Fonts

Postnet codes using METAFONT

John Sauter

Abstract

A reimplementation of Dimitri Vulis' Postnet bar codes.

1 Introduction

I was excited to read in *TUGboat* 12, no. 2, about Dimitri Vulis' work with the Postnet bar codes for envelopes. I was determined to include his work into my letter-writing software until I came to the last line of his article, where he says "The macros are copyrighted, though, and I intend to defend them strenuously against unauthorized commercial use."

I was disappointed. This stricture meant that I could not use Mr. Vulis' work, since I sometimes write letters on behalf of a small retail store near my home. I was determined to find a way to use the Postnet codes in spite of Mr. Vulis' limitation.

2 A Different Approach

To avoid violating Mr. Vulis' copyright on his macros, I decided to take a different approach to the problem of producing Postnet codes. I would implement the bar codes using METAFONT as much as possible, with only as much TEX macros as necessary for support. I started by visiting my local Post Office, so I could obtain the information for constructing the bar codes from the source, to avoid any accusation that I had violated Mr. Vulis' copyright by using the dimension information in his macros.

I was pleased to find that the Post Office has liberalized the rules for Postnet codes since Mr. Vulis did his work. The FIM is no longer necessary, and the Postnet code can be placed immediately above the addressee's name, as follows:

John Sauter
9-801128-09 Elizabeth Drive
Merrimack, NH 03054-4576

Placing the Postnet code here avoids the hassle of figuring out how to get the code to appear in the proper corner of the envelope.

3 The METAFONT Font

The Postnet font file begins with identification and setup information.

```
% Postnet font, for USPS barcodes.
%
mode_setup;
%displaying:=0;
"Postnet digits";
font_identifier "POSTNET";
font_coding_scheme "Digits";
```

As Mr. Vulis describes in his article, Postnet represents digits using long and short bars. The first order of business in the font, therefore, is to define the dimensions of the bars and the spacing between them. This information is taken from the United States Postal Service regulations as described in the Domestic Mail Manual (DMM) issue 39 (June 16, 1991) sections 551 and 552, as summarized and explained in *Bar Code Update*, a document provided to me by my Postmaster.

```
% Primary parameters, as specified by
% the U. S. Postal Service.
bar_width#      := 0.020in#;
   % plus or minus 0.005in
half_bar_height#:= 0.050in#;
   % plus or minus 0.010in
full_bar_height#:= 0.125in#;
   % plus or minus 0.010in
bar_spacing#    := 1/22in#;
   % 20 to 24 bars per inch
```

The tolerances placed on the bar dimensions are great enough that any reasonably modern printer should have no trouble producing acceptable bar codes.

I now define some secondary parameters, so called because they are based on the primary parameters.

```
% Secondary parameters
digit_width#:=5/22in#;
   % width of a digit
digit_height#:=full_bar_height#+0.04in#;
   % leave space above bars
digit_depth#:=0.040in#;
   % min space below bars
```

The font parameters are next. These parameters are very simple, since this is a fixed-width font and all the characters are the same height.

```
font_size digit_height#;
font_slant 0;
font_normal_space digit_width#;
font_normal_stretch 0;
font_normal_shrink 0;
```

```
font_x_height half_bar_height#;
font_quad digit_width#;
font_extra_space 0;
```

It is now time to specify pixel-dimensioned versions of the necessary parameters. These will be used when actually drawing characters.

```
define_pixels (bar_width);
define_pixels (half_bar_height);
define_pixels (full_bar_height);
define_pixels (bar_spacing);
define_pixels (digit_width);
define_pixels (digit_height);
define_pixels (digit_depth);
```

Here is an alternate version of Plain METAFONT's `makebox`, which provides more information when printing proofs. It is based on an example in *The METAFONTbook*, Appendix E.

```
def makebox(text r) =
 for y=0,full_bar_height,
   half_bar_height,digit_height,
   -digit_depth:
  r((0,y),(w,y)); endfor % horizontals
 for x=0 step bar_spacing until w:
  r((x,0),(x,h)); endfor % verticals
 r((w,0),(w,h));
enddef;
```

All of the real work in a bar code font is done by drawing bars. It therefore seems fitting to place the bar-drawing macro next. This macro has two explicit parameters, the bar number and the bar height. It defines two points, `t` at the top and `b` at the bottom of the bar, and uses the current pen to draw it. The macro also depends on `bar_pos` to be the left edge of the bar, and increments this value so that the next invocation of the macro will draw the next bar in the following position.

```
def draw_bar (suffix $)
      (expr bar_height) =
   lft x$t = bar_pos*bar_spacing;
   top y$t = bar_height;
   x$b = x$t;
   bot y$b = 0;
   draw (z$t -- z$b);
   labels ($t, $b);
   bar_pos := bar_pos + 1
enddef;
```

