Zpravodaj Československého sdružení uživatelů TeXu

Zdeněk Wagner

Anatomie virtuálních fontů

Zpravodaj Československého sdružení uživatelů TeXu, Vol. 14 (2004), No. 1, 3–16

Persistent URL: http://dml.cz/dmlcz/149940

Anatomie virtuálních fontů

Zdeněk Wagner

Článek představuje úvod do problematiky virtuálních fontů. Neklade si za cíl úplnost. Zaměřuje se na vysvětlení základních vlastností virtuálních fontů. Předvádí jednoduché metody, jak virtuální font vytvořit a předvádí vybrané možnosti, jak virtuální font využít.

Úvod

Základní stavební jednotkou, kterou potřebuje každý sázecí program, je font. Právě font obsahuje uspořádané množiny znaků, jimiž vytváříme texty. Fonty jsou dostupné v různých formátech a ne všechny sázecí systémy dovedou pracovat se všemi z nich. Nás bude v tomto článku zajímat pouze jedna třída, jíž jsou virtuální fonty. Jak název napovídá, nemají daleko k virtuální realitě. Virtuální font je popis objektu, který neexistuje. Až ve vhodném okamžiku je podložen fontem skutečným.

V české TEXové literatuře dosud nebyla problematika virtuálních fontů dostatečně pokryta. Přitom virtuální fonty mohou být velmi užitečné. V tomto článku si tedy vysvětlíme, co virtuální font je a jak se dá vytvořit a využít.

Článek obsahuje příklady maker a programů. Abyste si je nemuseli opisovat, jsou k dispozici jako elektronická příloha časopisu na `http://bulletin.cstug.cz/bul041.shtml`.

Jak TEX pracuje s fonty

TEX je dávkově orientovaný sázecí stroj, který na vstupu očekává zdrojový soubor, a výstupem je hotová sazba ve formátu nezávislém na zařízení (DVI – Device Independent). Ve výstupním souboru jsou tedy uloženy informace, kam se mají umístit jednotlivé znaky, ale jejich kresba tam není. V průběhu sazby tedy vůbec nemusíme vědět, jak bude který znak vypadat. Potřebujeme pouze znát rozměry znaků a vlastnosti ovlivňující dvojice sousedících znaků: selektivní prostrkání (kerning) a slitky (ligatury). Zmíněné informace jsou uloženy v metrickém souboru, který se tradičně označuje zkratkou TFM (Text Font Metrics). Skutečnou kresbu znaků vyžaduje až ovladač výstupního zařízení, který ze souboru DVI tvoří tištěnou stránku nebo jinou vizuální reprezentaci dokumentu. Zdánlivou výjimkou je pouze pdfTEX. Při výstupu do PDF totiž musí znát kresbu znaků

již během sazby. Je to dáno tím, že sázecí stroj i výstupní ovladač pro PDF jsou integrovány v jediném programu.

TEXový sázecí stroj se nezajímá o formát fontů, vyžaduje pouze metrické soubory. Pokud si tedy vymyslíme Private Nonsense Font Specification, k fontům vytvoříme odpovídající soubory TFM a napíšeme si ovladač, který s těmito fonty bude umět pracovat, můžeme je v TEXu použít.

Každý znak má čtyři rozměry: šířku, výšku, hloubku a kurzívní korekci. Při sazbě TEX umístí referenční bod znaku na aktuální pozici sazby. Poté posune aktuální bod sazby doprava o šířku znaku a pokračuje ve stejné činnosti. Ke změně dochází pouze v případě, že mezi danou dvojicí znaků má být nenulový kerning. V takovém případě se aktuální bod sazby posune o požadovanou hodnotu. Výška a hloubka jsou nutné pro sestavování řádků. Jakmile je odstavec nalámán, zjistí TEX výšku a hloubku každého řádku. Pokud by se řádky překrývaly, zvětší TEX řádkový proklad[1]. Kurzívní korekci vkládáme příkazem `\/`.

V TEXu používáme nejčastěji dva druhy fontů: bitmapové generované METAFONTem a PostScriptové ve formátu Type1. V následujícím odstavci si je stručně popíšeme.

METAFONT je program, který z matematického popisu fontu vytváří metrický soubor a bitovou mapu určenou pro konkrétní výstupní zařízení[2]. Některé metrické informace jsou uloženy i v bitmapě. Ovladač tedy nemusí číst soubor TFM, protože informace o rozměrech znaků jsou přítomny ve fontu samotném.

