

Belleek: A Call for METAFONT Revival

Richard J. Kinch

TRUETEX Software

6994 Pebble Beach Ct

Lake Worth, Florida 33467 USA

kinch@truetex.com

http://truetex.com

Abstract

Despite the importance of mathematical typesetting to the persistent popularity of T_EX, very few T_EX math fonts are available to complement the thousands of available text typefaces. Developing fonts is a very different enterprise from programming other software, requiring different tools and different skills, offering little reward in technical innovation, and often requiring a commercial price to justify the effort. To the corpus of public-domain T_EX software, we contribute *Belleek*, a new set of hand-drawn math fonts to complement Times, published simultaneously in METAFONT, Type 1, and TrueType formats, and compatible with the L^AT_EX `mathtime` package. We describe the difficult process of creating such software with a graphical editor, which motivates a second-look at METAFONT as a practical design tool. We examine Hoenig's METAFONT-based *MathKit* software as a paradigm of fitting math fonts to text typefaces, concluding that METAFONT will become practical only when it gains a visual editor for input and outline fonts for output.

These issues should be vital to mathematical publishing, because meta-math fonts will likely be the only economical source for math fonts to complement most of the universe of text typefaces.

Introducing the Belleek Fonts

Figures 1–3 set forth character-set tables and sample uses of the Belleek math fonts for use with Times text. Figure 4 shows various samples of math-mode usage of the fonts. These fonts are herewith contributed to the public domain, with hopes that T_EX will thereby gain some small measure of flexibility to adapt freely to typefaces other than Computer Modern. The shapes were hand-drawn using Fontographer 4.1, some as freehand originals and others by stretching the Computer Modern generic shape to fit the Times weight and aspect ratio. The following constraints and goals guided the crafting of the shapes:

1. The character set, encoding, and metrics must match the three math fonts underlying the L^AT_EX `mathtime.sty` package¹, thus allowing their compatible use with a single `\usepackage{mathtime}` command.
2. All characters must harmonize with the visual style and weight of Times text.

3. The character shapes must be original designs in those cases admitting sufficient latitude for originality, so as to avoid any appearance of infringement of existing proprietary designs.
4. The math symbol shapes (as opposed to Greek letters) should follow the general form of the Computer Modern meta-designs, to the extent possible while still strictly harmonizing with Times. This goal is an experimental excursion into the principle that a meta-typeface might serve as a basis for automatic generation of new math fonts.

Creating Belleek by Hand

Knuth said that success with METAFONT would depend on “collaborative efforts” between “artists and programmers” [Knuth 86, ?]. One wonders if these classes of people ever meet, because despite the earnest hopes of many, the elegant, mathematical and linguistic power of METAFONT has seen little application to font design, and has not been recently used by the commercial type-design industry. Taking in hand once again those beautiful Volumes C and E

¹ Namely, the MathTime fonts `mtex`, `mtsy`, and `rmtmi`.

Γ	Δ	Θ	Λ	Ξ	Π	Σ	Υ	Φ	Ψ	Ω	α	β	γ	δ	ϵ
ζ	η	θ	ι	κ	λ	μ	ν	ξ	π	ρ	σ	τ	υ	ϕ	χ
ψ	ω	ε	ϑ	ϖ	ϱ	ς	φ	\leftarrow	\rightarrow	\dashrightarrow	\dashleftarrow	\leftarrow	\rightarrow	$($	$)$
Γ	Δ	Θ	Λ	Ξ	Π	Σ	Υ	Φ	Ψ	\cdot	$,$	$<$	$/$	$>$	$*$
∂	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O

