Automatic Generation of Math Fonts for TEX

Charles Duan
Harvard University
Cambridge, Massachusetts
USA
\switchemailcduan@eecs.harvard.edu

Abstract

It is easy to use different fonts with TEX, and writers of technical documents enjoy this typographic flexibility. Unfortunately, there are few full math font sets available, and the current practice of mixing incompatible text and math fonts produces undesirable results. We describe a system that can automatically produce a set of math fonts immediately suitable for typesetting with TEX, such that those math fonts are compatible with a given text font. This font generator is designed to run automatically and without any technical knowledge on the part of the end-user. Using this system, a wide variety of mock math fonts can be generated, even for sans-serif fonts, and to great precision.

Introduction

Users of TEX are fans of high-quality typesetting and fine typography, and fans of fine typography enjoy the use of well-chosen, beautiful fonts for their documents. Digital fonts are available in great quantity and high quality, but these fonts nearly exclusively provide only the Roman alphabet. Of particular interest to the current audience, there are only about five fonts complete with mathematical symbols suitable for use with TEX [2]. What is the poor TEXnician to do, given this abundance of text fonts and corresponding void of mathematical symbols?

The most common option is to mix unrelated text and mathematics fonts. It is all too common to see *published* technical articles set in Times Roman text but Computer Modern math. However, these unrelated fonts are often *visually incompatible*: dominant visual features of the two fonts are inconsistent, and the reader is jarred and distracted by the visual discontinuity. With regard to Times Roman and Computer Modern, the former has thicker stems, less prominent serifs, and shorter capital letters than the latter.

Naturally, the optimal solution would be for all of us to hire type designers to make new, compatible math fonts for us on demand. Lacking the necessary resources to do so, it seems that an *automated approach to the creation of compatible math fonts* is worth investigation.

MathKit: A First Solution

Alan Hoenig provided a first stab at a solution with *MathKit* [3]. His system took advantage of one key observation: the Computer Modern font programs, developed by Knuth [4], are made up of sixty-two parameters that determine various visual elements of the fonts they generate, so with the right set of parameters, a math font can be *automatically created* to match a given text font, and using virtual fonts, the math symbols can be combined with the text font's italic letter set to produce a highly compatible set of math fonts for use with TEX or LATEX.

While MathKit is sufficient to generate compatible math fonts for many text fonts, it has three main drawbacks. First, in order to create math fonts as compatible with an arbitrary text font as possible, it is necessary to measure many minute aspects of the text font. MathKit provides three parameter sets (for Times Roman, Palatino, and Baskerville), but for other fonts the user must create the parameter set alone—a task that, in the present author's experience, involves learning the METAFONT language and carefully studying the Computer Modern font programs in order to achieve good precision.

Second, even with a great deal of knowledge, several of the font parameters used in the Computer Modern programs are difficult to measure by hand. For example, several parameters define the amount of curvature of round letters; those parameters are hard for a person to measure but fairly easy for a computer to measure. (Naturally, there are also plenty of parameters that are easier for a human

to measure than a computer.) MathKit deals with these values by not measuring them at all, simply taking the values out of Computer Modern's standard parameter sets.

Third, a quality math font also requires proper spacing the characters (e.g., how much extra blank space to put on the left and right of each character). MathKit provides a mechanism for setting these spacing values, but it makes no attempt to set them automatically. Without proper spacing values, the letters will rarely, if ever, be spaced correctly. However, given more than fifty-two letters to adjust, the process of spacing adjustment is tedious, time-consuming, and in general an unpleasant exercise.

Motivation for a New System

What we would like is a system for which a user without any formal knowledge of typography or programming in general, let alone the specific implementations of METAFONT and the Computer Modern programs, can produce math fonts reasonably compatible with a given text font. The user should merely have to provide the necessary font files, execute a program or two, and have a ready-to-use collection of math symbols that match well with the text.

Naturally, an automatic system for generating aesthetically pleasing shapes can do no better than an expert human designer, and it may not even surpass a reasonably knowledgable non-expert. However, the calculated values should at least provide a ballpark estimate. The automatic output should be acceptable for use, and the determined user's task should be one of fine-tuning, adding and subtracting small increments to already reasonable values, rather than one of guessing values until the output looks reasonable.