PostScriptové fonty Type1 jsou specializované programy, které vykreslují znaky na zařízeních, kde je tento jazyk podporován. Obrysy znaků jsou definovány jejich obrysy, jež jsou popsány matematickými rovnicemi. Závěrečným příkazem v popisu každého znaku je posun aktuálního bodu sazby. Při tisku tedy nemusíme znát metriku, ovladač potřebuje pouze posloupnost znaků, mezislovní mezery a kerningy a povely pro přechod na nový řádek. Rozměr znaku lze získat z něho samotného. PostScript obsahuje operátor `pathbbox`, který zjistí ohraničovací rámeček (bounding box) aktuální cesty tvořené sadou křivek. Takto zjištěný ohraničovací rámeček však může být příliš veliký. Proto nejprve použijeme operátor `flattenpath`. Šířka znaku v TEXovém smyslu však nemusí (a často také není) shodná s šířkou ohraničovacího rámečku. V souboru TFM potřebujeme údaj o posunu aktuálního bodu sazby. Ten zjistíme operátorem `stringwidth`. Metrické údaje lze tedy zjistit přímo z fontu. Lze k tomu použít makro `printafm.ps` z distribuce GhostScriptu, jímž vytvoříme soubor AFM (Adobe font Metrics). Většinou však soubor AFM dostaneme současně s fontem. To je lepší alternativa, protože soubor může obsahovat informace o kerningových párech. Tento údaj ve fontu není. Metriku z formátu AFM převedeme do TFM buď programem `afm2tfm` nebo makrem `fontinst`.

[1] Toto chování lze ovlivnit hodnotou parametru `\lineskiplimit` a dalších.

[2] Výstupní zařízení se od sebe liší nejen rozlišením, ale i tvarem a velikostí tiskového bodu. Proto dvě různá zařízení se stejným rozlišením vyžadují různé korekce. Parametry pro řadu zařízení najdete v souboru `modes.mf`, který byste ve své instalaci TEXu měli mít.

Vidíme, že u dvou nejběžnějších typů fontů jsou metrické informace uloženy na dvou místech: v souboru TFM pro TEX a v samotném fontu pro výstupní zařízení. Tyto informace musí být identické. Výstupní zařízení totiž nemůže tušit, že jsme TEXu podstrčili upravenou metriku. Pokud to uděláme, dostaneme na výstupu zmatek. Jestliže potřebujeme font se stejnou kresbou znaků, ale s jinými metrickými údaji, musíme použít metodu, kterou si ukážeme v následující kapitole.

Co je virtuální font

Ukázali jsme si, že TEX při sazbě používá pouze metrické údaje. Můžeme tedy TEXu podstrčit metriku fontu, který vůbec neexistuje. K čemu se to hodí? Je to užitečné tehdy, když vlastní font je vytvořen ve výstupním zařízení. Pojmem *virtuální font* však rozumíme předpis, kterým vytváříme kresbu znaků poskládáním fragmentů z jiných fontů. Virtuální font smí obsahovat vše, co může být v souboru DVI, včetně příkazů `\special`. Pokud však do virtuálního fontu vložíme příkaz `\special` fungující pouze na určitém zařízení, bude výsledný font na tomto zařízení závislý. Při tvorbě a použití virtuálních fontů proto musíme být obezřetní.

Virtuální font sám o sobě nedefinuje kresbu znaků. Výjimkou jsou pouze případy, kdy by daný znak byl tvořen pouze linkami, jež reprezentují TEXové primitivy `\hrule` a `\vrule`, nebo byl generován výhradně pomocí příkazů `\special`. Nejčastěji virtuální znak odkazuje na skutečný znak nebo znaky v nějakém jiném fontu nebo několika různých fontech. Tyto instrukce jsou uloženy v souboru VF a ovladač DVI je při zobrazování interpretuje. V době, kdy koncept virtuálních fontů vznikl, si ovladače DVI s nimi neuměly poradit. Proto byl napsán program `dvicopy`, který při kopii souboru DVI provede devirtualizaci. Instrukce ze souborů VF interpretuje a výsledný soubor pak obsahuje pouze odkazy na normální fonty. V dnešní době byste program `dvicopy` již neměli potřebovat, protože virtuální fonty jsou podporovány ve všech ovladačích, ale je dobré o této možnosti vědět.