\perp	\cdot	\times	$*$	\dagger	\diamond	\ddagger	\ddagger	\oplus	\ominus	\otimes	\oslash	\odot	\circ	\bullet	
\succ	\equiv	\sqcup	\sqcup	\leq	\geq	\llcorner	\llcorner	\sim	\approx	\sqcup	\sqcup	\lll	\ggg	\lrcorner	\lrcorner
\leftarrow	\rightarrow	\uparrow	\downarrow	\leftrightarrow	\rightrightarrows	\rightrightarrows	\rightleftarrows								
\simeq	∞	\in	\ni	\triangle	∇	$/$	$-$	\triangleleft	\triangleright	\perp	\emptyset	\Re	\Im	\top	\perp
\approx	\circ	\circ	$+$	$=$	\triangleleft	\triangleleft	$=$	$;$	\prime						

$()$	$[]$	$ $	$ $	$[]$	$[]$	$\{ \}$	$\{ \}$	$\langle \rangle$	$\langle \rangle$	$ $	\equiv	$/$	\backslash		
$()$	$()$	$[]$	$[]$	$[]$	$[]$	$\{ \}$	$\{ \}$	$\langle \rangle$	$\langle \rangle$	$\langle \rangle$	$\langle \rangle$	$/$	\backslash		
$()$	$[]$	$[]$	$[]$	$[]$	$[]$	$\{ \}$	$\{ \}$	$\langle \rangle$	$\langle \rangle$	$/$	\backslash	$/$	\backslash		
$(\backslash$	$[]$	$[]$	$[]$	$ $	$ $	$[]$	$[]$	$\langle \rangle$							
$\backslash)$	$ $	$ $	$\langle \rangle$	$\langle \rangle$	\sqcup	\sqcup	\oint	\oint	\odot	\odot	\oplus	\oplus	\otimes	\otimes	
Σ	Π	\int	\cup	\cap	\oplus	\wedge	\vee	Σ	Π	\int	\cup	\cap	\oplus	\wedge	\vee
Π	Π	$\hat{\quad}$	$\hat{\quad}$	$\hat{\quad}$	$\tilde{\quad}$	$\tilde{\quad}$	$\tilde{\quad}$	$[]$	$[]$	$ $	$ $	$[]$	$[]$	$\{ \}$	$\}$
\surd	\surd	\surd	\surd	\surd	$ $	$[]$	\equiv	\uparrow	\downarrow	\prime	\prime	\prime	\prime	\uparrow	\downarrow

Figure 3: Belleek Math Extension (b1ex) 10 pt at 600 dots/inch.

$$\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}\right)|\varphi(x + iy)|^2 = 0$$

$$\pi(n) = \sum_{m=2}^n \left[\left(\sum_{k=1}^{m-1} \lfloor (m/k) / \lceil m/k \rceil \rfloor \right)^{-1} \right].$$

$$p_1(n) = \lim_{m \rightarrow \infty} \sum_{v=0}^{\infty} (1 - \cos^{2m}(v!^n \pi/n)).$$

Let H be a Hilbert space, C a closed bounded convex subset of H , T a nonexpansive self map of C . Suppose that as $n \rightarrow \infty$, $a_{n,k} \rightarrow 0$ for each k , and $\gamma_n = \sum_{k=0}^{\infty} (a_{n,k+1} - a_{n,k})^+ \rightarrow 0$. Then for each x in C , $A_n x = \sum_{k=0}^{\infty} a_{n,k} T^k x$ converges weakly to a fixed point of T .

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix}$$

$$\prod_{j \geq 0} \left(\sum_{k \geq 0} a_{jk} z^k \right) = \sum_{n \geq 0} z^n \left(\sum_{\substack{k_0, k_1, \dots \geq 0 \\ k_0 + k_1 + \dots = n}} a_{0k_0} a_{1k_1} \dots \right).$$

$$\Pi_R \begin{bmatrix} a_1, a_2, \dots, a_M \\ b_1, b_2, \dots, b_N \end{bmatrix} = \prod_{n=0}^R \frac{(1 - q^{a_1+n})(1 - q^{a_2+n}) \dots (1 - q^{a_M+n})}{(1 - q^{b_1+n})(1 - q^{b_2+n}) \dots (1 - q^{b_N+n})}.$$

$$\int_{-\infty}^{\infty} e^{-x \cdot x} dx = \sqrt{\pi}$$

$$X = \sum_i \zeta^i \frac{\partial}{\partial x^i} + \sum_j x^j \frac{\partial}{\partial \dot{x}^j}$$

