

FONT FORUM

Georgia K.M. Tobin

The ABC's of Special Effects

Special effects such as reverse video, pattern fills and pseudo shadowing can be fairly easily achieved in a METAFONT font by making use of `picture` variable manipulation. Since the way I did this incorporated the special effect at generation time and thus used the same code which generates my standard fonts, these specialty fonts retain much of the flexibility inherent in any meta-font, i.e. they can be generated for different point sizes and resolutions.

My basic plan of attack was to define a subroutine called *pattern* which in turn redefined the subroutine *endchar*. (Nota Bene: It would not do, under any circumstances, to simply overwrite the definition of *endchar* in plain.mf. METAFONT is quite content to redefine any subroutine whose name appears subsequent to its initial occurrence.) In this way, I can simply input the appropriate pattern code and invoke *pattern* at run time; each letter which is subsequently cranked through by METAFONT will be done in the special effect specified.

The simplest effect is reverse video. A step-by-step consideration of the creation of such a font should make the preceding generalities much clearer. I created a file called *pattern.reversevideo.mf* which contains the following definition of *pattern*:

```
def pattern=
def endchar=
cullit;
picture NormalChar;
  NormalChar=currentpicture;
clearit;
fill (0,-desc-2vo)--(w+ho,-desc-2vo)--
(w+ho,cap+2vo)--(0,cap+2vo)--cycle;
picture BlackBox;
  BlackBox:=currentpicture;
picture ReverseVideo;
  ReverseVideo=BlackBox-NormalChar;
currentpicture:=ReverseVideo;
% The rest is from standard endchar
scantokens extra_endchar;
```

```
chardx:=w;
shipit;
if displaying>0: showit; fi
endgroup;
enddef;
enddef;
```

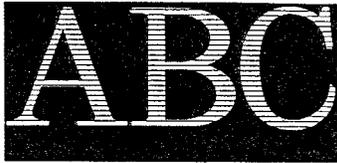
After loading all the normal font- and style-specific stuff, but *before* inputting any character descriptions, I input this file and then call *pattern*; the effect of this is to ensure that the `endchar` routine defined therein is run at the conclusion of each character.

So, prior to executing this `endchar`, METAFONT has just drawn a character in the usual manner. We recall that, depending upon the way in which the drawing was done, a blackened pixel may have any value greater than or equal to 1, and a white pixel any value less than or equal to zero. Since this can cause complications later as we add and subtract `pictures`, the first thing to do is a bit of housekeeping: all black bits are set equal to 1, and all white bits are set equal to zero. In other words, we capture the image of the character we have drawn in a `picture` variable *NormalChar* which is composed solely of 0's and 1's. Then, `currentpicture` is zeroed out, in preparation for a new drawing. We fill the whole letter grid completely — that is, set all bits to 1 — and set a `picture` variable *BlackBox* equal to the thus blackened grid. We then create yet another `picture` variable *ReverseVideo* and set it equal to *BlackBox* less *NormalChar*; i.e., the grid of all ones minus the grid with ones only at pixels that are part of the character. The result is:



An obvious and easy variation on this theme is a

greyed reverse video font. One way of achieving this is with a striped overlay:



We can produce this by creating a file which is called `pattern_reversegreyed` which replaces the line `currentpicture:=ReverseVideo`; in the preceding code with these lines:

```
clearit;
pickup MinPen;
for f=-desc-2vo step HugeStep
  until cap+2vo:
    draw (0,f)--(w,f);
endfor;
picture StripeOverlay;
  StripeOverlay=currentpicture;
currentpicture:=
  StripeOverlay+ReverseVideo;
```

which simply defines and draws a picture `StripeOverlay` which is added to the `picture` variable for the reverse video character as drawn in the same way as above. If we produce a new pattern file wherein we change that last line of code above to:

```
currentpicture:=
  StripeOverlay-ReverseVideo;
```

we get:



Extending the basic idea here just a bit, it seems that we can put a character filled with one pattern on a background filled with another. The following is the salient portion of the code which defines `pattern_stripedot`, a character filled with dots on a striped background. (We may assume that pictures `NormalChar` and `ReverseVideo` have been defined as in the preceding examples.)

```
% pattern one: the background pattern
pickup MinPen;
for f=-desc-vo step MedStep
  until cap+vo:
    draw (0,f)--(w+ho,f);
endfor;
currentpicture:=
```