Tvoříme jednoduchý virtuální font

Virtuální font je tvořen binárním souborem VF. Programem `vftovp` jej převedeme na textový soubor VPL (Virtual Property List). Ke zpětné konverzi slouží program `vptovf`. Jeho výstupem je jak virtuální font VF, tak metrický soubor TFM, který vyžaduje TEX. Virtuální font můžeme tedy vytvořit tak, že si jej sami napíšeme v textové podobě a přeložíme programem `vptovf`. To však není příliš pohodlné, je výhodnější využít nějaké nástroje.

Jedním z jednoduchých nástrojů, jímž lze virtuální fonty vytvářet, je balíček qdTEXvpl od Eherharda Mattese. Hlavní program je psán v jazyce C (jako jeden z mála programů E. Mattese je dodáván ve zdrojovém kódu). Součástí balíčku je TEXové makro a fiktivní font. Princip spočívá v tom, že se požadované znaky vygenerují TEXovými prostředky. Dodaným makrem je označkujeme, přeložíme do DVI a z něj se následně generuje virtual property list. Písmena *qd* v názvu programu znamenají *Quick & Draft*. Je tedy téměř jisté, že před přeložením VPL na VF budeme muset udělat další zásahy.

Celý postup si předvedeme na jednoduchém příkladu. Nejprve si napíšeme soubor, který nazveme `qdmftex.tex`:

```
\input qdtexvpl
\font\T cmr10
\font\M mplogo10
\texvpl{t}{\T \TeX}
\texvpl{m}{\M METAFONT}
\texvpl{c}{\kern2pt\vrule height5pt depth-1pt width4pt\kern2pt}
\bye
```

Soubor definuje tři znaky, v nichž se používají dva různé fonty. Prvním parametrem makra `\texvpl` je požadovaný znak, druhým parametrem je požadovaná realizace. Soubor zpracujeme TEXem a na DVI se můžeme podívat. Programem qdTEXvpl pak vytvoříme následující VPL:

```
(MAPFONT D 17
   (FONTNAME cmr10)
   (FONTCHECKSUM O 11374260171)
   (FONTDSIZE R 10.000000))
(MAPFONT D 18
   (FONTNAME mplogo10)
   (FONTCHECKSUM O 37045067476)
   (FONTDSIZE R 10.000000))
(CHARACTER C t
   (CHARWD R 1.861079)
   (CHARHT R 0.683330)
   (CHARDP R 0.215276)
   (MAP
      (SELECTFONT D 17)
      (SETCHAR C T)
      (PUSH)
      (MOVERIGHT R -0.166702)
      (MOVEDOWN R 0.215277)
      (SETCHAR C E)
      (POP)
```

```
      (MOVERIGHT R 0.388855)
      (SETCHAR C X)))
(CHARACTER C m
   (CHARWD R 5.133313)
   (CHARHT R 0.600000)
   (CHARDP R 0.000000)
   (MAP
      (SELECTFONT D 18)
      (SETCHAR C M)
      (SETCHAR C E)
      (SETCHAR C T)
      (MOVERIGHT R -0.022223)
      (SETCHAR C A)
      (SETCHAR C F)
      (MOVERIGHT R -0.044444)
      (SETCHAR C O)
      (SETCHAR C N)
      (SETCHAR C T)))
(CHARACTER C c
   (CHARWD R 0.800000)
   (CHARHT R 0.500000)
   (CHARDP R 0.000000)
   (MAP
      (MOVERIGHT R 0.200000)
      (MOVEDOWN R -0.100000)
      (SETRULE R 0.400000 R 0.400000)))
```

Vlastnost `MAPFONT` definuje použité fonty. Pak následují definice jednotlivých znaků. Každý z nich začíná specifikací rozměrů. Pokud je některý z rozměrů nulový (typicky hloubka), nemusí být uveden. Vlastnost `MAP` popisuje realizaci daného znaku. Pomocí `SELECTFONT` se zvolí font, `SETCHAR` slouží k usazení znaku, `MOVEDOWN` a `MOVERIGHT` k posunům a `SETRULE` k sazbě linky. Rozměry jsou uváděny v násobcích `DESIGNSIZE`, jejíž implicitní hodnota je 10 pt (uvedli jsme ji při volání programu qdTEXvpl). Soubor VPL přeložíme programem `vptovf`. Metriku TFM vložíme na místo, kde TEX očekává metrické soubory, VF vložíme tam, kde ovladač očekává virtuální fonty. V některých TEXových instalacích může být vyžadována obnova databáze, některé instalace jsou schopny číst uvedené soubory z aktuálního adresáře. Přepneme-li se nyní do fontu `qdmftex` a napíšeme tcm, dostaneme:

TEX ■ METAFONT

Takto vytvořený font neobsahuje mezislovní mezeru. Museli bychom do něj ručně doplnit hodnoty parametrů `\fontdimen`, o nichž si povíme později.

