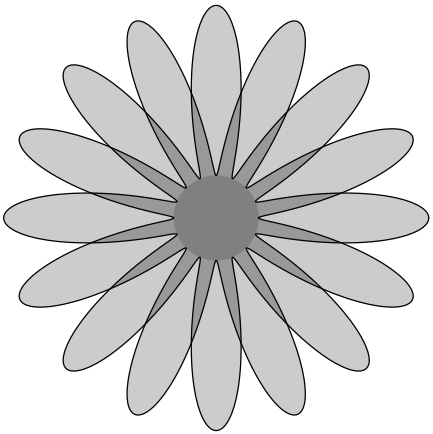
TEX METAFONT TEX METAFONT TEX METAFONT TEX METAFONT



2

ZPRAVODAJ 94

ČESKOSLOVENSKÉHO SDRUŽENÍ UŽIVATELŮ TEXU

OBSAH

Podle informace kolegy Petra Pražáka je možno ukládat zajímavé soubory i na BBS a rozšiřovat je tak mezi obecný lid. Pan Pražák je dále napojen i do sítě FIDO, takže je možno jeho prostřednictvím zasáhnout i tam. Příslušná čísla uveřejňujeme.

Petr Pražák `<pr@vema.cz>`
VEMA Brno tel: 5-4321 5524/106
Výstavní 17/19 fax: 5-4321 1946
603 00 Brno BBS5-4321 1633

Gates versus Mattes

Úvaha o postavení TEXu v komerčním prostředí

Petr Olšák

Prosím, berte tento text jako mou osobní úvahu, se kterou můžete, ale samozřejmě nemusíte souhlasit. Text byl napsán 28. 10. 1994.

Mnozí z nás si vzpomínají na počítačové začátky v naší republice. Není to tak dávno, co zde vládly jen sálové počítače typu EC (единная система), a pak to přišlo. Osobní počítače typu IBM PC zavalily nejen svět, ale i naší republiku. Tím se dostala dříve skoro nedostupná výpočetní technika na stoly koncových uživatelů. Největší přelom byl v dostupnosti počítačové grafiky a vzniku aplikačních programů spolupracujících s uživatelem na bázi vzájemné interakce.

Hned nad prvními počítači tohoto typu (XT) jsme začali jásat, že s nimi jde dělat věc naprosto nevídaná – počítačová sazba za pomocí programu TEX. Kdo se v té době do TEXu zamiloval, asi jej už neopustí, protože TEX je skutečně perfektně koncipovaný a myšlenkově bohatý a krásný program. Rychlost příchodu TEXu pro první osobní počítače byla dána tím, že autor programu Donald Knuth jej vytvořil tak, aby byl snadno implementovatelný pro libovolný počítačový systém, který splňuje jisté minimální kapacitní nároky na hardwarové vybavení počítače. Počítače XT byly na samotné hranici těchto nároků a při implementaci bylo potřeba udělat jistou netriviální práci a spokojit se s kapacitními omezeními programu samotného.

První TEXy na XT a později AT běhaly v naší republice zřejmě za použití komerčního balíku PC-TEX a teprve později se v našich myslích objevilo jméno studenta techniky v Německu – Eberharda Mattese. Tento pán doslova zruinoval firmu, která založila svou existenci na prodávání PC-TEXu (TEXu pro DOS), jednoduše tím, že dal svou implementaci TEXu pro DOS a OS/2 (včetně vlastních programů–ovladačů `dvi`) světu zadarmo. Implementace se jmenuje emTEX. Nakonec nás jeho jednání tolik nepřekvapuje, protože víme, že samotný Knuth daroval svůj program ve zdrojové podobě veřejnosti a tím nastartoval jeho šíření hlavně po nekomerčních cestách. Znamená to, že implementace pro různé operační systémy vznikaly většinou v rukou fandů pracujících vesměs na univerzitách a nikoli v Křemíkovém údolí.

V posledních několika letech ovšem toto TEXovské nadšení dostává vážnou ránu v podobě šíření komerčních programů, které řeší více či méně totéž: vytvořit v počítači sazbu. Těmto programům se většinou říká DTP (DeskTop Publishing). Navíc ještě existují programy, které se nejprve zrodily jako ASCII editory, pak začaly nabízet uživateli jednoduché možnosti na zpracování textu určeného pro vytištění (textové procesory) a nyní aspirují na to nahradit DTP systémy. Příkladem může být editor WordPerfect.

Všechny tyto programy sledují hardwarový vývoj osobních počítačů a pracovních stanic, zvláště v oblasti grafiky. S vývojem grafiky a zvyšování výkonu procesorů a pamětí přicházejí nové „lepší“ verze těchto programů, které naplno využívají hardwarové možnosti nových počítačů. To je samozřejmě správné. Za tyto nové verze se většinou draze platí a navíc datový formát, se kterým programy pracují, bývá v každé verzi jiný. Příkladem může být znova editor WordPerfect.

TEX ale zůstává stále stejný a jeho autor se dokonce odvažuje tvrdit, že TEX bude za 100 let dávat stejné výsledky, jako dává nyní, a bude jedno, jaká v té době bude pracovat výpočetní technika. Je to při pohledu na prudký počítačový rozvoj posledních let velmi překvapivé tvrzení. Mnoho lidí by si dokonce mohlo myslet, že to hraničí až s podivínstvím. Autor Donald Knuth ale ví, co říká. Je skutečností, že jeho program pracuje v podstatě v nezměněné podobě už od roku 1982 (tj. pokud jsme něco napsali v roce 1982 v TEXu, dnes to vysázíme naprosto stejně jako tenkrát). Takovou životnost (k dnešnímu dni 12 let stabilního života) nemá a nemůže mít žádný komerčně šířený program. Program TEX totiž stojí na myšlenkách (odvažuji se dokonce napsat Myšlenkách s velkým M) a nikoli na současné hardwarové a softwarové situaci, v jaké se momentální výrobky výpočetní techniky nalézají.

Investice naučit se TEX je tedy dlouhodobá investice, ačkoli toto učení může být ze začátku dosti náročné. Na druhé straně, pokud pošlete někoho na školení o konkrétním DTP, pak za toto školení vyhodíte nekřesťanské peníze a to, co se tento člověk naučil, může už za dva roky s klidným svědomím zapomenout.

