MathGen: Automatic Generation of Mathematical Symbol Fonts

Charles Duan
cduan@fas.harvard.edu

1 INTRODUCTION

The digital age has brought great advances to the world of typography. Typefaces are cheaply and conveniently available, produced in high quantity and high quality. However, the bulk of available typefaces are oriented toward the typesetting of plain text in roman characters. Authors with need for a wider range of symbols, in particular mathematicians and scientists who regularly use Greek letters and other symbols in written works, have only a limited set of fonts with all the necessary symbols from which to choose.

In the recent past, the traditional solution was to mix and match: a high-quality text font was chosen by the author, and the mathematical symbols were drawn from whatever font was available. The most common example of this is the proliferation of documents set in Times Roman, but using the Computer Modern set of math symbols. This creates inconsistent typography: the symbols have substantially different visual attributes compared with the text. In the case of Times Roman and Computer Modern, the Times Roman characters are substantially heavier, they tend to have less prominent serifs, and the capital letters are shorter than the Computer Modern symbols. Such inconsistencies create visual distractions and abberations, decreasing the overall appearance and readability of the text.

The optimal solution would be for an author to choose whatever text font he/she desired and then have a compatible set of math symbol fonts to use in conjunction with the text. However, we are not all font designers, so it is necessary for those compatible symbols to be *automatically generated* by inspection of the text font. This is the goal of MathGen: to automatically inspect a text font and then produce a high-quality set of math symbol fonts that are visually compatible with the original text font.

2 PREVIOUS WORK

The original concept of an adjustable font was developed by Donald Knuth in his METAFONT program and the Computer Modern typefaces designed with it [3, 2]. In those programs, the Computer Modern font was defined in terms of sixty-two changeable parameters, each of which specifies an attribute of the font, such as the thickness of letter stems and the height of lowercase letters (the x-height). In these fonts, Knuth implicitly makes two reasonable assertions. First, he asserts that all fonts carry a basic "shape": certain stems by convention will be thick and other stems thin, certain characters will be the same height as others, and so on. For example, in the letter "M," the leftmost stem is always thinner than the rightmost stem. Second, the aspects of character shapes that can change will change in uniform ways: if the thickness of the stem of the lowercase letter "l" is increased, then the thickness of the stems of other letters such as "m," "q," and "k" should increase as well.

These two assertions are not merely true for the fonts that can be generated from the Computer Modern programs. In fact, observation of many text fonts reveals that, in most ways, text fonts (that is, those designed for high readability over long passages of text, as opposed to those designed to have some unusual or idiosyncratic appearance) adhere to the general character shapes and uniformities that those sixty-two parameters and the Computer Modern programs define.

Because text fonts adhere to many of those general assumptions about character design encapsulated in the Computer Modern fonts, it should seem reasonable that, by measuring out the appropriate values for those sixty-two parameters, one could use the Computer Modern font programs to generate new sets of characters for any font. This idea was first put to use by Alan Hoenig in *MathKit* [1]. In MathKit, the user would measure each of the parameter values (by hand, with a ruler or on-screen calipers) and then execute a series of scripts that would generate the new math symbols and then combine them with italic letters to create a unified, compatible math font.

MathKit's approach will in fact produce good-quality math fonts with the appropriate amount of work; in many situations the output of MathGen will be practically the same as that from MathKit. There are several substantial differences, though. First, MathKit requires the end user to perform the measurements by hand. This is a difficult task, not because of the difficulty of actually performing the measurements, but because the user does not know what to measure. Although MathKit advertises that the user need not know METAFONT, the meaning of the parameters is written only in the first chapter *Computers & Typesetting*, Vol. E, and even there the parameters are not fully defined. The only way to understand what exactly to measure is by reading through the METAFONT programs to understand how the values are used. This requires not only learning METAFONT but also spending many hours interpreting the programs, which, as Knuth observes, are "actual 'optimized' code" rather than "straightforward... textbook examples." It is the author's personal experience that this is an unreasonable demand.

Second, even if the user is technically skilled enough and has sufficient time to perform the measurements by hand, there are some that are simply very difficult to actually perform. For instance, the *beak_darkness* parameter measures the "thickness of beaks," or the amount of ink used in filling the portion between the horizontal arm and its terminal serif of letters like the capital "E." Measuring this parameter requires a tricky construction of a triangle superimposed on the serif and then figuring out the fraction of the triangle filled by the serif's internal curvature. A computer can easily do the necessary construction, but it is fairly hard to do by hand. Even more difficult are the *superness* and *superpull* parameters, which measure the rate of curvature of circular letters. In MathKit the user is not even permitted to change these and other such values, although experimentation has shown that even small changes to these hard-to-measure numbers can produce substantially different results.

For these reasons it seems that an *automated* method of performing the necessary measurements is strongly preferable to measurement by hand. The question, then, becomes how to actually analyze a font's digitized form and produce an accurate and complete set of measurements.

3 AUTOMATED MEASUREMENT OF TEXT FONTS

The task of producing an automated mechanism for measuring qualities of text fonts is separated into two tasks. First, a library of simple routines for assessing general qualities of the shapes of characters is needed. That library should provide all of the necessary functions for one to perform measurements on the characters in a font. Second, we must use that library to write routines to measure the necessary values for our font generation system (i.e., the sixty-two parameters that define the Computer Modern fonts).

We begin by discussing the requirements and implementation of the general measurement library. We then consider how that library was used in measuring several parameters from a font.

3.1 Requirements for the Library

The goal of the measurement library is to take a common digital representation of a font and to perform programmable analyisis on it. In MathGen, the font representation of choice is the PostScript Type 1 format. This format is chosen because it is natively supported by the PostScript programming language, so it is not necessary to write or acquire routines for interpreting the font. As a natural consequence, the library and the routines written with it are all done in the PostScript language.

The Type 1 font format represents characters as outlines, series of conjoined curves and lines that define the edges of the character. The PostScript language provides a method for traversing and processing this outline, so the measurement routines must use this outline for inferring qualities of the characters. As a result, the library must offer mechanisms for identifying certain points on the outline, such as local extrema and intersections with other lines. Judicious identification of such points and their locations should allow the calculation of the necessary font parameters.

3.2 Functions Provided by the Library

The library provides functions for determining several properties of a given character outline. The more important functions are described below.

Intersections It is often useful to identify the intersection of a character's outline with a straight line. A routine is provided to find all the intersections between a line segment and the character, and the user of the function may then pick the relevant intersection by sorting the list of identified points.

Extremes It is also useful to identify extreme points on the curve (highest, lowest, furthest left or right). A general routine returns all local extrema on the curve; the user can identify the proper one by sorting through the list.

Subpaths Often it is desirable to perform an operation on only a small portion of the character outline rather than the entire outline. The subpath operation allows the user to select a portion of the outline and then perform operations on it alone. The subpath may be defined by a starting and ending point; the user can also specify a subpath to end when the direction in which the path is traveling meets a certain condition (e.g., the subpath should stop before it starts traveling straight upward).

Point Operations A general set of "convenience functions" are provided for adding, subtracting, scaling, averaging, and otherwise manipulating two-dimensional coordinates.

Line Operations Functions are also given that allow the user to draw various lines by specification, such as lines traveling through a point with a given angle. Such operations are useful for constructing lines to be fed into the path intersection routine above.

Iteration Although this is not specific to the library (the PostScript language provides iteration constructs), iteration is useful for performing, say, several measurements and then taking the minimum value found.

The above operations are, in fact, sufficient to perform reasonably accurate measurements of all the parameters for the Computer Modern programs; they also allow us to perform other measurements such as character spacing and accent placement, discussed later.

3.3 Parameter Measurements

Using the functions provided by the measurement library we can measure many of the parameters of the Computer Modern font. Here we will consider a few of these measurement algorithms.

The *stem* parameter specifies the width of upright stems in lowercase letters such as l, d, and t. That parameter's value can be easily measured by drawing a horizontal line halfway through the letter l, finding the (hopefully two) intersections between the line and the character outline, and finding the distance between those two intersections.

Figure 1 Measurement of the stem width from the lowercase letter l. The line $\overline{\alpha\beta}$ is drawn halfway up the height of the letter, and intersections γ and δ are found. The stem width is the length $\gamma\delta$.

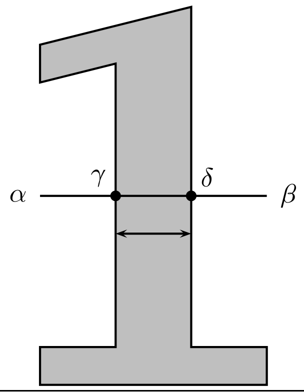
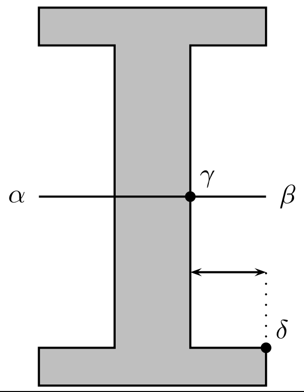


Figure 2 Measurement of *cap_jut* from an uppercase I. The line $\overline{\alpha\beta}$ is drawn halfway through the letter, and point γ is found. Point δ is located as the point on the extreme right of the letter. The jut, then, is calculated as $y_\delta - y_\gamma$.

