

The METAFONT Approach: Implicit, Relative, and Analytical Font Design

Timothy Hall

Introduction

A recent 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 chose 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 that should 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.

The coded path segments will usually look something like this:

```
z0 .. controls u1 and v1 .. z1
```

or perhaps like this:

```
z0 -- z1 -- ... -- z8
```

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

```
pth[3]=z2 .. controls u3 and v3 .. z3
```

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

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

Finally, this path is 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*:

```

1.numeric u; u = 1pt;
2.designsize := 10;
3.beginchar(99, 4.092pt#,
4.          5.80801pt#,
5.          0.264pt#); "myst";
6.fill (0.594u,-0.264u)
7. -- (0.594u,0.957u)
8. & (0.594u,0.957u)
9. .. controls (1.716u,2.046u)
10. and (2.211u,2.805u)
11. .. (2.31u,3.366u)
12. & (2.31u,3.366u)
13. .. controls (2.508u,3.927u)
14. and (2.409u,4.686u)
15. .. (2.013u,4.851u)
16. -- (1.617u,4.917u)
17. -- (0.693u,4.917u)
18. -- (0.693u,5.808u)
19. -- (1.881u,5.808u)
20. .. controls (2.805u,5.808u)
21. and (3.564u,5.412u)
22. .. (3.399u,2.508u)
23. & (3.399u,2.508u)
24. .. controls (3.531u,1.32u)
25. and (3.564u,0.693u)
26. .. (3.696u,0u)
27. -- (2.673u,0u)
28. & (2.673u,0u)
29. .. controls (2.574u,0.726u)
30. and (2.475u,1.353u)
31. .. (2.442u,1.782u)
32. & (2.442u,1.782u)
33. .. controls (1.881u,0.957u)
34. and (1.221u,0.198u)
35. .. (0.594u,-0.264u)
36. -- cycle
37. ;
38.cull currentpicture dropping (0,0);
39.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`). 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

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

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

METAFONT output 2002.11.05:0256 Page 1 Character 248 "myst"

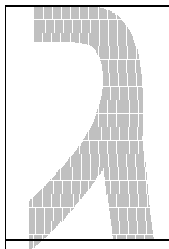


Figure 1: The Original

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, to 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 sensitivity 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 interesting in designing a font with hundreds of characters that followed the look and feel of the mystery character given in lines 1–36. 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 transformations we may apply to lines 1–36 (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 1–36 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  $\alpha_2$  and  $\beta_2$ ..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  $\alpha_3$  and  $\beta_3$ ..z4
--z5--z6--z7--z8

```

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

```

x1=x2=0.594u;
x3=x4+0.297u=2.31u;
x9+0.297u=x10=x11+1.023u=3.696u;
x12+0.231u=x11;
x12-x13=1.848u;

y1+0.264u=y2-0.957u=0;
y3=y4-1.485u=3.366u;
y10=y11=y9-2.508u=y12-1.782u=0;
y12-y13=2.046u;

z5=(1.617u,4.917u);
z6-z5=(-0.924u,0); z7-z6=(0,0.891u);
z8-z7=(1.188u,0);

```

The implicit form of lines 1–36 is now available.

```

70. numeric u; u = 1pt;
71. path pth[];
72. designsizes := 10;
73. beginchar(99,4.092pt#,
74.           5.80801pt#,
75.           0.264pt#);"myst";
76. x1=x2=0.594u;
77. x3=x4+0.297u=2.31u;
78. x9+0.297u=x10=x11+1.023u=3.696u;
79. x12+0.231u=x11; x12-x13=1.848u;
80.
81. y1+0.264u=y2-0.957u=0;
82. y3=y4-1.485u=3.366u;
83. y10=y11=y9-2.508u=y12-1.782u=0;
84. y12-y13=2.046u;
85.
86. z5=(1.617u,4.917u);
87. z6-z5=(-0.924u,0);
88. z7-z6=(0,0.891u);
89. z8-z7=(1.188u,0);
90.
91. pth[1]=z1--z2;
92. pth[2]=z2..
93.     controls(1.716u,2.046u)