Uživatel TEXu má též širokou podporu při řešení svých problémů. Může se obrátit na elektronické diskusní skupiny o TEXu vedené na univerzitních počítačových sítích (na národní i mezinárodní úrovni). Vždy se najde nějaký zkušenější uživatel, který v těchto skupinách odpoví. Má-li uživatel pocit, že něco nefunguje tak, jak má (nějaké makro nebo dokonce samotný TEX), má možnosti prostřednictvím elektronické sítě

dát tuto věc najevo. Většinou mu během krátké doby kvalifikovaně odpoví samotní autoři. Knuth vypisuje dokonce odměnu za to, když mu někdo v programu najde chybu. V současné době je TEX pravděpodobně program s nejméně chybami na světě (srovnáváme-li programy srovnatelné složitosti).

Vše, co tady dosud říkám, jsou argumenty, proč použít TEX. Dalo by se jich najít mnohem více. Jako poslední argument uvedu skutečnost, že s TEXem mohou pracovat lidé na různých úrovních poznání tohoto programu a vytvořit tak jakousi ideální dělbu práce.

TEX relativně snadno začnou používat uživatelé, kteří se naučí pár LATEXovských příkazů, mají vedle klávesnice tabulky některých příkazů (např. že `\alpha` způsobí α) a pro jejich potřeby (v případě písařky potřeby nadřízených) jsou tyto znalosti postačující.

Pak jsou uživatelé, kteří umějí efektivně používat vesměs celé makro LATEXu a sledují nejnovější vývoj tohoto makra. Bohatost grafických efektů, které použijí ve svých dokumentech je více méně závislá na existenci už vytvořených maker a případně dalšího softwaru, který daný efekt umožní realizovat. Tyto možnosti jsou velmi rozsáhlé. Pro ilustraci na univerzitních sítích se potulují (samozřejmě zadarmo) stovky až tisíce TEXovských maker a dalšího podpůrného softwaru, který řeší nepřeberné množství grafických požadavků na sazbu. Probrat se jenom informacemi o těchto podpůrných TEXovských prostředcích je mnohdy nadlidské úsilí. Veřejné TEXovské archivy obsahují řádově gigabajty nejrůznějšího podpůrného software k TEXu.

Na dalším stupni jsou uživatelé, kteří si umí psát sami makra a jednoduchý software podporující rozhraní mezi TEXovskou sazbou a grafickým prostředím dnešních dnů. Odtud už není daleko k programátorům `dvi` ovladačů pro všechna možná i nemožná výstupní zařízení a implementátorům TEXu pro různé nové operační systémy. To už ale nejsou uživatelé TEXu – stávají se jeho služebníky. Sami ze svého pera (přesněji ze své klávesnice) dávají k dispozici do univerzitních sítí makra a software, podporující různé grafické efekty v rámci sazby v TEXu. Tato dělba práce je více méně ideální. Pokud jsou tito programátoři dobře vychováni i v oblasti písma a typografické kultury, pak je šance, že uživatelé jejich maker budou mít sazbu po všech stránkách kvalitní. Takovou dělbu práce předpokládal samozřejmě samotný Knuth v době, kdy dával program k dispozici.

Citelně chybí jistá skupina uživatelů TEXu, kterou bychom mohli nazvat „profesionální sazeči v TEXu“. Tito lidé by měli převzít text od

písařky, která umí napsat `\alpha`, a zpracovat tento text do výsledné podoby. Měli by důkladně znát TEX a typografická pravidla, umět zařadit libovolný grafický efekt do sazby. Přitom by asi úzce spolupracovali s tvůrci nejnovějšího podpůrného software k TEXu. Kdyby takoví lidé pracovali v jednotlivých nakladatelstvích, pak by se TEX mohl snadno stát standardem pro vyjádření požadavků sazby a tyto texty by se mohly archivovat a znovu použít v dalších vydáních knihy. V dalším textu se pokusím rozvinout úvahu, proč tito lidé skutečně chybí.

Pusťme se nyní do rozboru zásadní nevýhody TEXu. Odvažuji se tvrdit, že tato nevýhoda spočívá ve vlastnostech samotného autora TEXu. Má představa totiž je, že je to člověk nadprůměrně inteligentní (neví někdo, zda je členem hnutí Mensa?). Lidé tohoto druhu mají většinou jednu nevýhodu. Pohybují se v prostředí lidí aspoň trochu srovnatelné inteligence a vzdělání (například v akademickém prostředí), komunikují s nimi a jsou do jisté míry izolování od lidí průměrně uvažujících. Výsledky intelektuální činnosti nadprůměrně inteligentních lidí mohou pak zpracovat a důkladně využít znova jen lidé, kteří se aspoň nepatrně přibližují schopnostem těchto géniů. Tyto výsledky jsou pro průměrně uvažující jedince vesměs nedosažitelné. Myslím, že to platí obecně pro libovolnou vědní disciplínu, nejen v případě Knutha a jeho TEXu. Stačí připomenout třeba kvantovou teorii ve fyzice.

Program TEX je výsledek intelektuální činnosti jednoho z géniů tohoto století. Podle toho, co jsem popsal v předchozím odstavci, plyne, že nutit TEX průměrně uvažujícímu uživateli je přinejmenším surovost. V době, kdy Knuth pracoval na svém projektu, se nepředpokládalo, že by se počítač dostal do ruky člověku bez schopnosti algoritmického uvažování a jistého nezbytného vzdělání. Dnes ovšem v souvislosti s prudce klesajícími cenami počítačových výrobků a díky snaze komerčního světa prodat tyto výrobky komukoli (třeba cvičené opici) se samozřejmě klávesnice počítače dostává do rukou i lidem bez nejmenšího šajnu o tom, co to je algoritmus. V takovém prostředí se TEX *nemůže* uživit a veškerá jeho propagace pro tyto vrstvy uživatelů je marná.