Design and Implementation

The process of automatic math font generation requires two steps: measurement and generation. In the measurement phase, the existing text font is analyzed, producing a set of numerical (or other) parameters that will be fed to the generation phase. In the generation phase, the set of parameters is applied to font generation programs (in this case, a modified version of the Computer Modern font programs) to create new math fonts.

The Measurement Phase We perform measurements on fonts in the Adobe Type 1 format. There are two good reasons for this: first, Type 1 fonts are well-supported within TEX [5]; second, they are natively understood in the PostScript language, so it is easy to perform analyses on them [1].

The measurement routines are divided into two levels. First, a library of "measurement primitives" is given, providing simple functions such as finding the intersection of a character's outline with a straight line or finding the highest point in a character. Second, a set of measurement routines is built out of that library, using those primitive actions to calculate the desired parameter. For example, to measure the upright stem width of uppercase letters, the program will draw a horizontal line 50% through the height of the letter I, find the (hopefully two) intersections between the line and the character outline, and finally find the distance between those intersection points. For each of (most of) the sixty-two parameters, one or more routines is used to measure that parameter value (and multiple measurements are averaged according to complex rules). Some parameters are also calculated based on several measurements.

A thought that occurred to the author after he was more experienced in METAFONT (and had already written the measurement routines) was that many of the primitives he had written were built-in functions in METAFONT already. Thus, another way of performing the measurements would be to translate the fonts into METAFONT (there are programs to do this) and then perform measurements there. It may be worthwhile translating the programs in this way if only for the speed increase (the built-in METAFONT primitives would be much faster than the coded PostScript ones).

The Generation Phase Once those parameters are calculated, they are passed to the font generation programs. In most cases this is essentially like running MathKit with the parameter set calculated from the measurement phase. However, there are two differences.

First, many alterations have been made to the original Computer Modern font programs. Many new parameters have been added to increase compatibility. Also, several of the character designs, most notably β, γ, and δ, have been changed to permit for a wider parameter space. Additionally, Knuth observes that the math symbol programs lack the "metaness" of the roman letters; that is, much of their appearance is hard-coded rather than determined by parameters. Several parameters have been introduced to provide some more "metaness."

Second, the measurement phase also provides measurements for proper letter spacing. The font

generation process takes into account those spacing measurements.

Range of Font Support

For which fonts can we automatically generate acceptable math fonts using this procedure? In general, any font that looks relatively like Computer Modern will produce a visually compatible math font. Additionally, support has been extended to sans serif fonts. All of the standard Adobe printer fonts are supported: Times Roman, Palatino, Bookman, New Century Schoolbook, Helvetica, Avant Garde, and Courier. Many other fonts have been tested successfully; some samples will be given below.

Fonts that will not work generally have one or more of the following qualities:

- They contain unexpected strokes or splotches of ink (e.g., a blackletter font with two strokes for a capital I)
- They have inconsistent stroke widths or character heights, or characters that do not line up with the baseline (e.g., a very unusual handwriting font)
- They tend to be used for display purposes instead of setting long passages of text

Luckily, people tend not to want or need math fonts for unusual display typefaces, so for any reasonable text font this program should be successful.

Samples

Below are several samples of generated math fonts. Please take the appearance seriously, but not the content!

Helvetica demonstrates a number of features of the new fonts. First, notice the new design of the δ character, whose top is flat rather than curved. Also, the loops of the β and other characters are squared off, as are many of the terminals of letters. This "squareness" option is provided to the user; fonts can be generated with either round or square ends.

Times Roman displays a number of interesting features. The serifs are very thin and short, as opposed to the relatively thick, prominent serifs of Computer Modern. The generated Λ and Γ characters capture this, as do the summation and product signs.

Optima is an interesting sans serif display font. The generated math characters properly capture the stem width contrast and elegant flares. However, it is also noteworthy what *cannot* be captured: the minute flaring and slightly curved terminals of the stems (compare the letters A and Λ). Qualities that are not captured tend to be so small that they do not show up at small text sizes, so for normal text (where mathematical symbols would most likely be used) these minor insufficiencies are unnoticeable.