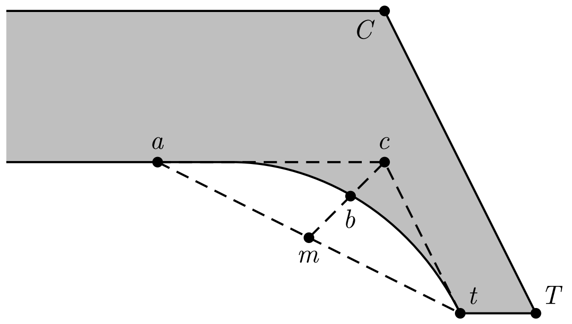


The *cap_jut* parameter specifies the distance by which serifs extend horizontally from the main character stem of uppercase letters. This is measured by drawing a horizontal line halfway up the capital I, finding the rightmost intersection of that line with the character outline, finding the rightmost point of the entire outline, and then subtracting to get the overall horizontal distance between those points.

The use of the uppercase I in measuring the *cap_jut* raises an interesting question: why measure the jut of the uppercase I, rather than that of, say, an M or E, for which the horizontal serif jut can be just as easily measured. The answer is found by looking at the predominant use of that specific parameter in the math font programs: the *cap_jut* parameter is used in the uppercase Greek letters Γ, Λ, Π, ϒ, Φ, and Ψ. Of these, the serifs on Γ and Λ require special treatment (as shown by experimentation), and the serifs on ϒ, Φ, and Ψ are most like those of an uppercase I. For the Π character, the serifs should actually look like those on a capital H, so a separate measurement is taken and applied only for the serifs on the Π character. The problem of the capital serif jut is discussed further in Section 5.1.

The *thin_join* parameter specifies the stroke thickness at the "join" locations between arches and upright stems on letters such as h, m, and n. Since it is not a simple task to analytically find the distance across a stroke given only the stroke's outline, we find the stroke's width by sampling many possible intersections. First we trace a subpath consisting of the underside of the arch of the letter. Next we find a

Figure 3 Measurement of the *beak_darkness* parameter. The solid line represents the outline of the beak serif; the dotted lines are constructed.



point near the top of the join location (at the notch at the upper left of the letter n, for example). Then a series of lines are drawn starting from that point and at different angles toward the lower right of the letter. For each line drawn, we find the intersection of the line with the subpath formed above and thus the distance between that intersection point and the join location point. The minimum distance is taken to be the stroke thickness and reported as the *thin_join* value.

Finally, the *beak_darkness* parameter, which measures the amount of curvature under beak serifs like the one terminating the top stroke of the letter F, provides an example of a relatively complex measurement routine. The explanation will reference the points in Figure 3. The measurement construction is based on the subroutine for drawing beaks, in *C & T* Vol. E, page 376 [2].

First, the points T, C, and t are determined by identifying extrema on the beak serif; this is done without great difficulty. Point a is defined as the point on the path whose x-coordinate is halfway between T and the left end of the beak. Point c is defined such that $\overline{ac}$ is horizontal and $\overline{ct} \parallel \overline{CT}$. Point m is then defined as the midpoint of $\overline{at}$, and b is the intersection of the character outline with line segment $\overline{mc}$. The *beak_darkness* parameter, then, is:

$$beak\_darkness = \frac{bc}{mc}$$

These and many other routines are used to measure the values of parameters for the generation of the math symbol fonts.

3.4 Assumptions about Text Fonts

Because MathGen attempts to distill the overall shape of a font into the sixty-two parameters defined by the Computer Modern programs, we must make several assumptions about the nature of letter shapes. These assumptions fall into two categories: assumptions that when violated, will break the program or otherwise cause erroneous measurements, and assumptions that, when violated, will allow the program to run successfully but will produce strange-looking or incompatible fonts.

In general, the more "standard" a font looks, the more successful MathGen will be, where "standard" is defined as Computer Modern. So long as the overall shapes of the letters are much like those of Computer Modern—serifs in the right places, thick strokes and thin strokes of uniform widths and in the same relative locations, character height uniformity, and so on—the measurements will be accurate and the resulting fonts will look correct.

The following are the major assumptions about the font; namely, those that must be true for MathGen to successfully run.

Serifs Naturally, fonts may have serifs, but the serifs must occur in the same places as they do in Computer Modern. The serifs may have a wide variety of shapes, but they must not be too large (e.g., if the serifs at the ends of the crossbar of a T should be less than 50% of the character's height). Additionally, serifs *must* be present on certain letters such as I and l (otherwise, the font is sans serif, and it should be denoted as so when running MathGen). Finally, mere flares at the ends of strokes (e.g., in Optima) are not considered serifs. Serifs must constitute a rapid change in stroke direction or width.

Strokes Letters should have no additional strokes beyond those that appear in the letters of Computer Modern. In particular, large swashes, loops, or unexpected splotches of ink (placed intentionally by the font designer, of course) will disrupt the measurements. For example, the letter I is assumed to have a single vertical stroke, and the letter O is assumed to consist of a single, closed, circular stroke. Fonts such as Linotext (see the samples at back) fail to meet this assumption.

Stroke Uniformity Strokes are assumed to have relatively uniform width and direction. For example, some display fonts, intending to imitate the appearance of old typewriters, place "pockmarks" or small holes in the otherwise straight stems of letters. These unexpected distortions will cause MathGen to not properly interpret the overall shape of characters.

Straightness Strokes that are normally straight should be as straight as possible. In particular, the capital I should be perfectly vertical, and the sides of the letters A and V should not be curved.

The following assumptions must hold for the output to look acceptable.

Inter-Character Uniformity The thickness of upright stems, curved strokes, and thin hairlines should be uniform for different characters of the font. Additionally, if two regions have the same thickness in the Computer Modern font, then in general, those two regions should have the same thickness in the text font being measured.

Italic and Roman Similarity The italic and roman fonts should be as similar as possible in terms of stroke widths, character heights and widths, and overall appearance. Since the parameters are measured out of the roman font, the resulting math characters will be as compatible as possible with the roman text; if the italic is not compatible with the roman, then the italic will not be compatible with the math fonts either.

Stroke Width Uniformity MathGen does not (yet) provide for simple, straight strokes of non-uniform width. For fonts where strokes tend to flare at the ends (e.g., Optima), MathGen will be unable to capture this change in stroke thickness.

Standard Serifs The Computer Modern programs provide for two types of serifs: horizontal serifs, which attach to vertical stems, and vertical beak serifs, such as those at the ends of the arms of E, F, and L. Six parameters define the horizontal serifs (ten in MathGen) and five define the beak serifs. Serif designs that do not fall within the space of these parameters cannot be perfectly duplicated by MathGen. For example, in Palatino, the beak serifs jut inward toward the letter and are slightly curved. MathGen can capture the inward jut but not the curvature of the serif.

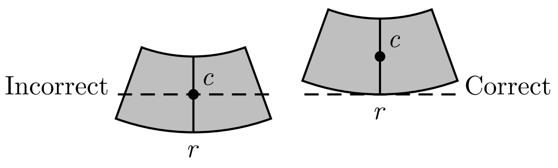
Although these assumptions may seem austere, practically all standard text fonts will follow these rules; the exceptions are generally unusual display fonts in which one would not often typeset mathematics. Additionally, MathGen has been designed to be as robust as possible, correcting egregious dimensional errors and handling unusual character shapes as gracefully, if not accurately, as possible.

4 CHANGES TO THE COMPUTER MODERN PROGRAMS

Although the programs that generate the Computer Modern fonts are touted as producing high-quality fonts for any reasonable set of parameters, experimentation has shown that, in particular with the mathematical and Greek symbols, much work was necessary to make this claim true. Since the goal of MathGen is to produce compatible math fonts for a wide range of typefaces, the original font programs had to be improved substantially in several areas.

Simplest among these changes were several bug fixes to errors in the font programs. However, many of the changes were design decisions, altering the overall look of the font to be more compatible with certain classes of text fonts. Finally, additional parameters were introduced, some to ensure greater compatibility with the text font (in which cases the parameters were measured) and others to allow the user greater control over the generated font's appearance (in which cases the parameters are specified by the end user).

Figure 4 Example of the general bug found in Computer Modern. The bottom edge of the gray stroke (point r) should line up with the dotted line. Instead, the programs specify the center of that stroke (point c) to align there.