Právě na tyto vrstvy uživatelů se zaměřil komerční svět. Podle toho vypadají manuály k aplikacím, které většinou neřeknou skoro nic, a algoritmicky uvažující uživatel při jejich četbě skřípe zubama. Zde se skutečně uživí DTP systémy, které uživateli nabízejí konečnou sadu jistých nástrojů v podobě grafických prvků na obrazovce, a veškeré operace, týkající se úpravy sazby do požadovaného formátu se provádějí pohybem myšího kurzoru po pracovní ploše obrazovky.

Mluvil jsem s jedním nakladatelem o argumentech pro a proti TEXu. Říkal, že se mu vyplatí zaplatit cvičenou opici, která zalamuje stránky pomocí nějakého myšoidního DTP systému, než platit člověka s takovým vzděláním, že dokáže účelně využít TEX a vždy v každém okamžiku vývoje výpočetní techniky dokáže promítnout do výsledné sazby současné možnosti grafiky (například při zařazování obrázků a grafických efektů do textu). Takových specialistů, kteří důkladně zvládají TEX (tj. mají k tomu řekněme buňky a chtějí to dělat), se potuluje po světě velmi málo. Zatímco cvičenou opici na DTP kdokoli kdykoli vyškolí. Tento redaktor má v zásadě pravdu, ale smutné a paradoxní (aspoň v naší republice) je skutečnost, že člověk s uvedenými schopnostmi většinou realizuje své nadání v univerzitním prostředí, kde má několikanásobně menší příjem než ona vyškolená cvičená opice. Nad tím by se ovšem měli zamyslet politikové, v jejichž rukách je osud státního rozpočtu. Jinak nám tyto hlavy budou stále utíkat do zahraničí, nebo budou investovat svou tvůrčí invenci do programování bankovních databází.

Mnozí z čtenářů si jistě přečetli Zajíčkův úvodník v Bajtu z ledna 1994 s názvem „Od TEXu ke škvarku“. Mohli jej považovat za počátek konce TEXu, protože tento úvodník vyzněl tak, že TEX je v dnešní době něco úplně neužitečného. To samozřejmě není pravda. Přesto ale s autorem úvodníku v základních myšlenkách souhlasím. Celá pravda je totiž řečena v této větě: „Nadšení z kouzel TEXu vyprchávalo s rostoucí jistotou, že tady pro něj neexistují sazeči, natož programátoři nezbytných (a hodně rozsáhlých) maker dost specifikovaného jazyka.“ V tomto článku nesouhlasím s autorem pouze v jednom detailu, že „…schopnosti TEXu převést bohatě zdobenou publikaci do barevných výtažků jsou minimální…“. To svědčí trochu o nepochopení principů TEXu. Samozřejmě, že TEX jako program nezávislý na pokroku ve výpočetní technice nebude mít v sobě implementovány algoritmy pro barevné výtažkování, které jsou nepochybně závislé na současném rozvoji grafiky. Existuje ovšem plno elegantních způsobů, jak v TEXu pracovat s barvou, zatímco realizace barevných separací se pochopitelně přenechává ovladačům výstupních zařízení (tj. v současné době nejčastěji PostScriptovému RIPu).

Z předchozích úvah snad vyplynulo, proč živná půda pro TEX je a zůstane pouze v univerzitním prostředí. Napomáhají tomu navíc tyto skutečnosti:

- Pro sazbu složitých matematických vztahů ve vědeckých publikacích asi dlouho nebude nic lepšího.

- TEXovské soubory jsou zpracovatelné TEXem s naprosto stejným výsledkem na nejrozličnějších operačních systémech, a v akademických institucích se ve stále větším rozsahu používá výměna vědeckých poznatků prostřednictvím počítačové sítě. TEXovský formát se tak stává jistým „komunikačním protokolem“ v rámci světové sítě počítačů, pokud chceme přenést nejen text, ale i informaci o sazbě tohoto textu.
- Vědecká publikace nemusí být plná pro pohled čtenáře efektních grafických triků, kterou by běžný uživatel, který nevidí do všech možností TEXu, těžko zvládal. Takové záležitosti jsou nutné v reklamních poutačích a v bulvárních plátcích. Proto je nabízejí (a zdůrazňují) veškeré komerční DTP systémy. Je to ovšem na škodu typografické kultury, ale takový je život.

Pokud TEX používají i mimo akademická pracoviště, pak je to proto, že výjimečně pochopili alespoň částečně smysl a hloubku Myšlenek TEXu, kterou uživatelé myšoidních programů nemají šanci pochopit z analogických důvodů, jako běžný řidič autobusu nechápe kvantovou teorii.

V závěru mých úvah bych se chtěl věnovat symbolu „Gates versus Mattes“, který jsem použil v nadpisu svého článku. Naznačuje to vztah mezi komerčním světem a světem počítačových nadšenců, kteří vědí, co dělají. Jejich výrobky často výrazně předčí komerční produkty vytvořené pro podobný účel. TEX samotný je toho jasným příkladem. Mezi dvěma zmíněnými muži samozřejmě vůbec není žádný spor (oba muži si dělají to, co považují za vhodné), ale pro nás uživatele může tento symbol „Gates versus Mattes“ nabýt docela konkrétních podob.

Mnozí čtenáři určitě četli článek pana Kroba o tom, že mattesovská instalace TEXu odmítá akceptovat DPMI, tj. standard správy paměti navržený firmou pana Gatese. Místo toho Mattesovy programy podporují jiné standardy, například VCPI. Veškerý tlak na pana Mattese je zbytečný, protože tento pán prohlásil, že nikdy v životě nechce mít nic společného s takovým výrobkem, jako je Windows pana Gatese. Pan Mattes má pravdu a já mu držím ze všech sil palce. Je ale pravda, že na rozdíl od pana Gatese nepatří pan Mattes mezi nejbohatší muže světa a neprodal miliony instalací svých produktů (za obrovské peníze). Obchodní politika pana Gatese je samozřejmě zaměřena na uživatele typu cvičená opice. Na druhé straně pan Mattes dělá implementace TEXu, takže jeho výrobek je vlastně určen lidem, kteří nemají se cvičenými opicemi příliš mnoho společného. Teoreticky by tedy nemělo docházet

k třecím plochám, ale bohužel dochází. Lidé, kteří chtějí používat TEX, totiž stále častěji potkávají lidi, kteří používají MS-Windows. Nejhorší pak je, když se taková věc sejde v jednom počítači – Gatesovy Windows a Mattesův TEX. Nezbývá, než bootovat jednou kvůli panu Mattesovi a jednou kvůli panu Gatesovi. Nebo používat komplikovaná a většinou výkonnost snižující náhradní řešení.

