When TEX and METAFONT Talk: Typesetting on Curved Paths and Other Special Effects

Alan Hoenig
Department of Mathematics, John Jay College
17 Bay Avenue, Huntington, NY 11743 US
(516) 385-0736
Bitnet: ajhjj@cunyvm

Abstract

It is possible to successfully ask TEX to typeset text on arbitrarily curved paths provided one enables TEX and METAFONT to communicate with one another in an appropriate manner. In this paper, we describe one method for setting text on convex paths. One possible application of this work may be toward setting text along the circular rims of institutional seals so that TEX can include such images in letterheads. We discuss this particular example in some depth, and also present some examples of fanciful typesetting made possible when TEX and METAFONT communicate with one another.

Old Work

A few years ago, I thought of a way to get TEX to typeset around a circle, and I spent some time teaching this trick to TEX [1]. Here's the basic idea. I imagined inscribing a regular n-gon inside the circle. The generality of METAFONT makes it easy to generate n rotated fonts, so that characters from the i^{th} font would sit properly on the i^{th} face of my polygon. For purposes of testing, I used `cmbx12` as the font to rotate, and I let $n = 32$.

Many people to whom I showed the end products were kind enough to applaud my feeble efforts. But the kindest of all was one individual who scolded me in no uncertain terms. I had arranged things so that each letter was centered on its polygon face. This might have been acceptable had I used a monospace font (such as `cmtt10`), but with a variable spaced font like `cmbx12`, it looked just like someone had used a *computer* to set type around a circle. This critic closed his review with a scathing remark: PostScript could do better!

A new and substantially more acceptable but different approach has since occurred to me, and it's this set of techniques that I will discuss today.

Communication between TEX and METAFONT. Here's the main problem. TEX would be better at typesetting in nonlinear baselines if it were able to do more advanced mathematics. METAFONT *does* do that kind of mathematics, and therefore one immediately envisions some kind of dialog between the two programs as they generate and exchange information with one another. But METAFONT's file handling abilities are greatly crippled when compared to TEX. Other than font pixel files, font metric files, and log files, METAFONT cannot write files. Furthermore, although METAFONT can read files, it cannot read records individually—it's the whole file or nothing. Therefore, we have to design an inter-program dialog with some care.

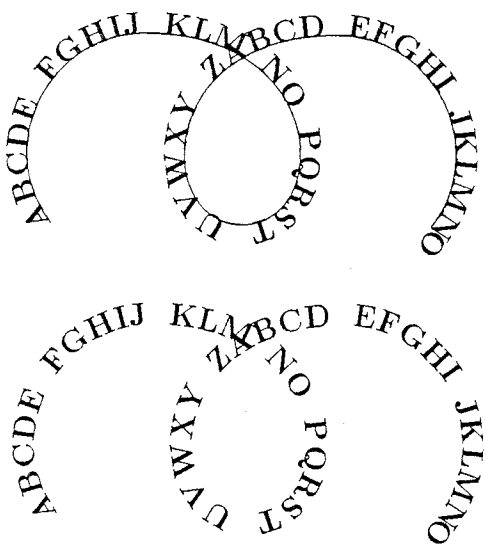

FIGURE 1. Curved typesetting.

In an earlier presentation [2], I had suggested that METAFONT might embed useful geometric information for later use by TEX in the `\fontdimen` parameters which accompany any font. This approach works, but more extensive testing revealed a problem. The syntax given in both *The TEXbook* and the METAFONT*book* suggests that there is no upper limit on the number of font dimens in any font, but the METAFONT program has a hard-coded upper limit of 50 such parameters per font. It looks like a simple change to the WEB listing could augment this value, but few users, not including myself, have ready access to WEB source (or to WEB expertise) which readily compiles in their operating system.

A better solution appeals to METAFONT's ability to store character kerning information in the `tfm` file. With old METAFONT, we were limited to 256 kern pairs, but the new limit with METAFONT2.7 is something like 32k or 64k—a much greater number of pairs.

Here's one way to pass numeric information from METAFONT to TEX using kern information. Suppose, for example, we need to tell TEX that the result of some important calculation is −14.2 pt. (There's nothing significant about this value; it was chosen purely for illustrative purposes.) We ask METAFONT to record that the kern between character 0 and character 1 (say) of a font be that value (−14.2 pt, in this example). The METAFONT code to do that is something like

```
ligtable 0: 1 kern -14.2pt#;
```

which should appear somewhere in the METAFONT driver program for this font.

