Zpravodaj Československého sdružení uživatelů TeXu

Zdeněk Wagner

METAFONT a velké znaky

Zpravodaj Československého sdružení uživatelů TeXu, Vol. 3 (1993), No. 2, 77–83

Persistent URL: http://dml.cz/dmlcz/149665

Literatura

[1] Klégr, A., a kol.: *Světem jazyků.* Praha, Albatros 1989.

[2] Vochala, J., Novák, M., Pucek, V.: *Úvod do čínského, japonského a korejského písma.* Praha, SPN 1975.

[3] Jalbert, F.: *Japanese* [LA]TEX *for DOS – User's Guide* (dokumentace k systému JemTEX, verze 2.00).

Zdena Zinková, Pavel Sýkora
sykora@csearn

METAFONT a velké znaky

Zdeněk Wagner

Nedávno jsem METAFONTem vytvořil několik obrázků. Zákazník ovšem požadoval, abych je zvětšil na dvojnásobnou velikost. V duchu jsem zajásal nad tím, že všechny rozměry jsem definoval jako násobek parametru u, jehož velikost byla 1 mm. Změnil jsem tedy přiřazení na začátku souboru na

```
u#=2mm#;
```

a doufal jsem, že je tím vše vyřešeno. Ukázalo se však, že moje radost byla předčasná. Program `dvidot` od Eberharda Mattesa je schopen zobrazit hodně velké znaky. Pokud ale požadujete otočení o 90°, může se vám stát totéž co mně. `Dvidot` mi oznámil:

```
Character too big
```

V tomto příspěvku chci stručně popsat metodu, kterou jsem tento problém vyřešil. Není to jistě metoda jediná a možná není nejlepší. Je to první řešení, které mě při mé neznalosti napadlo, a podařilo se mi je snadno dovést do funkčního stavu.

Nejprve jsem se rozhodl, že obrázky budu tvořit podobně, jako to provádí program BM2FONT. Zde je ale problém, jak obrázky rozdělit. Asi by práce byla jednodušší, kdybych s takovým záměrem začínal již při prvním návrhu. Nyní jsem potřeboval rozdělit již hotový obrázek.

Moje idea tedy spočívala v tom, že jsem nechal obrázek nakreslit tak, jako dříve. Výsledek jsem schoval do globální proměnné typu *picture* a z té jsem pak postupně vybíral jednotlivé části. Experimentováním jsem dospěl k dělení na osm částí, a to po čtyřech obrázcích ve dvou řadách nad sebou. Znaky v horní řadě budou mít vždy nulovou hloubku.

Když opomenu znaky vyjadřující písmena skutečné abecedy a matematické symboly, má každý znak v METAFONTu tři rozměry: šířku, výšku a hloubku. Pokud by výška nebo hloubka byla záporná (se šířkou si nejsem jist), METAFONT by se zlobil. Jak uvidíme později, záporná výška by se zde mohla při malé opatrnosti objevit. Pro jistotu navíc budeme požadovat, aby výška neklesla pod určitou kladnou hodnotu, např. 1.5 pt. Upravený soubor tedy začínal příkazy:

```
picture mypicture;
min_ht:=vround(1.5pt);
```

Kdybych měl upravovat jeden obrázek, asi bych všechno udělal přímo v jeho definičních rovnicích a nevyráběl bych žádná makra. Obrázků ale bylo dvanáct a zde už se makra vyplatí. Rozhodl jsem se předefinovat `beginchar` a `endchar`. Podíval jsem se do `plain.mf`, co tato makra dělají, a pak jsem jejich definice upravil podle své potřeby.

Upravená verze `beginchar` uschová své parametry do globálních proměnných a vypočte hodnoty, které budeme potřebovat později. Všimněte si výpočtu poloviční výšky. Zde se testuje, zda je poloviční výška menší než minimální povolená. Dojde k tomu tehdy, když je hloubka původního obrázku větší než jeho výška (v mém případě se to skutečně přihodilo). Nakonec zavoláme makro `begin_char`, které provádí zhruba to, co původní `beginchar` definovaný v `plain.mf`.

```
def beginchar(expr code,w_s,h_s,d_s) =
 c:=code;          w_sharp:=w_s;
 h_sharp:=h_s;     d_sharp:=d_s;
 fi_w_sharp:=1/4w_sharp;
 sec_w_sharp:=1/2w_sharp;
 thi_w_sharp:=3/4w_sharp;
 half_h_sharp:=.5(h_sharp-d_sharp);
 fi_w:=hround(fi_w_sharp*hppp);
 sec_w:=hround(sec_w_sharp*hppp);
```

```
 thi_w:=hround(thi_w_sharp*hppp);
 full_w:=hround(w_sharp*hppp);
 half_h:=vround(half_h_sharp*hppp);
 full_h:=vround(h_sharp*hppp);
 if half_h < min_ht:
   half_h:=min_ht;  half_h_sharp:=half_h/hppp;
 fi;
 dp:=vround(d_sharp*hppp);
 mypicture:=nullpicture;
 begin_char(c,fi_w_sharp,half_h_sharp,d_sharp,
                                fi_w,half_h,dp);
 scantokens extra_beginchar;
enddef;
```