Osmý bit schází nám

Písmena s diakritickými znaménky lze vysázet čistě TEXovými prostředky tak, že samotný akcent usadíme nad příslušné písmeno. Proč nás tedy zajímají fonty, kde jsou akcentovaná písmena kreslena jako samostatné znaky? Máme pro to dva důvody. První důvod je estetický. Primitiv `\accent` umisťuje diakritické znaménko na geometrický střed, což nevypadá vždy hezky. Závažnější je však skutečnost, že takto zapsaná slova neumí TEX dělit. Oba problémy lze řešit virtuálním fontem. TEX podle metrického souboru najde všechny znaky a ve virtuálním fontu můžeme usazení diakritických znamének doladit. Více se o tom dočtete v češtině v článku [4].

Podobný přístup, ale z jiného důvodu, byl kdysi použit při zpracování Zpravodaje. Pro jeho sazbu se používají $\mathcal{CS}$ fonty, které jsou vytvořeny METAFONTem. Pro soubor ve formátu PDF se však bitmapové fonty nehodí, je nutno použít PostScriptové fonty Type1. $\mathcal{CS}$ fonty ve formátu Type1 neexistovaly. Proto byl pro tyto účely použit virtuální font, který mapoval znaky z $\mathcal{CS}$ fontů na standardní Computer Modern, jejichž varianty Type1 byly volně dostupné. Virtuální font byl ovšem použit pouze pro účely převodu do PDF. Mohli jsme si to dovolit, protože virtuální fonty měly identickou metriku jako skutečné $\mathcal{CS}$ fonty.

Triky s fonty

Virtuální font umožňuje rozšířit možnosti TEXu. Klasickým případem, s nímž si TEX neumí poradit, je prostrkaný text. Existuje sice několik řešení na úrovni maker, jmenujme např. makro `\prostrkej` od Petra Olšáka [3], ale tyto metody se hodí spíše na samostatná slova, nejlépe v titulcích. Pokud bychom takto měli sázet větší úsek textu, potřebujeme jiný přístup. Vytvoříme si prostrkaný virtuální font. Metrický soubor TFM lze programem `tftopl` převést na textový soubor PL (Property List). Jeho formát se hodně podobá formátu VPL. Prostrkaný virtuální font si můžeme proto vytvořit jednoduchým prográmkem. Je napsán v Perlu, protože programy v tomto jazyce se snadno a rychle ladí a interpret existuje pro všechny platformy. Zde je výpis programu `spaced.pl`:

```
#!/usr/bin/perl
if ($#ARGV == 1) { $spacing = $ARGV[1]; }
else { $spacing = .1 }
if ($ARGV[0] =~ /\.pl$/) {
  $fontname = $‘;
  open(PL, $ARGV[0]) or die ’Cannot open ’ . $ARGV[0];
}
else { die ’Invalid font name’; }
$c = 0;
```

```
print "(VTITLE spaced version of $fontname)\n";
while (<PL>) {
  chop;
  next if /\(LIG C [fli]/;
  $_ = '   (SPACE R .4)' if /\(SPACE/;
  $_ = '   (STRETCH R .1)' if /\(STRETCH/;
  $_ = '   (SHRINK R .06)' if /\(SHRINK/;
  if (/^\((DESIGNSIZE.+)\)$/) {
    $dsize = "(FONT$1)";
  }
  if (/^\(CHARACTER\s+/) {
    $c = $';
  }
  if ($c && /\(CHARWD R (.+)\)$/) {
    $wd = $spacing + $1;
    $_ = "   (CHARWD R $wd)";
  }
  if ($c && $_ eq '   )') {
    print<<"EOMAP";
   (MAP
      (SETCHAR $c)
      )
EOMAP
    $c = 0;
  }
  print "$_\n";
  if (/^\((CHECKSUM.*)\)$/) {
    $checksum = "(FONT$1)";
    print<<"EOMAP";
(MAPFONT D 0
   (FONTNAME $fontname)
   $checksum
   (FONTAT R 1.0)
   $dsize
   )
EOMAP
  }
}
```