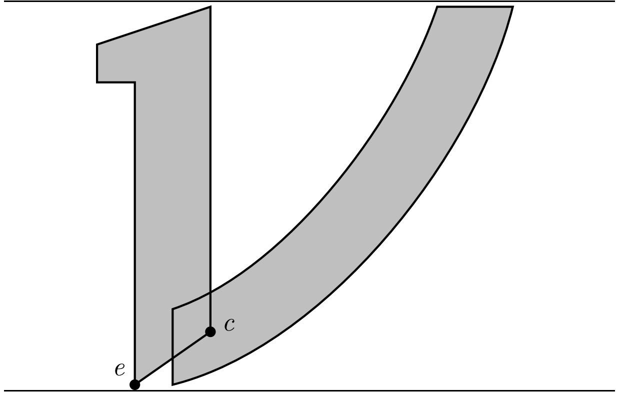
4.1 Bugs in the Computer Modern Fonts

During the process of reviewing the Computer Modern math fonts to make changes, I noticed several errors. They have since been reported to the proper authorities, and I am awaiting their response.

In METAFONT, a "stroke" can be drawn by defining the left and right edges around points along that stroke and then tracing through those edges. The locations of those points are usually defined by the extremities of those edges (e.g., the left edge of this stroke should have this x-coordinate). In several places in the program, the edge of a stroke is defined by the wrong side of the stroke. For example, in the ϕ character, the bottom and top of the circular bowl are placed with respect to the middle of the circular stroke rather than the stroke's edge. The same occurs in the β and δ. Also, the left upward stroke of the β is placed with respect to its right edge, but the code clearly implies that the placement should be with respect to the left edge.

Finally, there is a bug that I haven't found a definitive way to fix. The character ν is made up of a vertical stem and a diagonal stroke. The bottom of the

Figure 5 Example of the bug with the ν character in Computer Modern. The error is with the small notch in the lower left corner of the character. Note that, because the diagonal stroke is curved but $\overline{ec}$ is straight, there is no satisfactory location of c.



diagonal stroke is cut off so that it does not extend below the diagonal stroke, but it is cut off too much, so in certain conditions an unsightly “notch” will appear at the base of the character.

The reason that none of these bugs are problematic in the actual incarnation of the Computer Modern fonts is that the thin stroke widths are so small that the errors are barely visible. For fonts where the thin strokes are not so thin, such as sans serif fonts like Helvetica, these problems become much more obvious and troubling without correction.

4.2 Changes to Character Designs

In addition to many overarching changes to the general implementation of the Computer Modern character programs, a few specific design changes were made. These changes allowed for better character appearance under different parameterizations or simply better appearance in the opinion of the author.

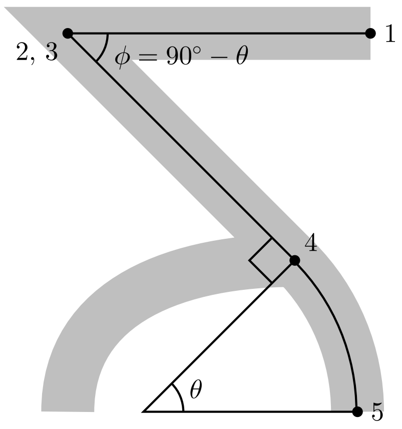
Under the second category of changes, the left stem of the β and the right upward stem of the γ were thickened. In high-contrast fonts, these characters tended to appear too light in comparison with the rest of the text. Additionally, it seems that someone (probably the American Mathematical Society) decided to slightly thin even the thicker areas of the β in the current programs; these areas were thickened to Knuth’s original values in *Computers & Typesetting*, Vol. E. Also, the thickness of the ϕ character was increased slightly. The ϕ should appear much like the letter “o,” but the cross-stroke would make it look too dark. As a result, Knuth counteracted the excess darkness by widening the character and thinning the thicker edges. It is the author’s opinion that only one of these changes is necessary, so the width of the character was left untouched but the extra thinning was undone.

Three characters were changed to improve their robustness under parametrization. The simplest was the κ character, whose upper diagonal was modified in two ways. First, in fonts with square ends (Section 5), the upper diagonal was made straight without a flare at the end. Second, the stroke was programmed in such a way that when the diagonal was relatively thick, the middle of the stroke might become too thin. This error was fixed by defining the control points on that stroke more robustly.

In the Υ character, the thickness of the two branches extending out of the central stem was originally defined as 3/5 the thickness of the central stem. For monowidth fonts such as Courier, where all of the strokes have the same thickness, this caused those two strokes to appear too thin compared to the rest of the text. This was fixed by redefining the branch width as 3/5 of the way between the font’s thinnest stroke width and the central stem’s width.

Finally, the δ character required significant revision. The upper hook of the character is defined by five points: the rightmost end of the hook (point 1), the top of the hook (point 2), the leftmost point on the hook (where it is traveling downward: point 3), the point at which the hook intersects the top of the lower loop (point 4), and the vertical tangent point at the right side of the loop (point 5). If the stroke widths at points 1, 2, and 3 are all about equal, then the hook becomes too bulky and some of the curves

Figure 6 Construction of the δ character. Points 1, 2, 3, and 5 are fixed, and the y-coordinate of 4 is fixed. The x-coordinate of 4 is chosen such that $\phi + \theta = 90^\circ$, so that the line from 3 to 4 is tangent to the circular arc between 4 and 5. The light gray line represents the outline of the character.



are positioned incorrectly.

Looking at other Greek fonts revealed that, for fonts with uniform stroke width, the upper hook was not drawn as a curve but rather as a straight horizontal stroke and then a diagonal line leading into the loop of the δ. The program for generating the character was then modified so, if the stroke width at point 3 was close to that at point 1, the character would take this shape. In the case of a square-end font, the upper horizontal stroke was squared off; otherwise it curved toward the diagonal stroke.

The next question, then, became how to ensure that the diagonal stroke entered into the loop. Experimentation showed that it was most aesthetically pleasing for the diagonal stroke of the hook to be perfectly straight, and it was also best for the part of the loop between points 4 and 5 to be as circular as possible. Points 3 and 5 were fixed by the character's design already. So the program was written to solve for the optimal location of point 4 such that, if a line were drawn from point 3 to point 4 and a circle through points 4 and 5, the angle of the line would be the same as the angle of the circle at point 4. The construction is shown in Figure 6.

In general, this calculation chooses a point about 60% of the way through the width of the character. In a font with very small x-height, such as Futura, this point is much closer to the right edge of the character; for a font with large x-height, such as Avant Garde, this point is closer to the center of the character's width. The generated δ characters for Futura and Avant Garde both look acceptable, showing that this placement calculation is robust for different parameterizations.

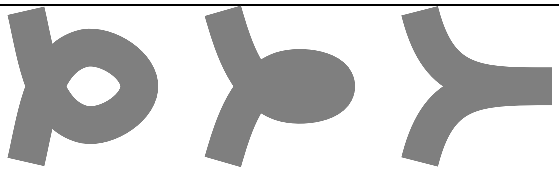
4.3 Loops in Characters

One of the larger changes to the font programs was the modification to small loops in characters. Loops appear in the β, ζ, and ξ characters; vertical loops occur in the γ and ω characters.

In Computer Modern, the hairline stroke width is small enough that an entire loop can be drawn, leaving some empty space in the middle of the loop. In other fonts such as Helvetica this is not the case. Because the edges of the loop cannot overlap (otherwise the stroke filling routine would cause an error), the Computer Modern programs originally just thinned the stroke width until the strokes of the loop no longer overlapped. This was unsatisfactory for monowidth fonts like Helvetica; the unexpected truncation of the strokes caused those letters with loops to look undesirably thin.

To solve this problem, an entirely new loop drawing routine was installed. If the edges of the loop did not

Figure 7 Three different styles of loops used in the Computer Modern Greek fonts. The square loop will be chosen if the *square_ends* parameter is chosen. Otherwise, if the thickness of the loop strokes is less than the desired loop width, an open, round loop will be made. If the strokes would be too thick, a closed loop will be made.



Open, round loop — Closed, round loop — Square loop

collide, then a loop was drawn as normal. Otherwise, the loop was treated as just an ellipse to be filled in, and the incoming strokes that would have made up the loop retained their widths and just terminated within the filled ellipse.

Additionally, modifications were made for fonts with square ends rather than round ones. In that case, loops were always closed so that the edges collided, and a rectangle was drawn rather than an ellipse. This made the "loops" appear stylistically much more consistent with other characters in the typeface.

As examples, the Helvetica and Palatino fonts use loops with square ends. The Didot and Tiffany fonts have round, open loops, and New Century Schoolbook demonstrates round, closed loops.

5 SQUARE STROKE ENDS

The Computer Modern fonts terminate many strokes with a circular cap. This does not match well with fonts like Helvetica, in which letters always are cut off squarely.

