Why didn't METAFONT catch on?

Dave Crossland
University of Reading, UK
dave (at) lab6 dot com
http://www.understandinglimited.com

Abstract

METAFONT is an algebraic programming language for describing the shapes of letters, designed and implemented by Knuth as part of the original TEX typesetting system. It was one of the earliest digital type design systems, and is completely capable of dealing with the letters of any writing system, has always been freely available, and is remarkably powerful. Yet it never caught on with type designers. Why?

"There are three kinds of people. Those that can count, and those that can't."

There is type in typography, but there is also type in psychology: Personality type.

There are many ways of thinking about personality type [1] and the famous *Myers–Briggs* typology places importance on four attitudes. First, there is our preference for competition or cooperation, or whether we tend to make decisions logically or emotionally. Second, our use of language reveals the way we think, with some people preferring more abstract language and others preferring more concrete language. Third is our attitude to time keeping, which may be exploratory or scheduling, and fourth is our orientation to socialising, where after a party we may feel drained or energised.

Put together, these four preferences between two options yield 16 personality types. The book *Please Understand Me 2* [2] puts them into a cohesive system that groups the 16 types into four temperaments, fleshed out by labels and personified by Greek gods: Epimethean 'Guardians,' Dionysian 'Artisans,' Apollonian 'Idealists,' and Promethean ~~'TEXXies'~~ 'Rationalists.'

Such broad theories for how people differ probably can't be taken too far, as ultimately people are all pretty much alike; *"what one man can do, another man can do."* But there is a common sentiment that some of us are more abstract in our language and more logical in our thinking than others.

Software is pretty abstract and logical, and people who become immersed in the world of software tend to be of a Promethean temperament. The arguments for software freedom especially have that kind of draw. TEX takes an abstract and logical approach to digital typography, from concept to usage, and METAFONT is no exception. But graphic design is not abstract and logical, for the most part; it is visual, concrete, more emotional than logical.

Throughout the long history of desktop publishing, people have generally not found TEX typesetting intuitive, preferring desktop publishing applications with graphical user interfaces. Even for those who go deep enough into graphic design to arrive at type design, METAFONT is almost entirely ignored — despite being freely available, completely capable, and remarkably powerful. I believe this issue of personality type is a primary reason why METAFONT has not caught on.

Let's consider type design divorced from the engineering of font software for a moment.

Design happens at various scales at the same time. In type design, the lowest visible level is that of the letter, where you are dealing mainly with the black shapes of the letter. It is obvious what those are, but there are also the 'white' shapes. If you are not sure what that means, imagine an image of a letter, and invert it so that the black becomes white and the white becomes black. Now, look around you to find a letter printed large on something like a poster or book cover. Looking at the letter, shift your awareness to the 'negative space' in and around the letter, and bring these white shapes into perceptual focus. It is hard to describe them, but they are there, and designing them is as important as designing the black shapes.

The next level up is that of words. Here there are not only the white shapes inside and around the letters, but those *between* the letters. At this level we can also see patterns in the black shapes *across* letters; things that look similar, yet are not exactly the same.

Consider the lowercase n and h. These contain several similar shapes, but looking closely, you will see that there are slight differences. Balancing these similarities and differences is a core part of the type design process. There are strong patterns in some sets of letters, weaker similarities in other sets, and some letters that are less typical, yet still look like they belong with the rest. The letter s is perhaps the most different, and Knuth wrote an interesting essay about the peculiarities of that letter [3].

Finally, there is the level of paragraphs. When a paragraph is set with a typeface, a different impression of the letters emerges. This must be taken into account at the other levels. Seen in this way, a type design is a collection of individual glyph shapes that fits together cohesively at all levels.

We can now see clearly the subtle distinction between a font and a typeface. The same typeface can be implemented in a variety of typesetting technologies — metal, software, even potato — with the end result appearing the same. A font is a typeface implemented in software. The term 'software' spans programs and data, and fonts are a peculiar kind of software because they are both programs *and* data, while normally the two have some separation. Examples of programs within fonts are TrueType hints and OpenType layout features; these instruct the computer to display the type in various ways. The data in a font is the glyph point data and metrics table data.