Můžete namítnout, že argument prodaných milionů kousků Windows je natolik pádný, že by se pan Mattes měl už konečně vzpamatovat. Domnívám se ovšem, že důvody pana Mattese, vedoucí k jeho tvrdohlavosti, stojí u samých kořenů problematiky vztahu dobrých a kvalitních programů šířených zadarmo a komerčního světa, který v případě pana Gatese vede do slepé uličky. Je potřeba si uvědomit, že pan Gates nastoupil do hodně rychle jedoucího vlaku, v jehož základech stojí dnes už naprosto překonaný operační systém DOS. To, že DOS stojí na technicky naprosto zastaralých principech, určitě pan Gates moc dobře ví, nemůže ovšem z tohoto vlaku s názvem DOS vystoupit, protože mezitím tam přistoupily miliony uživatelů po celém světě, kteří by jeho vystoupení těžko nesli, a vedlo by to k jeho pádu. Tak dojde k pádu až tehdy, kdy vlak pojede natolik rychle, že se stařičký DOS v jeho základech zavaří a přivodí pád celého vlaku. Mattes v tomto vlaku nechce být ani pasažérem a nechce svými produkty do tohoto vlaku vtahovat další uživatele. Po technické stránce věci ví moc dobře, proč.

Já spravuji heterogenní DOS–UNIXovou síť a mám své vlastní zkušenosti. Příslušný UNIX mi běží perfektně, DOS samotný, pokud nad ním běží nenáročná aplikace odpovídající jeho schopnostem, je taky bez problémů. Dokonce se ty dva systémy bezproblémově mezi sebou pomocí jistého software domluví. Pokud ovšem nějaký uživatel chce nastartovat Windows nad DOSem, začínají mi padat vlasy z hlavy. Windows se totiž začnou chovat stochasticky. Držím proto palce všem nadšencům, kteří pracují na vývoji operačního systému LINUX, který je k dispozici zdarma na univerzitní síti. Podle propozic by měl LINUX v rámci grafického X-windows prostředí (daleko povedenější standard Woken, ale pro UNIX-like systémy) umět spustit aplikaci projektovanou pro MS-Windows. Pokud by se podařilo odstranit všechny mouchy, které s tím souvisejí, možná by to přivodilo rychlejší pád vlaku, který pilotuje pan Gates. Nakonec by pan Gates za to mohl být vděčný, protože by nepadal v tak velké rychlosti. A kromě pana Gatese by z toho mohli těžit samozřejmě všichni uživatelé a kooperující firmy, kteří v tom vlaku v současné době jedou. Nenabili by si totiž tolik ciferník.

Program TEX zůstane tím, čím je, i za sto let. Je v něm totiž Myšlenka, která přetrvá všechny krachy a jiné geologické pohyby různých Křemíkových údolí. A v tom spočívá podstata rozdílu mezi programy šířenými zadarmo a komerčním světem. Kvalitní lidská myšlenka se totiž nedá zaplatit penězi a snaha po její materializaci formou přesného finančního ocenění je marná.

Virtuální fonty, accents a přátelé

Petr Sojka

Tento článek byl napsán s úmyslem částečně nahradit uživatelskou dokumentaci programu `accents` resp. `l2accent`, která dosud nevznikla. Čerpá ze zkušeností ze spouštění tohoto programu, z článku [2] a článku prezentovaném autorem programu na konferenci EuroTEX '91 [3].

Program `accents` používá koncept virtuálních fontů, který je pro pochopení programu podstatný. Virtuálním fontům je proto věnována první kapitola. Ve druhé kapitole se čtenář dozví podstatu programu `accents` a způsob jeho používání, aby ve třetí zjistil další vazby a možnosti při použití s dalšími programy. Článek předpokládá obeznámenost se základními pojmy z dané oblasti (font, metrika, znak fontu, `.dvi` soubor apod.).

1. Virtuální fonty

Virtuální fonty spolu s programy `vftovp` a `vptovf` vznikly (byly navrženy Knuthem) dlouho po vzniku TEX82 [1] (kolem roku 1987) spolu se vznikem nového jazyka pro popis stránky — `PostScriptu`. Pokusme se *neformálně* objasnit tento pojem.

1.1. Co *je* virtuální font

Virtuální fonty umožňují, jednoduše řečeno, nazývat *písmenem, znakem* i větší část vysázeného textu, např. znak č, což je v TEXu82 (přesněji Computer Modern fontech) obyčejně sekvence znaků `c` a `~`, vysázená např. pomocí `\v{c}`. Písmeno virtuálního fontu však může být jakákoli část `.dvi` souboru (třeba celá stránka). Zobecňují tedy pojem znaku na úrovni ovladačů výstupních zařízení — programů `dvi`*něco*.

Virtuální font neobsahuje žádné křivky, jde vlastně o popisy, jak skládat písmena (virtuálního fontu) z jiných (několika normálních) písmen (třeba i různých) fontů na základě metrických informací o znacích (podobně jako to dělá TEX).

1.2. Co virtuální font *není*

Virtuální fonty nebyly zavedeny až od nějaké verze TEXu číslo `x.yz` prostě proto, že je bylo (teoreticky) možno používat od prvopočátku (TEXu82).[1] Metrika virtuálního fontu se nikterak neliší od jiných (nevirtuálních) metrik (a s žádným dalším souborem s informacemi o fontu TEX nepracuje).

Virtuální fonty též nikterak nesouvisí s obrysy (`.pfb` resp. `.pfa` u `ADOBE` Type-1 fontů) či bitovými mapami (`.pk` soubory) jednotlivých písmen. Při vytváření virtuálních fontů se obrysy (bitmapy) jednotlivých komponent znaku nemění.

1.3. Vf-ware

Knuth napsal pro práci s virtuálními fonty programy `vftovp` a `vptovf`, Peter Breitenlohner pak `dvicopy` (vše v jazyku `web`, jak bývá zvykem).

