Tracing bitmap fonts in LMTX

Hans Hagen, Mikael P. Sundqvist

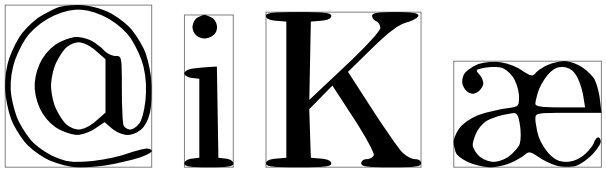
This is a follow up on potrace-generated outlines from bitmaps (see preceding article, "Unusual bitmaps"). Most of today's TEX users have probably never generated a document with bitmap fonts but back in the day we only had these, almost always Computer Modern. Because we don't use bitmap fonts in ConTEXt there is no need to support so-called PK (bitmap) fonts in the backend, but, because we did support them in MkIV, we still have it in LMTX. After all, what is TEX without bitmap fonts? We also owe it to Don Knuth.

It is a bit of challenge to load a traditional eight bit font using only a TFM file because in ConTEXt we map everything to Unicode. Regular Type 1 fonts are still supported, although they are declared obsolete, and vendors have moved on to OpenType fonts. When we define a font that has Type 1 resources, we load it as if it were an OpenType font. These eight bit fonts thus become wide (up to 64K glyphs) fonts internally. Normally we consult the AFM and PFB files and leave the TFM files, if present at all, for what they are. You can think of runtime `afmtotfm` conversion. An exception is the few traditional math fonts that we support: Antykwa, Iwona and Kurier; here the TFM files provide dimensions.

For the purpose of demonstrating what comes next it is enough to know that in principle one can still mess around with TFM values, either outline or bitmap, and that encoding files play a role in mapping them onto Unicode. We never set out to do something like this, but, because we had the loaders available anyway, some quick (few line) experiments of passing PK bitmaps to the MetaPost potrace helpers we got curious.

If you've run into an old TEX document on the web you might have noticed bitmap fonts being used. Often these are of a relatively low resolution. The reason that one never noticed that in print is that, when read on screen, we see glyphs large and can even zoom in, while in print they are seen small. When larger glyphs are used (say in a title) the scaled glyphs are actually different bitmaps: they have the same resolution as the smaller ones but more pixels because they are larger. Take these characters at 600 dpi, scaled up from their original 10 point size:

How do these patterns translate into an outline? For this we use the potrace library that we have available in LuaMetaTEX and interfaced to Metafun in ConTEXt (as discussed in the companion article), although for fonts we follow a more direct route.



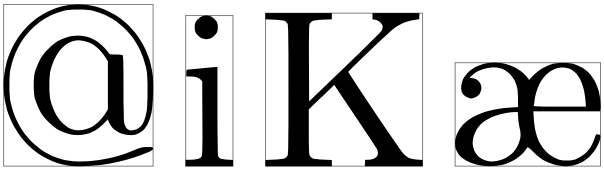
So why do these glyphs look somewhat different (smoother) from a normal outline Computer Modern? This is because the 600 may sound like a lot but actually isn't. Distributed over an inch we have 600 pixels on 25.4 mm so roughly one pixel per 0.05 mm, which might be acceptable in a small print but not when scaled. Here is how an 'm' in 600 dpi pixels is coded (64 by 37 pixels):

```
0000001111111000000001111111100000000000000011111111000000000000
1111111111110000011111111111110000000000011111111111100000000000
1111111111110000111100000011111100000001111000001111110000000000
1111111111110001110000000011111100000011100000000111111000000000
0000111111110011100000000000111111000111000000000001111110000000
0000001111111011100000000000011111100111000000000000111111000000
0000001111111111000000000000011111101110000000000000111111000000
0000001111111110000000000000011111111100000000000000011111100000
0000001111111110000000000000011111111100000000000000011111100000
0000001111111100000000000000011111111000000000000000011111100000
0000001111111000000000000000011111110000000000000000011111100000
0000001111111000000000000000011111110000000000000000011111100000
0000001111110000000000000000011111110000000000000000011111100000
0000001111110000000000000000011111110000000000000000011111100000
0000001111110000000000000000011111110000000000000000011111100000
0000001111110000000000000000011111110000000000000000011111100000
0000001111110000000000000000011111110000000000000000011111100000
0000001111110000000000000000011111110000000000000000011111100000
0000001111110000000000000000011111110000000000000000011111100000
0000001111110000000000000000011111110000000000000000011111100000
0000001111110000000000000000011111110000000000000000011111100000
0000001111110000000000000000011111110000000000000000011111100000
0000001111110000000000000000011111110000000000000000011111100000
0000001111110000000000000000011111110000000000000000011111100000
0000001111110000000000000000011111110000000000000000011111100000
0000001111110000000000000000011111110000000000000000011111100000
0000001111110000000000000000011111110000000000000000011111100000
0000001111111000000000000000011111111000000000000000111111100000
1111111111111111100000111111111111111111000001111111111111111111
1111111111111111100000111111111111111111000001111111111111111111
1111111111111111100000111111111111111111000001111111111111111111
```

