Zpravodaj Československého sdružení uživatelů TeXu

Karel Horák

Jak si poradit s velkými METAFONTovými obrázky

Zpravodaj Československého sdružení uživatelů TeXu, Vol. 4 (1994), No. 4, 154–162

Persistent URL: http://dml.cz/dmlcz/149724

CLB: Poslední otázka, pro vás ta nejprotivnější... Jaký je váš současný plán na dokončení všech sedmi svazků *The Art of Computer Programming*?

Knuth: Chystám se vydávat dvakrát ročně fascikly o zhruba 128 stranách. Nejdříve shromáždíme čtyři z nich a pak teprve vypustíme první dva; budeme si nechávat něco v záloze! První fascikly čekejte v roce 1995 nebo 1996; budou to testovací beta verze opravdových knih. Počítám, že bych mohl dokončit svazek IV (části A, B a C) v roce 2003, svazek V v roce 2008, pak přijít s novými vydáními svazků I, II a III, potom dělat na VI a VII. Bude i jedna verze jakožto „čtenářský výtah" ze svazků I až V.

CLB: Jaká by byla vaše kariéra a život, kdybyste nebyl ohlásil ten sedmisvazkový soubor?

Knuth: Počkat, nejdříve jsem to neohlašoval. Myslel jsem si, že píšu jedinou knihu. Ale i kdybych to nebyl udělal, počítám, že bych stejně hodně psal. Nějak se zdá, že po celou tu dobu jsem vždy nacházel zalíbení v pokusech vysvětlovat věci. Na střední škole jsem vydával studentský plátek; na vyšší škole jsem redigoval časopis. Vždycky jsem si hrál se slovy.

(překlad: *Ladislav Lhotka*)

Následující příspěvek byl původně napsán v angličtině pro EuroTEX 94. Vždycky jsem si myslel, že je pro autora snadné napsat pak totéž i česky, takže do posledního okamžiku jsem doufal, že mě to nebude stát víc energie, než kolik je potřeba na přeťukání. To, že článek vychází česky teprve teď, kdy hlavní důvody pro jeho napsání částečně pominuly, je bohužel jen moje ostuda.

Jak si poradit s velkými METAFONTovými obrázky

Karel Horák

Před nějakým časem referoval o podobném problému na stránkách tohoto občasníku kolega Zdeněk Wagner [2]. Řešení, které je v jeho článku navrženo, předpokládalo rozklad obrázku na několik menších obdélníků

a poté jejich následné složení dohromady při sazbě pomocí TEXových boxů. V podstatě šlo o stejný princip, jaký používá program `bm2font` F. Sowy: to je rozumné řešení, protože samotný TEX je natolik přesný, že se nemusíme obávat nějakých nepřesností, jež by čtenáři prozradily, že obrázek sestává z menších částí.

S podobnými problémy při tisku větších METAFONTových obrázků jsem se setkal i já, i když ne v takové míře, jak to kolega Wagner tehdy popisoval. Jak je asi všeobecně známo, dokázaly emTEXovské programy tisknout i hodně velké obrázky (skoro bych tvrdil, že větší, než často bylo možno samotným METAFONTem vzhledem k problémům s pamětí vůbec vyrobit). Jakmile ovšem bylo třeba tisknout otočeně či zrcadlově, byl konec pohody, protože v takových případech šlo tisknout jen charaktery, jejichž počet pixelů nebyl větší než cca 230 000, což při kvalitě běžné laserové tiskárny nestačilo ani na obrázky velké 5×5 cm (`bm2font`, aby vyhověl většině běžných ovladačů tiskáren, se omezuje jen na palcové charaktery). Tuto hranici jsem u svých obrázků překračoval dost často.

