The METAFONT approach: Implicit, relative, and analytical font design

Timothy Hall

Introduction

A past article in *Time*® magazine's *On-Line* monthly "submagazine" explored the world of do-it-yourself font creation and manipulation. The orientation of the article was to help a relative novice choose the right tools and techniques for whatever kind of font work was desired. The article was heavy on facts concerning a four-step process that might be familiar to readers of *TUGboat*:

1. Scan in a hand-drawn or copied glyph to form the basis for a character in your font.
2. Import the scanned image into a graphics software package, such as Illustrator, Fontographer, or CorelDraw.
3. Use the functionality of the package to trace the outline of the glyph.
4. Export the captured outline to an ASCII file that contains coded concatenated path segments.

In METAFONT syntax, the coded path segments will usually look something like this for a curved path:

$$z_0 \texttt{ .. controls } u_1 \texttt{ and } v_1 \texttt{ .. } z_1$$

or like this for straight lines:

$$z_0 \texttt{ -- } z_1 \texttt{ -- } \cdots \texttt{ -- } z_8$$

or perhaps combinations of these two forms. If each of these segments is named in a path array (**path** *pth[]*;), e.g., the third segment would be

$$\texttt{pth[3]=}z_2 \texttt{ .. controls } u_3 \texttt{ and } v_3 \texttt{ .. } z_3$$

then these path segments may be concatenated by the METAFONT operator "&" to form a cycle returning to the first point, such as this:

```
pth[0]=pth[1] & pth[2] & cycle
```

Finally, this path is typically filled to form the definition of the glyph:

```
fill pth[0];
```

As an example of this methodology, consider the following code for an arbitrary "mystery" character:*

```
numeric u; u = 1pt;
designsize := 10;
beginchar(99, 4.092pt#,
              5.80801pt#,
              0.264pt#); "myst";
fill  (0.594u,-0.264u)
 -- (0.594u,0.957u)
  & (0.594u,0.957u)
 .. controls (1.716u,2.046u)
    and (2.211u,2.805u)
     .. (2.31u,3.366u)
  & (2.31u,3.366u)
 .. controls (2.508u,3.927u)
    and (2.409u,4.686u)
     .. (2.013u,4.851u)
 -- (1.617u,4.917u)
 -- (0.693u,4.917u)
 -- (0.693u,5.808u)
 -- (1.881u,5.808u)
 .. controls (2.805u,5.808u)
    and (3.564u,5.412u)
     .. (3.399u,2.508u)
  & (3.399u,2.508u)
 .. controls (3.531u,1.32u)
    and (3.564u,0.693u)
     .. (3.696u,0u)
 -- (2.673u,0u)
  & (2.673u,0u)
 .. controls (2.574u,0.726u)
    and (2.475u,1.353u)
     .. (2.442u,1.782u)
  & (2.442u,1.782u)
 .. controls (1.881u,0.957u)
    and (1.221u,0.198u)
     .. (0.594u,-0.264u)
 -- cycle ;
cull currentpicture dropping (0,0);
endchar;
```

This form of code is perfectly acceptable to METAFONT, and this character definition (utilizing several unlisted parameter defaults) produces the glyph in Figure 1 (under `\mag=8`).

® *Time* is a registered trademark of AOL Time Warner, Inc.

* The material found herein does not depend on the particulars of any glyph definition.

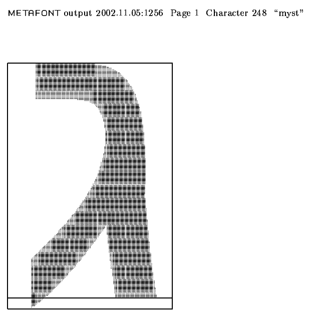

Figure 1: The original

However, there are several things “wrong” with the code in lines 6–36. METAFONT approaches font development from an implicit, relative, and analytical point of view, as compared to the explicit, absolute, and algorithmic calculations in Illustrator, Fontographer, and CorelDraw (among many others). In these other packages, Bézier curves are described in terms of control points, which are explicitly calculated within the software based on the absolute positions of critical points as parsed by the software. Illustrator, for one, when automatically tracing the outline of a glyph, decides where these critical points are, based on criteria and principles hidden from the user.