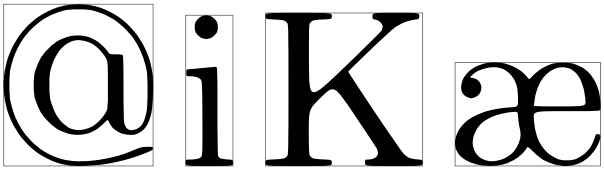
In the days of sending bitmaps to printers, possibly wrapped in a PostScript file, one could often recognize the characters from blobs like this. You might also remember some of that line printer art. At any rate, it shows that 600 dots per inch is much less than it sounds. A decent 2400 dpi (dots per inch) is more normal these days for printing on presses (with ink) but although there are 1200 dpi laser printers,

doi.org/10.47397/tb/45-1/tb139hagen-tracefonts

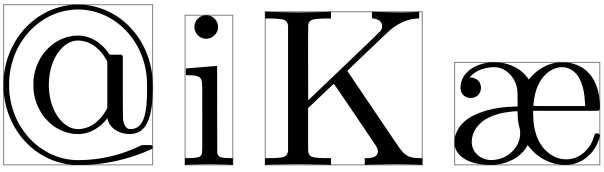
600 became the norm. After all, toner particles are not that small. For quite a while the authors used high speed OCE low temperature toner printer (first 508 dpi, later 600 dpi) and these were visually superior to most of what the competition had — but OCE never quite managed to do the same in color (only in lab testing, not reaching the market). Nowadays we use HP full width high speed inkjet printers that give a rather good quality full color experience at 600 dpi. This is what a scaled-up 2400 dpi bitmap look like:



This already looks more crisp although we can go back to the smoother variant by setting a higher threshold in potrace:



We can go higher. In the next rendering we use 7200 dpi bitmaps and now we see some details that didn't show up before. Keep in mind that subtle details might not be noticed in 10 point running text.



The top of the 'K' now has a little dent (likely not visible except with magnification). If you ever viewed a document with Latin Modern outlines you might recognize this. It is a side effect of outlines having that information while a bitmap needs to get the exact bits, which won't happen if such a dent stays beyond the pixel threshold. We now are ready for some real text. We define two font features to load bitmap and outlines, respectively:

```
\definefontfeature[whateverpk][default]
 [reencode=ontarget-cmr.enc,bitmap=pk]
\definefontfeature[whateverpt][default]
 [reencode=ontarget-cmr.enc,bitmap=outline]
```

and some colors:

```
\definecolor[pkcolor][r=1,t=.5,a=1]
\definecolor[ptcolor][b=1,t=.5,a=1]
```

We use the following three font definitions in three overlaid examples (figure 1). The results are close enough to justify a closer look at the possibilities.

```
\definefont[PKdemoA]
 [file:lmroman10-regular.otf*default sa 1.2]
\definefont[PKdemoB]
 [file:ontarget-cmr10.tfm*whateverpk sa 1.2]
\definefont[PKdemoC]
 [file:ontarget-cmr10.tfm*whateverpt sa 1.2]
```