Figure 4: Math mode samples (after *The T_EXbook*) using Belleek and Times New Roman, 10 pt at 600 dots/inch.

of *Computers and Typesetting*, and observing the erudition therein, one wonders how such magnificent engines have seen such little use. Have METAFONT and Computer Modern become museum pieces, like some polished-brass steam engine, impressive to look at, but long since superseded by higher-powered technology? Or were they perhaps ahead of their time, and not yet harnessed to their potential to create?

Whatever its virtues as a field of human endeavor, working with type as software seems to stifle one's yearning to abstract and perfect a physical enterprise in mathematical form. Instead, it seems to stir the passions for raw, un-parameterized, bare-handed manipulation of perimeters. You want to grab a shape like a piece of hefty rope, not tweeze some bits of code.

Thus fonts are today drawn using direct-manipulation, CAA (computer-aided agony) tools that somewhat speed the brutish task of digitizing and refining outlines. Big publishers have in-house software, and the small-time designers have GUI software such as *FontLab*, *Fontographer*, and *Type Designer*.

This author is a true believer in languages as a means to use computers. In respect of font design this could hardly be better implemented than through METAFONT. Yet when it came to the practical problem of creating a few fonts in the shortest time, even these near-absolute principles fell to the expedience of the GUI tools. It is indeed faster to just click and drag, just not to be recommended as a steady job, if you value your sanity. The T_EX world would have also been better off with a meta-version instead of a merely-specific version.

As miserable as creating font shapes is, the need to hint fonts for low-resolution use is even more so. Hinting is like undertaking. One applies grisly techniques to preserve the corpse from decay in the presence of trying conditions. Success is achieved when the casual observer comments on how natural the result looks. The Belleek fonts are auto-hinted (let us not carry the grim metaphor any further). Perhaps someone will have the ghoulish expertise to do a proper job on them.

Creating Math Fonts Automatically

Alan Hoenig has taken the right approach in his *MathKit* [Hoenig 98] and *MathInst* [Hoenig 98] packages. In principle we should be able to program (say, in METAFONT) meta-characters for math symbols, and fit them automatically (with, say, METAFONT) to a given text typeface. He

exhibits successful applications of this principle, instantiating the Computer Modern character programs with hand-measured characterizations (x-height, stem widths, etc.) of a few typefaces, such as Times, Baskerville, Jenson, and Caslon. Knuth's ancient (in software years) Computer Modern math symbols seem to have been endowed with a sufficient amount of meta-ness to cover a range of target typeface styles. Where meta-qualities are lacking in Computer Modern math, the METAFONT programs can be upgraded.

So if there is great demand for T_EX to typeset math in anything-but-Computer-Modern fonts, why has *MathKit* not received popular acceptance? It is not due to any shortcoming in the results, but rather to the utter mess that T_EX (and moreso L^AT_EX) have made of changing fonts, encoding, and styles. What a user wants is not a programming kit containing dozens of components for designing new fonts, but a single command that says "I mean to use Goudy, so please just make it so." Instead, the *MathKit* approach requires an almost super-human expertise in T_EX, PERL, NFSS, and not a few other sophisticated tools. This is not to criticize *MathKit* for being overly obscure; in examining its implementation one must admire the economy and efficiency it displays. The source of the complexity is just the nature of the instantiation task.