METAFONT, on the contrary, demands that the user decide where the critical points are located based on criteria and principles left to the designer, and expects the user to supply relative positions based on implicit relationships between the critical points. Only then will METAFONT calculate the control points for the glyph subpaths, and proceed to generate bitmaps for the image based on those calculations and numerous user-supplied parameter values.

In essence, simply stating a set of control points that blindly point to the outline of a glyph is not font design, and approaching font development this way is certainly not worthy of the amazing power and flexibility of METAFONT.

There is another thing “wrong” with the code in lines 3–5. The width, height, and depth of the character are stated as floating point numbers, one of which is given to five decimal places! This is definitely the work of a machine, and not a font designer. The control points listed in lines 6–36 also suffer from this feature. If the image had been held in a slightly different position when it was scanned using Illustrator, or a user had twitched slightly differently in positioning a serif in Fontographer, or a different version of the graphics software had been used with CorelDraw, or different hardware had been used in generating the numbers in lines 3–5, then slightly different numbers would be found there (certainly in the fifth decimal place).

This would not happen if the code development was under the control of a font designer who applied “the METAFONT approach.” No amount of alignment issues, mouse sentitivity settings, version control aspects, or platform dependencies will ever affect the implicit, relative, and analytical relationships between critical points in the description of a font glyph, so none of these should be present in a METAFONT description of the glyph.

The METAFONT approach

Suppose you were interested in designing a font with hundreds of characters that followed the look and feel of the mystery character given in lines 1–38. You could scan and trace all of your images, and hope that the variations introduced by this process do not get in the way of your font’s consistent look — or you might approach this task in the METAFONT way.

There are three straightforward transformations we may apply to lines 1–38 (and any others like them) to help reveal the underlying design structure of the mystery character, and help us apply it to all other characters in our desired font. This will go a long way towards ensuring a consistent look and feel to the entire font, and not simply be something applied to characters one by one. It is also possible that these transformations will reveal the absence of a consistent look and feel in the entire character set, thereby helping us modify lines 1–38 into a better glyph.

Implicit. The first transformation applies to all lines that contain explicit point definitions. If we label the first subpath as follows:

```
z1=(0.594u,-0.264u)
z2=(0.594u,0.957u)
z3=(2.31u,3.366u)
```

then we have:

```
x1=x2=0.594u; x3=2.31u;
y1+0.264u=y2-0.957u=0; y3=3.366u;
```

The resulting subpath would be:

z1--z2..tension α_2 and β_2..z3

The control points will be implicitly supplied by the **tension** statements during a later transformation.

The next subpath would be transformed as follows:

```
z4=(2.013u,4.851u)
z5=(1.617u,4.917u)
z6=(0.693u,4.917u)
z7=(0.693u,5.808u)
z8=(1.881u,5.808u)
```

. . . which produces:

```
x3=x4+0.297u=2.31u;
y3=y4-1.485u=3.366u;
z5=(1.617u,4.917u);
z6-z5=(-0.924u,0);
z7-z6=(0,0.891u);
z8-z7=(1.188u,0);
```