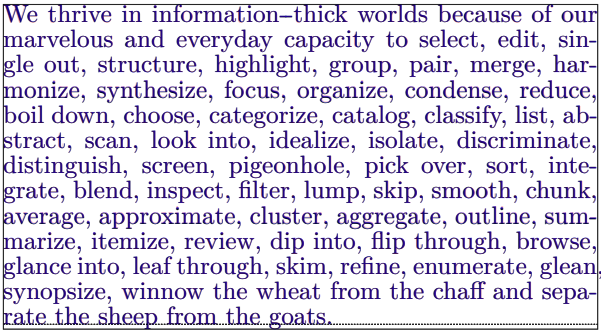


Figure 1: Overlaid regular, bitmap and potraced text.

In figure 2 we see from top to bottom: Latin Modern OpenType outlines, a bitmap Computer Modern and a potraced Computer Modern. The fourth line has the bitmap and potraced overlaid. Figure 3 shows a small portion of the last one as seen on screen. Blown up, some drift is seen but so far we didn't find a way to get rid of it. Some is due to the way glyph streams get rendered and synchronized (after spacing, for instance).

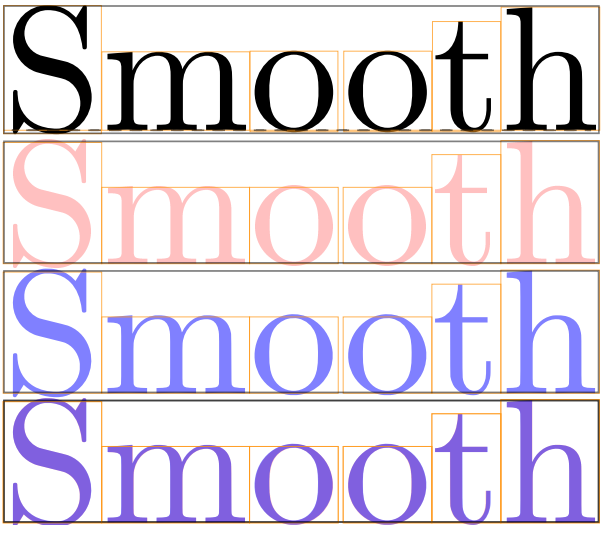


Figure 2: Bitmap and potraced compared; from top to bottom: OpenType, PK, potraced and overlaid.

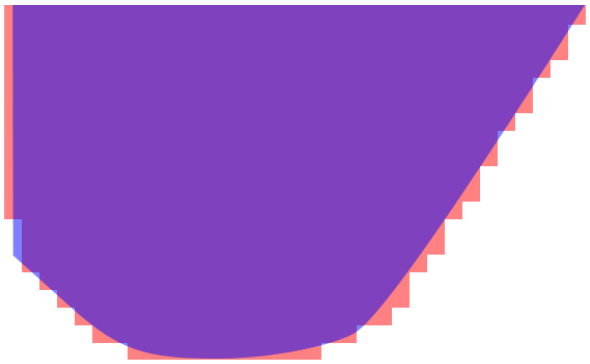


Figure 3: An enlarged clip of the overlay.

When not overlaid we get this for a normal Latin Modern Regular:

We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

And this for a Computer Modern Roman PK bitmap:

We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

The same Computer Modern Roman outlined by potrace gives this:

We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

Because these are outlines we can scale them nicely with `\glyphscale 800` here:

We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

We can scale further, for instance with an extra `\glyphxscale 1200`. This can of course also be done with bitmaps but outlines are a safer bet.

We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

The conversion happens in the backend and can have some impact on the runtime but we cache the outlines, so a subsequent run is faster. Of course outlines are more efficient than bitmaps in terms of bytes and viewers also tend to render them better than bitmaps, which, especially at a lower resolution, can look pretty bad (in some viewers).

One might wonder if bitmaps are worse than outlines. When we use a reasonable resolution there is no need to generate more than one size, and in ConTEXt we assume this anyway because we scale all fonts to 10 big points and scale that shared instance on demand. The 600 dpi 'm' that we showed has 64 pixels in the horizontal direction. So, 75 of these (a line of text) need 4800 pixels, which is more than enough for a 4–8 display. Unfortunately viewers still render a bitmap font somewhat badly.