Future Work

In its current state, as exemplified by the samples above, high-quality math symbol fonts can be produced for a wide variety of text fonts, and those symbols typeset with quality comparable to hand-tuned characters. However, there are several areas of potential and useful improvement.

First, the algorithms for performing spacing are somewhat naïve, measuring only a sample of points and not taking into account any prior knowledge of the shapes of characters. Better algorithms could be developed with some investigation.

Second, the algorithm for calculating spacing could also be applied to generating kerning pairs (fine adjustments to spacing between certain pairs of characters).

Third, additional math fonts could be generated. MathKit provides blackboard bold and Fraktur fonts; it should be possible to incorporate those as well.

Fourth, the math fonts are currently generated in bitmap form (as all METAFONT programs generate) and then traced using an outline tracing program to produce Type 1 fonts. A better method would be to use METAPOST and various associated utilities to produce Type 1 fonts without the tracing. This would require rewriting the requisite METAFONT font generation programs a great deal, but in theory the output fonts would be more efficient and appear better on digital displays.

Finally, although the Computer Modern programs are sufficient for generating very compatible math fonts, the resulting fonts tend not to have certain distinctive features of the original text font. For instance, the small hooks or serifs at the top and bottom of the stroke of the italic i are very distinctive for every font; such features should, optimally, be duplicated as best as possible.

There are two good arguments *against* duplicating such distinctive features. First, those features are carefully crafted to the individual letter, so a computer will not be able to successfully imitate them by some sort of generic transformation. Second, the math symbols ought to look unique anyway, so they might as well have different visual features.

Regardless of the camp in which you fall, there are two possible lines of work stemming from this observation. First, new font generation programs need

A Random Sample of Mathematical Typesetting

Charles Duan

May 25, 2004

Let α be a variable such that $\alpha \geq \alpha$ and $\alpha \leq \alpha$. There exists some β such that either $\alpha = \beta$ or $\alpha \neq \beta$, that is:

$$\forall \alpha \, \exists \beta : \alpha = \beta \vee \alpha \neq \beta$$

Consider vectors $\vec{\nu} = (\alpha, \ldots, \beta)$ and $\vec{\nu} = \nu \times \nu$. We wish to find some value Λ such that:

$$\Lambda = \pi \int_0^\infty \nu \cdot v \, d\theta$$

Applying the Γ transformation:

$$\Lambda = \sum_{i=0}^{\infty} \frac{\nu}{c\theta}$$

for some constant c.

We know that one of γ and δ is true. Applying a logical reduction:

$$\begin{aligned} \gamma \wedge \delta &\implies \gamma \wedge \delta \wedge \omega \\ &\implies \frac{\gamma \wedge \delta}{\omega'} \vee \neg\epsilon \\ &\implies \bot \end{aligned}$$

It then must logically follow that μ reduces to:

$$\ln \left[\lim_{z \to 0} \left(1 + \frac{1}{z} \right)^z \right] + \left(\sin^2(x) + \cos^2(x) \right) = \sum_{n=0}^{\infty} \frac{\cosh(y) \sqrt{1 - \tanh^2(y)}}{2^n}$$

revealing that $f^2 = g^2$.

A Random Sample of Mathematical Typesetting

Charles Duan

May 25, 2004

Let α be a variable such that $\alpha \geq \alpha$ and $\alpha \leq \alpha$. There exists some β such that either $\alpha = \beta$ or $\alpha \neq \beta$, that is:

$$\forall\alpha \, \exists\beta : \alpha = \beta \vee \alpha \neq \beta$$

Consider vectors $\vec{\nu} = (\alpha, \ldots, \beta)$ and $\vec{\nu} = \nu \times \nu$. We wish to find some value Λ such that:

$$\Lambda = \pi \int_0^\infty \nu \cdot \nu \, d\theta$$

Applying the Γ transformation:

$$\Lambda = \sum_{i=0}^{\infty} \frac{\nu}{c\theta}$$

for some constant c.