Soubor s virtuálním fontem má obvykle extenzi `.vf` (virtual font) a je jména shodného s názvem metriky. Je to binární soubor, který se dá konvertovat do ascii souboru s extenzí `.vpl` (virtual property list) programem `vftovp`. Po editaci `.vpl` souboru běžným ascii editorem můžeme vytvořit opět virtuální font programem `vptovf`.

Pro orientaci uveďme ukázku části `.vpl` souboru:

```
(VTITLE Created by afm2tfm pbscr -v pbscr)
(COMMENT Please edit that VTITLE if you edit this file)
(FAMILY TeX-rpbscr)
(CODINGSCHEME TeX text + FontSpecific)
(DESIGNSIZE R 10.0)
(DESIGNUNITS R 1000)
(COMMENT DESIGNSIZE (1 em) IS IN POINTS)
(COMMENT OTHER DIMENSIONS ARE MULTIPLES OF DESIGNSIZE/1000)
(CHECKSUM O 6541271040)
(FONTDIMEN
   (SPACE D 339)
```

[1] Pokud by existovaly potřebné ovladače.

```
   (STRETCH D 200)
   (SHRINK D 100)
   (XHEIGHT D 400)
   (QUAD D 1000)
   (EXTRASPACE D 111)
   )
(MAPFONT D 0
   (FONTNAME rpbscr)
   (FONTCHECKSUM O 31236506737)
   )
(LIGTABLE
   (LABEL O 41)
   (LIG O 140 O 16)
   (STOP)
   (LABEL O 47)
   (LIG O 47 O 323)
   (KRN C s R -19)
   (KRN C t R -19)
   (STOP)
   (LABEL O 55)
   (LIG O 55 O 320)
   (STOP)
   (LABEL O 77)
   (LIG O 140 O 17)
   (STOP)
   (LABEL C A)
   (KRN C T R 1)
   (KRN C V R -11)
   (KRN C W R -2)
   (KRN C Y R -18)
   (KRN O 47 R -29)
   (KRN C v R -50)
   (KRN C w R -57)
   (KRN C y R -59)
   (STOP)

.......

(CHARACTER O 0 (comment NUL)
```

```
   (CHARWD R 339)
   (CHARHT R 600)
   (CHARDP R 208)
   )
(CHARACTER O 1 (comment Delta)
   (CHARWD R 611)
   (CHARHT R 600)
   (CHARDP R 208)
   (MAP
      (SETCHAR O 306)
      )
   )

.......

(CHARACTER O 377 (comment DEL)
   (CHARWD R 339)
   (CHARHT R 600)
   (CHARDP R 208)
   (MAP
      (SETCHAR O 177)
      )
   )
```

Dalším programem, který je relevantní problematice, je program `dvicopy`, který konvertuje `.dvi` soubor obsahující virtuální fonty na `.dvi` soubor bez nich. To se hodí v případě, kdy oblíbený ovladač nezná virtuální fonty (což je dnes už zřídka pozorovat), nebo např. v případě, že chceme konvertovat naše *cs*fonty do *cm*fontů (o tom viz 3.5). O používání programu `dvicopy` viz 3.3.

Zmíněné programy *by měly být* součástí *každé* distribuce TEXu (jsou v emTEXu i `UnixTeXu`).[2]

1.4. Proč používat virtuální fonty?

1. Virtuální fonty umožňují jednoduše modifikovat *umístění akcentů* ve virtuálních fontech (např. vystředění háčku nad velkým `C` pomocí algoritmu pro příkaz TEXu `\accent` působí ve standardním TEXu přímo odpudivě).

[2] V CSTEXu jsou na disketě 19 v balíku `tools.zip`.

2. Virtuální osmibitové fonty mohou být použity na *překódování* — prostou permutaci — znaků v tabulce fontu, např. pro přechod od `ADOBE Standard Encoding` kódování do kódování TEXových textových fontů.
3. Při použití sedmibitových fontů nastávají[3] problémy s dělením v jazycích majících akcenty, neboť v sedmibitovém TEXu příkaz `\accent` znamenal interně pro TEX konec slova a tudíž ústil v nedokonalé *dělení slov*. Přechodem k osmibitovým (virtuálním) fontům tyto problémy ve většině[4] jazyků odpadají.
4. Použití virtuálních fontů umožňuje *úsporu místa na disku*, neboť bitové mapy (pokud existují) se uchovávají jen jednou (na rozdíl od řešení pomocí osmibitových normálních fontů, kde je informace např. o písmenu `e` zkopírována v `é` i `ě`).

S virtuálními fonty v současné době umějí pracovat všechny běžně rozšířené ovladače (`dvips`, `dviscr`, `dvidot`, `dvilj4`, ...).

2. Program `accents`

Program vlastně nahrazuje ruční používání programů `vftovp`, editaci `.vpl` souboru a `vptovf` (sloučením kódu obou programů a přidáním kódu doprostřed ostatně vznikl).

Program `accents`[5] můžeme použít na 2 věci:

1. na konverzi postscriptové metriky v kódování `ADOBE Standard Encoding` (např. `rptmr.tfm`), vygenerované programem `afm2tfm`, ale z verze `dvips` nejvýše 5.47, do *osmibitového* (českého) fontu a na současné umístění akcentů nad písmena.
2. na konverzi tradičního *sedmibitového* 128-znakového textového fontu v kódování *TEXovských textových* fontů, např. `cmr10.tfm`) do *osmibitového virtuálního* fontu, řekněme `vcmr10.vf`, spolu s odpovídající metrikou `vcmr10.tfm`. Virtuální font obsahuje na prvních 128 pozicích znaky (pokud je vstupní font obsahuje) v `CM` rozložení textových fontů (tj. OT1 v terminologii NFSS).

Program tedy může snadno *(polo)automaticky* generovat osmibitové virtuální fonty.

Program pozná obě uvedené varianty automaticky. Nesloží však samozřejmě znaky, ke kterým nemá ve vstupech komponenty — pracuje *pouze*

[3] Dlužno přiznat, že ne nutně; první verze CSfontů byla sedmibitová a s tímto problémem se vyrovnala.

