Understanding Type1 fonts – for fun and for profit

André Wobst

35 MV DANTE e.V.

14. September 2006

Rosenheim

Contents

This talks is about technical details of fonts.
„How is defined, how a letter looks like?“

- font types
- Type1 fonts (pfb-files + metric files)
- outline of a glyph
- decode hidden messages
- modify single glyphs
- path transformations
- questions

Font types

Typical font types in the TEX world:

- METAFONT fonts (developed by Knuth together with TEX)
- Type1 fonts (Adobe outline format for PostScript)
- TrueType fonts (Apple/Microsoft outline format)

Special features of METAFONT fonts

- METAFONT is a programming language for fonts
- METAFONT creates rasterized versions (pk files) for specific output devices (resolution and other device features)
- no mathematical description of the outline of the glyphs

Fonts in TEX

- TEX knows/understands almost nothing about fonts
- TEX knows the font metrics only (including ligatures and kerning; own format: tfm files)
- dvi files contain only font names and position information for each of the glyphs
- original idea of dvi drivers: create rasterized version of the complete page
- specials: no encoding, “virtual fonts”

rasterized output is typically **not** used nowadays any more, since computer modern fonts are available as Type1 fonts and those are resolution independend

Type1 fonts

A Type1 font is composed out of two files:

afm file: metric information and other meta data (e.g. ligatures, kerning) → clear text

pfb file: glyph information → encrypted (encryption is published now)

To use it in TEX:

- tfm file necessary (might be created from the afm file)
- backend support (e.g. dvips, pdfTEX and so on.; assignment by psfonts.map, pdftex.map)

Documentation from Adobe: t1_spec.pdf

Analyzing fonts

Why?

for fun: How is the outline encoded?

for profit: Embed used gylphs in the output file only!

Where? (or rather what the speaker can tell!)

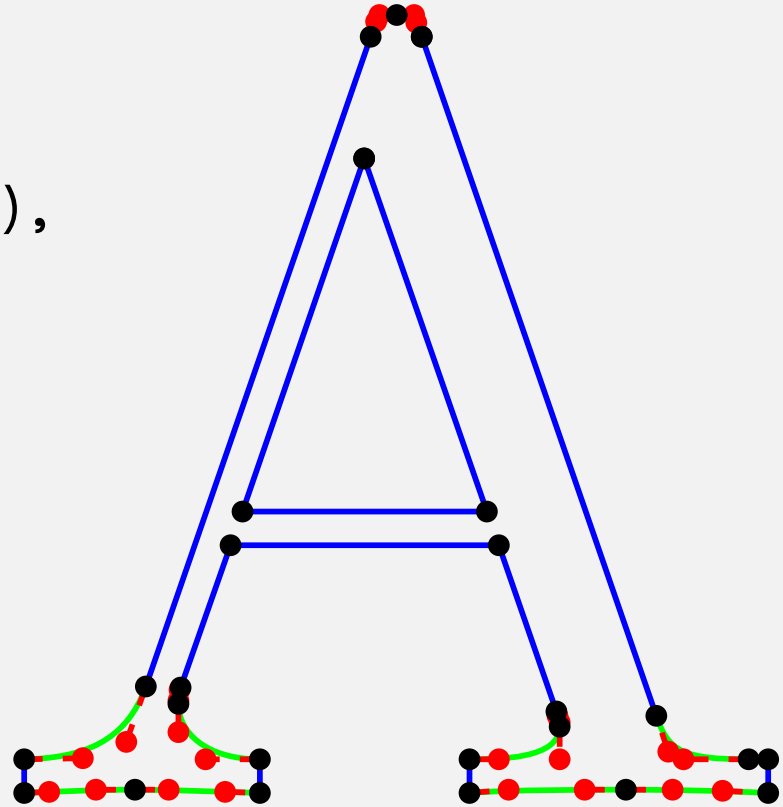
- PyX [http://pyx.sourceforge.net/]:
  Python package for the creation of PostScript and PDF
- very good TEX integration → process dvi files
- support for Type1 fonts

Filled glyph path

```
from pyx import *
from pyx.font.t1font import T1pfbfont

f = T1pfbfont("cmr10.pfb")

c = canvas.canvas()
c.fill(f.getglyphpath("A", 500))
c.writeEPSfile("glyphpath")
```

Elements of a glyph path

```
from pyx import *
from pyx.font.t1font import T1pfbfont

f = T1pfbfont("cmr10.pfb")

c = canvas.canvas()
c.draw(f.getglyphpath("A", 500),
       [deco.shownormpath()])
c.writeEPSfile("glyphpath")
```



What's hidden in cmr10.pfb?

Hidden message (including some additional line breaks to fit the line width of this presentation):

```
BKPH TeX CM
1992 Jul 21 16:14:27

Computer Modern fonts were designed by
Donald E. Knuth at Stanford,
based in part on (American) Monotype Corporation Modern 8A.
Font outlines generated by Doug Henderson
at Blue Sky Research,
using the ScanLab tool created by Ian Morrison
of Projective Solutions.
Character level hinting by Blenda Horn of Y&Y.
Adobe Type 1 encoding designed by Berthold K.P. Horn of Y&Y.
Barry Smith of Blue Sky Research coordinated the effort.
Long live TeX!
```

