References

[1] Ronald L. Graham, Donald E. Knuth, and Oren Patashnik, *Concrete Mathematics.* Addison-Wesley, 1989.

[2] Donald E. Knuth, "Mathematical typography," *Bulletin of the American Mathematical Society* (new series) **1** (1979), 337–372.

[3] Donald E. Knuth, *Seminumerical Algorithms*, second edition. Addison-Wesley, 1981.

[4] Donald E. Knuth, *The TEXbook*, Volume A of *Computers & Typesetting.* Addison-Wesley, 1984 and 1986.

[5] Donald E. Knuth, *The METAFONTbook*, Volume C of *Computers & Typesetting.* Addison-Wesley, 1986.

[6] Donald E. Knuth, *Computer Modern Typefaces*, Volume E of *Computers & Typesetting.* Addison-Wesley, 1986.

[7] Donald E. Knuth and Hermann Zapf, "AMS Euler — A new typeface for mathematics," *Scholarly Publishing*, to appear.

[8] David R Siegel, *The Euler Project at Stanford.* Computer Science Department, Stanford University, 1985.

Outline fonts with METAFONT

Doug Henderson

Lately I've been feeling like no one else out there is having any fun with METAFONT. I sure am. Recently, I came across a small gem in the rough in the METAFONTbook and decided to polish it up some. In chapter thirteen there is an interesting exercise (13.23, page 121) which calls for the user to replace a character by its "outline". Since it was an idea that appealed to me (double dangerous bend kind of fun) I set about applying the solution in various places in the METAFONT source code to see the affects. What I finally settled on was hooking it in as a definition extension of the endchar macro. I reasoned that for each Computer Modern character METAFONT produces in a font, there exists both a beginchar statement, with which you relate things like the height, width and depth, and an endchar statement, which tells the program, among other things, to create a grid box for proof characters and to ship the character off to the Generic Format file (GF file). So, the endchar definition seemed like a good spot to tell METAFONT to take whatever character image had been created and convert it to an outline, since it is called once per character, right before it's shipped out.

Here is the macro definition I used to create outline fonts with:

```
message"Loading the font outline macros.";
boolean outlining;
% only outline when told to
outlining:=false;

def outline =
 if outlining:
  cull currentpicture keeping (1,infinity);
  picture v; v:=currentpicture;
  cull currentpicture keeping (1,1)
   withweight 3;
  addto currentpicture also v - v
   shifted right -v shifted left - v
   shifted up - v shifted down;
  cull currentpicture keeping (1,4);
  if (pixels_per_inch >= 600) :
    addto currentpicture also currentpicture
     shifted left;
    addto currentpicture also currentpicture
     shifted up;
  fi
  showit;
 fi
enddef;

extra_endchar:=extra_endchar & "outline";
```

The first statement declares a boolean variable named outlining. The next line initializes outlining to be false, so we don't create outline fonts by default. The definition of outline includes the line if outlining: which tests to see whether the outlining feature is desired. If so, the macro proceeds to punch out the pixels on the inside of our character (leaving more than one pixel for the outline if using a high resolution printer or phototypesetter) and show the results, and, if not, ends the if statement with the fi statement. The last statement is interesting since it shows a nifty way to tack on new features when creating your characters. Instead of redefining the definition of the endchar macro with your special effects (in this case a character outline), just add to the definition of endchar with the extra_endchar statement. Some similar "hooks" exist for the beginchar and `mode_setup` macros; they are `extra_beginchar` and `extra_setup`.

This macro was intended to be used (by me anyway) as an extension of the `plain` and `cm` (Computer Modern) base files. This way I did not need to input it each time I wanted to make an outline font. Since I place all of my local extensions to either base file into a file named `local.mf`, that is where my outline macro went.

To load it into the plain base file using the initialization version of METAFONT, INIMF, use the following line:

```
inimf plain input local dump
```

Similarly, for the Computer Modern base file use:

```
inimf plain input cmbase input local dump
```

Now that it is a part of your base files, you simply need to set the boolean switch `outlining` to true to make an outline font. Here is one way to make a 17 point sans serif font:

```
mf &cm \mode=varityper;
outlining:=true; input cmss17
```

After making my first few outline fonts I was feeling quite pleased with them so I happily sent them off to Professor Knuth thinking I had some interesting results to share. Here is a small sample of how the characters had turned out.

Some were funny looking! # =

Notice the last two characters in the line above have fairly thin strokes, so the resulting outline characters appear mostly filled in. Professor Knuth suggested that the values for `rule_thickness` be increased to match the other characters, so I experimented a bit and came up with `rule_thickness=1pt` for the font sample I sent, cmss17 (a 17pt sans serif font). Another parameter which he suggested I change was `notch_cut`. Looking at the W reveals why. The `notch_cut` parameter helps "black" fonts look correct at the meeting place between diagonal strokes by removing pixels there. While I was at it, I also found that the `cap_notch_cut` parameter, when set to a very high value, made the outline fonts look less dark. Knuth's suggestion of setting the `notch_cut` value to `17pt#` (relative infinity) helped out considerably. Here is our reworked W.