For many of these letters, the circular cap is drawn by a routine named *circ_stroke*, so a first attempt at implementing square ends was to simply modify this routine. This worked reasonably well, but the exact positioning of the rounded end was often incorrect. In particular, rounded ends of vertical strokes would often "overshoot" the baseline or the character's intended height by a small amount to counteract the optical illusion that strokes with curved ends appear shorter than strokes with square ends. As a result, each stroke ending with a round end was manually repositioned and changed to a square end. The results of this transformation can be seen in the Palatino font as well as most of the sans serif fonts. For example, the vertical strokes of the κ and μ letters are normally rounded; in square-end fonts the strokes are squared off and slightly lowered (in relation to the baseline and character height).

More complicated were diagonal circular strokes, such as the shorter arm of the λ. In this case it was necessary to calculate the effective width of the stroke at a horizontal cutoff. If a stroke is to appear to have width ω at an angle θ from the horizontal, then the horizontal cutoff width is:

$$\omega_{\text{horizontal}} = \frac{\omega}{|\sin\theta|}$$

This formula was used for the diagonal strokes of λ and π as well as many of the math symbols.

Other changes were made to the designs of the fonts. For example, loops on letters such as γ and ε were stylistically made square, as discussed in Section 4.3. Also, the hook of the delta was modified in certain cases to appear more square.

Finally, many of the math symbols originally terminated in rounded ends (e.g. $+$, $\times$, $\approx$, $\in$). In particular, the equals and minus signs' strokes were rounded. This was problematic because symbols such as the long right arrows ($\longrightarrow$, $\Longrightarrow$) are made up of an equals sign and an arrow pulled closely together. If the strokes making up those symbols (defined by the *rule_thickness* parameter) was too large, then the diameter of the circular tip might have exceeded the overlap between the abutting symbols, and consequently the long arrows appeared to have small gaps.

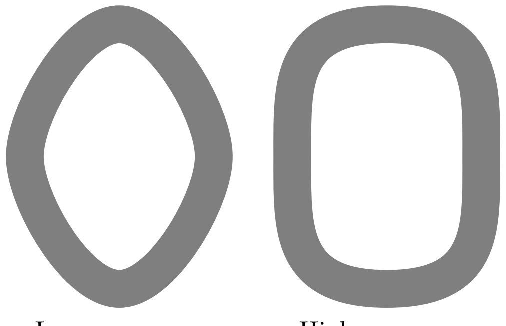
To resolve this problem, the equals and minus signs were redrawn with square ends. However, these two symbols then looked inconsistent compared to the other math symbols, so all of them were redrawn with square ends. At normal text sizes, this change is barely noticeable, but the change is significant at high resolutions or large sizes.

5.1 New Parameters Added

The Computer Modern fonts were built out of sixty-two parameters defining various dimensions and qualities of the letters. Naturally, even the most standard-looking text fonts do not conform with the constraints of these parameters, so in developing compatible math fonts it was necessary to add in new parameters.

Arm Thickness The thickness of the central bar in the Π character and the arms in the Σ and Γ characters were originally equivalent to the thickness of serifs. This produced undesirably thin lines in fonts like Times Roman, where the serifs are very thin but the arms are substantially thicker. The thickness of the Π bar was made equal to the thickness of the "T" crossbar, and the arms were made the same thickness as the upper arm in the letter "F."

Figure 8 The *superness* parameter. Low values will result in diamond-shaped bowls; high values will create square-shaped bowls. The value ranges from 0 to 1; a perfectly circular bowl is created with a *superness* values of $\sqrt{2}$.



Low *superness* High *superness*

Diagonal Serif Bracket The *bracket* parameter defines the amount of curvature between serifs and stems. This parameter is usually measured for a thick, vertical stem. When a serif is attached to a thin or diagonal stem, such as in the letters A or Λ, the amount of curvature must be adjusted. However, the adjustment is not parametrized, so for certain fonts the resulting curvature can appear fairly ridiculous.

Luckily, the only serifs that show this problem are in the Λ character, so a separate *bracket* parameter is measured based on the serifs of the letter A. This provides a very accurate measurement, and the serifs are much more compatible.

Superness of O The *superness* parameter determines how round or square circular arcs will appear, as explained in Figure 8. For the letter "O," the superness value is often different from the rest of the font, primarily because the letter is so round that it requires special treatment. As a result, the superness of the O is measured separately from the general superness value for the font, and that special value is used only for the Θ character.

Capital Serif Jut The amount by which serifs extend from the main stem of uppercase Greek letters is encapsulated in Computer Modern by a single parameter, the *cap_jut*. In most fonts, the serif jut varies depending on the character's shape. As a result, for the uppercase Greek letters, four different serif jut values are measured and applied to different characters. For the Γ character's bottom serif, the length is determined by the bottom serif of the uppercase F. For the Λ character the serif jut is determined by the capital A. For Π it is determined by H, and the others (Υ, Φ, Ψ) are determined by I.

5.2 Scaling Large Characters

Mathematics typesetting requires characters such as parentheses and summation signs in sizes much larger than the normal text. Such large characters should appear thicker than their normal-sized counterparts. How should the relative thicknesses of such characters be determined?

The original Computer Modern programs increased the size of large characters by adding ad-hoc dimensions that just happened to be the right sizes. A *dw* parameter is used as a small width increment in many of these characters. That parameter is defined as the difference in width between the font's *curve* parameter (the thickness of the sides of o) and *stem* parameter (thickness of l). This relation is completely arbitrary, and with any font other than Computer Modern it produces very strangely-sized large characters.

Instead, MathGen provides new "zoom" parameters. The idea behind these parameters is that the thickness of large characters should be proportional to their size, but not directly proportional. A character that is twice the height of another should be thicker but not twice as thick. The zoom parameters specify a fraction of the size increase that should contribute to the increased thickness. Say, for example, that one character has height α and another has height β; let the zoom parameter be ζ. Furthermore, assume that some feature of the character at size α has width ω_α. Then that feature for the character of size β will have width:

$$\omega_\beta = \omega_\alpha \cdot \left(1 + \zeta\left(\frac{\beta}{\alpha} - 1\right)\right)$$

If $\zeta = 0$, then the thicknesses of characters will be constant regardless of size; if $\zeta = 1$, then character thickness will scale linearly with size.

Two zoom parameters are provided. The first controls the increased thickness of delimiters (parentheses, brackets, braces, etc.); this is set by default at 0.25. The second controls the increased thickness of display operators (summation, integral, etc.); this is set at 0.55. Because these values cannot be measured out of just the roman font (for which all characters are about the same size), they must be specified by the user, although the default values seem to work well in most cases. Note that the text-size operators (summation, integral) are not zoomed at all, as they should appear compatible with the rest of the text.

6 CHARACTER WIDTH & ACCENT PLACEMENT

Simply creating a set of mathematical symbols and then inserting the italic letters as variables is not sufficient for high-quality math typesetting. The italic letter variables must also be properly fitted and spaced, and accents must be placed at their appropriate locations with those variables. The symbols generated from the Computer Modern fonts already contain these necessary width and placement adjustments, but the variables, taken from a text font without the necessary mathematical placement dimensions, need to be measured for proper placement and spacing.

Using our library of character measurement routines, we can implement a simple algorithm for measuring the appropriate spacing and accent location for characters. In the following sections we describe those algorithms; in the font samples we show the accuracy of the measurements.

6.1 Character Widths and Math Fitting

Characters in math mode require three parameters for proper fitting: the position of the left edge of the character, the position of the right edge of the character, and the position of the lower right edge where subscripts will be placed. These positionings are known as "sidebearings."

These measurements could be carried out naïvely by simply measuring the extremities of characters and placing the sidebearings at those extremities. This often does not result in good values. A character whose extremities are very thin serifs, for example, should have its edges defined exactly at those serif points. However, a character such as an "o" or an upright, sans serif "l," should have some extra space at the sides, or else adjacent characters will appear too close.

Our algorithm for the measurement of sidebearings is as follows. First, the absolute extremities of a character are measured. For the subscript positioning, only the lower portion of the character is considered. The sidebearings are required to be at least outside these absolute extremities.

Second, a field of n horizontal lines is drawn across the character, and the leftmost intersection of each line with the character is recorded. Those intersections are sorted by x-coordinate, producing a list of sorted x-coordinates $(x_1, x_2, x_3, \ldots)$. Some small unit of width w and an iteration maximum N is chosen. The sidebearing is then constrained by the following rule, where s is the position of the left sidebearing:

$$w - x_i \geq wi, \quad 1 \leq i \leq N$$

The right sidebearing and subscript positions are positioned using an analogous calculation.

If many points on the edge of the character are near the extreme left edge of the character, then the sidebearing will be pushed further and further away from the character. But if the extreme edge of the character is a thin serif, then most of the points will already be far away from the sidebearing position, and the sidebearing will end up right next to the serif.