Tyto problémy se přirozeně znásobily přechodem z 300 na 600 dpi. Do té doby jsem uvedený problém (např. při tisku otočeně pro sestavení brožurky anebo při zrcadlovém tisku na pauzák sloužící jako ofsetová předloha) řešil víceméně technicky: pomocí emTEXového ovladače dvidot s parametrickým souborem `pcx.dot` (původně emTEX obsahoval k tomu účelu samostatný program dvimsp, až později jej E. Mattes učinil součástí ovladače jehličkových tiskáren) jsem kritický obrázek převedl na pcx soubor a ten zas hned pomocí programu `bm2font` do bitové mapy ve tvaru `pk` (emTEX sice umí tisknout pomocí příkazu `\special` i grafické soubory typu pcx, ale to bychom byly tam kde předtím, protože na ně nefungují obvyklé transformace). Na tom mi trochu vadilo, že při novém začlenění takto upravených obrázků do textu se zdrojový text zaneřádil dočasně existujícími soubory (navíc svým charakterem silně závislými na zvoleném výstupním zařízení), které ovšem nestálo zato archivovat, protože se daly kdykoli znovu vyrobit popsaným způsobem. Později jsem proto začal občas dávat přednost tomu tisknout takto celé stránky, které obsahovaly problematické obrázky (což šlo díky schopnější verzi programu `bm2font`, které při rozlišení 300 dpi nevadila ani strana formátu B 5). Celý postup jsem snadno zautomatizoval jak ve své původní dávce, tak i v souboru `texbat.bat`, když jsem začal používat program mnu Petra Olšáka.

Po přechodu k LaserJetu 4 se 600 dpi se ovšem začal tento postup zdát málo efektivní, zvláště když jsem před konečným tiskem dospěl k nut-

nosti drobných oprav a musel tak uvedený postup několikrát opakovat. Na tom asi není nic překvapivého, když si uvědomíme, že najednou musel `bm2font` zpravovávat obrázky čtyřikrát větší. To byl patrně hlavní důvod, proč jsem začal raději uvažovat o jiném řešení.

Dospěl jsem tedy podobně jako kdysi kolega Wagner k přesvědčení, že bude potřeba obrázky nějak rozsekat na menší „podobrázky“. První užitečný nápad pramenil ze známé skutečnosti, že TEX zajímají pouze metrické informace skryté v souboru TFM a je mu úplně jedno, jak vypadají použité bitové mapy, které jediné při tisku činí potíže. Bude tedy nejjednodušší vyrobit všechny části budoucího obrázku jako charaktery se shodným referenčním bodem a s vlastní šířkou nulovou, protože TEX je pak postupně vysází všechny do téhož místa, jednotlivé části obrázku se budou překrývat, a nemusíme se tak starat o to, aby zvolený rozklad byl rozkladem i v matematickém smyslu, tedy na navzájem disjunktní části. Naopak, některé části se v takovém případě mohou klidně i opakovat. Samozřejmě že nejpohodlnější je zůstat u obdélníků, ale nemusíme se už tolik starat o jejich (nenulovou) společnou hranici.

Zůstaneme tedy u obdélníků, jak ale mají být veliké? To samozřejmě závisí především na schopnostech zvoleného ovladače. Ve starší verzi Mattesových ovladačů (1.4x, x $\leq$ s) byla problematická hranice někde kolem 480 pixelů (tedy pokud jsme chtěli tisknout s použitím některé ze sedmi netriviálních transformací — při normálním tisku jsem nikdy na problémy nenarazil, snad jedině že natažení fontu do tiskárny trvalo trochu déle než u malých písmenek). Pokud tedy zvolené rozměry našeho obrázku (šířka {width}, výška {height} a hloubka {depth}) odpovídají skutečným rozměrům toho, co nakonec METAFONT vyrastruje, pak určitě můžeme volbu odpovídajících obdélníků vcelku snadno zautomatizovat: následující výpočet v rámci METAFONTového programu dá postačující počet $n\_x \times n\_y$ obdélníků, které daný obrázek pokryjí,

```
n_x[charcode]:=ceiling(w/480);
n_y[charcode]:=ceiling((h+d)/480);
```