The resulting subpath would be:

z3..tension α_3 and β_3..z4
--z5--z6--z7--z8

Continuing this process, we arrive at the implicit description of the glyph, excerpted here.

```
55. numeric u; u = 1pt;
56. path pth[];
57. designsize := 10;
58. beginchar(99,4.092pt#,
59.              5.80801pt#,
60.              0.264pt#);"myst";
61. x1=x2=0.594u;
62. x3=x4+0.297u=2.31u;
63. x9+0.297u=x10=x11+1.023u=3.696u;
64. x12+0.231u=x11; x12-x13=1.848u;
65.
66. y1+0.264u=y2-0.957u=0;
67. y3=y4-1.485u=3.366u;
68. y10=y11=y9-2.508u=y12-1.782u=0;
69. y12-y13=2.046u;
70.
71.   [...]
72.
73. pth[1]=z1--z2;
74. pth[2]=z2..
75.        controls(1.716u,2.046u)
76.             and(2.211u,2.805u)
77.        ..z3;
78. pth[3]=z3..
79.        controls(2.508u,3.927u)
80.             and(2.409u,4.686u)
81.        ..z4--z5--z6--z7--
82.        z8..
83.        controls(2.805u,5.808u)
84.             and(3.564u,5.412u)
85.        ..z9;
86.
87.   [...]
88.
89. fill pth[1]
90.      for i=2upto6: & pth[i] endfor
91.      --cycle;
92. cull currentpicture dropping(0,0);
93. endchar;
```

The choice of points assigned to a given subpath is arbitrary, as long as the order of progression is retained. Indeed, the glyph produced by lines 55–93 looks exactly like that produced by lines 1–38 (see Figure 1 again), and would remain so should, for example, pth[3] be split into two subpaths each associated with a set of control points.

Relative. The next transformation enables the relative advantages of METAFONT design: Stating all points in terms of positions that are relative to the height, width, and depth of the character. In this way, you don't have to recalculate the points z1, z2, etc., every time you rescale the character to a different design size. Indeed, enabling the incredible variety of font characteristics found in the Computer Modern family is only possible when relative design policies are utilized.

The first question to answer in relative font design is, "on which dimension should the font be based?" If the font is to be monospaced, then standardization on the width would be appropriate. This would mean replacing the dimensions in lines 58–60 with

```
ds*pt#,
ds*(0.580801/0.4092)*pt#,
(0.264/0.4092)*pt#
```

in their respective positions, where ds is the design size. If the font is to have a uniform height, then the replacement dimensions would be

```
ds*(0.4092/0.580801)*pt#,
ds*pt#,
(0.264/0.580801)*pt#
```

Although it would be an odd design aspect, it is possible to standardize on a uniform depth, with the corresponding dimension adjustments as before. Finally, other standardizations are possible, with different corresponding numerical calculations on the dimensions, which would not change the overall look and feel of the font, as long as the aspect ratio ($\mathit{height} + \mathit{depth}/\mathit{width}$) remains the same.

For our purposes here, let's use a constant width equal to the design size, and make all other dimensions proportional to the width. Even though the width is the same for each character in the font design, each glyph need not occupy the same horizontal space available to a character.[1] The implicit and now relative form of lines 1–38 can now be seen.

```
100. path pth[];
101. designsize := 10;
102. ds         := designsize;
103. beginchar(99,ds*pt#,
104.           ds*(0.580801/0.4092)*pt#,
105.          (0.264/0.4092)*pt#);"myst";
106. x1=x2=0.594/(0.4092*ds)*w;
107. x3=x4+0.297/(0.4092*ds)*w
108.   =2.31/(0.4092*ds)*w;
109. x9+0.297/(0.4092*ds)*w
110.   =x10=x11+1.023/(0.4092*ds)*w
111.   =3.696/(0.4092*ds)*w;
112. x12+0.231/(0.4092*ds)*w=x11;
113. x12-x13=1.848/(0.4092*ds)*w;
114.
115. y1+(0.264/0.264)*d=0;
116. y2-0.957/(0.580801*ds)*h=0;
117. y3=y4-1.485/(0.580801*ds)*h
118.   =3.366/(0.580801*ds)*h;
119. y10=y11=y9-2.508/(0.580801*ds)*h
120.    =y12-1.782/(0.580801*ds)*h=0;
121. y12-y13=2.046/(0.580801*ds)*h;
122.
123.   [...]
124.
125. pth[1]=z1--z2;
126. pth[2]=z2..
127.        controls(1.716/(0.4092*ds)*w,
128.                 2.046/(0.580801*ds)*h)
129.             and(2.211/(0.4092*ds)*w,
130.                 2.805/(0.580801*ds)*h)
131.        ..z3;
132.
133.   [...]
134.
135. fill pth[1]
136.      for i=2upto6: & pth[i] endfor
137.      --cycle;
138. cull currentpicture dropping(0,0);
139. endchar;
```