Program má dva parametry. První je povinný a obsahuje jméno soouboru PL včetně přípony. Druhý, nepovinný parametr specifikuje požadovaný proklad v jednotkách DESIGNSIZE. Standardně použijeme hodnotu 0.1. Při převodu pro-

vádíme několik činností. Nejprve si zapíšeme `VTITLE`. Je to pouze nepovinný komentář, ale je vhodné si poznačit, co jsme vytvořili. Prostrkaný font nesmí obsahovat ligatury. Proto odstraníme záznamy `LIG`, pokud je druhým znakem některý z množiny f, l, i. Ponecháme ligatury, jimiž se vytvářejí dlouhé pomlčky. Prostrkaný text by měl mít větší mezislovní mezery, než se používají v běžném textu. Změníme proto hodnoty parametrů `SPACE`, `STRETCH` a `SHRINK`, jež určují mezeru, její roztažitelnost a stažitelnost. Tyto hodnoty ovšem lze modifikovat dodatečně TEXovým primitivem `\fontdimen`, případně můžeme velikost mezislovní mezery nastavit pomocí `\spaceskip`. Hodnotu `DESIGNSIZE` si pouze zapamatujeme, abychom ji ve vhodný okamžik mohli zkopírovat do `FONTDESIGNSIZE`. Jakmile přečteme `CHECKSUM`, můžeme zapsat informaci o použitém fontu `MAPFONT`. Zmíníme se o hodnotě `CHECKSUM`. Je to kontrolní součet, který je uložen v metrickém souboru TFM, TEX jej zapíše do DVI a METAFONT jej zapisuje i do vlastního bitmapového fontu. Nezáleží na způsobu, jak je tento kontrolní součet vypočítán. Důležité je pouze to, aby na všech místech byla stejná hodnota. Ovladač podle toho zkontroluje, zda používá font, jehož metriku použil TEX při sazbě. Při tvorbě virtuálního fontu musíme tedy uvést správnou hodnotu `FONTCHECKSUM`, na hodnotě kontrolního součtu virtuálního fontu nezáleží. Můžeme použít stejnou hodnotu nebo třeba nulu. Hlavní činnost provedeme při zpracování vlastnosti `CHARACTER`. Kód znaku si uložíme pro pozdější použití. Vlastnost `CHARWD` obsahuje šířku znaku. Tu musíme zvětšit o požadovaný proklad. Narazíme-li na ukončovací pravou závorku, zapíšeme instrukce, jak se má znak vytvořit. Vysadíme znak se stejným kódem z implicitního fontu. Proto nemusíme uvádět `SELECTFONT` a `SETCHAR` bude obsahovat kód znaku, který jsme si uložili. Program jsme aplikovali na font csr10. Použili jsme příkaz:

```
spaced.pl csr10.pl > scsr10.vpl
```

Nyní již můžeme sázet proloženě. Samozřejmě jsme virtual property list museli překompilovat na virtuální font a soubory nakopírovat do správných adresářů. Protože pracujeme s osmibitovým fontem, funguje i dělení slov. Ve fontu jsme ponechali všechny kerningy a definice ligatur pro pomlčky. Skutečně – pomlčka na půlčtverčík funguje a dokonce — máme i dlouhou čtverčíkovou pomlčku.

Druhým problémovým případem je podtrhávání. Řešení pomocí maker má svá omezení. Vytvoříme si tedy virtuální font programem `underline.pl`:

```
#!/usr/bin/perl
$overlap = .05;  $ulpos = .24;  $ulheight = .04; # defaults
$overlap = $ARGV[1] if $#ARGV > 0;
$ulpos = $ARGV[2] if $#ARGV > 1;
$ulheight = $ARGV[3] if $#ARGV > 2;
$ovstring = sprintf("%0.4f", -$overlap); # must be fixed real
```

```
$ulht = sprintf("%0.4f", $ulheight);
if ($ARGV[0] =~ /\.pl$/) {
  $fontname = $`;
  open(PL, $ARGV[0]) or die 'Cannot open ' . $ARGV[0];
}
else {
  die 'Invalid font name';
}
$c = $f = 0;
print "(VTITLE underlined version of $fontname)\n";
while (<PL>) {
  chop;
  if (/^\((DESIGNSIZE.+)\)$/) {
    $dsize = "(FONT$1)";
  }
  $f = 1 if /^\(FONTDIMEN/;
  if (/^\(CHARACTER\s+/) {
    $c = $';
  }
  if ($c && /\(CHARWD R (.+)\)$/) {
    $wd = 2 * $overlap + $1;
  }
  if ($c && $_ eq '   )') {
    print<<"EOMAP";
   (MAP
      (PUSH)
      (MOVEDOWN R $ulpos)
      (MOVERIGHT R $ovstring)
      (SETRULE R $ulht R $wd)
      (POP)
      (SETCHAR $c)
      )
EOMAP
    $c = 0;
  }
  if ($f && $_ eq '   )') {
    $f = 0;
    $th = $ulheight - $ulpos;
    print<<"EOMAP";
   (DEFAULTRULETHICKNESS R $th)
   (BIGOPSPACING1 R $ulpos)
EOMAP
```

```
  }
  print "$_\n";
  if (/^\((CHECKSUM.*)\)$/) {
    $checksum = "(FONT$1)";
    print<<"EOMAP";
(MAPFONT D 0
   (FONTNAME $fontname)
   $checksum
   (FONTAT R 1.0)
   $dsize
   )
EOMAP
  }
}
```