Hotový obrázek pak můžeme uschovat do proměnné typu `picture` (v našem případě to bude `pic_[charcode]`), ten přidáme k příslušnému vyplněnému obdélníku a to, co bude mít obdélník s naším obrázkem společného, uchováme příkazem `cull currentpicture keeping (2,2)`:

```
picture pic_[];
```

```
beginchar(1,45mm#,45mm#,mm#);
z1=origin;z2=(w,y1);z3=(.7w,h);
pickup pencircle scaled .2mm;
 draw z1--z2--z3--cycle;
cullit;
pic_[charcode]:=currentpicture; clearit;
n_x[charcode]:=ceiling(w/480);
n_y[charcode]:=ceiling((h+d)/480);
endchar;
```

Zde jsme pro jistotu použili příkaz `cullit`, abychom měli jistotu, že uschovávaný obrázek (`currentpicture`) nebude mít „násobné“ pixely. Následujícím příkazem `clearit` vyčistíme obsah aktuální proměnné `currentpicture`, což znamená, že samotný charakter `\char1` nebude obsahovat jediný černý bod, jediný pixel. Nakonec v následujícím cyklu vygenerujeme všechny potřebné charaktery s částmi našeho obrázku, přičemž všechny budou mít (pro TEX) nulovou šířku:

```
for $=0 upto n_x1*n_y1-1:
beginchar(incr charcode,0,0,0);
fill unitsquare scaled 480
 shifted(480*($ mod n_x1-.5,floor($/n_x1)));
addto currentpicture also pic_1;
cull currentpicture keeping (2,2);
endchar;
endfor
```

Až dosud jsme se nemuseli příliš starat o počet vzniklých znaků, ale v okamžiku, kdy je máme v dokumentu vysadit, už by se tato informace asi hodila. Ovšem při znalosti počtu $n\_x \times n\_y = n$ výsledných pravoúhelníků lze použít TEXovskou smyčku (`\loop`). Předpokládáme-li opět, že $n = 9$, jak tomu bylo v našem ilustračním příkladu pro rozlišení 600 dpi, stačí uvést

```
\newcount\N \N=1
\loop
 \ifnum\N<9\char\the\N\advance\N 1
\repeat
```

Naštěstí ale můžeme TEX k témuž výsledku přivést i jinak, bez znalosti skutečného počtu vzniklých obdélníků. Využijeme-li nové ligační schéma TEXu 3.1... a odpovídajícího METAFONTu 2.7..., kde D. E. Knuth poprvé zavedl možnost „okrajových“ (boundary) ligatur [1], můžeme v METAFONTovém zdrojovém textu napsat

```
boundarychar:=255;
```

a pak přidat následující schéma (pořád ještě předpokládáme, že předchozí METAFONTový obrázek byl rozdělen na $3 \times 3 = 9$ částí):

```
ligtable 1:   255  =:| 2;
ligtable 2:   255 |=:| 3;
ligtable 3:   255 |=:| 4;
ligtable 4:   255 |=:| 5;
ligtable 5:   255 |=:| 6;
ligtable 6:   255 |=:| 7;
ligtable 7:   255 |=:| 8;
ligtable 8:   255 |=:| 9;
ligtable 9:   255 |=:| 10;
ligtable 10:  255 |=:  1;
```

což se dá zjevně ještě zestručnit:

```
ligtable 1:   255  =:| 2;
for $=0 upto n_x1*n_y1-2:
    ligtable $+2:         255 |=:| $+3;
endfor
    ligtable n_x1*n_y1-1+2: 255 |=:  1;
```

takže pak v TEXovém souboru stačí napsat

```
\font\fig  figures
{\fig\char1 }
```

kde mezera tvořící hranici tohoto triviálního slova zároveň způsobí vysazení `\char2\char3...\char9\char10\char1`. Ve skutečnosti není ovšem mezera potřebná, jak jsem si zprvu myslel, TEX „pozná“ konec „slova“ i bez ní. To jsem si však uvědomil až mnohem později, protože psát na

tomto místě mezeru jsem byl zvyklý už dávno (pro obrázkové charaktery mezeru (`\space`) obvykle nedefinujeme, proto je její výskyt v uvedené závorce bezvýznamný. Na druhou stranu podléhám nyní pocitu, že právě zmíněný zvyk psaní mezery byl počátečním impulsem ke vzniku popisovaného řešení.