```

```

94.         and(2.211u,2.805u)
95.     ..z3;
96. pth[3]=z3..
97.     controls(2.508u,3.927u)
98.         and(2.409u,4.686u)
99.     ..z4--z5--z6--z7--
100.    z8..
101.    controls(2.805u,5.808u)
102.        and(3.564u,5.412u)
103.    ..z9;
104. pth[4]=z9..
105.    controls(3.531u,1.320u)
106.        and(3.564u,0.693u)
107.    ..z10--z11;
108. pth[5]=z11..
109.    controls(2.574u,0.726u)
110.        and(2.475u,1.353u)
111.    ..z12;
112. pth[6]=z12..
113.    controls(1.881u,0.957u)
114.        and(1.221u,0.198u)
115.    ..z13;
116.
117. fill pth[1]
118.   for i=2upto6: & pth[i] endfor
119.   --cycle;
120. cull currentpicture dropping(0,0);
121. 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 70–121 looks exactly like that produced by lines 1–36 (see Figure 2), and would remain so should, for example, `pth[3]` be split into two subpaths each associated with a set of control points.

METAFONT output 2002.11.05:1256 Page 1 Character 348 "myst"

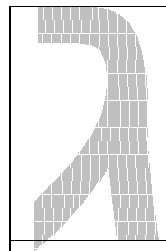


Figure 2: METAFONT Implicit

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 won't have to recalculate z_1 , z_2 , etc., every time you rescale the character to a different design size. Indeed, enabling the incredible variety of font characteristics found in the CM 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 73–75 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#
```

Even though 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 $\left(\frac{\text{height}+\text{depth}}{\text{width}}\right)$ 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 character¹. The implicit and now relative form of lines 1–36 is now available.

```
128.path pth[];
129.designsize := 10;
130.ds         := designsize;
131.beginchar(99,ds*pt#,
132.          ds*(0.580801/0.4092)*pt#,
133.          (0.264/0.4092)*pt#);"myst";
134.x1=x2=0.594/(0.4092*ds)*w;
135.x3=x4+0.297/(0.4092*ds)*w
136.  =2.31/(0.4092*ds)*w;
137.x9+0.297/(0.4092*ds)*w=x10=
138.  x11+1.023/(0.4092*ds)*w=
139.  3.696/(0.4092*ds)*w;
140.x12+0.231/(0.4092*ds)*w=x11;
```

¹ This consideration is the basis for the `adjust_fit` function in Plain METAFONT.

```
141.x12-x13=1.848/(0.4092*ds)*w;
142.
143.y1+(0.264/0.264)*d=0;
144.y2-0.957/(0.580801*ds)*h=0;
145.y3=y4-1.485/(0.580801*ds)*h
146.  =3.366/(0.580801*ds)*h;
147.y10=y11=y9-2.508/(0.580801*ds)*h
148.  =y12-1.782/(0.580801*ds)*h=0;
149.y12-y13=2.046/(0.580801*ds)*h;
150.
151.z5=(1.617/(0.4092*ds)*w,
152.    4.917/(0.580801*ds)*h);
153.z6-z5=(-0.924/(0.4092*ds)*w,0);
154.z7-z6=(0,0.891/(0.580801*ds)*h);
155.z8-z7=(1.188/(0.4092*ds)*w,0);
156.
157.pth[1]=z1--z2;
158.pth[2]=z2..
159.    controls(1.716/(0.4092*ds)*w,
160.             2.046/(0.580801*ds)*h)
161.    and(2.211/(0.4092*ds)*w,
162.         2.805/(0.580801*ds)*h)
163.    ..z3;
164.pth[3]=z3..
165.    controls(2.508/(0.4092*ds)*w,
166.             3.927/(0.580801*ds)*h)
167.    and(2.409/(0.4092*ds)*w,
168.         4.686/(0.580801*ds)*h)
169.    ..z4--z5--z6--z7--
170.    z8..
171.    controls(2.805/(0.4092*ds)*w,
172.             5.808/(0.580801*ds)*h)
173.    and(3.564/(0.4092*ds)*w,
174.         5.412/(0.580801*ds)*h)
175.    ..z9;
176.pth[4]=z9..
177.    controls(3.531/(0.4092*ds)*w,
178.             1.320/(0.580801*ds)*h)
179.    and(3.564/(0.4092*ds)*w,
180.         0.693/(0.580801*ds)*h)
181.    ..z10--z11;
182.pth[5]=z11..
183.    controls(2.574/(0.4092*ds)*w,
184.             0.726/(0.580801*ds)*h)
185.    and(2.475/(0.4092*ds)*w,
186.         1.353/(0.580801*ds)*h)
187.    ..z12;
188.pth[6]=z12..
189.    controls(1.881/(0.4092*ds)*w,
190.             0.957/(0.580801*ds)*h)
191.    and(1.221/(0.4092*ds)*w,
192.         0.198/(0.580801*ds)*h)
193.    ..z13;
194.
```

```

195. fill pth[1]
196.   for i=2upto6: & pth[i] endfor
197.   --cycle;
198. cull currentpicture dropping(0,0);
199. endchar;