Struktura programu je velmi podobná, popíšeme si tedy jen odchylky. Program nyní má čtyři parametry, z nichž pouze první, obsahující jméno původního fontu, je povinný. Dalšími parametry definujeme polohu podtrhávací linky, její tloušťku a přesah. Mezi některé dvojice písmen se totiž vkládá kladný kerning. Přesah slouží k tomu, aby linka nebyla přerušena. Parametry mají standardní hodnoty. Možná vás zaujme použití funkce `sprintf`. Je-li hodnota čísla menší než 0.1, Perl ji zapíše ve vědecké notaci. Takové číslo však program `vptovf` neumí přečíst, proto si proměnné, kde se dá očekávat malá hodnota, naformátujeme sami. Zajímavá jsou dvě místa programu. Při vlastní sazbě znaku si nejprve příkazem `PUSH` uložíme aktuální bod sazby, pak vysadíme linku, jejíž délku jsem vypočetli z šířky znaku, pak se příkazem `POP` vrátíme na původní bod sazby a vysadíme znak. Na konci programu modifikujeme sekci `FONTDIMEN`. Paramery `DEFAULTRULETHICKNESS` a `BIGOPSPACING1` jsou v TEXu dostupné jako `\fontdimen` 8 a 9. Ukládáme si do nich informaci o podtrhávací lince. Analogicky si vytvoříme font ucsr10. Nyní již můžeme sázet podtrženě, ale ještě to není úplně ono. Nepodtrhávají se mezery.

V této části jsme udělali tři změny. Především sázíme text do dvou sloupců a dovolíme TeXu, aby na nový řádek oddělil i jen dvě písmena. Můžeme si tak lépe demonstrovat, že TeX dělí podtržená slova. Druhou změnou je zvětšení řádkového prokladu o 15%, aby text nebyl příliš slitý. Poslední úpravou jsme dosáhli toho, že se podtrhávají i mezery. Ve virtuálním fontu nelze podtrženou mezeru vytvořit, protože TeX mezeru jako znak nepoužívá. Velikost mezislovních mezer se vezme z parametrů, o nichž jsme se zmínili při vytváření prostrkaného fontu. Jedinou možností, jak mezery podtrhnout,

je vytvoření vhodného makra. Postup si popíšeme již v normálním, nepodtrženém textu. Všimněte si, že řešení není dokonalé, logo TEX se podtrhává špatně, protože písmeno E zasahuje pod řádek.

Definici provedeme uvnitř prostředí `multicols`, některé změny budou lokální. Prvními kroky v definici je nastavení limitu, aby logo TEX, použité na konci dvousloupcové sazby, nerozhodilo řádkový rejstřík. Dále zvětšíme řádkový proklad, přepneme na podtržený font a povolíme dělení dvě písmena před koncem slova.

```
\lineskiplimit \minus5dd
\linespread{1.15}\selectfont
\underlined \righthyphenmin2
```

Nyní si nadefinujeme makro pro podtrženou mezeru:

```
\def\hrulespace{\leaders\hrule height\fontdimen8\the\font
                                depth\fontdimen9\the\font
    \hskip\fontdimen2\the\font plus \fontdimen3\the\font
                                minus \fontdimen4\the\font}
```

Makro definujeme pomocí `\leaders`. Tento primitiv se chová stejně jako *glue*. Je to odstranitelný element a je v něm povolen zlom, pokud mu nepředchází jiný odstranitelný element. Za řídícím slovem `\leaders` následuje *rule* nebo *box*, druhým objektem je mezera. Začněme vysvětlování odzadu. Velikost mezislovní mezery je uložena v metrice v parametru `\fontdimen2`, `\fontdimen` s čísly 3 a 4 obsahují roztažitelnost a stažitelnost. Použitím těchto hodnot v primitivu `\hskip` dosáhneme stejných výsledků, jako kdybychom použili nepodtržený font, z něhož jsme je do virtuálního fontu převzali. Při popisu programu `underline.pl` jsme si řekli, že polohu a tloušťku podtrhávací linky uložíme do parametrů `\fontdimen` 8 a 9. Ve skutečnosti `\fontdimen8` neobsahuje tloušťku linky, ale tloušťku zmenšenou o polohu linky. Přesně tuto hodnotu očekává `\hrule` za klíčovým slovem `height`. Tím jsme zajistili, že mezera bude podtržena stejnou linkou jako ostatní znaky.