[4] Autor si není jist, jak je tomu v jazycích jako japonština apod. `:-)`

[5] O jeho modifikaci s názvem `l2accent` viz. 3.5.

na úrovni metrických informací. Umístění háčků a čárek dělá dle standardních definic maker \v a \', pokud není řečeno jinak ve speciálním souboru (adjustačním souboru .adj).

2.1. Příklad první

Příkazem

```
accents cmr10.tfm vcmr10
```

vytvoříme osmibitový font vcmr10.vf a mu odpovídající metriku vcmr10.tfm. Tyto soubory uložte do patřičných adresářů (\emtex\vf a \emtex\tfm v emTEXu), aby je našel TEX a ovladače, nebo přidejte do seznamu prohledávaných adresářů těchto programů i běžný adresář.[6]

Zkuste

```
C:>tex testfont
Name of the font to test > vcmr10 scaled 1200
*\table\bye
```

a vytiskněte testfont.dvi běžným způsobem (ovladač musí znát virtuální fonty).

2.2. Příklad druhý — adjustace akcentů

Předpokládejme, že jsme si koupili ADOBE Type-1 font s názvem *BrushScript* a chceme s ním sázet české texty.[7] Máme tedy soubory BrushScript.afm a BrushScript.pfb resp. pod MSDOSem pbscr.afm a pbscr.pfb (nebo jeho ascii podobu pbscr.pfa). V dalším budu používat zkrácená jména v konvencích MSDOSu.

Programem afm2tfm vytvoříme metriku v ADOBE Standard Encoding:

```
C:>afm2tfm pbscr rpbscr
rpbscr BrushScript
```

[6] V CSTEXu se rovněž jedná o adresář tfm, ovšem pro vyhledávání ve vf adresáři je nutno přidat položku do souborů cfg*.cnf např. ve tvaru: /vf:{$TEXDIR:vf\,}@f.

[7] Pokud jste omylem dostali macintoshovskou disketu, je zapotřebí na začátek ještě několik kroků, ale o tom snad někdy příště.

Řádek, co nám `afm2tfm` vypsal na obrazovku si zapamatujte, bude se hodit později. Tím z `pbscr.afm` vznikla metrika `rpbscr.tfm` v `ADOBE Standard Encoding` kódování.

Programem `accents` vytvoříme metriku v jiném, TEXu příjemnějším kódování (Cork[8] alias DC alias EC kódování):

```
C:>accents rpbscr vrpbscr
```

Program píše po obrazovce jakási čísílka, kterých si (prozatím) nemusíme všímat. Z `rpbscr.tfm` vznikl `vrpbscr.vf` a odpovídající metrika `vrpbscr.tfm`. Font již můžeme použít v TEXu (po uložení `vrpbscr.tfm` do patřičného adresáře).

```
C:>tex testfont
Name of the font to test > vrpbscr at 13pt
Now type a test command (\help for help):)
*\table\bye
[1]
Output written on testfont.dvi (1 page, 10828 bytes).
```

Abychom však mohli dvi soubor `testfont.dvi` vytisknout, musíme použít nějaký ovladač TEXu pro PostScript, např. `dvips`.

Aby `dvips` znal nový font, musí znát

1. metriky `rpbscr.tfm` a `vrpbscr.tfm`
2. virtuální font `vrpbscr.vf`
3. patřičný obrys písma — `pbscr.pfb`

Je třeba nastavit příslušné cesty, umístit soubory do patřičných adresářů a do souboru `psfonts.map` přidat řádek podobný tomu, co vypsal program `afm2tfm` s udáním místa souboru `pbscr.pfb`:

```
rpbscr BrushScript   <c:\emtex\ps\pbscr.pfb
```

Nyní příkazem

```
C:>dvips -o prn testfont
```

[8] Dle místa vzniku této normy na konferenci TUGu.

vytiskneme tabulku fontu na postscriptovou tiskárnu připojenou k počítači. Podíváme se na výsledek a *nebude* se nám líbit umístění akcentů. Chceme s nimi pohnout. Nastudujeme notaci `vpl`. Vyjádříme svá přání v této notaci a vytvoříme soubor `rpbscr.adj`:[9]

```
(COMMENT Toto je soubor rpbscr.adj )
(COMMENT Muze slouzit jako vzor adjustacniho souboru)
(COMMENT Uziti:)
(COMMENT accents vstupni.tfm vystupni [tento_soubor])
(DESIGNUNITS R 15)
(COMMENT (USERSCHEME)
  (COMMENT uncomment this to generate output in KOI-8 encoding))
(COMMENT (GLOBAL (RIGHT R 0.5))
  (COMMENT uncomment this if you need global accent movements))
(CHARACTER O~201 (RIGHT R 1.5))           (COMMENT - A-ogonek)
(CHARACTER O~202 (RIGHT R .5))            (COMMENT - \'C)
(CHARACTER O~206 (RIGHT R 1.25)(UP R .5)) (COMMENT - E-ogonek)
(CHARACTER O~213 (RIGHT R .5))            (COMMENT - \'N)
(CHARACTER O~221 (RIGHT R .35))           (COMMENT - \'S)
(CHARACTER O~231 (RIGHT R .35))           (COMMENT - \'Z)
(CHARACTER O~241 (RIGHT R 1.75))          (COMMENT - a-ogonek)
(CHARACTER O~242 (RIGHT R .25))           (COMMENT - \'c)
(CHARACTER O~246 (RIGHT R 1)(UP R .4))    (COMMENT - e-ogonek)
(CHARACTER O~253 (RIGHT R .35))           (COMMENT - \'n)
(CHARACTER O~261 (RIGHT R .35))           (COMMENT - \'s)
(CHARACTER O~271 (RIGHT R .35))           (COMMENT - \'z)
(CHARACTER O~323 (RIGHT R .5))            (COMMENT - \'O)
(CHARACTER O~363 (RIGHT R .25))           (COMMENT - \'o)
(CHARACTER O~211 (LEFT R 3))              (COMMENT - \'L)
(CHARACTER O~264 (LEFT R 2.5)(UP R 2))    (COMMENT - \'t)
(CHARACTER O~244 (LEFT R 1))              (COMMENT - \'d)
(CHARACTER O~251 (LEFT R 1))              (COMMENT - \'l)
(CHARACTER O~203 (RIGHT R .75)(UP R .4))  (COMMENT - C-hacek)
(CHARACTER O~243 (RIGHT R .25))           (COMMENT - c-hacek)
```

Jak tušno, čísla znaků jsou vesměs zadávána oktalově[10] (ale je možno i dekadicky či binárně), vše je v pseudolispovském zápise s docela srozumitelnou sémantikou.