```

Note how 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 128–199 looks exactly like that produced by lines 1–36 (see Figure 3).

METAFONT output 1992.11.12.219 Page 1 Character 28 "myst"

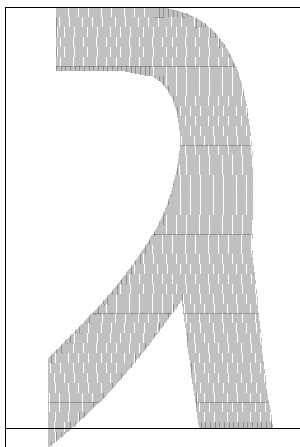


Figure 3: METAFONT Implicit and Relative

The values in the critical and control points need not be left as explicit reminders whence they came; simplified values, such as those found in lines 200–255 are often more convenient.

```

200. path pth[];
201. designsize := 10;
202. ds        := designsize;
203. beginchar(248,ds*pt#,
204.   ds*(1.419357)*pt#,
205.   0.645*pt#); "myst";
206. x1=x2=0.145161w;
207. x3=x4+0.07258w=0.564516w;
208. x9+0.07258w=x10=x11+0.25w
209.   =0.903226w;
210. x12+0.056452w=x11;
211. x12-x13=0.451613w;
212.
213. y1+d=0;
214. y2-0.164772h=0;
215. y3=y4-0.255681h=0.579545h;
216. y10=y11=y9-0.431817h

```

```

217.   =y12-0.306818h=0;
218. y12-y13=0.352272h;
219.
220. z5=(0.3951613w,0.846589h);
221. z6-z5=(-0.225806w,0);
222. z7-z6=(0,0.153409h);
223. z8-z7=(0.290323w,0);
224.
225. pth[1]=z1--z2;
226. pth[2]=z2..
227.   controls(0.419355w,0.352272h)
228.   and(0.540323w,0.482954h)
229.   ..z3;
230. pth[3]=z3..
231.   controls(0.612903w,0.676135h)
232.   and(0.58871w,0.806817h)
233.   ..z4--z5--z6--z7--
234.   z8..
235.   controls(0.685484w,0.999998h)
236.   and(0.870968w,0.931816h)
237.   ..z9;
238. pth[4]=z9..
239.   controls(0.862903w,0.227272h)
240.   and(0.870968w,0.119318h)
241.   ..z10--z11;
242. pth[5]=z11..
243.   controls(0.629032w,0.125h)
244.   and(0.604839w,0.232954h)
245.   ..z12;
246. pth[6]=z12..
247.   controls(0.459677w,0.164772h)
248.   and(0.298387w,0.034091h)
249.   ..z13;
250.
251. fill pth[1]
252.   for i=2upto6: & pth[i] endfor
253.   --cycle;
254. cull currentpicture dropping(0,0);
255. 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` utility². Documentation for using this

² Even if a user does not have access to MATLAB, the freely available source code may be easily

utility is included with its distribution. For example, for `pth[6]` (lines 246–249), using

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

Note how all pairs are processed in terms of `(w,h)`, even when $d > 0$, without explicitly mentioning the value of `w` or `h`. This is a fortunate feature of the `cp2ab` utility: 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 (see below).