Tak, teď už je snad vše jasné, takže se můžeme podívat na definitivní implementaci odpovídajících METAFONTových maker. Podíváme-li se na důležitá makra `beginchar` a `endchar` v METAFONTbooku, snadno též napíšeme obecnou proceduru, která vše včlení do makra `endchar` prostřednictvím `extra_endchar`.

```
%%%%%%%%  divide.mf  %%%%%%%%
boundarychar:=255;
picture pic_[];

max_charcode:=0;
def char_i(expr c)=
    i:=c; CHars[i]=1;
    max_charcode:=max(c,max_charcode);
enddef;

extra_beginchar:=
 extra_beginchar&"char_i(charcode);";

def divide_pic=
 cullit;
 pic_[charcode]:=currentpicture; clearit;
 n_x[charcode]:=
   ceiling(charwd*hppp/480)+1;
% show n_x[charcode];
 n_y[charcode]:=
   ceiling((chardp+charht)*hppp/480)+1;
 n_d[charcode]:=
   ceiling(chardp*hppp);
% show n_y[charcode];
enddef;

extra_endchar:=
 extra_endchar&"divide_pic;";
```

```
def do_pictures=
 def divide_pic= enddef;
 charcode:=max_charcode;
 for ii=0 upto charcode: save Charcode;
 Charcode:=ii;
 if known CHars[ii]:
 ligtable ii: 255  =:| charcode+1;
 for $=0 upto n_x[ii]*n_y[ii]-1:
 beginchar(incr charcode,0,0,0);
 fill unitsquare scaled 480
 shifted(n_d[ii]*down+480*
  ($ mod n_x[ii]-.5,floor($/n_x[ii])));
 addto currentpicture also pic_[ii];
 cull currentpicture keeping (2,2);
 ligtable charcode: 255
 if $<n_x[ii]*n_y[ii]-1: |=:| charcode+1
 else: |=: Charcode fi;

 endchar;
 endfor    fi
 endfor
enddef;
```

Máme-li teď nějaké větší obrázky, se kterými jsou při tisku potíže, stačí načíst soubor `divide.mf` a použít před koncem souboru `end` příkaz `do_pictures;`. Ale i toho se můžeme zbavit, když v souboru `divide.mf` vhodně změníme definici konce programu:

```
inner end, bye; let saveEnd=end; let saveBye=bye;
def end= do_pictures;
    saveEnd;
enddef;
let bye=end;
outer end, bye;
```

Samozřejmě při tom všem musíme ještě trochu dát pozor na jiná chytrá makra, která s oblibou používáme — v mém případě např. skvělý `labtex` Alana Hoeniga, kterým od EUROTEXu'92 soustavně umisťuji popisky

do obrázků. Ten totiž zas používá kerningové informace na přenos souřadnic popisovaných bodů prostřednictvím `TFM` z METAFONTového do TEXového souboru. Takže stačilo přesunout tento úkol z prvních tří znaků 0, 1, 2 na poslední 253, 254, 255 (začínat kvůli tomu číslování vlastních obrázků vždy až od 3 mi nepřipadalo příliš přirozené, zvlášť když jsem celé popisované řešení považoval jen za dočasné).

Jiným takovým případem mohou být chytrá makra Olina Ulrycha `incpic`, která používají jiný bezrozměrný souřadný systém a proměnnou `current_transform`, aby zprostředkovala vztah mezi skutečnými a uživatelskými jednotkami. V tomto případě postačí doplnit příkaz `notransforms` před použitím `do_pictures`.

Možné problémy a závěrečné poznámky

Především je třeba si uvědomit, že ukládání obrázků do proměnné `picture` má své meze, neboť rychle zaplňuje paměť, kterou má METAFONT k dispozici, a i při používání Mattesova `mf386` na konečnou kompilaci málokdy svedeme takto víc než 20 obrázků najednou. A s méně mocnými verzemi METAFONTu asi přiměřeně míň.

