Fell Types in ConTEXt

Luigi Scarso

Sommario

In questo articolo vengono illustrate installazione ed uso di un font OpenType con ConTEXt-`mkiv`, i *Fell Types*, descritti da M. Dominici nell'articolo "Utilizzo di caratteri TrueType con LATEX. Un esempio pratico: i *Fell Types*". Viene inoltre descritto un particolare problema causato da un parametro di progettazione dei font e discussa una soluzione offerta da ConTEXt-`mkiv`.

Abstract

In this paper we will briefly show how to install and use an OpenType font with ConTEXt-`mkiv`. We will use the *Fell Types* fonts as in M. Dominici's paper "Utilizzo di caratteri TrueType con LATEX. Un esempio pratico: i *Fell Types*". A problem with an unusual font parameter is described and a solution offered by ConTEXt-`mkiv` is discussed.

1 Introduzione

Come è noto, LuaTEX è nato anche poter gestire in modo efficace i font OpenType. Giunto ormai alla versione `0.52` possiamo ritenerlo adeguato per esplorare questa parte, ma piuttosto che addentrarci nei dettagli ed ipotizzare un possibile utilizzo ad alto livello, è preferibile guardare subito il formato che, procedendo in simbiosi con lo sviluppo di LuaTEX, offre all'utente finale la possibilità di un uso immediato: ConTEXt-`mkiv`.

Nell'articolo "Utilizzo di caratteri TrueType con LATEX. Un esempio pratico: i *Fell Types*" (Dominici, 2007), M. Dominici illustra come utilizzare in generale i font TrueType in LATEX, ed in particolare descrive una famiglia di font, i Fell Types, disponibili in due formati, OpenType con *outline* in PostScript (ovvero curve di Bézier cubiche) e TrueType con le proprie outline (curve di Bézier quadratiche).

In questo articolo ci concentreremo sulla versione OpenType, sebbene ConTEXt-`mkiv` possa gestire *direttamente* i font Type1, TrueType ed OpenType. Il motivo risiede nel fatto che la versione OpenType ha un `FontMatrix` pari a 2048 mentre usualmente è pari a 1000, e vedremo le conseguenze di questa scelta da parte dell'autore del font.

2 Breve panoramica sui font OpenType

Un font OpenType permette di gestire con un unico file binario tre fondamentali tipi di fonts:

1. `Type1`, un formato di font definito da Adobe in cui il disegno del carattere (il glifo) è essenzialmente descritto da curve di Beziér di terzo ordine espresse in un sottoinsieme del linguaggio PostScript (font vettoriale);
2. `TrueType`, un formato di font vettoriale definito da Apple e MicroSoft, in cui il glifo è descritto da curve di Beziér di secondo ordine con un linguaggio dedicato;
3. bitmap, un tipo font in cui il glifo è una immagine in bianco e nero. Le specifiche TrueType permettono di incorporare un file tipo immagine nella definizione di un glifo e queste specifiche sono valide anche per i font OpenType.

OpenType permette di gestire anche *collezioni TrueType* di font (font `.ttc`), ma non di "fondere" glifi `Type1` con glifi `TrueType` in un unico file e neppure collezioni miste `Type1` e `TrueType`; è invece possibile definire glifi bitmap e glifi vettoriali `TrueType`, sia in collezioni che in un font singolo. Il numero massimo di glifi è fissato a 65536.

Rimandando a Wikipedia (2010) per una rapida panoramica ed a Haralambous (2007) e OpenType specification per approfondimenti, interessa sottolineare in questo contesto i seguenti aspetti:

• è possibile definire un *encoding* (la mappa tra il nome del carattere del font ed il suo glifo) compatibile con Unicode. Dato che a sua volta LuaTEX accetta testo sorgente in formato Unicode con codifica UTF-8 (la codifica più compatta per le lingue latine, ma non necessariamente la migliore in assoluto, cfr."Getting Ready for Unicode" (Knuth, 2010)) è quindi naturalmente predisposto verso una codifica Unicode del font: in questo modo i caratteri di un testo sorgente `.tex` in formato Unicode vengono mappati in modo semplice verso i corrispondenti glifi;