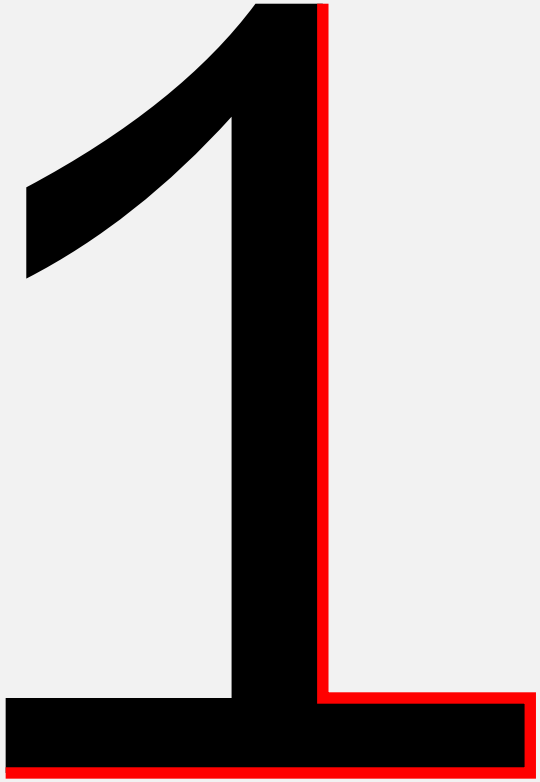
Decrypt hidden messages

- data in pfb files are encrypted (originally for license sale)
- the encryption contains random numbers (seeds)
- might contain hidden messages, which are decrypted together with the real content

```
from pyx.font.t1font import T1pfbfont, decoder

f = T1pfbfont("cmr10.pfb")
f._data2decode()
data = []
for code in f.subrs:
    data.append(decoder(code, f.charstringr, 0)[:4])
for glyph in f.glyphlist[:81]:
    data.append(decoder(f.glyphs[glyph], f.charstringr, 0)[:4])
print "".join(data)
```

Changing a glyph

org.pfb:



mod.pfb:

Changing a glyph

```
from pyx.font.t1font import T1pfbfont

f = T1pfbfont("org.pfb")
cmds = f.getglyphcmds("one")
for i, cmd in enumerate(cmds):
    print "Zeile %2i: %s" % (i, cmd)

print "----"
cmds[5] = 250
cmds[7] = 84
cmds[9] = 708
cmds[11:15] = []
cmds[29:33] = []

f.setglyphcmds("one", cmds)
for i, cmd in enumerate(cmds):
    print "Zeile %2i: %s" % (i, cmd)
f.outputPFB(open("mod.pfb", "wb"))
```

output (part):

```
Zeile  5: 2
Zeile  6: hmoveto
Zeile  7: 483
Zeile  8: hlineto
Zeile  9: 69
Zeile 10: vlineto
Zeile 11: -191
Zeile 12: hlineto
Zeile 13: 639
Zeile 14: vlineto
----
Zeile  5: 250
Zeile  6: hmoveto
Zeile  7: 84
Zeile  8: hlineto
Zeile  9: 708
Zeile 10: vlineto
```

Simple small caps

Faked small caps:

IowanOldStyleBT

Simple small caps

```
from pyx import *
from pyx.font.t1font import T1pfbfont

f = T1pfbfont("biwr8a.pfb")
c = canvas.canvas()
x1_pt = x2_pt = 0
for s in "IowanOldStyleBT":
    if s.islower():
        p, wx_pt, wy_pt = f.getglyphpathwxwy_pt(s.upper(), 50)



    else:
        p, wx_pt, wy_pt = f.getglyphpathwxwy_pt(s, 100)


    c.fill(p, [trafo.translate_pt(x1_pt, 100)])
    x1_pt += wx_pt
c.writeEPSfile("caps")
```

Better small caps

Faked small caps without and with width correction:

IowanOldStyleBT

IowanOldStyleBT

. . . by PyX path transformations

```
from pyx import *
from pyx.font.t1font import T1pfbfont

f = T1pfbfont("biwr8a.pfb")
c = canvas.canvas()
x1_pt = x2_pt = 0
for s in "IowanOldStyleBT":
    if s.islower():
        p, wx_pt, wy_pt = f.getglyphpathwxwy_pt(s.upper(), 50)
        c.fill(p, [trafo.translate_pt(x2_pt, unit.topt(0.04)),
                   deformer.parallel(-0.04, sharpoutercorners=1)])
        x2_pt += wx_pt + 2*unit.topt(0.04)
    else:
        p, wx_pt, wy_pt = f.getglyphpathwxwy_pt(s, 100)
        c.fill(p, [trafo.translate_pt(x2_pt, 0)])
        x2_pt += wx_pt
    c.fill(p, [trafo.translate_pt(x1_pt, 100)])
    x1_pt += wx_pt
c.writeEPSfile("caps")
```

Nice small caps

Faked small caps without and with width correction at good parameters:

IowanOldStyleBT

IowanOldStyleBT

. . . by useful parameters

```
from pyx import *
from pyx.font.t1font import T1pfbfont

f = T1pfbfont("biwr8a.pfb")
c = canvas.canvas()
x1_pt = x2_pt = 0
for s in "IowanOldStyleBT":
    if s.islower():
        p, wx_pt, wy_pt = f.getglyphpathwxwy_pt(s.upper(), 75)
        c.fill(p, [trafo.translate_pt(x2_pt, unit.topt(0.02)),
                   deformer.parallel(-0.02, sharpoutercorners=1)])
        x2_pt += wx_pt + 2*unit.topt(0.02)
    else:
        p, wx_pt, wy_pt = f.getglyphpathwxwy_pt(s, 100)
        c.fill(p, [trafo.translate_pt(x2_pt, 0)])
        x2_pt += wx_pt
    c.fill(p, [trafo.translate_pt(x1_pt, 100)])
    x1_pt += wx_pt
c.writeEPSfile("caps")
```

Questions?

contact@wobsta.de
http://www.wobsta.de/