Let's dive a little into why bitmaps might render suboptimally in PDF viewers. Consider these three lines typeset in our three fonts:

```
\PKdemoA Smooth \vskip.1ex
\PKdemoB Smooth \vskip.1ex
\PKdemoC Smooth \vskip.1ex
```

These 'Smooth' lines (this time in 12pt) end up in the PDF file as follows :

```
BT
/F1 10 Tf
1.195514 0 0 1.195514 0 30.316793 Tm
 [<000100020003>-28<000300040005>] TJ
/F2 10 Tf
1.195514 0 0 1.195514 0 15.415656 Tm
 [<010203>-28<030405>] TJ
/F3 10 Tf
1.195514 0 0 1.195514 0 0.514763 Tm
 [<010203>-28<030405>] TJ
ET
```

This code first switches to font `/F1`, a wide outline font so we have four-byte indices (in angle brackets). Next we trigger `/F2`, the bitmap variant and finally the potraced `/F3`; these are both Type 3 fonts so they get two-byte indices. We don't scale except to the 10 big points design size. After such a switch comes lines of text and there we do scale, here by 1.195514 in both directions. We're slightly off 1.2 because the PDF font system (by tradition) is set up in PostScript (big) points so we need to scale up a little from 12pt to 12bp. Scaling an outline is translated (in the end) to some factor and

the renderer can keep the device into account when it comes to rounding. With bitmaps it's different, because these are not mathematically-defined fonts, but some image that gets scaled. This can introduce the first inaccuracy. An inline bitmap in PDF is given between `ID` and `EI` operators, as demonstrated in the next two charproc entries for the Type 3 bitmap font (reformatted and abridged to save space):

```
21 0 obj
<< /Length 40708 >>
stream
556 0 55 -22 498 703 d1
q
442 0 0 725 55 -22 cm
BI
  /W 442 /H 725 /IM true /BPC 1 /D [1 0]
ID ...bytes...
EI
Q
endstream endobj
```

Notice the difference in the `/Length`:

```
22 0 obj
<< /Length 42784 >>
stream
833 0 33 0 809 440 d1
q
775 0 0 440 33 0 cm
BI
  /W 775 /H 440 /IM true /BPC 1 /D [1 0]
ID ...bytes...
EI
Q
endstream endobj
```

In contrast, a character in the potraced outline font looks like this:

```
31 0 obj
<< /Length 3678 >>
stream
556 0 55 -22 498 703 d1
q
1 0 0 1 55 -22 cm
172 723.96045697 m 144.84445441 720.6382363 ...
Q
endstream endobj
```

The length is much smaller and the outline shape is just a sequence of moveto (`m`), lineto (`l`) and curveto (`c`) operators mixed with numbers.

```
32 0 obj
<< /Length 4260 >>
stream
833 0 33 0 809 440 d1
q
1 0 0 1 33 0 cm
68.5 434.44174379 m 32.2 431.50947593 1.9375 ...
Q
endstream endobj
```

Experiments demonstrated that it's better to use rounded widths because otherwise (at least in SumatraPDF) we get some accumulated drift. The bitmap variants have a transform matrix like this:

```
442 0 0 725 55 -22 cm
775 0 0 440 33   0 cm
```

and the potraced outlines have:

```
1 0 0 1 55 -22 cm
1 0 0 1 33   0 cm
```

This means that a bitmap again gets scaled, luckily by an integer, but still there is some inaccuracy. In the end, we get the bits put on screen and especially at small scales we end up with artifacts in positioning and scaling. Where the font renderer is optimized for (indeed) rendering fonts, the bitmap renderer isn't. In figure 4 we see bitmaps being rendered bolder when they become smaller, because in the end, even at high resolutions, we're not talking pixels but bits (that can occupy multiple pixels). Outline fonts talk pixels, bitmap fonts speak in bits.

smallest small normal

Figure 4: Three zoom levels compared.

We haven't yet discussed how we got the bitmaps that we used. You might have noticed in the examples that there are some differences with the outline when it comes to dimensions. This is partly due to the fact that where Latin Modern is an OpenType font with no limits to dimensions, Computer Modern has to accommodate the limited number of heights, depths and widths that the TFM format permits. Think of arbitrary values of height compared to categories of height.