• un font può contenere delle varianti attivabili a richiesta. Il termine "variante" non è da intendersi nel senso tradizionale di variazione del peso (light,normal, boldface etc.) od inclinazione (ad es. roman vs. slanted) ma alla possibilità di controllare il posizionamento del glifo e di sostituire il glifo stesso in modo sensibile al contesto *ed in modo organizzato, coerente e normato*. Per un utente METAFONT non sono concetti nuovi, solo molto più estesi e sofisticati: in Haralambous (2007) vengono descritte poco meno di cento "varianti" o meglio *feature*, per usare il termine corretto, organizzate in base al tipo di linguaggio latino, se-

mitico, indiano/sud-est asiatico ed ideografico. È importante sottolineare che *le varianti del font non richiedono la modifica del testo sorgente per essere attivate*;

• un glifo è definito in un sistema di coordinate (x, y) intere adimensionali. Normalmente queste coordinate individuano una regione quadrata del piano, e per motivi storici i font `Type1` definiscono usualmente un quadrato di 1000 unità mentre i font `TrueType` definiscono usualmente un quadrato di 2048 unità (in ogni caso, le specifiche impongono una ragione limite data dal rettangolo definito dai vertici (-16384,-16384) e (16383,16383)). Per i font `Type1` la regione è definita tramite una matrice di trasformazione `FontMatrix=[sx 0 0 sy 0 0]` in cui normalmente `sx=sy=0.001`: in questo modo ogni parametro relativo a curve, spostamenti etc. pari a d unità del font diventano $0.001 * d$ punti PostScript (il lettore orientato verso aspetti matematico-computazionali potrà trovare interessante il corso di Computer Aided Geometric Design (Sederberg e Peterson, 2010)).

Infine è da notare che il formato OpenType è anche un descritto in Information technology.

3 Installazione font

ConTEXt viene distribuito in due modi:

1. attraverso la TEXlive;
2. attraverso la `minimals`.

Per questo progetto utilizzeremo la `minimals`, installabile seguendo le istruzioni descritte in `http://wiki.contextgarden.net/ConTeXt_Minimals`.

La struttura della `minimals` è

```
bin
tex
 texmf
 texmf-cache
 texmf-context
 texmf-linux
 texmf-local
 setuptex
 setuptex.bat
 setuptex.csh
 setuptex.tmf
```

ed andiamo a copiare i file OpenType

```
FeDPit27C.otf
FeDPrm27C.otf
FeENit27C.otf
FeENrm28C.otf
FeFCit27C.otf
FeFCrm27C.otf
FeFlow1.otf
FeFlow2.otf
FeGPit27C.otf
FeGPrm27C.otf
FePIit27C.otf
FePIrm27C.otf
FeTLrm27C.otf
```

all'interno della cartella

```
tex/texmf-fonts/fonts/
                  opentype/public/felltypes
```

A questo punto aggiorniamo la `minimals` della presenza dei nuovi font con

```
#> mtxrun --script fonts --reload
#> luatools --generate
```

e testiamo il tutto con

```
%%test.tex
\font\test=FeDPit27C
\starttext
\test\input tufte\par
\stoptext

#> context test.tex
```

4 Uso

I Fell Types contengono 6 famiglie di font, ciascuna nella variante Roman e Italic:

1. THE ENGLISH ROMAN AND ITALIC, da usare nella dimensioni di 13.5 punti,
2. THREE LINES PICA, 48 punti
3. FRENCH CANON, 39 punti,
4. DOUBLE PICA, 21 punti,
5. GREAT PRIMER, 17 punti,
6. DE WALPERGEN'S PICA, 12.5 punti

Prima di mostrare come ConTEXt-`mkiv` organizza i font utilizziamo un semplice programma in LuaTEX per ricavare alcune informazioni.
Il cuore del programma è la funzione LuaTEX `font,warning=fontloader.open(<filename>)` che permette, in caso di successo, di caricare un font e di renderlo disponibile con `my_fontdata=fontloader.to_table(font)`.
Nel nostro caso salviamo le informazioni in un file `.txt` (`FeDPrm27C.txt` nell'esempio), per poterle consultare in un secondo momento.

```
%%my_font_loader.tex
\startluacode
local function my_loader_otf()
 rep = io.open('FeDPrm27C.txt','w')
 local f,w = fontloader.open('FeDPrm27C.otf')
 my_fontdata = {}
 if w  then
  rep:write("Warning:\n")
  for k,v in pairs(w)
   do rep:write(string.format("%s = %s\n",
                              k,tostring(v)))
  end
  rep:write("=========\n")
 end
 if f then
   my_fontdata = fontloader.to_table(f)
   fontloader.close(f)
 end
 for k,v in pairs(my_fontdata) do
  rep:write(string.format("%s = %s\n",
                       k,tostring(v)))
```

```
 end
 rep:close()
end
my_loader_otf()
\stopluacode
\starttext\relax\stoptext
```

Eseguendo
`#> context my_font_loader.tex`
verrà scritto un report minimale relativo al font `FeDPrm27C.txt`. Non è difficile estendere il programma per il report di font arbitrario (cfr. LuaTEX development team (2010)).