This algorithm contains three parameters, n, w, and N. A higher value of n means greater accuracy in measurement but a longer computation time (since more points on the character are tested). A higher value of w means that the sidebearing will tend to be pushed further away from the character. And the value of N can be used to enforce a "maximum distance" of the sidebearing from the character, as the sidebearing will be no further than wN from the edge of the character.

This algorithm demonstrates the versatility of the simple functions provided by the character measurement library. An even more complex scheme, also supported by the library, would be to divide the character's edge into small segments and find the extreme points on each segment, rather than simply looking at sampled points on the edge of the character. This should provide an even more accurate positioning.

Although it has not been implemented, a variation of this algorithm could be used for determining kerning pairs between letters (the amount of horizontal space to be added or removed between two characters for optimal fitting). Although the results would not be perfect, it would give a good first impression of kerning values between characters without requiring weeks of manual tuning.

6.2 Placement of Accents

Another use of the measurement library is in determining the placement of accents on the tops of characters. It is often not satisfactory to simply place accents at the center of the character, in particular for characters with ascenders (lowercase "b," "d") and slanted or italic letters.

To measure the proper accent positioning, we draw a line at about 90% of the height of the character and find its intersections with that character's outline. We then take the extreme left and extreme right intersection and average them, taking that to be the approximate center of the character. The accent is placed directly above that average point, skewed to compensate for the italic slant.

This method works well for most characters, but does not work if the letter has an open bowl or a left stem but no corresponding right (e.g., "c," "G", "E"). For those characters, we simply take the middle point of the letter and place the accent directly above it (compensating for italic slant, again). For the letter "T," the measurement is taken at 50% of the letter height, so the accent is placed directly over the stem of the T.

For letters like "b," "d," "h," and "L," the accent is placed directly over the stem of the character. It is questionable whether or not this is desired behavior. On the one hand, the accent is skewed very far to one side, so it is obviously off-center. On the other hand, placing the accent directly above center would place it over blank space, which looks strange. The author's decision was to place the accents off-center but not over blank space; if one should desire the opposite effect, it would not be difficult to rewrite the routine using the functions of the measurement library to do so.,

The resulting sidebearings and accent locations are not perfect. Many of the characters are placed too tightly together (in particular, the right edge of the lowercase c is too small), and some of the accents are placed incorrectly (in particular, the accent is too far to the left on the f in Palatino). However, hand-tuning these values is not difficult, and with the automatically measured values as a starting point, the job is made much easier for the end user.

7 NUMERICAL ANALYSIS

Because the measured parameters have generally the same semantic meanings as the parameters for the original Computer Modern programs, it is possible to compare the measured values with the hand-chosen ones to see how accurately, from a numerical standpoint, the generated fonts should match. We can do so with the original Computer Modern fonts, as those parameters are readily available. Additionally, the parameter sets provided by Alan Hoenig's *MathKit* provide us with more comparison values.

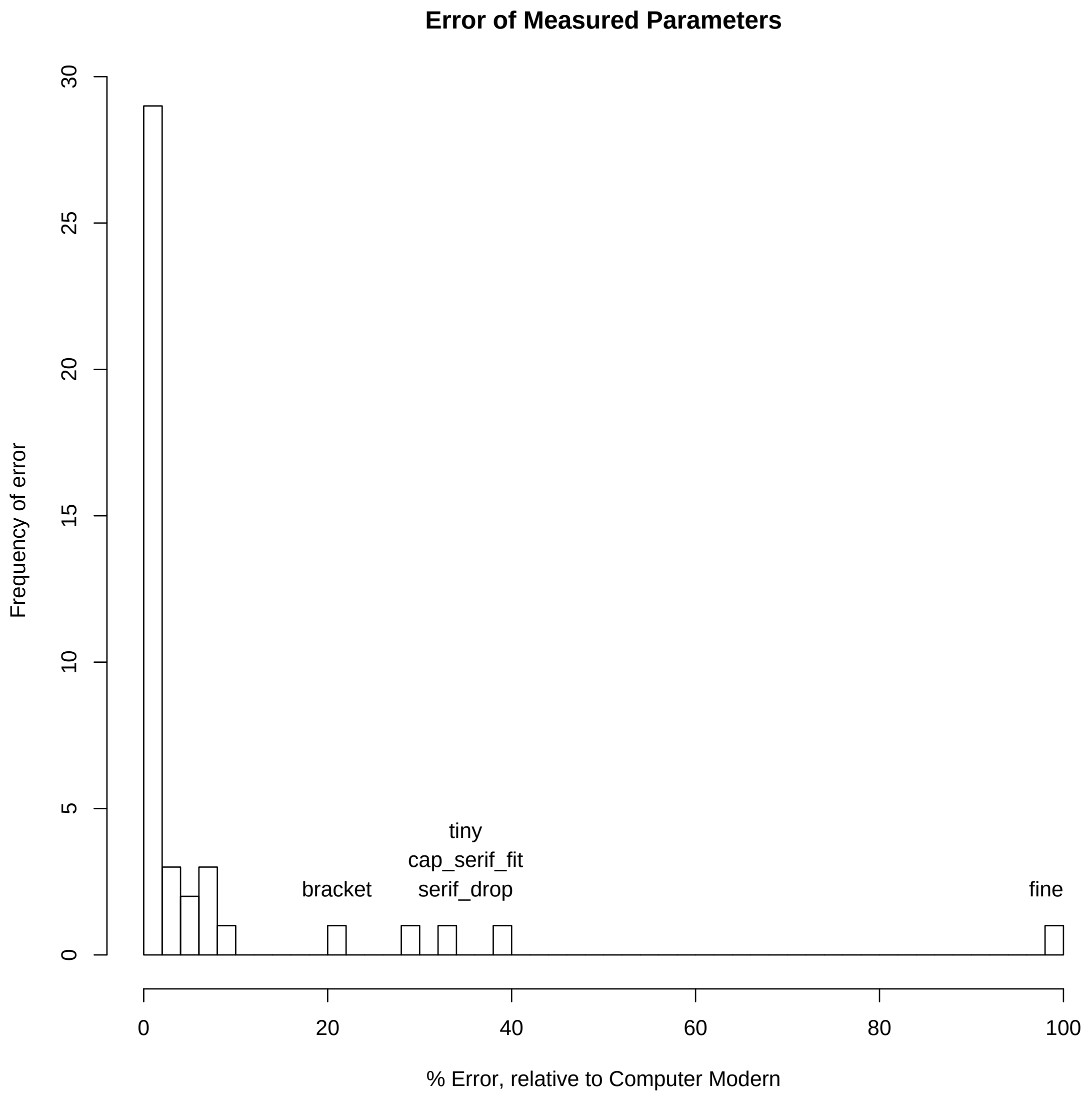
Naturally, numerical comparisons cannot tell us the aesthetic success of the generated fonts. Following this analysis are numerous visual samples of generated math fonts, which the reader may inspect to assess the degree of success of MathGen.

7.1 Computer Modern

Various companies have created Type 1 versions of the Computer Modern fonts; those versions can be run through MathGen just as easily as any other Type 1 font. Comparison of the generated parameters with those of Computer Modern reveal that MathGen has successfully captured, with good accuracy, the parameters of Computer Modern.

Figure 9 shows the percent error of the measured parameters, compared against the original parameters of Computer Modern. As the graph shows, most of the parameters are measured within a few percent

Figure 9 Histogram of the relative errors of the measured parameters of the Computer Modern font, calculated by the percent difference between the original value and the measured one.



accuracy. The four main outliers are named there. The outliers are as follows:

- The *fine* and *tiny* parameters define the curvatures around very small corners; their values change the stylistic appearance of the math fonts but not compatibility with the text.

- The *cap_serif_fit* parameter measures the amount of extra space surrounding the Π, Γ, and Λ characters. Currently that parameter is chosen in a somewhat ad-hoc fashion; further improvements could make that value more accurate.

- The *serif_drop* parameter measures the amount by which serifs at the tops of letters (i, l) are slanted. It is unfortunate that this value is so incorrect for Computer Modern (it is more accurate for nearly every other font tested), but it is not too significant, as it only affects one generated letter (ν).

- The *bracket* parameter measures the amount of curvature between a horizontal serif and the upward stem. It turns out that, because of the way serifs are drawn, the error will be divided by three, so it is not terribly worrisome. Additionally, serif bracket tends to be somewhat fickle to measure, but as long as a reasonable value is chosen, the serif will look correct.

We can also calculate error relative to the font's overall size. In that case, practically all the measurement errors are below 1%; the only substantial exception being the parameter *fig_height*, which is never used in MathGen.

7.2 Times Roman

Although Times Roman does not have an "official" set of METAFONT parameters, we can use the set prepared by Alan Hoenig as a reference for comparing our measured parameters [1]. There is no sense of an "incorrect" measurement because of this; Hoenig may just as well have erred in measurement.