Podstrčíme tento soubor programu `accents`, který nám odkryje dosud netušené možnosti:

[9] Komu se tato kovbojka líbí a chce na rozuzlení přijít sám, může v tomto místě článek odložit.

[10] Protože většinou pracujeme s tabulkami fontů vyprodukovanými pomocí `testfont.tex`

```
C:>accents
Input file [.TFM]:rpbscr
Output file [Vpbscr.VF]:vpbscr
Adjustment file (optional) [rpbscr.ADJ]:rpbscr.adj
This is ACCENTS, Version 1/DOS-TP 1
Copyright (C) 1990 Jiri Zlatuska
Distributed under terms of GNU General Public License
Input TFM is ADOBE file encoding scheme.
'041 '044 '045 '046 '047 '050 '051 '052
'054 '055 '056 '057 '060 '061 '062 '063
'064 '065 '066 '067 '070 '071 '072 '073
'077 '101 '200 '201 '300 '301 '302 '303
'304 '305 '102 '103 '202 '203 '307 '104
'204 '105 '205 '206 '310 '311 '312 '313
'106 '107 '207 '110 '111 '235 '314 '315
'316 '317 '112 '113 '114 '210 '211 '212
'115 '116 '213 '214 '321 '117 '216 '322
'323 '324 '325 '326 '120 '121 '122 '217
'220 '123 '221 '222 '223 '124 '224 '225
'125 '226 '227 '331 '332 '333 '334 '126
'127 '130 '131 '230 '335 '132 '231 '232
'233 '133 '135 '140 '141 '240 '241 '340
'341 '342 '343 '344 '345 '142 '143 '242
'243 '347 '144 '244 '145 '245 '246 '350
'351 '352 '353 '146 '147 '247 '150 '151
'354 '355 '356 '357 '152 '153 '154 '250
'251 '252 '155 '156 '253 '254 '361 '157
'256 '362 '363 '364 '365 '366 '160 '161
'162 '257 '260 '163 '261 '262 '263 '164
'264 '265 '165 '266 '267 '371 '372 '373
'374 '166 '167 '170 '171 '270 '375 '172
'271 '272 '273 '074 '134 '014 '015 '173
'042 '076 '022 '023 '136 '176 '026 '025
'137 '177 '027 '030 '175 '024 '174 '035
'037 '036 '032 '020 '034 '033 '031
I had to round some heights by 0.0139999 units.
I had to round some depths by 0.0054998 units.
```

Nyní již stačí jenom ověřit tiskem tabulky fontu `vpbscr`, jak se nám podařilo počeštit font. Jupíí!

2.3. Možné problémy

Na tomto místě bych si dovolil upozornit na možné problémy s používáním výsledků programu `accents` resp. virtuálních fontů vůbec.

Knuth optimalizuje umísťování písmen (ve vzoru ovladačů `dvitype`) při sazbě „slova“ tak, že ovladače zaokrouhlují mezipísmenný posun dle rastru aktuálního výstupního zařízení místo zaokrouhlování absolutní pozice písmena spočítané v přesných interních jednotkách v `.dvi` souboru. Tato akumulace zaokrouhlování někdy způsobí až několikabodové posunutí umístění písmena (třeba posledních písmen slova) od „absolutní pozice“. Dosud žádný problém, ale ouha!

Virtuální font může obsahovat při sestavování virtuálního znaku poměrně „velké“ posuny, což ovladač interpretuje jako „konec slova“ a uprostřed virtuálního znaku zaokrouhlí, čímž může vzniknout několikabodový posun při sazbě ď mezi d a háčkem, což je zřetelně viditelné. Projevuje se to zejména na zařízeních s nižší (300 DPI) rozlišovací schopností.

Prozatímním řešením je vypnutí drift algoritmu (např. přepínačem `-e0` u dvips) nebo používáním výstupních zařízení s vyšším rozlišením (jemnější rastr a tudíž menší zaokrouhlovací chyby a menší drift).

Příklad na demonstraci tohoto chování lze nalézt na `ftp.muni.cz` v adresáři `/pub/tex/fontware/accents/drift.zip`.

3. Využívání přátel virtuálních fontů

V tomto oddíle se zmíníme o praktickém využívání relevantních podpůrných programů — přátel virtuálních fontů.

3.1. `afm2tfm`

Tento program je distribuován společně s programem `dvips`. Rozsáhlá dokumentace k programu je v manuálu `dvips`, proto se zde nebudeme opakovat.

Pro soužití s `accents` je vhodná verze z balíku `dvips` s číslem ≤ 5.47 (tj. ne mladší). Novější verze s nástupem virtuálních fontů již do `raw` metriky nevkládá kerningy a ligatury, ty jsou ve vytvořeném virtuálním fontu. Program `accents` však byl navržen tak, že vychází z `raw` metriky a tyto informace jsou potřebné. Je tedy potřeba je z virtuálního fontu

vytvořeného `afm2tfm` získat, např. pomocí programu `vftovp`. Jinou, přímější cestou je program

3.2. `afm2pl`

Je to vlastně modifikace programu `afm2tfm`, která produkuje potřebný font přímo (na výsledný `.pl` soubor je třeba použít `pltotf` a ten pak používat jako vstup pro `accents`). Program `afm2pl` najdete na `ftp.muni.cz:/pub/tex/fontware/accents/afm2pl.zip` spolu s distribuční verzí programu `accents`.