Now, following the example of Computer Modern, I have defined macros to provide the beginning and end of each character. These macros are quite simple because of the simple nature of the font. All of the digits have the same width, height and depth, so the only parameter is the character code of the digit. The `end` macro is for aesthetics.

```
def beginpostnetchar (expr char_code) =
  beginchar (char_code, digit_width#,
      digit_height#, digit_depth#);
    bar_pos := 0;
    pickup stdpen;
enddef;

def endpostnetchar =
  endchar;
enddef;
```

We use only a single pen, with a simple shape: it has no height and is the width of a bar. We will use this pen only for vertical strokes.

```
pen stdpen;
stdpen = penrazor xscaled bar_width;
```

Well, it seems I lied about this font only containing digits. We need an additional full bar at the beginning and end of numbers, and as long as we need a separate full bar we should in fairness also have a half bar. We can use the `draw_bar` macro, but not the others since they assume a complete digit.

```
"full bar";
beginchar ("f", bar_spacing#,
    digit_height#, digit_depth#);
  bar_pos := 0;
  pickup stdpen;
  draw_bar (0, full_bar_height);
endchar;

"half bar";
beginchar ("h", bar_spacing#,
    digit_height#, digit_depth#);
  bar_pos := 0;
  pickup stdpen;
  draw_bar (0, half_bar_height);
endchar;
```

Now that the preliminaries are out of the way we can proceed with the digits themselves. Each digit consists of five bars, two full height and three half height. The pattern for each digit is described in *Bar Code Update*.

```
"Digit Zero";
beginpostnetchar ("0");
  draw_bar (0, full_bar_height);
  draw_bar (1, full_bar_height);
  draw_bar (2, half_bar_height);
  draw_bar (3, half_bar_height);
  draw_bar (4, half_bar_height);
endpostnetchar;
```

```
"Digit One";
beginpostnetchar ("1");
  draw_bar (0, half_bar_height);
  draw_bar (1, half_bar_height);
  draw_bar (2, half_bar_height);
  draw_bar (3, full_bar_height);
  draw_bar (4, full_bar_height);
endpostnetchar;

"Digit Two";
beginpostnetchar ("2");
  draw_bar (0, half_bar_height);
  draw_bar (1, half_bar_height);
  draw_bar (2, full_bar_height);
  draw_bar (3, half_bar_height);
  draw_bar (4, full_bar_height);
endpostnetchar;

"Digit Three";
beginpostnetchar ("3");
  draw_bar (0, half_bar_height);
  draw_bar (1, half_bar_height);
  draw_bar (2, full_bar_height);
  draw_bar (3, full_bar_height);
  draw_bar (4, half_bar_height);
endpostnetchar;

"Digit Four";
beginpostnetchar ("4");
  draw_bar (0, half_bar_height);
  draw_bar (1, full_bar_height);
  draw_bar (2, half_bar_height);
  draw_bar (3, half_bar_height);
  draw_bar (4, full_bar_height);
endpostnetchar;

"Digit Five";
beginpostnetchar ("5");
  draw_bar (0, half_bar_height);
  draw_bar (1, full_bar_height);
  draw_bar (2, half_bar_height);
  draw_bar (3, full_bar_height);
  draw_bar (4, half_bar_height);
endpostnetchar;

"Digit Six";
beginpostnetchar ("6");
  draw_bar (0, half_bar_height);
  draw_bar (1, full_bar_height);
  draw_bar (2, full_bar_height);
  draw_bar (3, half_bar_height);
  draw_bar (4, half_bar_height);
endpostnetchar;

"Digit Seven";
beginpostnetchar ("7");
  draw_bar (0, full_bar_height);
  draw_bar (1, half_bar_height);
  draw_bar (2, half_bar_height);
  draw_bar (3, half_bar_height);
  draw_bar (4, full_bar_height);
endpostnetchar;

"Digit Eight";
beginpostnetchar ("8");
  draw_bar (0, full_bar_height);
  draw_bar (1, half_bar_height);
  draw_bar (2, half_bar_height);
  draw_bar (3, full_bar_height);
  draw_bar (4, half_bar_height);
endpostnetchar;

"Digit Nine";
beginpostnetchar ("9");
  draw_bar (0, full_bar_height);
  draw_bar (1, half_bar_height);
  draw_bar (2, full_bar_height);
  draw_bar (3, half_bar_height);
  draw_bar (4, half_bar_height);
endpostnetchar;
```