As Figure 10 shows, the measurements for Times are substantially different from those provided by Hoenig. In fact, on the *cap_serif_fit* parameter there seems to be the greatest disagreement: Hoenig's parameters increase the whitespace around the relevant uppercase Greek characters, while MathGen removes space (i.e., the parameter is large and positive in MathKit but negative in MathGen).

It is difficult to resolve some of these disparities. For example, Hoenig's values for the *o* and *apex_o* parameters are both substantially larger than MathGen's computed values.[1] However, careful inspection of the font samples and actual hand-measurement of those parameters reveal that, if anything, they should be *smaller* than those reported by MathGen. The author's hypothesis is that Hoenig's measurements were taken on a slightly different version of Times Roman.

8 CURRENT STATUS OF THE SYSTEM

The MathGen system is currently fully functional and in the process of being made public. Currently the most important area of work is the user interface.

The program consists of seven Perl scripts and numerous auxiliary files that perform the measurements and construct the font. Information on which font to generate and special attributes of that font (e.g. whether or not the font is sans serif) are read out of a special file that the user is expected to create. Obviously this is an inconvenient system; a simple improvement would be an interview-like program that would ask the user to locate the appropriate font files and then create this configuration file automatically.

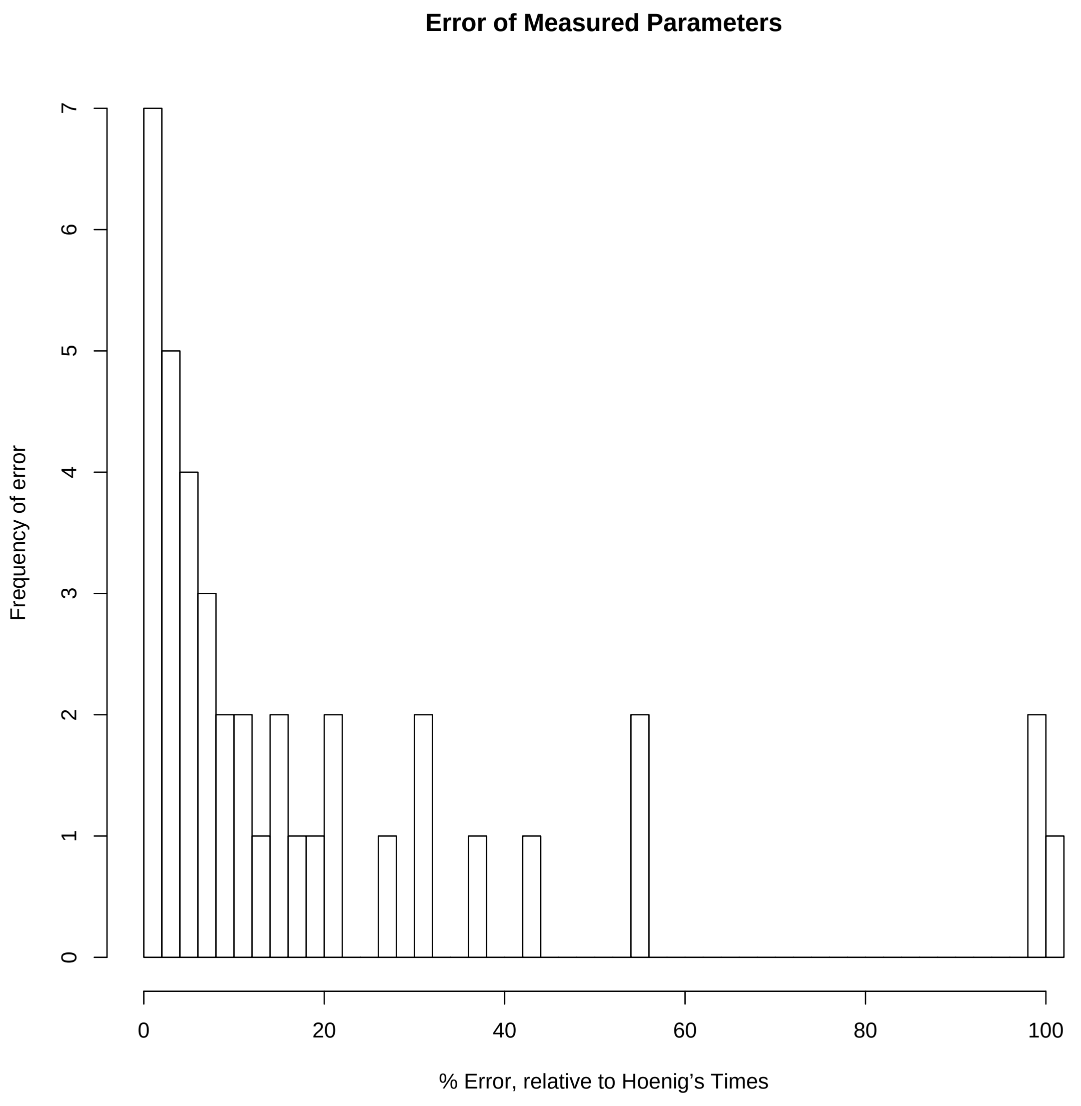
Additionally, although MathGen is intended to be run as-is, there are numerous "advanced features" sprinkled throughout the programs. The interface to these disparate expert parameters should be consolidated into a single file. In fact, there is already a main configuration file for constant parameter values not usually altered by the end user; those advanced parameters will soon be moved into that file and carefully documented.

Usage difficulty and current lack of documentation notwithstanding, MathGen has proven through tests with numerous fonts to be a robust, successful system for the generation of compatible math symbol fonts for a wide variety of text typefaces. The ease of font generation and the aesthetic appeal of the output will hopefully make its use popular and thus eliminate the common typographic dissonance of incompatible math fonts that is all too common today.

As a testament to that success, the math fonts of this paper were not the original Computer Modern math fonts; they were automatically generated by MathGen.

[1]These parameters specify the amount by which curved letters and angled letters (e.g., o and v) should "overshoot" their intended heights and depths.

Figure 10 Percent error of measured parameters relative to Alan Hoenig's parameter set, as provided in MathKit.



Samples of Generated Fonts

1 COMPUTER MODERN

As discussed before, it is instructive to compare generated math fonts based on Computer Modern against the original Computer Modern fonts. In most respects, the generated fonts are indistinguishable from the originals. The two obvious distinctions, the thicker stems of the β and γ characters, are stylistic changes made in response to surveys observing that the stems of those two characters in the original font were too thin. Additionally, the large display operators (summation, product) are somewhat lighter than in the original fonts. This is a matter of taste for the user; the operators' darkness could be increased by simply raising the appropriate zoom factor.

Under *extremely* careful inspection on high-resolution output, there is one other difference. Small corners, such as the small hooks on the terminals of letters like η, ι, and υ are sharper in the generated fonts than in the originals. This is due to the fact that, in Computer Modern, the corners of letters, in particular the corners of serifs, are sharp in the roman font but rounded in the italic. Since the "corner roundness" parameters are measured from the roman font rather than the italic, the generated math fonts have sharp corners as well. Note that this sharpness only affects very thin strokes, so it is barely noticeable at normal sizes.

One possible way to correct this would be to measure corner diameter from the italic font instead. However, italic fonts tend to have a less-defined structure: the style of hooks, serifs, and other features differ a great deal between italic fonts. As a result, it is difficult to make assumptions about the shapes of characters for italic fonts, and so it is hard to find consistent ways of measuring fine parameters such as corner diameter.

2 TRADITIONAL TEXT FONTS

Most standard text fonts follow certain conventions for typeface design: the use of serifs at the ends of letters, stem thickness differences in predictable places, consistent letter heights, and so forth. For these fonts it is not an unusual style of design or anomalous-looking characters that differentiate them; instead, differences in width, height, thickness, curvature, and other measurable parameters make up the bulk of the uniqueness of such fonts. Because these differences are so measurable and uniform, MathGen is adept at measuring and mimicking them.

To ensure that MathGen is successful at producing aesthetically compatible math fonts for many different text fonts, it was tested on a wide range of fonts. Table 1 lists some of the unique qualities of each of those fonts.

2.1 Bookman

Bookman was designed to be a highly legible typeface for books, modeled after Caslon but with larger x-height and width and less prominent serifs. Its no-nonsense, highly traditional style make it an excellent candidate for automatic math font generation.

The generated fonts capture many of the key features of Bookman. As expected, the stem and hairline widths are appropriate, as are the x-height and average letter width. The highly flared beak on the letter "F" is also mimicked in the "Γ" character, although not perfectly—the base of the beak is angled in the "F" but flat in the "Γ," as Computer Modern assumes that the tip of the beak is always flat. However, this is not too disconcerting.

2.2 Courier

Courier is a monospaced font designed to look like a typewriter. It displays very consistent stroke width and rounded serifs (in the version distributed with GhostScript).

The appearance of the generated letters is satisfactory given the unusual challenge of measuring a monospaced font, as the dimensions are not as one would expect (e.g. the serif jut is substantially different between the "I" and "M"). However, the programs capture the necessary features nicely, and the generated fonts match well with the original text.