There are generally two approaches to implementing typefaces in software. The 'outline' approach involves drawing each letter by interactively placing points along its outline. This attempts to be a direct facsimile of drawing letters on paper. Interpolation between sets of outlines means this approach can handle the creation of large typeface families.

The 'stroke' approach is where each letter is constructed by specifying points along the path of a pen's stroke, and the attributes of the pen's nib at those points. Archetypal pieces can be designed and used like Lego blocks to construct whole glyphs, with refinements made for the individual requirements of each letter. With parametrisation to make the shared values of shapes easily adjustable, such as widening stems or modifying serifs, this approach can handle a large typeface family in a cohesive and powerful way.

Today the outline approach is dominant because it gives instant visual feedback and exacting control; it is direct and visceral. This means designing type at the level of individual letter shapes is intuitive and a typeface emerges quickly.

It is especially suited to implementing existing type designs where all the aspects have already been thought out; the TEX community provides a clear example of this in the *AMS Euler* project [4], where a team of Stanford students attempted to digitise a new type design for mathematics that Zapf had drawn on paper; the developers tried both approaches and felt tracing outlines was most appropriate. FontForge [5] is a vigorously developed free software font editor application for working in this way today.

While not suitable for implementing existing type designs, METAFONT's abstract and logical nature makes it powerful for dealing with type at the level of words. While initially slow, it speeds up later stages of the design process, especially when covering very large character sets. I think it is ideally suited to developing new type designs where the designer is not sure of the precise look that they are trying to capture and want to experiment with a variety of sweeping changes to their design.

TEXworks [6] attempts to make TEX typesetting more visual and interactive. While still abstract and logical compared to desktop publishing applications like Scribus [7], its user interface design and the SyncTEX technology [8] tightly interconnect the code and the document, making TEX more visual, interactive, concrete and emotional.

Today METAFONT source code is written, various programs are run to generate graphics, then another program is used to view them. These programs may be METAFONT, or METAFONT and then `mftrace` [9], or METAPOST [10] with MetaType1 [11]. All involve a whole long process that is similar to writing TEX documents in the traditional manner. But with TEXworks, the TEX source code is rendered into a document in near real-time, so there is a very quick Boyd cycle [12] between adjusting the typesetting code and seeing the document rendered.

Perhaps if there was a graphical user interface to visualise METAFONT code in near real-time, type designers who feel writing code is unintuitive could be more confident about doing so. The simple GUI shown by Sherif & Fahmy in their Arabic design work is an example of this [13]. It might even be feasible to have two-way interaction between code and rendering, as Inkscape [14] achieves for SVG. Perhaps then, METAFONT might catch on.

Dave Crossland is an international public speaker on software freedom and fonts, runs a small business doing type and information design and systems administration, and is a committee member of UK-TUG. He is currently studying at the University of Reading's Department of Typography on the MA Typeface Design programme.

References

[1] `http://en.wikipedia.org/wiki/Category:Personality_typologies`

[2] Keirsey, D. *Please Understand Me II: Temperament, Character, Intelligence*. 1998: Prometheus Nemesis.

[3] Knuth, D. *Digital typography*. 1999: CSLI.

[4] Knuth, D. & Zapf, H. *AMS Euler: A New Typeface for Mathematics*. 1989: Scholarly Publishing.

[5] `http://fontforge.sourceforge.net`

[6] `http://tug.org/texworks/`

[7] `http://www.scribus.net`

[8] `http://itexmac.sourceforge.net/SyncTeX.html`

[9] `http://lilypond.org/mftrace/`

[10] `http://www.tug.org/metapost.html`

[11] `http://www.ctan.org/tex-archive/fonts/utilities/metatype1/`

[12] Osinga, F. *Science, Strategy and War: The Strategic Theory of John Boyd*. 2006: Routledge.

[13] Sherif, A. and Fahmy, H. Meta-designing parameterized Arabic fonts for AlQalam. In this volume, 435–443.

[14] `http://www.inkscape.org`

[15] `http://metafont.tutorial.free.fr`