```
z1 =
0.1452 + 0.2339i
0.5645 + 0.8226i
u1 =
0.4194 + 0.5000i
0.6129 + 0.9597i
v1 =
0.5403 + 0.6855i
0.5887 + 1.1452i
w1 =
0.5645 + 0.8226i
0.4919 + 1.1855i

[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
```

ported to many analytical calculation applications, such as MAPLE and S. A user may even use the code to make manual calculations.

```
0.4919 + 1.1855i
-1.0000
diru1 =
44.1449    55.0882
70.5599   -1.0000
dirv1 =
55.0882    79.9923
157.3801   -1.0000

z2 =
0.4597 + 1.4194i
0.8306 + 0.6129i
u2 =
0.6855 + 1.4194i
0.8629 + 0.3226i
v2 =
0.8710 + 1.3226i
0.8710 + 0.1694i
w2 =
0.8306 + 0.6129i
0.9032

[a2,b2,ppath2,diru2,dirv2]
=cp2ab(z2,u2,v2,w2)

a2 =
0.9599    1.2854    1.4312
0.7046   -1.0000   -1.0000
b2 =
1.0041    0.7524    0.7430
1.1942   -1.0000   -1.0000
ppath2 =
0.4597 + 1.4194i
0.6652 + 1.3680i
0.8026 + 1.1374i
0.8306 + 0.6129i

0.8306 + 0.6129i
0.9032
-1.0000
-1.0000
diru2 =
0.0000   -34.3494   -75.9637
-83.6600   -1.0000   -1.0000
dirv2 =
-34.3494   -75.9637   -93.2519
-79.2157   -1.0000   -1.0000

z3 =
0.6532
0.5968 + 0.4355i
u3 =
0.6290 + 0.1774i
0.4597 + 0.2339i
```

```

v3 =
  0.6048 + 0.3306i
  0.2984 + 0.0484i
w3 =
  0.5968 + 0.4355i
  0.1452 - 0.0645i

[a3,b3,ppath3,diru3,dirv3]
  =cp2ab(z3,u3,v3,w3)

a3 =
  0.8177
  0.9276
b3 =
  1.3927
  1.1893
ppath3 =
  0.6532
  0.5968 + 0.4355i

  0.5968 + 0.4355i
  0.1452 - 0.0645i
diru3 =
  97.7653
  -124.2156
dirv3 =
  94.3989
  -143.6158

```

Lines 373–401 list the results of the analytical transformation, and Figure 4 displays the resulting glyph. This figure also demonstrates how yet another transformation has not changed the appearance of the glyph.

```

373. pth[1]=z1--z2;
374. pth[2]=z2{dir44.1449}
375. ..tension0.8004and1.2963
376. ..{dir55.0882}
377. (0.4486*ds*pt,0.5766*ds*pt)
378. ..tension0.7771and1.3193
379. ..{dir79.9923}z3;
380. pth[3]=z3{dir70.5599}
381. ..tension1.0488and1.2967
382. ..{dir157.3801
383. }z4--z5--z6--z7--
384. z8{right}
385. ..tension0.9599and1.0041
386. ..{dir-34.3494}
387. (0.6652*ds*pt,1.3680*ds*pt)
388. ..tension1.2854and0.7524
389. ..{dir-75.9637}
390. (0.8026*ds*pt,1.1374*ds*pt)
391. ..tension1.4312and0.7500

```