However, there is one stylistic element that makes them slightly incompatible. Because Courier is so light, serifs are placed on practically every terminal point of every letter, including the top of the capital "A." However, the Computer Modern lowercase Greek letters have almost no serifs. As a result, the lowercase letters tend to look like "stick figures" or simple line drawings. While that is not problematic for mathematical formulas, where Greek characters are interspersed with roman, the generated Greek fonts would be highly unsuitable for text, more so with the Courier mock math fonts than other generated fonts.

2.3 Didot

The font Didot is a stark example of the modern style of type design, with its high stem width con-

Table 1 Unusual qualities of each of the standard text fonts tested.

Bookman:	wide, large x-height, heavy stems
Courier:	monospaced, no stem thickness contrast
Didot:	very high stem contrast, strong vertical stress
Adobe Garamond:	low stem contrast, small x-height, narrow
New Century Schoolbook:	heavy stems, thick serifs
Times Roman:	high stem contrast, thin serifs
Utopia:	squarish curved letters (high superness)

trast, strong vertical stress, and unadorned serifs. As the Computer Modern fonts are also drawn in the modern style, one would expect the programs to be highly capable of creating matching fonts.

The generated math fonts are surprisingly accurate. One distinctive feature of Didot is the rapid thinning of thick strokes into thin, observed in the lowercase and uppercase "O"; this thinning is captured in the *superpull* parameter. As a result, similar circular strokes such as in θ and Θ display the same rapid stroke thinning. The prominent darkness of beak serifs is also mimicked nicely.

One unfortunate characteristic of Didot is that the italic letters are substantially thinner than the upright ones. This causes the Greek letters to appear much too dark. Additionally, the font seems to have reported its italic slant incorrectly (the slant is read directly from a coded parameter in the font rather than measured).

The designer of Didot chose to make the *rule_thickness* parameter (which governs the thickness of strokes in mathematical symbols) about equal to the hairline thickness. It is questionable whether that is desirable, as the math symbols become barely visible at that thin size.

2.4 Adobe Garamond

Adobe Garamond is a very fine font designed after the typefaces of Claude Garamond. It exhibits a liberal use of curves, a relatively low stem thickness contrast, and a small x-height.

The italic face of Adobe Garamond is significantly narrower than the roman face, so the Greek letters look too wide in comparison. Otherwise, the characters blend well in terms of darkness and character shape. The generated fonts are not able to capture the oblique interior of the "O" in the Θ character; this would be a good place for improvement and additional parametrization.

One difficulty with Garamond is that, because it exhibits very long ascenders and descenders *and* it has a very sharp italic angle, the spacing is fairly incorrect. In particular, letters with ascenders and descenders tend to "push" other characters away, creating unsatisfactory visual whitespace. This could be solved by introducing kerning pairs into the math font, which would require using the measurement libraries to determine an optimum kerning between pairs of characters.

2.5 New Century Schoolbook

New Century Schoolbook is the typeface that is most similar in style to Computer Modern. Both are "modern" typefaces, exhibiting strong stroke contrast and vertical stress, but they are also both designed for readability over lengthy passages. Interestingly, though, many features of New Century Schoolbook are very different from those of Computer Modern although the general shapes are similar: the x-height is much higher and the letters are much darker than in Computer Modern.

Because of the similarity in shapes, the generated math fonts match practically perfectly with the text fonts. Stylistically, the only major difference is that the κ character terminates with a bulb while the k has a loop as the upper diagonal. The uppercase Greek letters are virtually indistinguishable from the originals.

2.6 Times Roman

Times Roman is one of the most popular fonts in common use today. One reason for its popularity is its very simple, traditional design, which makes it a good candidate for font measurement.

The resulting math fonts match very well with the text font. Unexpectedly, the Times italic font is somewhat lighter than the upright, so the Greek letters match better with the upright text than the italic. The capital Greek letters match practically perfectly; in particular the hairline-thin serifs are captured properly.

Because the Times fonts are so commonly used with the Computer Modern math fonts, it is worth noting why they are so *incompatible.* The Times fonts are significantly darker than Computer Modern; in particular the hairlines are much more prominent.

However, the serifs in Times Roman are much finer and less prominent than those in Computer Modern. The x-height matches fairly well, but the capital letters in Times are smaller than those in Computer Modern.

2.7 Utopia

Utopia is a text font much like New Century Schoolbook or Times Roman. Because of its very standard appearance, the generated math fonts look very compatible with it.

Unsurprisingly, the stem widths match well, and the font generally looks acceptable. One odd quality is the unusually high value of the *superness* parameter, causing the circular bowls to appear squarish. It seems that the parameter is almost over-exaggerated, with the ϕ letter appearing more square than circle. However, the squareness works well for characters such as α and δ.

3 SANS SERIF FONTS

As the original math fonts were originally designed as serif fonts—the sans serif roman designs were tacked on after the original programs were written—the production of quality sans serif math fonts presents a new challenge for the reworked Computer Modern font programs and MathGen.

To ensure good coverage of the general space of sans serif fonts, we select from several different styles of sans serif type. The geometric style utilizes simple shapes such as circles and triangles; Avant Garde and Futura are the two examples tested. The grotesque or gothic style, exemplified by Helvetica, is designed for simplicity and clarity. Finally, humanistic sans serif fonts utilize elements of serif font design to achieve elegance and readability; Optima is a humanistic sans serif font that we test.

Finally, Hardwood is a display-like sans serif font that exhibits several unusual display-like qualities. The generated math fonts for Hardwood reveal many of the successes and limitations of MathGen's use with sans serif fonts.

3.1 Avant Garde

Avant Garde is a "geometric" sans serif typeface, meaning that its letters are drawn from simple shapes like circles and straight lines. It is characterized by very strictly monowidth strokes, an unusually large width and x-height, and widely open counters and bowls (parts of letters surrounded partially or entirely by the letter, such as the inside of an "n" or "p").

The generated math font is easily able to capture the consistent stroke width and large x-height and width. Additionally, the *superness* parameter allows for many of the circular letters, such as σ and Θ, to appear roughly circular as well. One would expect the ϕ character to be circular as well, but it generated as wider on the horizontal axis. That is unfortunate but expected, as ϕ should be wider than o to make up for the additional visual color from the cross-stroke.

The Computer Modern Greek letters were not designed out of simple geometric shapes as Avant Garde is. As a result, although the generated letters are visually compatible with Avant Garde, they look stylistically very different. In a sense, Avant Garde is something of a display font, working under a certain design constraint. If the use of geometric typefaces is common enough, it might be worth designing a set of math metafonts that work under this constraint of simple geometric shapes; that could be used to generate math fonts that are both visually and stylistically compatible with a range of geometric typefaces.

3.2 Futura

Futura is a geometric sans serif font, drawn out of simple geometric shapes such as circles and triangles. It shares many properties with Avant Garde, but it differs in one key respect: Futura has an unusually low x-height. Early tests revealed that this property made several revisions to the character designs necessary; in particular, the upper hook of the δ was drastically modified to fit properly.

Although most of the letters look appropriate, the small x-height and correspondingly large ascender height (the increase in height of letters such as "l" and "d") cause several of the characters with ascenders to look oddly long. In particular, the cross-strokes of ϕ and ψ look somewhat disproportionate. It may be worthwhile setting a cap on their height to offset this odd appearance.

The uppercase Greek letters are slightly disproportionate in width—the Θ is too narrow and the Γ too wide—but on the whole they match well in appearance. In particular, the Δ and Λ characters display the characteristic sharp tip of the "A" and "V."

3.3 Helvetica

Helvetica is a "gothic" or "grotesque" sans serif font, due to its simple but non-geometric appearance. It exhibits a very strong apparent stroke regularity—but not actual stroke regularity; for example, the thickness of the joins of the arches in the "h," "m,", "n," and "u" are, upon close inspection, significantly thinner than the rest.

The optical uniformity of stroke widths is captured in most places and unexpectedly exaggerated in others in the generated math fonts. For example, the downward curve of the ζ and ξ characters is slightly but noticeably thinner than other strokes. This is based on the fact that the thickness of the top of the lowercase "o" is less than that of the sides, but unlike that letter, the ζ and ξ characters "enlarge" much more slowly. However, it doesn't look too disconcerting to me; in fact, I actually rather like it that way.

Another point about those same two characters: in most hand-designed sans serif fonts, the upper loop is represented as a straight bar, like the top of the redesigned δ, and the terminal hook generally does not turn upward. This is another example where a general-purpose redesign of the Computer Modern math fonts for sans serif compatibility would be in order. Possibly the revision of the δ character represents a start of that process.

Finally, close inspection of the roman Helvetica alphabet shows that almost all of the stroke cutoffs are horizontal or vertical. This is true for many of the strokes in the generated math fonts, but not all of them. However, this doesn't affect the compatibility of the fonts.

