FONT MATCHING WITH FLEXIFONTS*

David Mellinger, Interleaf, Inc. and Stanford University
Katherine Nitchie, Interleaf, Inc.

Abstract. The font–matching problem arises when you wish to show a font on a wide variety of output devices of different resolutions. How do you make the different representations appear as similar as possible, both in font style and in set width? The Flexifont system solves this problem by having a large pool of hand–crafted characters which are used to assemble matching fonts. The pool has versions of each character in different styles, so that a matching character can be chosen to have an appearance near to that of the font being matched. Within styles, the pool has several different–width versions of each character to allow choosing a character well–matched in width. The characters in the pool must be carefully designed so that they can be created in different–width versions, and so that they will match existing typefaces well.

Proofing High-Resolution Fonts on Low-Resolution Devices

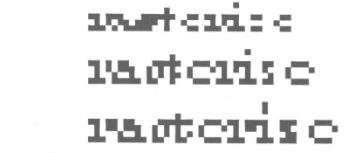
Many text-processing applications require that a particular typeface be accurately depicted in several different media. For example, What-You-See-Is-What-You-Get text editors typically need to show a bitmapped font on a 75 dot-per-inch CRT monitor, a 300 dpi laser printer, and an 800 to 4000 dpi typesetter. In order to maintain the WYSIWYG concept, these depictions need to be as similar as possible given the resolution limitations of the various output devices. The creators of text-processing systems have generally taken one of three approaches towards filling the need for fonts that match across a range of output devices.

The simplest method, and the one that produces the best-matching and highest-quality fonts, is simply to hand-craft each letter form of each font for each different output device in each point size, weight, and slant desired. The problem with this method is that it requires an enormous number of characters. A typical font used in WYSIWYG applications has 100 letters in four weight/slant combinations. Creating such a font for three output devices would require that 1,200 characters be created; for a typical range of ten point sizes, 12,000 characters. This approach is obviously impractical in labor and disk space cost if you need to match more than a few typefaces.

A second approach uses an outline font as a resolution independent master. Rasterization techniques use the master to create bitmap characters on the fly at the appropriate point size and resolution as they are called for by the proofing device. Although creating an accurate outline font can be more time-consuming and difficult than creating a bitmap font, it is more efficient in overall time and storage. To create outlines for the family described in the previous paragraph, an artist working on an outline font has to create 400 characters rather than 12,000. However, there are two major disadvantages to rasterizing. First, rasterizing small point sizes at low resolutions often creates illegible characters (see figure on next page). A second, more serious problem in

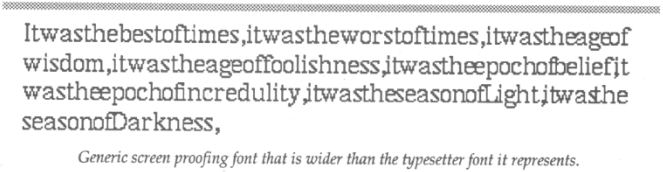
* Flexifont is a trademark of Interleaf, Inc.

practical applications is that the outline for a font may not be available to the creator front-end system. Font libraries are typically created at great expense by typesetter manufacturers, who sell fonts for use in their typesetters but do not release the outlines for use by others. The only information made public about a font is the widths of the characters in it so that typesetter users can space the characters properly. Therefore, there are no outlines available to make screen and laser printer versions of a font.



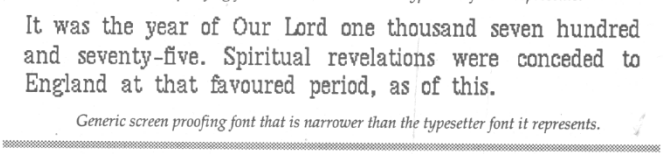
Here is the word "rasterize" rasterized to 6, 8, and 9 point 75 dpi screen resolution.

Overcoming these problems led to a third method of handling proofer fonts – generic fonts. A generic font is a single font used on one device to represent many fonts on another device. For example, a single generic font is used to display many different typesetter fonts on a screen, and a single generic laser-printer font is used for printing proof copies of the typesetter fonts. The generic representation is never perfect, but often it is adequate in situations where pixel-perfect accuracy is not necessary. At the resolution of a CRT monitor, there are not enough pixels available in a ten point font to show the stylistic differences between two serif fonts.