Bylo by nepohodlné, kdybychom `\hrulespace` museli do každé mezislovní mezery zapisovat ručně. Mezeru proto uděláme aktivní a vytvoříme vhodnou definici. Aktivní mezera je však dvojsečná zbraň. Abychom nemuseli dávat při definicích zvlášť velký pozor, použijeme trik. Vykřičníku nastavíme, že odpovídající malé písmeno je mezera. Dále učiníme vykřičník aktivním. Celou definici pak vložíme do `\lowercase`. Tento primitiv převede všechna písmena na malá, ale zachová jejich kategorii. Z vykřičníku se tedy stane aktivní mezera:

```
{\lccode`\!32
```

```
\catcode`\!13
\lowercase{%
\gdef!{\ifhmode\expandafter\checkspace\fi}
\gdef\checkspace{\futurelet\next\maybespace}
\gdef\maybespace{%
  \let\hsp\hrulespace
  \ifx!\next \let\hsp\empty
  \fi
  \hsp}
}}
\def~{\nobreak\hrulespace}
```

Mezery chceme podtrhávat pouze v horizontálním režimu. Jsme-li v jiném režimu, neprovedeme nic. V textu můžeme mít více mezer za sebou. Za normálních okolností je vstupní procesor TEXu nahradí jedinou mezerou, ale neprovede to v okamžiku, až bude mezera aktivní. Proto se pomocí `\futurelet` podíváme na následující token. Je-li jím aktivní mezera, neprovedeme nic, v opačném případě vložíme podtrženou mezeru. Pak si ještě nadefinujeme vlnku tak, aby vkládala nezlomitelnou podtrženou mezeru.

Text odstavce se může nacházet na několika řádcích. Prázdným řádkem je odstavec ukončen. Pokud bychom do tohoto mechanismu nezasáhli, přechod na nový řádek ve zdrojovém souboru by vytvořil nepodtrženou mezeru. Ošetříme to následujícím způsobem:

```
\catcode`\^^M13 %
\def^^M{\ifhmode\expandafter\checkpar\fi}%
\def\checkpar{\futurelet\next\maybepar}%
\def\maybepar{%
  \let\hsp\hrulespace %
  \ifx^^M\next \let\hsp\par %
  \fi %
  \hsp}
```

Znak konce řádku jsme nejprve učinili aktivním. Od tohoto okamžiku musíme všechny řádky ukončovat procentem, jinak se TEX promění v zuřivého lva, který nám přeplní zásobník. Opět nebudeme provádět žádnou činnost v jiném než horizontálním režimu. Zvláštní ošetření vyžaduje posloupnost dvou konců řádků, proto se opět podíváme na následující token. Následuje-li konec řádku, místo sazby podtržené mezery ukončíme odstavec. Za poslední pravou závorkou definice makra `\maybepar` již procento nepotřebujeme. Zbývá už jen změnit kategorii mezery pomocí `\catcode`\ 13` a můžeme tisknout podtrženě. Makro není zcela robustní. Nejsou ošetřeny mezery na začátku a na konci řádku. V anglické sazbě se nevloží delší mezera za tečku, protože nekontrolujeme hodnotu `\sfcode`. Dokonalejší makro si můžete udělat za domácí cvičení.

Další problém je v uvedených perlovských skriptech. Neimplementují plný parser souborů ve formátu property list, ale jsou pevně svázány s výstupem z programu `tftopl`.