How can TEX read that information? Let's suppose that the files `special.tfm` and `special.pk` store the information on this font. We can say something like

```
\font\specfont=special
```

in the TEX source document. To access this value, we say something like

```
\setbox0=\hbox{specfont\char0 \char1}
\setbox2=\hbox{\specfont
 \hbox{char0 }\hbox{\char1 }}
```

in the TEX file. The difference in the widths `\wd0` `\wd2` of these two boxes will be the number TEX needs. In practice, it is straightforward to create batch files which perform the necessary METAFONTing and which then invoke TEX, and to embed the details of the computation into a macro so this cumbersome routine is workable.

With these observations in hand, let's return to the main problem—how to typeset along any convex path, not just a circle.

Convex Paths

Here's just a quick word on what we mean by a *convex path*. Imagine that a tiny bug drives along the path in a tiny car, and that the bug has started at the beginning of the path and proceeds towards the endpoint without backing up at all. We say the path is *convex* wherever the bug turns the steering wheel to the right to stay on the path.

For typesetting purposes, convex paths are easier to treat then concave paths. The bottoms of adjacent letters butt against each other on convex paths. (On concave paths, the letters butt together at the top, and there are thorny problems in deciding where the bottoms of the letters will sit. That's why this paper only considers convex paths.)

A Three-Pass Method

I have been able to adapt a three-pass method to accomplish curvilinear typesetting. The end product of the three passes will be a new special purpose font, created just for the purpose of printing the curvaceous message. The characters in the font will not be those of the standard font layout, but will rather be the individual characters of the message, each one rotated or transformed by an amount appropriate for its position along the curved path.

Step One. The first pass belongs to TEX. In this step, TEX creates two files for later use by METAFONT.

TEX first examines the text of curvilinear material but does not typeset it. Rather, it examines each character, places it in an `\hbox` to measure its width, and writes this information into a file which METAFONT will use in the second step. TEX has adequate file handling abilities, so it's a straightforward task to create a file whose lines and records conform to METAFONT syntax. This file will be `widths.mf`.

By the way, the code to examine individual characters in a list is identical to the answer to exercise 11.5 in *The TEXbook* [3, page 67]. (The only difference lies in the definition of the macro `\\`, which I used to write the width information to an auxiliary file.)

The second file is `letters.mf` and contains essentially the individual characters of the message

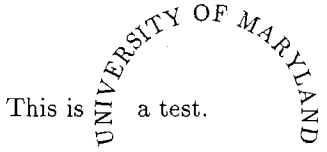

FIGURE 2. Text for a university seal.

gussied up with additional information that METAFONT will soon use to create the letter with its special rotation.

Step Two. In the second step, we invoke METAFONT. We make METAFONT use the information in the TEX output file `widths.mf` to create new information for the actual typesetting that TEX will do in the third (and final) step.

In our new font, I use `\char0` to store the representation of the actual curved path. (In this way, we can typeset the path as well as the curved text if we so choose.) Since I don't expect there to be any normal kerning between adjacent characters on a curved path, I am free to use all kerning information to transmit information back to TEX for the next step.

Now, for each character in your message, METAFONT performs a sequence of steps which I describe below. The purpose of these steps is to determine the position of this letter on the path, and to record this information for subsequent retrieval by TEX.

First, suppose the point z_0 marks our current position on the curve. Then we use METAFONT's extraordinary `solve` macro to find the point z_1 such that the length of the chord $z_1 - z_0$ is the same as the width of the current character. (We need to decrease the `tolerance` when using `solve`. Plain METAFONT sets `tolerance=0.1`; we need a smaller value, such as `tolerance=0.0001`.) Using other METAFONT commands, we can easily determine the angular orientation of this chord, and therefore the amount by which the letter should be rotated. (The chord is really an imaginary construct. We never draw it.) Really, we are *approximating* our path by a series of straight chords which "inscribe" the convex path such that each face of this approximate path will be the exact width of each letter or character.

TEX will eventually need two pieces of information about each letter in order to typeset it properly—the x- and y-offsets of that letter from the previous letter. We can pass this information to TEX using kerning pairs.

But there are other modifications we need to make to some standard METAFONT files such as `romanu.mf` (and other program files). `romanu.mf` contains the actual programs which METAFONT uses to construct the uppercase characters. This file is organized as a series of programs for each letter, one after the other. Here's how the program for A begins and ends.

```
cmchar "The letter A";
beginchar("A", 13u#, cap_height#,0);
...
penlabels(0,1,2,3,4,5,6); endchar;
```

The ellipsis denotes the details of the construction which are not important here. We add some lines to each such program as follows.

```
def A_(expr rotation_angle)=
currenttransform:=identity rotated
 rotation_angle;
def t_=transformed currenttransform
 enddef;
cmchar "The letter A";
beginchar("A", 13u#, cap_height#,0);
...
penlabels(0,1,2,3,4,5,6); endchar;
enddef;
```

That is, we embed the program for each letter within a METAFONT subroutine. (As you can see, METAFONT macro syntax differs from that of TEX.) The argument for each subroutine is the angle by which the letter needs to be rotated.