When you generate a bitmap you rely on scripts that do the work and these work together with so-called printer modes as defined in the METAFONT file `modes.mf` (`https://ctan.org/pkg/modes`). These modes are for printers which means that there can be compensation going on: rounding up or down of points exceeding bounding box edges, the size of printer pixels (toner, ink) and accuracy of positioning them, etc. In our case, when we went for 8000 dpi we ended up with a device that did more compensation than needed. Better is to investigate time in figuring out how to control the machinery to cook up (maybe) 7200 dpi bitmaps because down the inclusion route there is some division by 72. If we decide to play

a bit more, we might as well first figure out how to control the bitmap generation and see if we can come up with this 7200 resolution. Not only does it divide nicely by 72 (for display) but also by 600 (for the average printer). To what extent that matters is to be seen.

A bitmap font (instance) is generated by METAFONT and driven by a (printer) mode defined in `modes.mf`. We added this one:

```
mode_def potrace =
    mode_param (pixels_per_inch, 2 * 3600) ;
    mode_param (blacker, 0) ;
    mode_param (fillin, 0) ;
    mode_param (o_correction, 1) ;
    mode_common_setup_ ;
enddef;
```

The `2 * 3600` is a trick to get around METAFONT maxing out at 4096 but internally being capable to deal with larger numbers. Of course we forgot to run `fmtutil-sys --byfmt mf` which is needed to get these modes in the format file, but eventually we managed to generate the 7200 dpi PK file for `cmr10`. Generating is easiest done with pdfTEX with `\pdfmapfile {}`, which wipes the mapping to a Type 1 file.

We mentioned using MetaPost in our first attempts to get an idea how well potrace can vectorize the bitmaps. Here is how that is done:

```
\startluacode
  local f = fonts.handlers.tfm.readers.loadpk
              ("cmr10.pk")
  if f then
    local g = f.glyphs[string.byte("R")]
    if g then
      local b
        = fonts.handlers.tfm.readers.showpk(g)
      potrace.setbitmap("demo",b)
    end
  end
\stopluacode

\startMPpage[offset=1ts]
    draw lmt_potraced [
        stringname = name,
        value      = "1",
    ];
\stopMPpage
```

In figure 5 we demonstrate three such renderings. Watch how the number of points increases as the shape gets better. In figure 6 we show the potraced variant alongside the OpenType outline. Left is the default potrace output, next comes the regular OpenType (here CFF) outline, and at the right we see two potraced results, both with `optimize = true` passed; the first has the default tolerance of 0.2 and

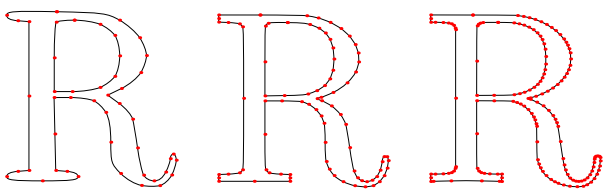


Figure 5: Using MetaPost for analysis.

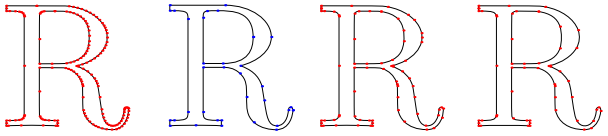


Figure 6: Comparing potraced bitmaps with Type 1.

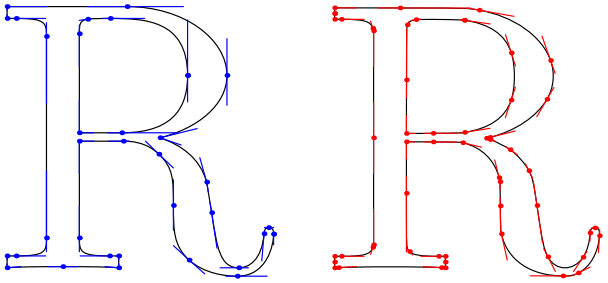


Figure 7: Comparing control points.

the second uses 0.5 and thereby has less points. For sure the middle one has less points which is nice, but the potrace ones are not that excessive so we can live with it. We've run into cases (especially math) where the regular outlines are also not perfect. Most will go unnoticed anyway given the small size at which glyphs are normally rendered. When you look closely at the rightmost output you'll notice that the bend in the stems makes for more points which is an indication that the METAFONT output might actually be more subtle. In figure 7 we also show the controls and you see subtle differences in the angles there. Also note that when there are no lines that indicates that there is likely a lineto instead of a curveto.

