Different Approaches to Lively Outlines

Erik VAN BLOKLAND† & Just VAN ROSSUM‡

† *Laan van Meerdervoort 1f*
2517AA The Hague, The Netherlands
‡ *Hasselaarsplein 7*
2013GB Haarlem, The Netherlands

Abstract: Some ideas about the creation of rough and lively outlines in typefaces, how to make them flexible, and control detail levels.

Keywords: Flexible, rough outlines, self modifying fonts, caching.

1 General

The roughness, or texture of an outline is a factor that can be controlled and used as a parameter in, for instance, graphic design. We have developed some schemes that create rough outlines in high resolution digital formats. Although there seems no direct technical demand for this (some of the schemes are rather counterproductive where speed is concerned), we do believe there is an aesthetic demand for new typefaces, that take advantage of the possibilities of their digital formats. Instead of re-creating old designs at a quality and resolution they were never meant for, we think that using the available technology in a different way can create new type and typography. We do not necessarily look for speedy programming though sometimes even we are forced to efficiency. We will present different approaches to creating this kind of outlines.

Liveliness and texture in letterforms have been lost during the mechanization of writing. We attempt to find out how they work and how we can make them again integral parts of lettering. We do not consider roughness and liveliness a technique that is just for simulating old type. Hopefully we'll find systems that will add these qualities to new typefaces, but have contemporary characteristics as well.

2 Blunt flexibility

A technique that can be applied to every outline. Here, like in the other processes we will describe, the printer is programmed to modify the outline. A process we (perhaps for the wrong reasons) call randomizing. Every point is translated in a semi-random direction and distance. Every point will therefore move in a different direction and make the type look distorted (Figure 1). The amount of distortion depends on how far each individual point is allowed to move, arbitrary values that can be found with some experimenting. The semi-random number generator used in PostScript, rand, suffices for this process [Adobe 87].

To make each instance of a character different, the font cache mechanism has to be switched of. A font cache rasterizes a specific character once and stores

Figure 1. Random distortion of an outline. [FontShop 90]

the bitmap for later use. When that specific character is called for again, the font cache will have the rasterized picture ready and save time. This would mean repetition of the distorted letterforms, not really the effect we're after. So we have to trade processor time for weirdness. This can be done by using the `setcharwidth` operator instead of the `setcachedevice` operator [Adobe 87, André 89]. Now the caching mechanism is bypassed and every character will be rasterized again when it is called for. Every call of a character will result in a specific combination of translation vectors, therefore a specific letterform, one that will almost certainly never repeat itself (that does not mean it cannot be reconstructed).

This technique only affects the points that are already in the outline. Small changes in the translation vector can create large differences in weight. The whole weight of a stem for instance is controlled by only four points. A small change in the relative position of these points can alter a character a lot. Large changes will affect the very legibilty, distorting the outline into an unfamiliar shape.

The overall color of the page typeset in a font like this remains very much the same. The differences in weight in individual characters will perhaps not cancel each other out, but they will find a balance. The same effect but on a smaller scale can be observed with printed type: the texture of the paper distorts the individual letterforms, but on the whole the page is clean.

3 Subtle flexibility

3.1 Self detailing fonts

To achieve a more smooth roughness (something that will affect the texture of the outline rather than it's weight) we can calculate intermediate points between the points on a plain outline.

An example. First, the outline is scaled to the desired size. The curved parts are then converted to straights (for instance using the PostScript `flattenpath` command, [Adobe 87]), and the longer straight lines are cut up into a series of shorter lines with the same tangent. This way the whole outline becomes a sequence of short vectors. To kick all these points around, we can apply the same mechanism that moves the points in the blunt flexibility scheme, or a more complex algorithm (Figure 2). The outline gets a rather lively texture. The number of vectors on such an outline will define the detail level. This means we can program details up to printer resolution in the outline independent of character size, by making the vectors