In the Postnet code a number is more than a string of digits. To be a proper number a string of digits must have a full height bar before and after it. We can use the new facilities of METAFONT version 2 to provide these additional bars as ligatures.

```
%
% ligature table for Postnet font.
% provide tall bars at the beginning
%  and end of numbers.
%
boundarychar := 32;
beginchar (boundarychar, 0, 0, 0);
  endchar;

ligtable ||:
  "0" =:| "f",
  "1" =:| "f",
  "2" =:| "f",
  "3" =:| "f",
  "4" =:| "f",
  "5" =:| "f",
  "6" =:| "f",
  "7" =:| "f",
  "8" =:| "f",
  "9" =:| "f",
```

```
"0": "1": "2": "3": "4": "5": "6":
  "7": "8": "9": 32 |=: "f";
```

And with that, the font description is complete.

```
bye;
```

4 The TEX macros

The font itself is adequate for simple examples, like the one earlier in this article. However, as explained by Mr. Vulis, each number also ends with a check digit. I considered, very briefly, trying to do the check digit computation as a ligature table in the font. I came to the conclusion that METAFONT is the wrong language for such a computation, since the size of the ligature table gets very large with 11-digit numbers. Therefore, I decided to write the check digit code using TEX macros. Mr. Vulis used a very clever technique, but because of his copyright I had to use a different method. After some hunting I found an example of almost exactly what I needed in *The TEXbook*, Appendix D.

In the following pair of macros, `\postnetcheckdigit` takes as its argument a string of digits followed by a vertical bar. It uses `\getpostnetcheckdigit` to set `\count0` to the sum of the digits, and then arranges for token register `\Postnetchecktoken` to be set to the correct checksum digit for the string.

```
\def\getpostnetcheckdigit#1{\ifx#1\end
  \let\next=\relax
  \else\advance\count0 by #1%
    \let\next=\getpostnetcheckdigit\fi
  \next}
\def\postnetcheckdigit#1|{{\count0=0
  \getpostnetcheckdigit#1\end
  \count1=\count0
  \divide\count1 by 10
  \multiply\count1 by 10
  \advance\count0 by -\count1
  \count1=10
  \advance\count1 by -\count0
  \ifnum \count1>9
    \advance\count1 by -10\fi
  \aftergroup\Postnetchecktoken
  \aftergroup=\aftergroup{%
  \expandafter\aftergroup\number\count1
  \aftergroup}%
  }}
```

The check digit must be combined carefully with the rest of the digits so that the ligatures work correctly, placing a full bar before the first digit and after the check digit. In addition, it is convenient to have a macro which specifies the Postnet digits so they can be printed wherever in the letter the style requires. In some cases the Postnet code will be unknown or inappropriate, and so should not be printed.

To accommodate these needs I have a macro `\Postnetdigits` which accepts the digit string, and `\Postnetline` which is used from my letter formatting macros to set a line of Postnet bar codes.

```
\def\Postnetdigits #1{%
  \Postnettoken={#1}%
  \postnettrue}
\def\Postnetline{\ifpostnet
  \expandafter\postnetcheckdigit
  \the\Postnettoken|%
  \Postnettoken=\expandafter\expandafter
  \expandafter{\expandafter
  \the\expandafter\Postnettoken
  \the\Postnetchecktoken}%
  {\Postnetfont \the\Postnettoken\hfil}%
  \fi}
```

In support of the above macros we must declare the token registers and the condition.

```
\newtoks\Postnettoken
\newtoks\Postnetchecktoken
\newif\ifpostnet
\postnetfalse
```

My letter formatting macros are based on *The TEXbook*, Appendix E. I have placed the Postnet code only on the envelope, so only macro `\makelabel` needs modification. The Postnet bar code goes just above `\theaddress`, as follows:

```
\def\makelabel{\endletter\hbox{\vrule
    \vbox{\hrule \kern6truept
     \hbox{\kern6truept
       \vbox to 2truein
       {\hsize=6truein
         \smallheadfont
         \baselineskip9truept
         \returnaddress
         \vfill\vbox {%
           \hskip 2truein\Postnetline}%
         \moveright 2truein
         \copy\theaddress\vfill}%
       \kern6truept}%
     \kern6truept\hrule}%
    \vrule}
  \pageno=0\vfill\eject}
```

5 Conclusion

I thank Mr. Vulis for the motivation his article gave me to re-implement his Postnet bar codes.