3.4 Optima

Optima is a "humanistic" sans serif font, named such because it takes many features from the design of serif fonts. Optima is unusual for its visible stroke contrast and its very subtle calligraphic styling, evident in the slight flaring of some of the thinner strokes.

The math fonts are able to capture much of the elegance of Optima, in particular its stroke contrast. However, because the programs assume a consistent width for strokes, the generated letters do not capture the slight flaring that occurs, so they do not appear as "calligraphic." The subtle width difference is most noticeable in the uppercase Λ, when compared to an uppercase "A." At low resolutions or small sizes this is not too significant, though.

Surprisingly, the shapes of the Computer Modern letters work very well with Optima. In particular, after the modifications to the routines that make the small loops on the β, ζ, and ξ, they match very well with the text, and the δ is unexpectedly elegant.

3.5 Hardwood

Hardwood is a sans serif font designed in an art deco style. Its main characteristics are a large x-height, very consistent stroke width, and fairly narrow characters.

Naturally the generated characters capture all three of these qualities well. Hardwood's characters do have somewhat non-traditional widths, so some of the letters are narrower than expected (in particular ρ, which should be almost as wide as a "p").

Because Hardwood is only available in an upright form, the Greek letters are not slanted. This demonstrates the fact that the Greek letters really aren't meant to be drawn in an upright style. The λ, δ, γ, and other characters extend too far to the left; this would not be noticeable if they were slanted to the right.

4 SEMI-STANDARD FONTS

Many fonts fall in between the categories of standard text fonts and display fonts. Such fonts generally exhibit the normal qualities of text fonts assumed by MathGen but also contain unique, unmeasurable qualities of appearance that cannot be easily duplicated. The question then becomes how distracting the lack of those distinctive qualities becomes when combining the text font with those qualities with a generated math font lacking them but similar in other aspects.

We look at three such semi-standard typefaces as examples. Palatino is in most respects a standard text font, but it was designed to mimic a chiseled pen. Tekton is designed to look like an architect's handwriting, but its strokes are consistent enough for MathGen to duplicate its overall appearance well. Tiffany also follows the general conventions for a standard text font, but it also includes large flourishes at the ends of many letters, giving it an unusual character.

The overall conclusion, at least in the author's opinion, is that while the generated math fonts lack the distinctive qualities of the original fonts, they are nonetheless compatible and acceptable for use. Those distinctive features augment the aesthetic quality of the text font, but they are not noticeably missed when not present in the Greek or mathematical symbol text.

4.1 Palatino

Palatino is a serif font with a strongly calligraphic design. It was originally designed for display use, but it is now commonly used for text as well. Although it is generally like a standard text font, it shows many unique display characteristics that are difficult to duplicate automatically.

The overall appearance of the resulting math fonts is acceptable; the stem weights and letter widths match properly. One minor problem is that the Palatino italic letters are somewhat lighter than the upright roman letters; since the Greek letters take after the uprights, they too are a bit darker than the italic (although they match nicely with the roman text).

Many of Palatino's unique calligraphic effects, though, are not captured. The angled interior of letters like "o" and "p" is not reflected, and the terminals of letters (e.g., k and u) are more angled while the Greek letters are more horizontal. Additionally, although the inward-jutting beak of the "F" is present in the "Λ" character, the slight curve of it is not.

Palatino's math set was generated with the *square_ends* parameter on, so that letters would not end in bulbs but rather with rigid cutoffs. This improved the font compatibility significantly.

4.2 Tekton

Tekton is a font designed to mimic an architect's handwriting. It exhibits a very consistent stroke width and an unusual upward angle to many of its letters, such as the capital "T" and "E."

The generated math fonts capture the stroke width very well, and the fonts look very much like the original roman characters. However, the slight flares at the tips of letters, mimicking small blobs of ink from a stopped pen, are not captured, as they are an unusual display feature of the font. This is not so obvious at small sizes, and it does not detract from the compatibility of the math characters.

More distracting is the lack of the upward slant of the characters. The horizontal bars of the characters Γ and Σ do not fit well with the slanted F and E. This is unfortunate, but there isn't much that can be done about it in any general manner.

On the whole, though, the generated font is very compatible with Tekton. As this font seems like one that might be used for informal mathematics documents, it is good that a compatible math font can be generated.

4.3 Tiffany

Tiffany is an unusual text font, exhibiting extreme stem thickness contrast and humorous serifs. The generated math fonts capture the stem contrast beautifully, and they exhibit some of the unusual qualities of the serifs, such as the unusually large angle of the upper serif of the "F". However, the overdone flare at the top of the serif isn't captured. However, that seems to be the only unusual quality of Tiffany, and all of the other letters, in particular the lowercase Greek, match very nicely.

5 DISPLAY FONTS

The following fonts deviate greatly from the assumptions that MathGen makes about letter shapes. As a result, many of the measurements taken from them are erroneous or unreasonable, and the fonts produced match much less well than those generated from traditionally-shaped typefaces.

These examples are useful for two reasons. First, they demonstrate the limitations of MathGen, as examples of what cannot be easily measured or imitated. Second, they demonstrate what MathGen will do in such "emergency" cases of fonts that fail to meet the assumptions of letter shapes. It is noteworthy that (after some debugging) MathGen may produce highly unnatural results with these fonts, but it will not fail.

5.1 Linotext

Linotext is an old-English blackletter font. As a very non-traditional display font, it exemplifies the limits of MathGen pushed beyond the extremes.

Practically every assumption about letter shape made by MathGen is violated by Linotext, which made it an interesting experiment. The attempt to measure stem width from the capital letter "I" was thwarted by the fact that Linotext's "I" has two stems. Even stranger, the non-circular "o" caused the *superness* measurement to be completely wrong, making all the circular bowls in the lowercase Greek letters (α, β, θ) appear like diamonds. The resulting font is in some ways reminiscent of Donald Knuth's "funny" font, in which he chose arbitrary parameter values to see what the font would do.

It is hard to conceive of any parametrization of the Computer Modern font that would actually look compatible with this font. It is clear, though, that computerized measurement of the letter features would be nearly impossible without some assistance from the human eye.

5.2 Khaki

Khaki is a font designed to look as if it were painted with a watercolor brush (an alternate version includes random splatters as well). Since this is a display font, perfect results are not expected, and it is pleasantly surprising to see that the generated math fonts are actually reasonable.

Khaki violates some of the stem darkness assumptions of the font generation program. In particular, on the letter "O," the right side is thin and the bottom thick; this is the opposite of what one would expect with a text font. Additionally, the letters have a degree of "texture" to them: the stems are slightly wavy, to mimic a watercolor brush.

The resulting math fonts have very similar stem thicknesses: the wide stems are as wide as the widest in the original and the thin stems are equally thin. However, the designer of Khaki used an unusually wide size for the thicker stems and prevented the font from becoming too dark by making many areas that would traditionally be thick into thin areas. For example, the lowercase "b" has an unusually thick stem, but the adjacent bowl is entirely very thin, unlike any traditional text font. Consequently, the math fonts look extraordinarily dark, being unaware of this change. Also, naturally, the generated fonts do not capture the texture of the strokes.

The generated characters do not look *too* strange; clearly they are at least better than using any off-the-shelf math font such as Computer Modern. Possibly in this case it would be best to use the Computer Modern font programs as a starting point, changing them to reflect the odd usage of stem widths, in order to produce a more acceptable-looking font.

6 EXPLANATION OF THE FONT SAMPLES

For each font, three samples are provided. First, a mixture of the font's text and generated math characters is given along with several sample equations that only use the generated characters. Below them is a sample of the font interspersed with the Computer Modern math symbols. Note that the variables in that sample are also in Computer Modern. The intent of the second sample is to reveal the aesthetically displeasing appearance of mixed font use.

Note that, in the case of Palatino, the Euler fonts are used in the second sample rather than Computer Modern. This gives a sense of how the generated math fonts compare to a math font drawn to specifically match the given font.

Finally, a sample of the system's success at character width and accent placement is given. This sample contains three parts. In the first part, each character is surrounded by two vertical rules representing the minimal edges of the character. The second part demonstrates the placement of superscripts and subscripts on characters, and the third shows the placement of accents.

It is noteworthy that the accent and width measurement routines are not executed on the generated math fonts. In particular, the accent positionings are hard-coded in the font programs. As a result, it is often the case that accents are more properly placed over the italic variable characters than over the Greek letters.

REFERENCES

[1] Hoenig, Alan. *MathKit*: Alternatives to Computer Modern mathematics. In *TUGboat*, Vol. 20, No. 3 (1999).

[2] Knuth, Donald E. *Computers & Typesetting.* Vol. E. Reading, MA: Addison Wesley, 1992.

[3] Knuth, Donald E. *The METAFONTbook.* Reading, MA: Addison Wesley, 1992.