```

392. ..{dir-93.2519}z9;
393. pth[4]=z9{dir-83.6600}
394. ..tension0.75and1.1942
395. ..{dir-79.2157}z10--z11;
396. pth[5]=z11{dir97.7653}
397. ..tension0.8177and1.3927
398. ..{dir94.3989}z12;
399. pth[6]=z12{dir-124.2156}
400. ..tension0.9276and1.1893
401. ..{dir-143.6158}z13;

```

METAFONT output 2012.12.12.02.18 Page 1 Character 28 "myst"

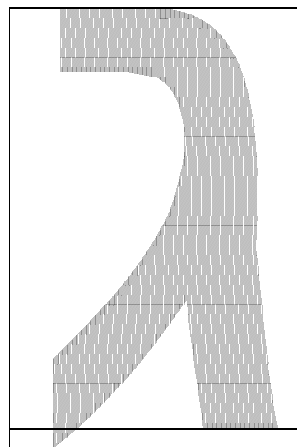


Figure 4: METAFONT Implicit, Relative, Analytical

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 step 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 402–449.

```

402. path pth[];
403. designsizes := 10;
404. ds          := designsizes;
405. beginchar(248,ds*pt#,
406.          ds*(1.419357)*pt#,
407.          0.645*pt#);"myst";
408. x1=x2=0.145161w;
409. x3=x4+0.07258w=0.564516w;
410. x9+0.07258w=x10=x11+0.25w
411.    =0.903226w;
412. x12+0.056452w=x11;
413. x12-x13=0.451613w;

```

```

414.
415. y1+d=0;
416. y2-0.164772h=0;
417. y3=y4-0.255681h=0.579545h;
418. y10=y11=y9-0.431817h
419.   =y12-0.306818h=0;
420. y12-y13=0.352272h;
421.
422. z5=(0.3951613w,0.846589h);
423. z6-z5=(-0.225806w,0);
424. z7-z6=(0,0.153409h);
425. z8-z7=(0.290323w,0);
426.
427. pth[1]=z1--z2;
428. pth[2]=z2{dir44}..tension0.8and1.3
429.   ..{dir55}(0.4486*ds*pt,0.5766*ds*pt)
430.   ..tension0.75and1.3..{dir80}z3;
431. pth[3]=z3{dir70}..tension1and1.3
432.   ..{dir157}z4--z5--z6--z7--
433.   z8{right}..tension1and1
434.   ..{dir-34}(0.6652*ds*pt,1.3680*ds*pt)
435.   ..tension1.3and0.75
436.   ..{dir-76}(0.8026*ds*pt,1.1374*ds*pt)
437.   ..tension1.3and0.75..{dir-93}z9;
438. pth[4]=z9{dir-84}..tension0.75and1.3
439.   ..{dir-79}z10--z11;
440. pth[5]=z11{dir98}..tension0.8and1.3
441.   ..{dir94}z12;
442. pth[6]=z12{dir-124}..tension1and1.3
443.   ..{dir-144}z13;
444.
445. fill pth[1]
446.   for i=2upto6: & pth[i] endfor
447.   --cycle;
448. cull currentpicture dropping(0,0);
449. endchar;

```

METAFONT output: 0112.0125.02.07 Page 1 Character 28 "mya"

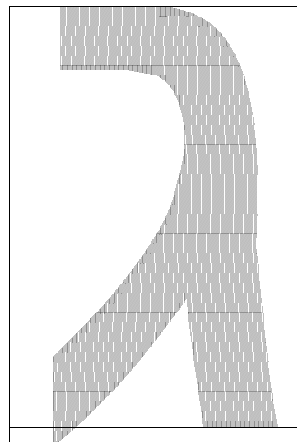


Figure 5: Final Form

approach imposed by popular fontmaking applications.

◇ Timothy Hall
PQI Consulting
P. O. Box 425616
Cambridge, MA 02142-0012
www.pqic.com/TUG/tmfa.htm

Summary. The METAFONT approach combines the irreplaceable advantages of implicit definitions (such as when the position of critical points refer to each other), relative policies (such as when dimensions are stated relative to the width and height of the bounding box), and analytical considerations (such as the use of 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