Some were better looking! # =

Another issue to be addressed is that of font naming conventions. The method supplied above has the side-effect of creating METAFONT runs with the names "cmss17.600" for a GF file, "cmss17.tfm" for a TFM file, and "cmss17.log" for the log file METAFONT creates. Since what we are doing is modifying Computer Modern fonts, we should also change the name of our outlined fonts. Besides, if you installed them in your font directories, you would overwrite the real cmss17 CM fonts!

I believe the best way around this name collision is to create unique fonts by copying the original name, say cmss17.mf, and copying it to another test file you can then work with. I put forth the suggestion that outline fonts have the letter "o" prepended such that an outline font created with `cmss17` would thus become `ocmss17`. This way the names and values of fonts which Knuth has perfected will remain untouched and we can happily play with the font "ocmss17.mf" without fear of reprimand. Below is a sample of some of the type of information I would change at the beginning of the file ocmss17.mf.

```
% This is OCMSS17.MF in text format,
% as of September 10, 1988.
% Computer Modern Sans Serif Outline 17.28 pt
if unknown cmbase: input cmbase fi

outlining:=true;

font_identifier:="OCMSS"; font_size 17.28pt#;

% rule_thickness#:=.6pt#; % old value
rule_thickness#:=1pt#; % new value
 % thickness of lines in math symbols
% notch_cut#:=32/36pt#; % old value
notch_cut#:=17pt#; % new value
 % maximum breadth above or below notches
% cap_notch_cut#:=46/36pt#; % old value
cap_notch_cut#:=17pt#;  % new value
 % max breadth above/below uppercase notches

% and of course the remaining fifty-eight
% parameters in a parameter file
```

The title is changed to reflect our new name and date of creation. The line `outlining:=true;` is used here to show an alternative to placing it on the command line, and the font_identifier is also changed to have the outline present (OCMSS). The other changes to the file are changing the values for `notch_cut`, `cap_notch_cut`, and `rule_thickness`, as they are.

This small amount of tuning I have found to be adequate for most of the outline fonts I have created. Here are some samples to show how well (or not) particular types of Computer Modern fonts look in outline form.

Computer Modern Roman Outline

5pt the lazy brown fox jumped
THE LAZY BROWN FOX JUMPED

10pt the lazy brown fox jumped
THE LAZY BROWN FOX JUMPED

12pt the lazy brown fox jumped
THE LAZY BROWN FOX JUMPED

17pt the lazy brown fox
THE LAZY BROWN FOX

Computer Modern Sans Serif Outline

8pt the lazy brown fox jumped
THE LAZY BROWN FOX JUMPED

10pt the lazy brown fox jumped
THE LAZY BROWN FOX JUMPED

12pt the lazy brown fox jumped
THE LAZY BROWN FOX JUMPED

17pt the lazy brown fox
THE LAZY BROWN FOX

Computer Modern Typewriter Outline

9pt the lazy brown fox jumped
THE LAZY BROWN FOX JUMPED

10pt the lazy brown fox jumped
THE LAZY BROWN FOX JUMPED

12pt the lazy brown fox jumped
THE LAZY BROWN FOX JUMPED

We can see from these samples that fonts which have a large degree of variance in the `stem`, `hair`, `curve`, and `vair` parameters, such as Computer Modern Roman fonts, do not make particularly good-looking outline fonts; often the horizontal bars come out looking black (or nearly so). This is also true for most variations of the Roman family, including CM bold (CMBX), slant (CMSL), and italic (CMTI).

Here is a sample of OCMBX12, OCMSL12, and OCMIT12:

12pt the lazy brown fox jumped
THE LAZY BROWN FOX JUMPED

12pt the lazy brown fox jumped
THE LAZY BROWN FOX JUMPED

12pt the lazy brown fox jumped
THE LAZY BROWN FOX JUMPED

The sans serif fonts, on the other hand, with nearly uniform thickness in `hair`, `stem`, and `curve` are better subjects for outlines than the serifed fonts and, I thought, came out rather nicely. Personally I thought that the CMSSDC (Computer Modern Sans Serif Demi-Bold Condensed) font came out looking the best of those that I tested. It looks like this:

10pt the lazy brown fox jumped over the smelly green dog
THE LAZY BROWN FOX JUMPED OVER AND OVER

12pt the lazy brown fox jumped over the smelly
THE LAZY BROWN FOX JUMPED

17pt the lazy brown fox jumped
THE LAZY BROWN FOX

I hope others will do some experimenting with outline fonts and share their results with the rest of us.

A few general observations that may be obvious but need to be said anyway are that the larger the font size, the better the "outline" due to the overall thickness of the digital pen strokes. This means that the five through ten point sizes are not really ideal for outline fonts for a 300dpi laser printer. I will be working on changing some parameters to make better low-resolution fonts and report on the results in a future issue. For now, though, we can see for ourselves how they look with the resolution of the APS phototypesetter (722.909 pixels per inch) since this article was typeset with them.