Note that the u unit is no longer needed, as it is now a constant function of the chosen design size, and all dimensions are expressed in terms of the width, height, and depth, which are in turn multiples of the design size. Once again, the glyph produced by lines 100–139 looks exactly like that produced by lines 1–38 (so once again, see Figure 1). The fractions in the critical and control points need not be left as explicit reminders whence they came; simplified values, such as those found in the following excerpt, are often more convenient.

```
140. path pth[];
141. designsize := 10;
142. ds         := designsize;
143. beginchar(248,ds*pt#,
144.           ds*(1.419357)*pt#,
145.               0.645*pt#);"myst";
146. x1=x2=0.145161w;
147. x3=x4+0.07258w=0.564516w;
148. x9+0.07258w=x10=x11+0.25w
149.   =0.903226w;
150. x12+0.056452w=x11;
151. x12-x13=0.451613w;
152.
153.   [...]
154.
155. pth[6]=z12..
156.        controls(0.459677w,0.164772h)
157.             and(0.298387w,0.034091h)
158.        ..z13;
159.
160. fill pth[1]
161.      for i=2upto6: & pth[i] endfor
162.      --cycle;
163. cull currentpicture dropping(0,0);
164. endchar;
```

Analytical. The final transformation finishes the numerical calculations begun in earlier work. In particular, the explicit control points must be changed to implicit tension statements. This ensures that the glyph shape is defined by analytical considerations between critical points, and not by arbitrarily chosen "magic" control points.

The transformation from control points to tension statements is accomplished by the Matlab-based `cp2ab` utility.[2] Documentation for using this utility is included with its distribution. For example, for `pth[6]` (lines 155–158), using

```
z12,
(0.459677,0.164772*1.419357),
(0.298387,0.034091*1.419357), and
```

[1] This consideration is the basis for the `adjust_fit` function in plain METAFONT.

[2] If a user does not have access to Matlab, the freely available source code for `cp2ab` may be easily ported to many analytical calculation applications, such as MAPLE and S. A user may even use the code to make manual calculations.

```
    z13
```

as input (so that all calculations take place in units of `ds*pt`), the corresponding tension statement would be

```
    z12{dir-124.2156}
       ..tension0.9276and1.1893
       ..{dir-143.6158}z13
```

The `cp2ab` utility has the feature that all results are invariant under scaling. The `cp2ab` utility also accepts vector input (with corresponding vector output) so that all sequential post- and pre-tension values may be calculated simultaneously, as shown in this partial Matlab session.

```
    z1 =
       0.1452 + 0.2339i
       0.5645 + 0.8226i
    u1 =
       0.4194 + 0.5000i
       0.6129 + 0.9597i
    [...]

    [a1,b1,ppath1,diru1,dirv1]
           =cp2ab(z1,u1,v1,w1)

    a1 =
        0.8004    0.7771
        1.0488   -1.0000
    b1 =
        1.2963    1.3193
        1.2967   -1.0000
    ppath1 =
       0.1452 + 0.2339i
       0.4486 + 0.5766i
       0.5645 + 0.8226i

       0.5645 + 0.8226i
       0.4919 + 1.1855i
      -1.0000
    diru1 =
       44.1449   55.0882
       70.5599   -1.0000

    [...]
```