Per utilizzare il font in ConTEXt-`mkiv` bisogna poi definire una `typeface`, ovvero un set coordinato di font identificato da un nome.
Ad esempio, pensiamo al "typeface" `postscript` composto dai "fonts" `times`, `helvetica` e `courier`. I suoi elementi (ad es. `times`) saranno a loro volta collezioni di font che descrivono le varianti (come Serif, SerifItalic, SmallCaps nelle rispettive dimensioni) e questi elementi sono a loro volta definiti tramite un `typescript`.

Nel nostro caso definiamo a titolo di esempio le typeface `felldewalpergenpica`, `felldoublepica`, `fellgreatprimer`, `fellenglish`, nelle varianti Serif e SerifItalic e le associamo al corrispondente file non tramite il nome del file `.otf` (cosa per altro possibile) ma tramite il `fontname` del font. Questo è uno dei parametri del font ottenibile con lo script precedente e memorizzato in una cache interna di ConTEXt-`mkiv`.

Ogni set contiene dunque un solo elemento; potevamo comunque seguire una strada opposta, ovvero definire un unico set `felltype` e "comporlo" con gli elementi, `felldewalpergenpica`, `felldoublepica`, `fellgreatprimer` e `fellenglish`.

```
\starttypescript[serif][felldewalpergenpica]
%
 \definefontsynonym
 [Serif][name:IM_FELL_DW_Pica_PRO_Roman]
 \definefontsynonym
 [SerifItalic]
 [name:IM_FELL_DW_Pica_PRO_Italic]
%
\stoptypescript

\starttypescript[serif][felldoublepica]
%
 \definefontsynonym
 [Serif][name:IM_FELL_Double_Pica_PRO_Roman]
 \definefontsynonym
 [SerifItalic]
 [name:IM_FELL_Double_Pica_PRO_Italic]
%
\stoptypescript

\starttypescript[serif][fellgreatprimer]
%
 \definefontsynonym
 [Serif][name:IM_FELL_Great_Primer_PRO_Roman]
 \definefontsynonym
 [SerifItalic]
 [name:IM_FELL_Great_Primer_PRO_Italic]
%
\stoptypescript

\starttypescript[serif][fellenglish]
%
 \definefontsynonym
 [Serif][name:IM_FELL_English_PRO_Roman]
 \definefontsynonym
 [SerifItalic]
 [name:IM_FELL_English_PRO_Italic]
%
\stoptypescript
```

La definizione di ogni set (typeface) è poi banale: dato che il set ha un solo elemento, scegliamo come nome del set il nome del suo elemento.

```
\definetypeface[felldewalpergenpica]
       [rm][serif][felldewalpergenpica]
\definetypeface[felldoublepica]
       [rm][serif][felldoublepica]
\definetypeface[fellgreatprimer]
       [rm][serif][fellgreatprimer]
\definetypeface[fellenglish]
       [rm][serif][fellenglish]
```

A questo punto i font sono disponibili:

```
\setupbodyfont[fellenglish,
               serif,13.5bp]
\starttext
\rm \input browne.tex
\it \input browne.tex

{\switchtobodyfont[fellgreatprimer,
                   serif,17bp]
\rm \input browne.tex\\
\it \input browne.tex
}

{\switchtobodyfont[felldewalpergenpica,
                   serif,12.5bp]
\rm \input browne.tex \\
\it \input browne.tex
}

{\switchtobodyfont[felldoublepica,
                   serif,21bp]
\rm \input browne.tex\\
\it \input browne.tex
}

\stoptext
```

4.1 Accesso alle feature

L'accesso alle feature di un font OpentType non è difficile: essenzialmente si definisce una fontfeature (ad es. `featI`) attivando le feature che ci interessano:

```
\definefontfeature
   [featI]
   [script=latn,
```

```
    aalt=yes,% Access All Alternates
    dlig=yes,% Discretionary Ligatures
    hist=yes,% Historical Forms
    kern=yes,% Kerning
    liga=yes,% Standard Ligatures
    salt=yes,% Stylistic Alternates
    ss01=yes,% Stylistic Set 1
    ss02=yes,% Stylistic Set 2
    ss03=yes,% Stylistic Set 3
    ss04=yes,% Stylistic Set 4
    protusion=no,
    expansion=yes
    ]
```