Stejně jako u programu `bm2font` závisí rozsekávání obrázku víc na počtu pixelů než na jeho skutečných rozměrech, takže konečná rozlišovací schopnost, pro kterou obrázky připravujeme, hraje důležitou roli. Tak, jak jsou makra napsána, dostaneme pro různou rezoluci různé metrické údaje (a samozřejmě také úplně jiné rozložení charakterů ve fontu). Samozřejmě se tomu můžeme vyhnout tím, že dělení obrázků zvolíme v závislosti na nejvyšším rozlišení, které používáme, a nebudeme je pro nižší rozlišení měnit. Také můžeme používat pro různá rozlišení různé adresáře a měnit přístup TEXu do nich změnou příslušné systémové proměnné.

Uvedený postup se samozřejmě může občas hodit i v jiných případech. DOSovská verze Rokického `dvips`, která teď stejně jako emTEXovské programy používá Mattesův ovladač emx, si sice už tak často nestěžuje, že není schopna zpracovat větší charakter kvůli limitu paměti 64 kB, nicméně myslím, že si stále ještě neporadí s tak velkými charaktery, které je METAFONT schopen vytvořit… Ovšem spíš se může tento postup hodit při rastrování větších ploch METAFONTem, kdy si METAFONT začne velmi brzy stěžovat na nedostatek paměti a začne být neúměrně pomalý.

Téměř přesně po roce, kdy se objevila první z nových verzí ovladačů Eberharda Mattese řady 1.5, které dovedou download na šestisetbodové tiskárny HP, se ve verzi 1.5h zřejmě podařilo odstranit nedostatky při tisku velkých charakterů při použití transformací `/tr1`, `3`, `4`, `6`, které

jsem až dosud musel obcházet právě popsaným způsobem. U této zatím poslední verze přestala konečně zlobit i hláška `internal error 3073: XMS error: 164`. Po intenzívním ročním používání tak snad budu moci odložit `divide.mf` ad acta. V každém případě mi však tato netradiční možnost využití ligačních schopností TEXu a METAFONTu přinesla radost a potěšení z toho, jak je ten TEX s METAFONTem pěkně vymyšlen!

A nejlepší (zejména pro příznivce METAFONTu) zpráva nakonec. Přesto, že popsaný postup překonává některé potíže při používání METAFONT k rýsování obrázků, některá omezení kompilátoru tohoto skvělého programovacího jazyka jsou zřejmě nepřekonatelná. Tedy pokud zůstane METAFONT jenom tím programem, jak ho (pro návrh písma ovšem) navrhl jeho autor, Donald Knuth. Tedy přesněji řečeno, pokud bude METAFONT schopen pouze generovat bitové mapy až do určitého počtu pixelů. Naštěstí jeden z Knuthových žáků, John Hobby, dokázal vnutit jazyk METAFONTu také PostScriptu a vytvořil program METAPOST, který rozumí příkazům METAFONTu, ale místo bitové mapy produkuje PostScriptový výstup. A od ledna je tento program také *in public domain*, tedy volně dostupný a šiřitelný. Zatím jen ve své UNIXové verzi, ale John Hobby zároveň slibuje v dohledné době připravit i verzi METAPOST pracující pod DOSem. Takže můžeme dál využívat skvělého a geometricky názorného Knuthova jazyka a nestarat se už tolik o závislost na rozlišení příslušného výstupního zařízení. Až se budu moci s METAPOST seznámit na vlastní kůži, rád poskytnu lepší odpověď na otázku v titulu tohoto článku, než se mi dosud mohlo podařit.

Literatura:

[1] Donald E. Knuth: *The new versions of TEX and* METAFONT, *TUGboat* 10 (1989), 325–328.

[2] Zdeněk Wagner: METAFONT *a velké znaky*, *TEXbulletin* (1993), 77–83.