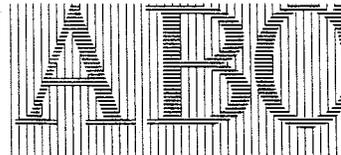
```
currentpicture+NormalChar;
cullit;
picture StripedGround;
  StripedGround:=currentpicture;
clearit;
% pattern two: the character fill
pickup PinPointPen;
for g=0 step BigStep until w:
  for f=-desc-vo step BigStep
    until cap+vo:
      draw (g,f);
  endfor;
endfor;
cullit;
currentpicture:=
  currentpicture-ReverseVideo;
cullit;
picture DottedChar;
  DottedChar:=currentpicture;
clearit;
currentpicture:=
  DottedChar+StripedGround;
and yields:
```



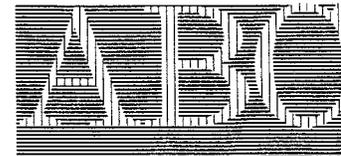
Clearly, by modifying what goes in the slots for pattern one and pattern two, we can produce



or



or



or even



When I was outlining this article, I had intended to say something along the lines of ‘you are limited only by your cleverness in coding patterns’ at this juncture; but as I produced the patterned fonts to use, I discovered that you are also limited by METAFONT’s capacity. In particular, the vertically oriented patterns fly in the face of the underlying idea of GF files, and the GF files get very big very fast.

* * * * *

In addition to doing arithmetic with `picture` variables to produce pattern filled characters, we can also manipulate pictures with rotations and shifts. We can produce a mirror image font



with a shifted reflection of `NormalChar`:

```
picture MirrorImage;
MirrorImage:=NormalChar
reflectedabout ((0,0),(0,h))
shifted (w,0);
```

A reverse video mirror image is accomplished by producing `ReverseVideo` in the same way as shown in the first example, and then doing a `reflectedabout` followed by `shifted` as before:



And, not to belabor the point, the same approach can produce any random pattern `reflectedabout` any line (that does not result in a transformation that METAFONT deems “too hard”) and `shifted` any amount.



* * * * *

Finally, I combined the two approaches — arith-

metic on `picture` variables and manipulation of them via `shifts` — to get a ‘pseudo-shadow’ effect. First, I `shifted` the character image to the right and down and subtracted the non-shifted character image from that shadow (giving the `picture ShadowOnly`). Then I laid a pattern-filled character on top. More precisely, I said:

```
f:=-desc-2vo;
pickup MinPen;
for f=-desc-2vo step HugeStep
  until cap+2vo:
    draw (0+ho,f)--(w,f);
endfor;
currentpicture:=
  currentpicture-ReverseVideo;
cullit;
picture StripedChar;
  StripedChar:=currentpicture;
clearit;
currentpicture:=NormalChar
  shifted (.5ucHairP,-.5ucHairP);
picture Shadow;
  Shadow:=currentpicture;
clearit;
currentpicture:=Shadow-NormalChar;
cullit;
picture ShadowOnly;
  ShadowOnly:=currentpicture;
clearit;
currentpicture:=
  ShadowOnly+StripedChar;
```

to get:



It should go without saying that any pattern may be used for the shifted shadow image, and any pattern may be used for the non-shifted image; likewise, the shifting may be in any direction, although the amount will doubtless be small in any case.

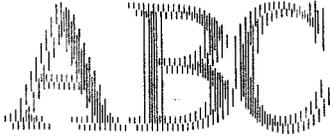


You’ll want to use discretion in combining patterns and shifts: such combinations quickly lead to effects that are not so much ‘special’ as rather wozy, as

in this dotted character with a horizontally striped shadow



or as in this font I call *HangOver*:



As a reaction to such excesses, I like the rather ethereal look of a shifted black shadow with *no* pattern in the character:



The code for each pattern given here can be applied to *any* font: I can do italic or bold Schoolbook as well as the text Schoolbook shown in the samples with no

changes. A completely different face – my decorative Uncial face



or a sans serif I'm working on



or my experimental Hebrew



– will need only minimal changes, viz. the names of pens and character parts; the same is true of CMR fonts.

I hope that these samples will serve as a springboard for my readers to generate special effect fonts of their own.

ART
begins where
GEOMETRY
ends,
and imparts to letters
a character transcending
mere measurement.

Paul Standard 1954