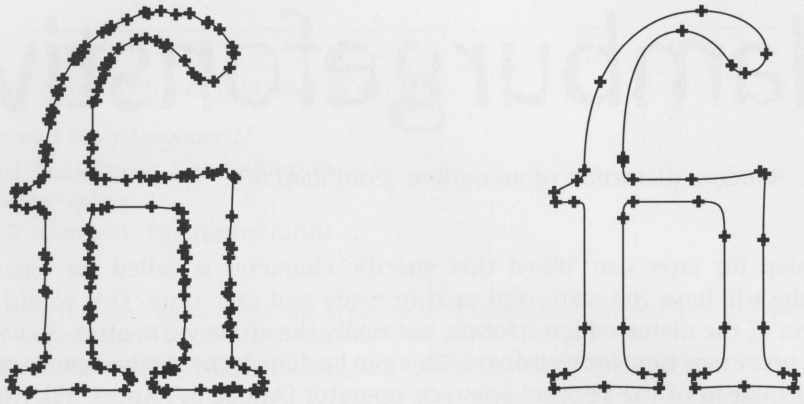

Figure 2. Note the number of points (marked by +'ses) on a normally digitized outline on the right [Adobe 89], and the same outline with a custom texture added, on the left.

shorter. But this will increase processing time dramatically.

3.2 *Fixed detail fonts*

Although it is not possible to reduce the number of points needed for a certain effect, we can at least streamline the procedure. The calculations for the intermediate points can be done outside the printer, for instance, in a typedesign program or in a custom "outline filter". During the design of the font a level of detail is chosen and all the outlines are converted to short straight vectors with a appropriate lengths. That does mean that the details will be scaled with the outline.

The resulting font still needs to calculate the random translation vectors in the printer, so its speed depends largely on the number of points on the outline. But it will still be noticeably faster than the self detailing font. If you know the size that will be used most commonly, it might be faster to make different fonts for different sizes than to wait for one to print.

4 Multiple caching

Speed versus memory. A mechanism that will cache randomized outlines but keeps alternative versions of the same outline in stock. A selection mechanism will jump through different caches (Figure 3). The number of caches that should be kept depends on available memory, and the things you want to do with the typeface. To make sure an outline is never used twice in a row (repetition of the same outline would at that moment become obvious), two cached versions will suffice for most languages, but three or four leave more room for variation.

Text printed with a multiple caching random font will still look like a "normal" random font. Performance of this font will be slower in the beginning (due to the

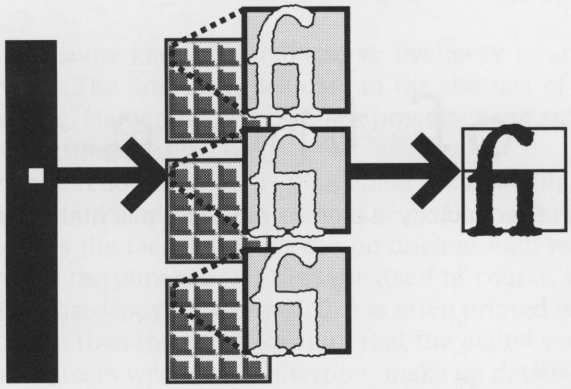

Figure 3. Multiple cache mechanism. The selecting device (l), a number of different versions of the same font, from which one instance is selected and printed.

extra caching), but gets to fairly normal speeds after that.

5 Wobbly Baselines

Not only the letter, but also it's orientation towards the page and other letters can be controlled and used. We tried various possibilities of making a variable baseline, as this is one of the aspects of handwriting that makes it lively (Figure 4).

First we tried a typewriter typeface by just shifting every single character in vertical direction with a random distance. This really creates the effect of an old, rusted typewriter. The shifting of the letters is very irregular.

Then we used a font with digitized handwritten letterforms. The vertical shift of a character is relative to the character on the left, and if there isn't any, to the original baseline. The letters increasingly move away from their baseline as they get closer to the end of the line (Figure 5).