We know that one of γ and δ is true. Applying a logical reduction:

$$\begin{aligned} \gamma \wedge \delta &\implies \gamma \wedge \delta \wedge \omega \\ &\implies \frac{\gamma \wedge \delta}{\omega'} \vee \neg\epsilon \\ &\implies \bot \end{aligned}$$

It then must logically follow that μ reduces to:

$$\ln\left[\lim_{z\to 0}\left(1+\frac{1}{z}\right)^z\right] + \left(\sin^2(x) + \cos^2(x)\right) = \sum_{n=0}^{\infty} \frac{\cosh(y)\sqrt{1-\tanh^2(y)}}{2^n}$$

revealing that $f^2 = g^2$.

A Random Sample of Mathematical Typesetting

Charles Duan

May 25, 2004

Let α be a variable such that $\alpha \geq \alpha$ and $\alpha \leq \alpha$. There exists some β such that either $\alpha = \beta$ or $\alpha \neq \beta$, that is:

$$\forall \alpha \, \exists \beta : \alpha = \beta \vee \alpha \neq \beta$$

Consider vectors $\vec{\nu} = (\alpha, \ldots, \beta)$ and $\vec{v} = \nu \times \nu$. We wish to find some value Λ such that:

$$\Lambda = \pi \int_0^\infty \nu \cdot v \, d\theta$$

Applying the Γ transformation:

$$\Lambda = \sum_{i=0}^{\infty} \frac{\nu}{c\theta}$$

for some constant c.

We know that one of γ and δ is true. Applying a logical reduction:

$$\begin{aligned} \gamma \wedge \delta &\implies \gamma \wedge \delta \wedge \omega \\ &\implies \frac{\gamma \wedge \delta}{\omega'} \vee \neg\epsilon \\ &\implies \bot \end{aligned}$$

It then must logically follow that μ reduces to:

$$\ln \left[\lim_{z \to 0} \left(1 + \frac{1}{z} \right)^z \right] + \left(\sin^2(x) + \cos^2(x) \right) = \sum_{n=0}^{\infty} \frac{\cosh(y) \sqrt{1 - \tanh^2(y)}}{2^n}$$

revealing that $f^2 = g^2$.

to be written. Computer Modern was designed in a "modern" style; many of the features of the modern type style (e.g. strong vertical stress; the lowercase "o" is symmetrical) are simply hard-coded into the font programs. It would be interesting to develop a set of old-style font generation programs, for example, one based on the Garamond fonts, but taking the same parameters for font design variation. The end user could then choose whether to make a Computer-Modern-style mock math font or a Garamond-style one; in both cases the visual weight, character height, and other dominant aspects would match well, but the style would differ.

Second, the current parameters are all numeric, with semantic meanings of lengths, distances, or ratios of lengths or distances. What about a parameter specifying a subpath of a character? For example, a "parameter" could be the outline of the upper hook of the italic n; that hook design could then be adapted for generated characters such as η.

Conclusion

In an ideal world, every font designer would produce a full complement of math fonts for every text font designed. In practice, the use of mathematical symbols is sufficiently limited that math fonts are rarely designed. The automatic generation of math symbol fonts by computerized measurements presents an ideal alternative, producing high-quality math fonts without great difficulty.

Possibly the most interesting revelation of this work is the fact that aesthetically pleasing math fonts can be produced by simple, static measurement algorithms. In particular, it is not necessary to employ artificial intelligence methods, even though the ultimate goal of the system is to produce output comparable to what a human would produce. In a sense, this mirrors a major contribution of TEX itself: that aesthetics can, in many cases, be discretized, quantified, and transformed into simple, deterministic algorithms.

References

[1] Adobe Systems Incorporated. *The PostScript Language Reference Manual*. Addison-Wesley Publishing Company, third edition, 1999.

[2] Thierry Bouche. Diversity in math fonts. *TUGboat*, 19(2):120–134, June 1998.

[3] Alan Hoenig. *MathKit*: Alternatives to Computer Modern Mathematics. *TUGboat*, 20(3):282–289, September 1999.

[4] Donald Erwin Knuth. *Computers & Typesetting*, volume E — Computer Modern Typefaces. Addison-Wesley Publishing Company, 1986.

[5] Philipp Lehman. The font installation guide. April 2004.