Si associa la fontfeature tramite un `typescript`:

```
\starttypescript[serif]
                [fellenglishfeatI][default]
 \definefontsynonym
  [Serif]
  [name:IM_FELL_English_PRO_Roman]
  [features=featI]
 \definefontsynonym
  [SerifItalic]
  [name:IM_FELL_English_PRO_Italic]
  [features=featI]
\stoptypescript
```

oppure si può attivare la fontfeature nel corpo del testo con la macro `setff`:

```
\setupbodyfont[fellenglish,13.5bp]
\starttext
{\rm assalire \it assalire}\\
{\setff{featI}
\rm assalire \it assalire}
\stoptext
```

che produce

assalire *assalire*

assalire *assalire*

Anche in questo caso è possibile ricavare le informazioni riguardanti le feature con uno script LuaTEX:

```
--
-- my_fontdata come
-- nello script precedente
--
local gsub = my_fontdata['gsub']
rep:write("GSUB\n")
for k,v in pairs(gsub) do
 rep:write(
  string.format("  %s = %s\n",k,tostring(v)))
 for k1,v1 in pairs(v) do
  rep:write(
   string.format("   %s = %s\n",
                 k1,tostring(v1)))
  if k1 == 'features' then
   for k2,v2 in pairs(v1) do
     rep:write(
      string.format("    %s = %s\n",
                    k2,tostring(v2)))
     for k3,v3 in pairs(v2) do
      rep:write(
       string.format("     %s = %s\n",
                     k3,tostring(v3)))
     end
    end
   end
  end
 end
end
```

5 Un bug di Adobe Reader?

I Fell Types definiscono una `FontMatrix` di 2048 unità e outline descritte da curve di Bézier cubiche. Questa caratteristica è piuttosto rara: normalmente la `FontMatrix` per un font Type1 è pari a 1000 unità, mentre per una outline quadratica è 2048 oppure 1024 unità, ma sono possibili altri valori. AdobeReader 7 non ha alcuna difficoltà a mostrare correttamente questi font particolari, ma, a partire dalla versione 8 fino a quella attuale, Acrobat (e Reader) distorce la visualizzazione del glifo in modo da rendere impossibile la lettura: ecco ad esempio la parola *assalire* come appare

e come invece dovrebbe apparire

assalire

L'aspetto interessante è che l'effetto *non* si manifesta se il font viene salvato all'interno del pdf come font Type1 CFF (cfr. Compact Font Format), mantenendo inalterato il valore di 2048 unità; anche la soluzione di convertire il font ad un Type1 con una `FontMatrix` pari a 1000 unità elimina il "bug", ma la conversione introduce arrotondamenti non voluti ed in ogni caso non sembra permessa dalla licenza del font.

La conversione in Type1 comporta in ogni caso la creazione di un encoding a 256 slot e sostanzialmente si deve percorrere la stessa strada indicata per i TrueType nell'articolo di M. Dominici (Dominici, 2007), ovvero creare tante "versioni" Type1 del font quante sono le codifiche necessarie. Questa strada inoltre è difficile da percorre in ConTEXt-`mkiv`, perché per default i font Type1 vengono comunque salvati nel pdf come se provenissero da un equivalente OpenType al fine di mantenere l'encoding Unicode. Infine, questo "bug" si manifesta anche con `ghostscript`, ma non con `mupdf` ne con `xpdf`.

L'autore ha segnalato la cosa al team di sviluppo di luatex nel marzo del 2009 e, dopo un anno, è stata trovata una soluzione accettabile: `luatex` è stato modificato per

1. normalizzare la `FontMatrix` a 1000 unità, ovvero `FontMatrix=[sx 0 0 sy 0 0]` con `sx=sy=0.001`. Questo comporta però una distorsione: il font risulta ingrandito di circa due volte (per la precisione `2.048`) ;
2. correggere la distorsione introdotta riducendo di 2.048 volte i caratteri del testo (ovviamente solo quelli che hanno la `FontMatrix` inusuale).

In questo modo non si modifica il font originale con il rescaling, riducendo errori di arrotondamento, e non si passa attraverso la conversione al formato `Type1`, evitando ogni problematica di encoding. Questa modifica al programma `luatex` è stata recepita da H. Hagen ed è già disponibile nella nuova minimals; inoltre gli sviluppatori, nonostante le osservazioni precedenti, non sono contrari ad implementare la conversione *automatica* del font in formato `Type1`, riprendendo quindi la strada descritta in Dominici (2007).