Itwasthebestoftimes,itwastheworstoftimes,itwastheageof wisdom,itwastheageoffoolishness,itwastheepochofbelief;it wastheepochofincredulity,itwastheseasonofLight,itwasthe seasonofDarkness,

Generic screen proofing font that is wider than the typesetter font it represents.



It was the year of Our Lord one thousand seven hundred and seventy-five. Spiritual revelations were conceded to England at that favoured period, as of this.

Generic screen proofing font that is narrower than the typesetter font it represents.

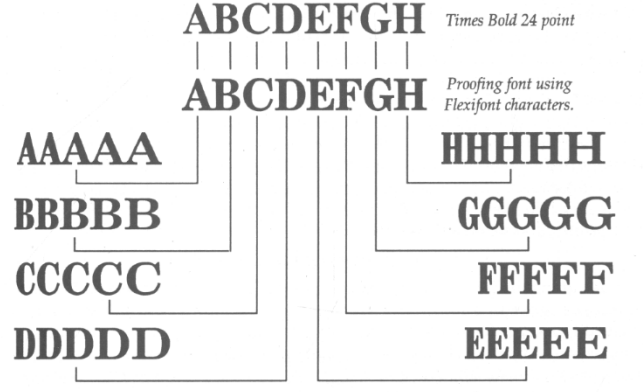
Of these three solutions to the proofing font problem, the generic font approach is the easiest, most accessible, and most economical for the maker of text-processing equipment to support, both in software code and in type design. However, it too has its problems. The generic font approach generally ignores all of the design quirks of a particular typeface. Can you imagine a single font being used to represent such diverse type styles as Palatino, Bodoni, Times Roman, Rockwell, and Helvetica? It also ignores the varying line lengths and set widths of the type styles that it represents. If

the generic font is not the same width as the typesetter font that it is representing, the difference is usually accommodated in the inter-letter or inter-word spacing on the proof. When the generic font is narrower than the typesetter font that it represents, the generic proof will have characters and words that are too widely spaced. When the generic font is wider than the typesetter font that it represents, the generic proof will have characters and words that over-print each other. Neither of these results is attractive, and both are difficult to read.

Flexifonts

The Flexifont system takes the generic font scheme several steps further in resolving its greatest disadvantages. Flexifonts are created from a pool of generic font characters, all of different widths. For each character in the typesetter font, there are several corresponding characters in the Flexifont pool, each of a different width. The software that composes the proofing font looks through the range of available characters and picks the one with the closest match to the character in the typesetting font.

Suppose we have a given font to be matched, 24 point Times Bold, for example, as seen in the accompanying illustration. We would compose our proofing font by looking at the width of each character in the typesetter font and then choosing the appropriate Flexi-character from the pool. Starting with the Times Bold 24 point capital A, we would scan the available capital A's and pick the one with the best match in width. The chosen Flexi-capital A will not look particularly like the typesetter capital A, but its width will be close enough that it will space in a similar manner. This way, we can avoid excessive word and letter spacing or word and letter crashing on the proof output. After picking the capital A, we will go through the same process with the capital B, and so on until the entire font has been composed.



Proofing font at 300 dpi (second row) for Times Bold (top row) beginning to be composed from Flexifont masters (below).

Sometimes, the width of a character to match may fall exactly between the widths of two Flexifont characters in the pool. Choosing either available character may give a bad width match. In this case, it is necessary to pick one of the two characters and adjust its side bearings (the white spaces on the sides of each character) so that its width matches more closely. This introduces the possibility, if such adjustments are too large, of having such adjusted characters in a word crash into each other or be too widely spaced. The solution to this is to have enough Flexifont characters in each pool that such adjustments are small and infrequent. Adjustments will tend to cancel each other out, since approximately the same number of Flexifont characters chosen will be adjusted to a larger size as to a smaller size.

Also, in choosing a particular Flexifont character to match a given character, it is not clear what the "best" width match is. It is not necessarily best to just choose the character with the nearest width to the given character, since a sequence of matching characters in a word may all err in the same direction, causing large accumulated width errors. A better algorithm might need to have information about how characters are distributed in words, and words in lines, in order to pick the best width for each matching character. Finding the best width-matching algorithm is under current investigation. In any case, the choice of algorithm is independent of the Flexifont idea of selecting the best character from a pool of available characters.