METAFONT finishes the second stage by using these subroutines together with the letter information passed to it in `\letters.mf` (first step using TEX) to generate the special purpose font.

Step Three. Finally, it's TEX's turn again. TEX takes your message text, and, character by character, it typesets it on the page. It extracts the information from the kerning pairs in the way I suggested earlier.

A frivolous example of curved typesetting appears in figure 1. The typesetting appears twice—with and without its path. If you look closely, the curved typesetting here looks a bit ragged. The reason is that I used an inferior method, in which it was only possible to get letters to match rotations to the nearest "quantum" of rotation, which was 360/32. In the improvement to that method, which is the method I just discussed, the fit would be better.

Getting to the Point. There is one application which may be of interest to curvilinear typesetters.

FIGURE 3. An approximation to π. We generate this figure by simultaneously rotating and magnifying numerals. We see the figure with and without its path.

Many university and institutional seals employ some text in a circle, such as that of the University of Maryland (shown in this article). It's a reasonably straightforward matter to use METAFONT to create the pictorial elements of any seal, but we would still need a method of setting the text, and of centering this text along the circle. (It's reasonably easy to perform this centering since we can take advantage of the circle's having constant curvature everywhere.)

Figure 2 displays a circular inscription suitable for a TEX letterhead. Unfortunately, the rest of the pictorial components for the seal have not yet been METAFONTed!

Breakthrough in Thinking

It was hard for me to get used to the idea that a single font could be created for one-shot uses. I am used to thinking of fonts as sacred collections that can serve long and honorably in many contexts. Nevertheless, if special-effect typesetting is needed, these special-purpose, one-shot fonts may be quite versatile.

It is clear, for example, that we can vary other of METAFONT's parameters at the time we do the rotation. In figure 3, we see the effect of simultaneously shrinking each numeral at the same time as we rotate it.

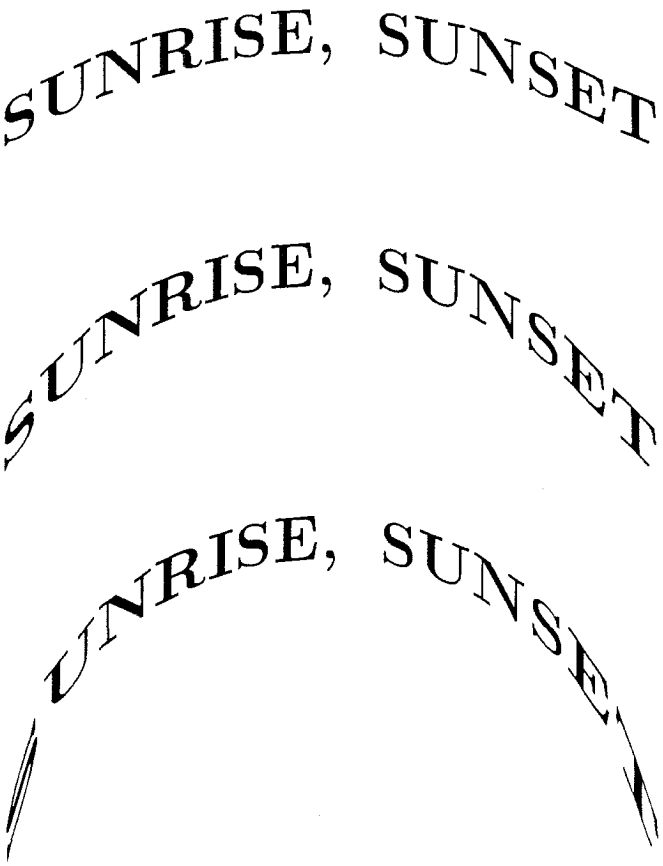

FIGURE 4. With the proper instructions, METAFONT will gladly generate these curve letters. We can adjust the rise of the "sunrise" as we see here.

But of course, we need not rotate the characters at all. We can use METAFONT to apply whatever special effect we want, character by character. Figure 4 is one such example.

Bibliography

1. Hoenig, Alan. "Circular reasoning: typesetting on a circle, and related issues" *TUGboat*, **11**#2 (June 1990), pages 183–190.
2. Hoenig, Alan. "Labelling Figures in TEX Documents" *TUGboat*, **12**#1 (March 1991), pages 125–129 (TEX90 Conference Proceedings).
3. Knuth, Donald E., *The TEXbook*. Reading, MA: Addison-Wesley, 1984.

Copyright 1991 Alan Hoenig