Makro `begin_char` provádí velmi jednoduchou činnost. Úkolem je pouze vynulovat paměť pro znak a přiřadit správné hodnoty vnitřním proměnným.

```
def begin_char(expr _c,_w_s,_h_s,_d_s,_w,_h,_d) =
 begingroup
 charcode:=if known _c: byte _c else: 0 fi;
 charwd:=_w_s; charht:=_h_s; chardp:=_d_s;
               w:=_w; h:=_h; d:=_d;
 charic:=0; clearxy; clearit; clearpen;
enddef;
```

Makro `endchar` ukončí definici znaku tak, jak je obvyklé. Navíc přidáme příkaz `cullit`. Tím zajistíme, že pixely, které mají být černé, budou mít hodnotu 1, což bude v dalším postupu nezbytné. Dále uložíme hotový obrázek do proměnné *mypicture* a závěrečnou práci provede makro `ship_char`, které si vysvětlíme později.

Makro `endchar` pak pokračuje řádky, které vytvářejí zbývajících sedm částí znaku. Jedná se vždy o trojici příkazů. Nejprve se zavolá `begin_char` s příslušnými parametry, které definují rozměry znaku. Potom si natáhneme dříve uložený obrázek se současným posunem a závěr opět přenecháme makru `ship_char`.

```
def endchar =
```

```
 scantokens extra_endchar;
 cullit;    mypicture:=currentpicture;   ship_char;
 %
 begin_char(c+1,sec_w_sharp-fi_w_sharp,half_h_sharp,
  d_sharp,sec_w-fi_w+1,half_h,dp);
 currentpicture:=mypicture shifted (-fi_w,0);  ship_char;
 %
 begin_char(c+2,thi_w_sharp-sec_w_sharp,half_h_sharp,
  d_sharp,thi_w-sec_w+1,half_h,dp);
 currentpicture:=mypicture shifted (-sec_w,0);  ship_char;
 %
 begin_char(c+3,w_sharp-thi_w_sharp,half_h_sharp,d_sharp,
  full_w-thi_w+1,half_h,dp);
 currentpicture:=mypicture shifted (-thi_w,0);  ship_char;
 %
 begin_char(c+4,fi_w_sharp,h_sharp-half_h_sharp,0,
                              fi_w,full_h-half_h+1,0);
 currentpicture:=mypicture shifted (0,-half_h);  ship_char;
 %
 begin_char(c+5,sec_w_sharp-fi_w_sharp,
  h_sharp-half_h_sharp,0,sec_w-fi_w+1,full_h-half_h+1,0);
 currentpicture:=mypicture shifted (-fi_w,-half_h);
                 ship_char;
 %
 begin_char(c+6,thi_w_sharp-sec_w_sharp,
  h_sharp-half_h_sharp,0,thi_w-sec_w+1,full_h-half_h+1,0);
 currentpicture:=mypicture shifted (-sec_w,-half_h);
                 ship_char;
 %
 begin_char(c+7,w_sharp-thi_w_sharp,h_sharp-half_h_sharp,0,
  full_w-thi_w+1,full_h-half_h+1,0);
 currentpicture:=mypicture shifted (-thi_w,-half_h);
                 ship_char;
enddef;
```

Úkolem makra `ship_char` je výběr odpovídající části znaku. Ta je definována vnitřními proměnnými w, h, d, které získávají svoji hodnotu v makru `begin_char`. Nyní tuto oblast vyplníme a poté vymažeme dostatečně velký obdélník, v němž je zcela jistě celý znak obsažen. V původním

znaku jsme zajistili, že pixely, které mají být černé, mají hodnotu 1. Po provedení příkazu `fill` budou mít černé pixely hodnotu 2 a bílé pixely hodnotu 1. Příkaz `unfill`[1]) pak zmenší hodnotu všech pixelů o jednotku. Kladnou hodnotu pak budou mít pouze černé pixely v obdélníku, který jsme vyplnili. Tím tedy máme vybránu osminu obrázku. Dále následují příkazy převzaté z původní definice `endchar`.

```
def ship_char =
 wait;
 fill (0,-d)--(w,-d)--(w,h)--(0,h)--cycle;
 wait;
 unfill (-2full_w,-2full_h)--(2full_w,-2full_h)--
   (2full_w,2full_h)--(-2full_w,2full_h)--cycle;
 if proofing>0: makebox(proofrule); fi
 chardx:=w;     % desired width of the character in pixels
 wait;
 shipit;
 if displaying>0: makebox(screenrule); showit; fi
 endgroup
enddef;
```

Nakonec si nadefinujeme pomocné makro, které nemá vztah k porcování obrázků. Během testování se chceme na své obrázky podívat, a proto je musíme na obrazovce na chvíli zastavit. Nenapadlo mě nic lepšího než následující makro:

```
def wait =
 if wait_n>0:
  oldpen:=savepen;      pickup nullpen;
  for j:=0 upto wait_n: drawdot origin; endfor;
  pickup oldpen;
 fi;
enddef;

if unknown wait_n: wait_n:=0; fi;
```

[1]) Rychlejší a bezpečnější (není nutno počítat rozměry) je asi „`cull currentpicture keeping (2,2);`“, ale kvůli spěchu jsem to již netestoval.