The simplification of this daunting complexity is not as simple as gathering the output of *MathKit* for a given typeface into a ready-to-run package, creating a little archive that the user can drop in the TDS tree. Because *MathKit* in part depends on METAFONT to rasterize fonts, *MathKit* must necessarily impose several layers of scripts and programs to guarantee that, for example, METAFONT can generate bitmaps for the fonts in the sizes eventually called-up in the user's document. METAFONT seems to be the dowdy aunt who is welcomed at first, but then doesn't know when to leave.

Minimizing the user's task requires something more, namely conversion of the *MathKit* instantiation of the meta-math fonts into scalable outlines. This reduces the components of a new style to a few outline font files, a few T_EX virtual fonts, and some T_EX or L^AT_EX macros, all in a ready-to-run distribution. The key, therefore, is the ability to convert METAFONT designs to outlines.

METAFONT: The Flaw

If the ability to convert METAFONT designs to outlines is key, why has it been lacking? Indeed, we reach a startling conclusion: METAFONT is fundamentally flawed, and this flaw has inhibited its acceptance as a font-design tool, namely:

*METAFONT should produce outlines,
not bitmaps, as output!*

Confirming this assertion is the software-tools philosophy. METAFONT is at heart a language for the expression of mathematically abstract shapes. As a properly demarcated tool, METAFONT should convert that abstraction form to another abstraction, and do no more and no less. Knuth's problem with METAFONT in 1982 was that he, his students, and colleagues were unable to practically solve the computational-geometry problem of converting stroked elliptical pens to outlines, and of overlapping shapes to outlines. Instead he relied upon rasterization as an expediency [Knuth 86].

Also confirming this assertion is the existence and popularity of METAPOST [Hobby 89], which converts METAFONT code to PostScript code, something closer to (although still not quite) outlines. METAPOST works by intercepting METAFONT's internal data structures before they are bitmaps. In essence it is taking the proper output from METAFONT and expressing it in an intermediate form using the more primitive PostScript language.

METAFONT: The Redemption

If METAFONT should produce outlines, but Knuth sidestepped the problem, then what are we to do? Ask him to try again, but harder? There would seem to be two routes to getting outlines, instead of bitmaps, from METAFONT:

Outlines from Overlapping Shapes

METAPOST converts METAFONT code to PostScript code, expressing the same overlapping shapes in a more primitive geometric form. MetaFog [Kinch 95] first exhibited a practical solution to the further task of reducing overlapping, stroked PostScript shapes to non-overlapping outlines. We are tantalizingly close to a "MetaFog 2" that completes the theoretical solution and implements it in robust form. This would allow fast and complete conversion of METAFONT code to non-overlapping outlines, such as are required for outline font formats.

Proper Outlines from Curve-Fitting Polygons or Bitmaps

The MetaFog research attempts to solve a generalization of an already-solved problem, that of removing overlaps in polygons or bitmaps. If we convert the overlapping METAFONT shapes to polygons or bitmaps, then we can apply well-understood algorithms to compute the equivalent non-overlapping polygons or bitmaps. Indeed, in the bitmap domain we have merely described what METAFONT now does, namely, it computes a single bitmap resulting from the rasterization of any number of overlapping shapes.

If we consider a polygon (or bitmap) as a digital sampling of an underlying analog shape, then it would appear consistent with sampling theory that the band-limited analog shape underlying the polygon (or bitmap) should be recoverable, given that we have sufficient resolution and absence of noise in the polygon coordinates (or bitmap pixels). This "given" is assured in the case of METAFONT, since we can scale its output noiselessly to any desired resolution.

While there are many published algorithms for practical curve-fitting of bitmap edges ("autotracing") [Schneider 90], none attempts the possibility of recovering the exact mathematical curves underlying a noiseless rasterization such as METAFONT generates. (The typical application tries to fit approximating curves to a noisy scan of an irregular physical object). In this matter, this author is again tantalizingly close to a curve-fitter that will solve the problem and implement it in robust form. Among other wonderful applications, this would allow conversion of METAFONT shapes to outlines by "mere" curve-fitting.