Using Interleaf's Flexifont System

Users start with a font on the typesetter which they wish to use in the Interleaf document-production software. They obtain a table of character widths for the font, either by getting it from the company that created the font or by loading the table directly from the typesetter onto the Interleaf system. They then specify what style to use, either serif or sans serif, and what point sizes and faces (roman, bold, etc.) they will want used as proof representation. Then they run the font-matching program, which goes through the process described above for each size/face combination - namely, it takes each character width of the typesetter font, then looks at the five Flexifont matching characters and picks the one closest in width. After doing this for each character in the typesetter font, it assembles the chosen characters into a new font and writes it to disk. Then it proceeds to the next size/face combination until all the combinations are finished. For a typical type family of ten sizes and four faces, this process takes about 15 minutes. With eight categories, in ten different point sizes, for two output devices, with five different width versions of each character, we ended up with 800 fonts total. These 800 fonts fit into about 15 megabytes of disk space, an amount easily stored on a personal workstation. When this is done, the newly-assembled fonts can be used in the Interleaf system like any other fonts – they can be shown on the screen, proofed on a laser-printer, and set on a typesetter.

Designing the Flexifonts

In designing the pool of Flexifonts, the group of Interleaf designers had to classify typeface designs based upon styles. Much of that division is self-explanatory. First, we grouped typesetter fonts into their two broadest classifications: serif and sans serif. Then, we subdivided each of those two categories into groups of roman, italic, bold, and bold italic. Most of the available text typefaces will fall into one of these categories, excepting unusual and novelty styles such as script and display faces.

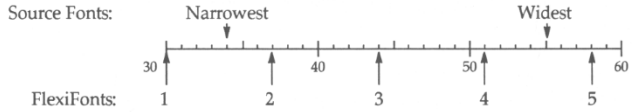
We were designing this system of proofing fonts for a printing language that used bitmapped images only. That meant that the fonts we delivered had to be in pixel

form. This imposes physical limits upon the type design system. You can only deliver a finite number of fonts in certain discrete point sizes. To make this project possible in the number of production hours available, we limited our output devices to two: a 75 dpi workstation monitor screen and a 300 dpi (industry standard) laser printer. We decided to offer a range of ten point sizes, 6, 8, 9, 10, 11, 12, 14, 18, 24, 36, feeling that most text typesetting would be in one of those ten point sizes. Presently, we are considering adding some other point sizes to widen the range of sizes that we offer.

Having decided on the heights of our fonts, we had to decide how many widths of each character we would offer, and how varied those widths would be. In order to make a considered and practical scale of widths, we had to do some research on the range of line lengths and average character widths available in typefaces commonly used today. So using data on character widths that we gathered from major typesetter vendors, we discarded the typefaces that seemed abnormally condensed, abnormally wide, or not applicable to text typesetting, to arrive at a range of figures for each of our eight style categories.

When we plotted the typefaces that we had chosen as our basic designs along our high-low graph (see illustration), they seemed to fall fairly consistently somewhere between 80 and 90% along the range. Looking at the results, we decided that we could comfortably cover our targeted range of fonts with 5 or 6 regular increments along the width graphs. It worked out that with even increments of 13.5% from our starting font, we could cover our range with a set of five fonts: our starting font, a wider font at 113.5% of the original, three narrower fonts at 86.5, 73, and 60% of the original. That covered our intended range quite well.

Widths of Source Fonts and Matching Flexifonts



Graph shows average character widths, in pixels at 300 dpi, of serif roman 24–point typefaces.

We also had to choose some typeface designs to use as the basis of our Flexifont pool. This was difficult. We wanted designs that were straight forward and fairly anonymous in look. If these designs were very stylized, their visual identification with a particular type face design might get in the way of their use as proofing fonts. If these designs incorporated of flourishes or other bits of decorative details, they might get lost or mangled in either the rasterization process or the condensing/expanding process. We wanted a design whose counters were fairly symmetrical an a vertical axis so that their shapes would not get destroyed in the condensing/expanding process. The shapes could be further preserved if we could find a design whose rounds were somewhat "squared" in appearance, rather than very rounded. Condensing a perfectly circular round shape would result in something that had little points or pimples at its top and its bottom. Finally, we wanted to find a design whose hairlines were not so thin that they would bleed out to zero width at the smallest (6, 7, and 8) point sizes. We needed to consider this for those occasions when our 300 dpi proofing device might be a very light printing printer. So after filtering many designs through the criteria that we had

established, we chose a Century-like face as our base for a set of serif fonts and a Helvetica-like face as the basis for our san-serif set.

ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz
ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz

Bitstream Swiss

ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz
ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz

Bitstream Century Schoolbook

Producing the Flexifonts

Interleaf creates its digital fonts by rasterizing an outline design to a specific pixel height. Font designers then edit the individual characters within each font smoothing out bumpy curves, checking weights, optimizing spacing. When it came time to start the actual production work of these Flexifonts, we had to decide when in the process of creating the bitmap characters, we would perform the condensing/expanding operations: during rasterization or while pixel editing the individual characters. At first we tried to condense the fonts while rasterizing the outlines. Particularly at larger point sizes, if we could make this feasible in producing acceptable results, it would be a significant time-saver. But we encountered a couple of problems.

When a designer creates a condensed or expanded version of a typeface, the designer tries to maintain a relationship in the weight/color of the typeface elements. This means that the stem weights of the condensed version are very close to (if not the same as) the stem weights of the original version. What is more likely to get condensed are the white areas of the character: the counters and bowls that capture internal white shapes and the side-bearings and set-widths that determine the white space around each character. Additionally, the horizontal strokes remain the same weight as those of the original version as well.

The first problem that we encountered when we tried to condense characters while rasterizing them was that condensing a character horizontally condenses *all* of the character's vertical pieces - stems, rounds, counters, side-bearings - while leaving the horizontal pieces intact. This creates a design whose vertical stem weights are significantly lighter than those of the original font and whose horizontal strokes are heavier than its vertical strokes, contrary to the principles of good type aesthetics. We were not pleased with these results. The rasterizing algorithm was smartened up to allow user-specifiable controls for stem weights. Then, we were able to rasterize fonts on an individual basis, batching together groups of characters with similar stem weights: OQCDG, HIJKLRTU, WAXV, ZNM, oce, bdpq, rnmihu, lktf, etc.

The second problem that we encountered was much more difficult to control. It involved the angle of italic fonts. As you rasterize an italic font to a proportionately tighter and tighter em box, the angle gets more and more upright. Unfortunately, we didn't notice this until we had started to combine characters from the various italic fonts to create proofing fonts. The results were not acceptable: capital H, capital I, and capital E were not parallel to each other. Engineering provided the design team with a program that would re-italicize a bitmapped font after it had been rasterized and condensed. The results at larger sizes and on certain characters were not too bad, but at smaller point sizes, they weren't very good.

Results

This proofing system has been optimized for the highest quality output possible on the final output device. Text produced by the system is of high typographic quality. The Flexifont method works well for creating fonts that match existing fonts in appearance and in set width. It avoids the pitfalls of other font-matching schemes, in that (1) characters have high aesthetic value; (2) text displayed with Flexifonts is the same width as the typeset text, so that it does not have letter- and word-crashing problems; (3) the text is quite legible, even at low resolutions and small point sizes; (4) the Flexifont system runs quickly and fits easily onto a personal workstation; and (5) running the system to add new fonts requires little work by the user. Since there are only two different styles of Flexifonts, one potential problem is that fonts which look different on the typesetter, such as Century and Bodoni, might be indistinguishable on the screen, since they are both represented by the same style of Flexifont. This has not turned out to be too much of a problem in practice, since documents usually don't mix typefaces in streams of text. Another problem occasionally noticeable is that adjacent characters seem to have different "fatnesses" – a character from a narrow source Flexifont (like number 1) will appear next one from a wide source (like number 5) – but the effect is not objectionable. Interleaf has successfully used the Flexifont approach to enable it to support several different typesetting machines (by Compugraphic, I.I.I., Linotronic, and Monotype) and other printers with built-in fonts (like Postscript and Interpress).

It was the best of times, it was the worst of times, it was the age of wisdom, it was the age of foolishness, it was the epoch of belief, it was the epoch of incredulity, it was the season of Light, it was the season of Darkness, it was the spring of hope, it was the winter of despair, we had everything before us, we had nothing before us, we were all going direct to Heaven, we were all going direct the other way - in short, the period was so far like the present period, that some of its noisiest authorities insisted on its being received, for good or for evil, in the superlative degree of comparison only.

Text sample showing screen Flexifonts emulating a Times Roman face.

The authors wish to thank the following contributors to the Flexifont project: David Anderson, Kimbo Peebles-Mundy, Les Snow, Natalie Bergman, Joseph Scaro, Anthony Kulesa, Jean Evans, Martina Mikulka, John Martin, Morissa Rubin, Anne Pease, Jeanne Williamson, Marie Lazarra, John Lawrence, Robert Morris, and Lawrence Bohn.

This paper was prepared with Interleaf's Technical Publishing System. The text font is Paladin 10–point.