUGboat* 16.3 (1995), pages 233–243.

[17] Berdnikof A.S. and Turtia S.B., *mmf.sty: Computer Modern Typefaces as the multiple master fonts*, file mff.dvi, (1997), in any CTAN repository.

[18] Pakin S., *mf2pt1 – Produce PostScript Type 1 fonts from METAFONT source*, file mf2pt1.pdf, (2001), in any CTAN repository.

[19] Jackowski B., Pianowski P. and Ryćko M., "Package for removing overlaps and expanding strokes", file roex.zip, (1998), in /graphics/MF-PS in any CTAN repository.

◇ Claudio Beccari
Politecnico di Torino, Turin, Italy
claudio.beccari@polito.it