Other Meta-Design Formats

Besides METAFONT, there are other formats for specifying some degree of meta-design to typefaces, such as the multiple master extension to Type 1. But none of these match the potent ability of METAFONT to express meta-ness in far more sophisticated ways than mere linear interpolation. Linear interpolation may be sufficient for a limited range of variation, such as stem weight, slant, or even the presence or absence of serifs. But the non-linear and programmatic possibilities of METAFONT provide a much wider range of possible variations; and still more powerful is METAFONT's ability to stroke and overlap shapes.

On the other hand, some anecdotal experience has resulted in failures when attempting

satisfactory METAFONT designs [Siegel 90]. One lesson from such experience is that a single meta-character is not necessarily able to represent wildly different characteristics, particularly as might vary in letters. In the case of nearly all non-letter math symbols, however, one can expect that the possibilities of variation are restricted enough to permit meta-characterization to a degree sufficient to cover a wide range of text typefaces. It might be necessary to produce differing math meta-characters for serif versus sans-serif typefaces, or other gross variations; indeed this was the effect of many of Knuth's conditionals in Computer Modern.

The Future: T_EX, and the Web

The future of math publication, whether in T_EX or on the Web, will depend in part on the variety of math fonts available. It would appear inevitable that math fonts will always lag seriously behind text fonts, if creation of quality math fonts necessarily involves manual design. The present tools for meta-font design suffer from a fundamental flaw in that they cannot produce parametric output, only bitmaps. This approach is impractical, if for no other reason than it necessarily involves intractable complications for users, who cannot be expected to deal with the vagaries of bitmapped fonts.

The tasks of defining font encodings, building a symbol inventory, designing meta-math programs, and writing style-switching code are all substantial, yet well-understood. None of those problems will ultimately impede the adoptions of T_EX, HTML, or any other markup language. They involve complicated details which can be managed by the experts and well-hidden from the user.

It is therefore our conclusion that:

- Quality math meta-fonts will be crucial to success in math publishing, because hand-drawn shapes are too costly.
- Implementing quality meta-fonts reduces to two fundamental problems:
 - A language for meta-design, which we believe is superbly extant in the METAFONT language.
 - A processor for that language which can produce non-overlapping outlines. This requires solving one of two open research problems: topological analysis of stroked, overlapping shapes; and exact curve-fitting of arbitrary-resolution rasterizations.

References

- [Hobby 89] Hobby, John D. "A METAFONT-like System with PostScript Output." *TUGboat* **10** (4), pp. 505–512, 1989. See also the software on CTAN.
- [Hoening 98] Hoening, Alan. "Hundreds of New Math Fonts with MathKit (version 0.7)." CTAN `fonts/utilities/mathkit`.
- [Hoening 98] Hoening, Alan. "The MathInst Package (version 0.8): New Math Fonts for T_EX." CTAN `fonts/utilities/mathinst`.
- [Kinch 95] Kinch, Richard J. "MetaFog: converting METAFONT shapes to contours." *TUGboat* **16** (3), pp. 233–243, 1995. Current version available with TRUET_EX.
- [Knuth 86] Knuth, Donald E. *The METAFONTbook*. Addison Wesley, Reading MA, 1986.
- [Knuth 86] Knuth, Donald E. *METAFONT: The Program*, Section 524, "Elliptical Pens". Addison Wesley, Reading MA, 1986.
- [Schneider 90] Schneider, Philip J. "An algorithm for automatically fitting digitized curves." In *Graphics Gems*, Andrew S. Glassner, editor, pp. 612–626 and 797–807. Academic Press, Cambridge MA, 1990. See also the on-line archives at <http://www.acm.org>.
- [Siegel 90] ("Illuminating little booklet" by Dave Siegel on negative experience attempting to metafont-ize the Euler font at Stanford, cited by Berthold Horn in `news://comp.fonts` ca. Aug 1997.)