The following excerpt shows the results of the analytical transformation, which still has not changed the appearance of the glyph from Figure 1.

```
202. pth[1]=z1--z2;
203. pth[2]=z2{dir44.1449}
204. ..tension0.8004and1.2963
205. ..{dir55.0882}
206. (0.4486*ds*pt,0.5766*ds*pt)
207. ..tension0.7771and1.3193
208. ..{dir79.9923}z3;
209.
210.   [...]
```

Final product. To complete the METAFONT approach to font design, an effort should be made to minimize the number of tension values used in a glyph definition. This ensures a consistent look and feel from one character to another, especially along edges. For example, the use of 1.2963 and 1.2967 may be replaced by a single 1.3 value, such as in `pth[2]` and `pth[3]`. The final form of the mystery glyph, completely utilizing the METAFONT approach, may be found in Figure 5, and is listed in lines 211–258.

```
211. path pth[];
212. designsize := 10;
213. ds         := designsize;
214. beginchar(248,ds*pt#,
215.           ds*(1.419357)*pt#,
216.               0.645*pt#);"myst";
217. x1=x2=0.145161w;
218. x3=x4+0.07258w=0.564516w;
219. x9+0.07258w=x10=x11+0.25w
220.   =0.903226w;
221. x12+0.056452w=x11;
222. x12-x13=0.451613w;
223.
224. y1+d=0;
225. y2-0.164772h=0;
226. y3=y4-0.255681h=0.579545h;
227. y10=y11=y9-0.431817h
228.    =y12-0.306818h=0;
229. y12-y13=0.352272h;
230.
231. z5=(0.3951613w,0.846589h);
232. z6-z5=(-0.225806w,0);
233. z7-z6=(0,0.153409h);
234. z8-z7=(0.290323w,0);
235.
236. pth[1]=z1--z2;
237. pth[2]=z2{dir44}..tension0.8and1.3
238.    ..{dir55}(0.4486*ds*pt,0.5766*ds*pt)
239.    ..tension0.75and1.3..{dir80}z3;
240. pth[3]=z3{dir70}..tension1and1.3
241.    ..{dir157}z4--z5--z6--z7--
242.    z8{right}..tension1and1
243.    ..{dir-34}(0.6652*ds*pt,1.3680*ds*pt)
244.    ..tension1.3and0.75
245.    ..{dir-76}(0.8026*ds*pt,1.1374*ds*pt)
246.    ..tension1.3and0.75..{dir-93}z9;
247. pth[4]=z9{dir-84}..tension0.75and1.3
```

```
   ..{dir-79}z10--z11;
pth[5]=z11{dir98}..tension0.8and1.3
   ..{dir94}z12;
pth[6]=z12{dir-124}..tension1and1.3
   ..{dir-144}z13;

fill pth[1]
     for i=2upto6: & pth[i] endfor
     --cycle;
cull currentpicture dropping(0,0);
endchar;
```

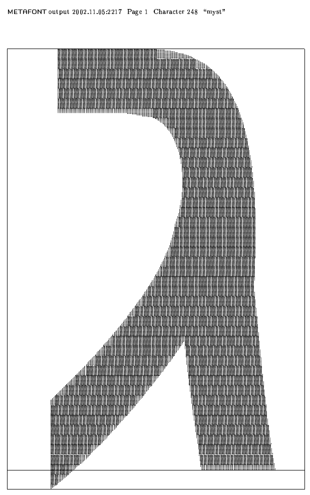

Figure 5: Final form

Summary. The METAFONT approach combines the irreplaceable advantages of implicit definitions (making the position of critical points refer to each other), relative policies (stating dimensions relative to the width and height of the bounding box), and analytical considerations (using tension statements rather than control points) to ensure the consistency and style of the resulting glyphs across all characters in a font.

With *practice, practice, practice,* and the right analytical tools, such font design far outshines, in flexibility and functionality, the let-us-do-it-for-you approach imposed by popular fontmaking applications.

◇ Timothy Hall
PQI Consulting
P. O. Box 425616
Cambridge, MA 02142-0012
`info@pqic.com`
`http://www.pqic.com/TUG`