If you like the look of these shapes, take a look at Volume E of Don Knuth's "Computers & Typesetting" series. An incredible amount of work went into the details of the fonts and you'll run into brackets, beaks, crisps, notches, slabs, juts, dishes and more elements and parameters that are used. For instance the serifs as seen in the 'R' are actually made programmatically (so that they can be discarded in the sans shapes). If you check out the proof sheet of the 'R' you will only see the basic points that describe

the character, not the points we see above, after conversion to (any kind of) outline.

So now that we can turn a PK into a proper outline we're done, right? Well, not entirely. Because we have relatively simple shapes (moveto, lineto and curveto) we can directly go to CFF and avoid the MetaPost to Type 3 conversions. Because we already can load (and adapt) CFF outlines it is not that complicated to do the reverse. The backend already can include them so we can also borrow code there.

When going from potrace output to CFF, we need a high resolution, so we started out again with 7200 dpi. Although in the end we were quite satisfied with the results, we tested with 20000 dpi and thought it looks even better than the Latin Modern successor (although one can argue about it). Some first experiments showed that it was doable but it actually took a whole day to (test and) decide how to get better results. For instance, MetaPost uses floats while a renderer uses integers, so we run into rounding issues if we delegate that (although we can include floats in CFF, it is not a success). When overlaying the MetaPost output and the CFF shapes the latter was quite disappointing: weird protrusions and bubbles. Of course one may doubt the implementation, but double and triple checking showed that we use the same numbers.

The results improved a lot when the results from potrace were multiplied by 10 (or 20) effectively giving very huge glyphs (on a 10000 unit canvas) and in the page stream scaling down by that factor. By then using rounded values we got enough precision to get the CFF results close to the (always) high quality MetaPost rendering. In figure 8 we can see how close they became.

The next question is, how do we control this: PK, MetaPost Type 3 or native CFF? Even more challenging is that we had wanted to use different resolutions, and mix these three methods, if only because we want to be able to experiment and document the mix. That means that it has to be available not only in the font feature mechanism, which is rather trivial, but also in the backend, which then involves a font with the same name (say `CMR10`) to be rendered differently, so we need different handlers, distinctive caching of streams, etc. In the end we got there. It is even possible to mix variants with different potrace parameters in one document.

We start with CFF definitions. In figure 9 we show three resolutions overlaid, with the 7200 dpi variant below. In figure 10 we show the default and optimized (fewer points) traces.

```
\definefontfeature[CMRCFF]
  [reencode=ontarget-cmr.enc,bitmap=cff]
```

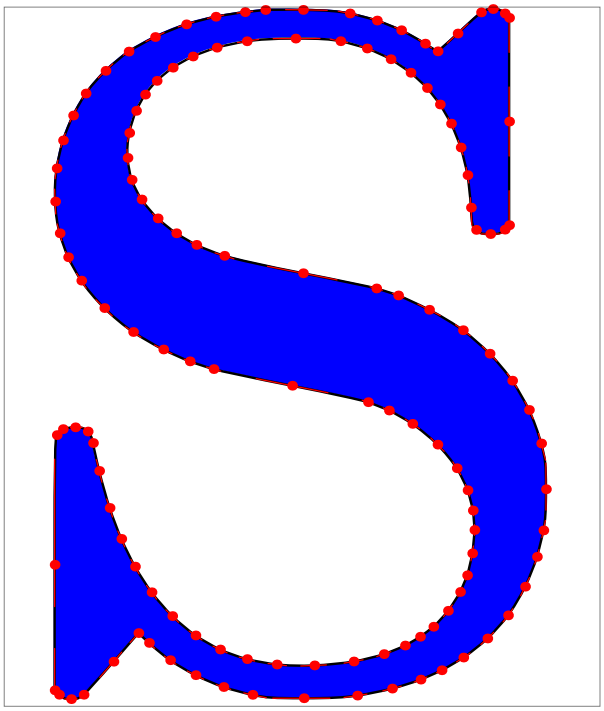


Figure 8: A MetaPost rendering on top of its CFF counterpart.