6 A Point of View

The way most computers and printers treat fonts now, the font data and typographic layout data are kept separate until a page is actually printed. Then both the font and the page are downloaded to the printer. That means that if a font wants to find out how large it is going to be (to modify it's level of detail for instance), it has to wait to print time. We have to leave all of the processing to the printer. But most printers will not be able to handle the number of calculations in a reasonable period of time (within two hours per page). As designers we would like to see a font/page system would allow computer and printer both to take

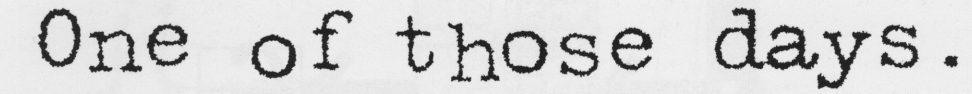

Figure 4. Victory of technology: a high resolution typewriter typeface

Figure 5. A growing shift in the baseline adds a natural unevenness to the text.

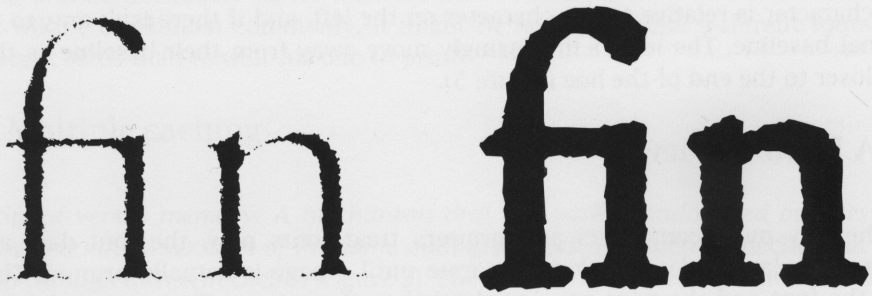

Figure 6. Digitally rendered outlines, simulated underexposure (r) and too much virtual ink (l). Both pictures were calculated from the same outline [Adobe 89].

outline roughness, texture and detail normal design parameters, just like size and orientation.

Methods that add some kind of roughness or liveliness to an outline can be as diverse as we want. The limitation lies only in the abilities of the printer and the patience of the user. Mathematically complex processes can simulate problems from real world typesetting (Figure 6).

Uninspired reconstruction however is happening a lot already, like the revival of hot metal typefaces in digital formats. One of the problems that arises when old designs are digitized is the fact that there are no original high resolution images in existence. There are the punches and the type itself of course, but these are no printed formats. The "hardcopy" that is available is often printed on paper that has a much coarser surface than the paper (or film) that the digital version is going to be printed on. The digiteers will have to interpret, make up details themselves and try to reconstruct somebody's thoughts from long ago.

The result is undoubtedly a very nice looking typeface, an image as smooth as state of the art printers can render, but completely lacking the beauty and charm of the original. Perhaps the only true way to digitize an old typeface is to get all the noise, dirt and smudges in as well. Perhaps the only true way to design a new typeface is to take advantage of modern technology and use it to the full extent (not to repeat the past faster). High resolution should not automatically mean smoothness. It means more detail to whatever you want to print (including pretty curves of course).

References

[Adobe 87] Adobe System Incorporated, *PostScript Language Reference Manual* Addison-Wesley Publishing Company, Reading MA, 1987.

[Adobe89] *Outlines from Adobe Garamond*, copyright Adobe Systems Incorporated, 1989.

[André89] J. André & B. Borghi, Dynamic fonts, *Raster Imaging and Digital Typography* (J. André & R.D. Hersch eds.), Cambridge University Press, 1987, p.198–204. (See also *The PostScript Language Journal - International Edition*, vol.2, no.3, 1989, pp.6–8).

[FontShop90] *Outlines from BeoSans*, copyright FontShop International GmbH, Berlin, 1990.