A conoscenza dell'autore, attualmente ConTEXt-mkiv è l'unico sistema in grado di gestire i font OpenType con outline Postscript e FontMatrix diverso da 1000 unità.

Una osservazione è però d'obbligo: si tratta di un bug o no? Le specifiche PDF Reference recitano testualmente a pag. 394 (grassetto dell'autore):
"The glyph coordinate system is the space in which an individual character's glyph is defined. All path coordinates and metrics are interpreted in glyph space. ***For all font types except Type 3, the units of glyph space are one-thousandth of a unit of text space****; for a Type 3 font, the transformation from glyph space to text space is defined by a font matrix specified in an explicit FontMatrix entry in the font."*

Il documento riguardante i font `Type1` (Adobe Font Format) recita poi a pag.13 :
"The program inserts eight items (FontInfo, FontName, PaintType, FontType, FontMatrix, Encoding, FontBBox, and UniqueID) into the dictionary. The 1000 to 1 scaling in the Font- Matrix as shown is typical of a Type 1 font program and is highly recommended."

Inoltre nella descrizione del formato CFF (Compact Font Format) a pag. 15 il valore di `FontMatrix` è di *default* `0.001 0 0 0.001 0 0` ma null'altro viene detto.

Infine il fatto che la versione 7 del Reader non mostri questo effetto può essere spiegato con il fatto che questa versione è antecedente alle specifiche PDF Reference le quali, bisogna ricordarlo, sono allineate allo standard ISO-32000.

Questi elementi, e la personale esperienza dell'autore, portano a concludere che il design di font OpenType tipo PostScript con `FontMatrix` non canonica di 1000 unità *è fortemente sconsigliato.*

6 Conclusioni

In questo articolo abbiamo rapidamente visto come è possibile installare un font OpenType e come renderlo disponibile a ConTEXt-`mkiv` tramite typescripts. L'utilizzo delle feature di un font non è di per sé difficile ma non bisogna dimenticare che Unicode ha un vastissimo repertorio di "caratteri" e questi possono interagire in modo inaspettato con le feature di un font.

Abbiamo poi visto come la parte Lua di LuaTEX possa essere usata per ottenere informazioni su un font senza l'ausilio di programmi esterni, anche se attualmente le curve delle outline del glifo non sono memorizzate da LuaTEX e quindi programmi come `ttx` sono ancora superiori. Comunque, visto che luatex si basa su `fontforge`, è possibile che in futuro vengano aggiunte le routine per la memorizzazione delle outline, magari al fine di poter creare font artificiali più ricchi.

7 Ringraziamenti

L'autore desidera ringraziare i membri del team di sviluppo di luatex ed in particolare Taco Hoekwater ed Hartmut Henkel per il loro sostegno ed Hans Hagen per aver preparato una prima versione di ConTEXt-`mkiv` ad-hoc per questo articolo.

Riferimenti bibliografici

Adobe Font Format (1993). *Adobe Type 1 Font Format*. Addison-Wesley. URL `http://www.adobe.com/devnet/font/pdfs/T1_SPEC.PDF`.

Compact Font Format (2003). «The compact font format specification». Technical Report 5176. URL `http://www.adobe.com/devnet/font/pdfs/5176.CFF.pdf`.

Dominici, M. (2007). «Utilizzo di caratteri TrueType con LaTeX. Un esempio pratico: i *Fell Types*». *ArsTEXnica*, (4).

Haralambous, Y. (2007). *Fonts & Encodings*. O'Reilly.

Information technology (2009). *Information technology — Coding of audio-visual objects — Part 22: Open Font Format*. ISO/IEC, seconda edizione. URL `http://standards.iso.org/ittf/PubliclyAvailableStandards/c052136_ISO_IEC_14496-22_2009(E).zip`.

Knuth, D. E. (2010). «Mmix news». URL `http://www-cs-faculty.stanford.edu/~knuth/mmix-news.html`.

LuaTEX development team (2010). *LuaTEX Reference Manual*.

OpenType specification (2010). «OpenType specification». URL `http://www.microsoft.com/typography/otspec/`.

PDF Reference (2008). *PDF Reference*. Adobe Systems Incorporated, sesta edizione. URL `http://www.adobe.com/devnet/acrobat/pdfs/pdf_reference_1-7.pdf`.

Sederberg, T. W. e Peterson, C. (2010). «Computer aided geometric design». URL `http://tom.cs.byu.edu/~557/`.

Wikipedia (2010). «Opentype». URL `http://en.wikipedia.org/wiki/OpenType`.

▷ Luigi Scarso
Logo S.r.l.