Hodnotu `wait_n` nesmíme přehnat; doporučuji pro první pokusy použít `wait_n=5`.

Obrázky jsme tedy vytvořili. Nyní je musíme vysázet. Opět jsem si vzal inspiraci z programu BM2FONT.

Následující makra jsou psána pro LaTeX. Lze je snadno upravit i pro plain TeX. Je pouze nutné nahradit makra definovaná v souboru `latex.tex` nebo jejich definice doplnit do zdrojového souboru. Pouze makro `\setfig` je určeno pro uživatele, ostatní jsou pouze pro experty. Proto se ve jménech maker objevuje znak „@“. Nezapomeňte změnit jeho `\catcode` na 11, jinak nebudou definice správně fungovat.

Makro `\setfig` má jeden parametr. Tím je kód první části obrázku. Chceme-li tedy vysázet obrázek, který je složen ze znaků s kódy 16–23, zadáme ve svém dokumentu:

```
\setfig{16}
```

Makro nejprve uloží kód do čítače `\@fig` a poté přepne font `\fig`. Ten musí obsahovat definice obrázků. Vodorovné řady se pak vloží do boxů 253 a 254, což si v LaTeXu mohu dovolit (nesmím použít box 255).

```
\def\setfig#1{\global\@fig=#1\relax{\fig
  \parindent\z@ \parskip\z@ \offinterlineskip
  \setbox254=\hbox{\put@fig}\global\fig@wd=\wd254%
  \setbox253=\hbox{\put@fig}
  \vbox{{\hsize=\fig@wd
   \copy253\copy254}}}}
```

Pro sesazení obrázku si nadefinujeme dvě drobná makra, které za nás provedou část práce.

```
\def\put@fig{\put@@fig\put@@fig\put@@fig\put@@fig}
\def\put@@fig{\char\@fig \global\advance\@fig\@ne}
```

Nesmíme zapomenout na proměnné, které budeme při skládání potřebovat.

```
\newcount\@fig   \newdimen\fig@wd
```

Text maker obsahoval `\z@`, což v LaTeXu[2]) znamená současně rozměr 0 pt i hodnotu 0, a `\@ne`, což je hodnota 1.

Tím je vše skončeno a vynaložené úsilí již může nést ovoce.

Zdeněk Wagner
`wagner@csearn`

Dr. Halo, UFO a TeX

Radovan Wiszt

V predchádzajúcich bulletinoch sa už neraz objavili články o tom, ako kresliť v TeXu. Drvivá väčšina autorov však začala a skončila METAFONTom. Ako začiatočník pracujúci s TeXom však potrebujem kresliť obrázky už teraz a hlavne s niečím čo už poznám v tejto chvíli.

Najznámejším kresliacim programom je asi Dr. Halo. Predpokladám, že k Dr. Halo má čitateľ tohoto článku aspoň trochu priateľský vzťah. Povedzme si, ako dostať obrázok z jeho výsostných vôd do vôd TeXu. Predstavme si, že sme vytvorili pomocou Dr. Halo obrázok ufáka letiaceho v tanieri. Ak ho uložíme pomocou symbolu diskety, dostaneme súbor ufo.pic, ktorý je však pre tlač v TeXu nanič. Obrázok treba uložiť pomocou symbolu nožníc. Tým dostaneme súbor ufo.cut. Má to len jednu nevýhodu. Nožnicami totižto nenačítame celú kresliacu plochu Dr. Halo, a preto je veľkosť obrázku obmedzená maximálnym rámikom nožníc, t.j. asi 40 % celkovej plochy.

V ďalšom kroku je potrebné súbor ufo.cut premeniť do formátu programu PC Paintbrush. Využijeme pritom sharewarový program gws. Po spustení gws.exe sa zjaví výpis aktuálneho adresára. Presunieme sa do adresára, kde je uložený súbor ufo.cut. Nakoľko PaintBrush pracuje s bielym pozadím ako implicitným a Dr. Halo s pozadím čiernym, zídeme kurzorom na ufo.cut a stlačíme „F6 to reverse“. Vyberieme z ponuky formát PCX (PC Paintbrush) a po potvrdení sa vytvorí súbor r_ufo.pcx. Ak máme nastavené cesty podľa pražských inštalačných diskiet, presunieme súbor r_ufo.pcx do adresára c:\tex\dvi.

[2]) i v plainu