We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

Figure 9: Above: 7200, 2400, 600 dpi CFF variants overlaid; bottom: cf. 7200 dpi CFF variant alone.

```
\definefontfeature[MyFontCffA]
  [default,CMRCFF][resolution=7200]
\definefontfeature[MyFontCffB]
  [default,CMRCFF][resolution=2400]
\definefontfeature[MyFontCffC]
  [default,CMRCFF][resolution=600]
\definefontfeature[MyFontCffD]
  [default,CMRCFF][resolution=7200,
                   potrace={optimize=true}]
```

The overlays look fuzzy, demonstrating the need for high resolutions. Font definitions are done as follows; we only show one definition:

We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

Figure 10: Above: Normal and optimized 7200 dpi variants overlaid; below: cf. 7200 dpi variant alone.

We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

Figure 11: Above: 7200, 2400, 600 dpi PK variants overlaid; below: cf. 7200 dpi PK variant alone.

```
\definefont[MyFont]
  [file:ontarget-cmr10.tfm*MyFontCffA]
```

These definitions are used in figure 11. The differences aren't immediately apparent in small print but zoom in (if you're online) and you'll understand why we need more than 600 dpi to feel comfortable.

Finally we show the MetaPost-generated threesome in figure 12 and these look quite reasonable. One thing to keep in mind when wrapping shapes into a Type 3 font is that one has to make sure that color keeps working.

So, how useful is all this? Maybe it's time for a revival of METAFONT. Or maybe we can get some old designs out of the archives where they got tagged obsolete and use them again. Or maybe it's just for the fun of it. We started out with PK bitmaps that we need to support anyway. Next we were curious how well a traced outline would look, and for that using a MetaPost Type 3 font makes sense. Then we took the challenge to turn potrace output into a CFF OpenType font. We could now make a whole tool chain but it makes little sense: we can do it in ConTEXt, so we're fine, and we don't expect widespread usage

Figure 12: Above: 7200, 2400, 600 dpi MetaPost variants overlaid; below: cf. 7200 dpi MP variant alone.

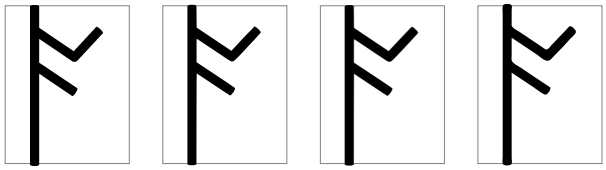


Figure 13: A test with one of the allrunes fonts. Left is using the PK at a 7200 dpi resolution. Next is the potraced 7200 PK original outline. Third from left is the generated CFF. Right is the shipped PFB.

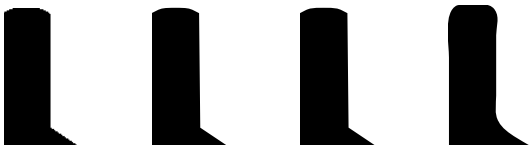


Figure 14: Enlarged clips of the variants in figure 13.

outside the TEX ecosystem. Next on the agenda is to locate some interesting METAFONTs and see if they can be put to use. In case you wonder how these eight bit fonts fit into the Unicode ecosystem, don't worry, we can use the virtual font mechanism to hook shapes into existing fonts (which is what we do anyway) or create combined fonts. We can even make variable fonts using METAFONT, but for that we need to become experienced designers first. As a teaser we show a character from the runes font by Carl-Gustav Werner in figures 13 and 14. Mikael, knowing the author, promised to come up with a proper encoding for that one, so stay tuned.

◇ Hans Hagen
Pragma ADE

◇ Mikael P. Sundqvist
Department of Mathematics
Lund University
mickep (at) gmail dot com