Překódování fontu

Pro češtinu a slovenštinu se používá několik různých kódování. Může se tedy stát, že máme sice font s akcentovanými znaky, ale v jiném kódování, než vyžaduje TEX. Na úrovni PostScriptu lze font překódovat operátorem `ReEncodeFont`. Jinou metodu, která však není zcela ekvivalentní s předchozí[3], nabízí virtuální font. Pro příklad nemusíme chodit daleko. Představme si, že chceme používat mezinárodní LATEX, jehož fonty mají kódování T1, ale typograficky se nám více líbí $\mathcal{CS}$ fonty s kódováním IL2. Můžeme si proto vytvořit virtuální font, jehož kódování bude T1, ale bude odkazovat na znaky z $\mathcal{CS}$ fontu. Provedeme to v několika krocích. Můžeme začít podobným perlovským skriptem, jaký byl popsán v předchozí kapitole. Nebudeme upravovat parametry fontu ani rozměry znaků, jen jejich kód. Například písmeno Č má v IL2 oktalový kód 310, v T1 má oktalový kód 203. Při konverzi tedy všechny výskyty řetězce `O 310` nahradíme řetězcem `O 203`. Do popisu znaku Č pak vložíme instrukci `(SETCHAR O 310)`. Podobně naložíme s ostatními znaky, jejichž kód se liší. Pokud chceme psát pouze česky a slovensky, můžeme být s výsledkem spokojeni. Kódování T1 však obsahuje znaky, které v $\mathcal{CS}$ fontech nejsou. Při jejich vytváření nám pomůže qdTEXvpl. Znaky s diakritickými znaménky, např. španělské ñ, vytvoříme snadno pomocí `\~n`. Některé znaky však v $\mathcal{CS}$ fontech nemáme, příkladem je Đ. Vezmeme si jej z odpovídajícího DC fontu. Oba VPL nyní zkombinujeme. Ve třetím kroku musíme doplnit kerningy. Můžeme je zkopírovat z $\mathcal{CS}$ fontu od podobně vypadajících znaků, nebo je převezmeme z DC fontu. Čtvrtým krokem bude doladění polohy akcentů a kerningů. Návod vypadá na první pohled jednoduše, ale mohou nás potkat problémy, které popisuje Petr Sojka [4].

Literatura

V této kapitole uvádíme seznam základní české i cizojazyčné literatury, která se problematice virtuálních fontů zabývá. Kniha [3] se sice virtuálním fontům nevěnuje, ale ohsahuje popisy formátů souborů TFM a DVI.

1. Donald Knuth: *Virtual Fonts: More Fun for Grand Wizards.* TUGboat (11) 1990, No 1 (April), pp. 13–23.

[3] PostScriptový font může obsahovat více než 256 znaků. Pomocí `ReEncodeFont` lze přidělit kódy i těm znakům, které v původním fontu žádný číselný kód neměly. Pomocí virtuálního fontu to možné není.

2. Petr Olšák: **Typografický systém TEX**, 2. vydání. Konvoj, Brno 2000. ISBN 80-85615-91-6.
3. Petr Olšák: **TEXbook naruby.** Konvoj, Brno 1997. ISBN 80-85615-64-9.
4. Petr Sojka: *Virtuální fonty, accents a přátelé.* Zpravodaj Československého sdružení uživatelů TEXu, **4** (2), 56–69 (1994).
5. Jiří Zlatuška: *Automatic generation of virtual fonts with accented letters for TEX.* Cahiers GUTenberg No. 10, září 1991.

Summary: Anatomy of Virtual Fonts

The article is a brief introduction to the concept of virtual fonts. It is first explained how TEX works with fonts. Afterwards a simple tool for building a virtual font, namely qdTEXvpl, is presented. Finally usage of virtual fonts is demonstrated by typesetting spaced and underlined text. The macros and Perl scripts described in this article are available from the web page of the Bulletin.

Wordové plug-iny související s TEXem aneb Možnosti a schopnosti produktů Word2TeX a TeX2Word

Aleš Pavelka

Úvod

V současné době kdy většina uživatelů používá produkty firmy *Microsoft*, vzniká tím vedlejší efekt nutící uživatele k vzájemné dohodě na formátu, ve kterém zpracovávají své dokumenty. Je smutné, že ne vždy se lidský vývoj ubírá tím „nejlepším“ směrem a nevybírá to nejlepší pro člověka. Musíme si otevřeně přiznat, že pro nezkušeného nebo začínajícího uživatele počítače je poněkud jednodušší napsat a upravit nějaký dokument v prostředí *MS Wordu* než v TEXu či LATEXu. Tento problém je však daleko širší a kdo ví, kdy bude vyřešen. Se stavem takového dvojího světa vzniká i otázka: „Existuje kvalitní a rychlá možnost přechodu mezi těmito formáty?“ Při hledání odpovědi na tuto otázku nám pomohou konvertory, pluginy či dodatky *MS Wordu* – produkty Word2TeX a TeX2Word.

Článek původně vznikl jako semestrální práce předmětu „Publikační systém TEX“