3.3. `dvicopy`

Jak již bylo řečeno, tento program „odvirtualizuje“ `dvi` soubor, tj. nahradí odkazy na virtuální fonty jejich expanzí, tedy „obyčejnými“ fonty. Krátkým náhledem do `dvicopy.exe` zjistíme, že program hledá virtuální fonty v adresářích nastavených proměnnou `TeXvfonts` a metriky v `TeXfonts`. Nastavením pomocí MSDOS příkazu `set`

```
C:>set TeXvfonts=d:\emtex\vf
C:>set TeXfonts=d:\emtex\tfm
```

docílíme, aby program našel vše potřebné, a pak stačí zadat

```
C:>dvicopy csdvi
```

kde `csdvi` je název `dvi` souboru. Program `dvicopy` vytvoří `dvi` soubor s názvem `csdvi.cpy`, který již na virtuální fonty *ne*odkazuje.

3.4. Užití PostScriptových písem na nepostscriptových tiskárnách

Krátce upozorníme na možnost používat postscriptová písma pro ty, kteří nevlastní postscriptovou tiskárnu. V podstatě jsou dvě cesty:

1. PostScriptový soubor vytvořený pomocí `dvips` tisknout pomocí programu `ghostscript` (nebo prohlížet pomocí `ghostview`).
2. generování bitových map písem z postscriptových fontů public domain programem `ps2pk`.[11] V obou případech však pro dosažení kvalitního tisku potřebujete mít originální `.pfb` resp. `.pfa` soubory. Ty jsou vesměs licencované firmou `ADOBE`, ale existuje několik rodin, které jsou volně dostupné (např. na `ftp.muni.cz` v adresáři `/pub/tex/CTAN/fonts/postscript`):

[11] Na `ftp.muni.cz` v adresáři `/pub/tex/CTAN/fonts/utilities` najdete kromě `ps2pk` ještě několik podadresářů s programy pro práci s fonty.

```
./utopia/putri.pfa
./utopia/putb.pfa
./utopia/putr.pfa
./utopia/putbi.pfa
./courier/couri.pfa
./courier/courbi.pfa
./courier/courb.pfa
./courier/cour.pfa
./urw/uaqrrc.pfa
./urw/unmr.pfa
./urw/unmrs.pfa
./urw/ugqb.pfa
./msym/msym10.pfa
./ascii/cour.pfa
./charter/bchr.pfb
./charter/bchbi.pfb
./charter/bchb.pfb
./charter/bchri.pfb
./msym/msym10.pfb
./postscript/cm/logobf10.pfb
./postscript/cm/msbm10.pfb
./postscript/cm/msam10.pfb
./postscript/cm/linew10.pfb
./postscript/cm/logo9.pfb
./postscript/cm/logo8.pfb
./postscript/cm/logo10.pfb
./postscript/cm/lcirclew.pfb
./postscript/cm/line10.pfb
./postscript/cm/lcircle1.pfb
./postscript/cm/lasyb10.pfb
./postscript/cm/lasy9.pfb
./postscript/cm/lasy8.pfb
./postscript/cm/lasy7.pfb
./postscript/cm/lasy6.pfb
./postscript/cm/lasy5.pfb
./postscript/cm/lasy10.pfb
./postscript/cm/logosl10.pfb
./postscript/cm/eusm5.pfb
./postscript/cm/eusm10.pfb
```

```
./postscript/cm/eusb7.pfb
./postscript/cm/eusb5.pfb
./postscript/cm/eusb10.pfb
./postscript/cm/eurm7.pfb
./postscript/cm/eurm5.pfb
./postscript/cm/eusm7.pfb
./postscript/cm/eurm10.pfb
./postscript/cm/eurb7.pfb
./postscript/cm/eurb10.pfb
./postscript/cm/eufm7.pfb
./postscript/cm/eufm5.pfb
./postscript/cm/eufm10.pfb
./postscript/cm/eufb7.pfb
./postscript/cm/eufb5.pfb
./postscript/cm/eurb5.pfb
./postscript/cm/euex10.pfb
./postscript/cm/cmvtt10.pfb
./postscript/cm/eufb10.pfb
./postscript/cm/cmtt9.pfb
./postscript/cm/cmtt8.pfb
./postscript/cm/cmu10.pfb
./postscript/cm/cmtt10.pfb
./postscript/cm/cmti9.pfb
./postscript/cm/cmbx7.pfb
.... atd. (CM fonty)
```

3.5. Od $\mathcal{CS}$fontů k cmfontům — l2accent

V balíku $\mathcal{CS}$TEX najdete místo programu `accents` program `l2accent`, což je upravený program `accents` tak, aby pracoval v kódování ISO--Latin-2, které je použito v $\mathcal{CS}$fontech. Autor vytvořil tuto variantu svého programu právě pro uživatele $\mathcal{CS}$TEXu. Program se liší od programu `accents` pouze kódováním výstupních fontů. Virtuální české Computer Modern fonty, které takto vzniknou, mají stejné šířky jako $\mathcal{CS}$fonty z $\mathcal{CS}$TEXu.

Protože ovladač znalý virtuálních fontů se pídí v první řadě po nich, pro nahrazení $\mathcal{CS}$fontů virtuálními českými Computer Modern fonty stačí o nich dát vědět nastavením příslušných cest. Pak spuštěním programu `dvicopy` na `dvi` soubor s (dočasně virtuálními) $\mathcal{CS}$fonty dostaneme `dvi`

soubor *pouze* s odkazy na sedmibitové Computer Modern fonty a tento `dvi` soubor se již nemusíme obávat nikam poslat.*

Není to nic neskutečného, ty virtuální fonty, ne?

Odkazy

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

[2] Timothy Murphy, `<tim@maths.tcd.ie>` School of Mathematics, Dublin, Ireland: Making virtual fonts with Jiří Zlatuška's 'Accents' program, článek na `tex-news` z 4. 6. 1993.

[3] Jiří Zlatuška, `<zlatuska@muni.cz>`: Automatic generation of virtual fonts with accented letters for TEX, Cahiers GUTenberg No. 10, září 1991.

Příloha — tabulka extenzí souborů

`.vf`	virtuální font	virtual font
`.tfm`	TEXová metrika (virtuálního) fontu	TEX font metric
`.afm`	metrika `ADOBE` Type-1 fontu	`ADOBE` font metric
`.pl`	ascii výpis metriky	property list
`.vpl`	ascii výpis metriky virt. fontu	virtual property list
`.adj`	soubor posunů znaků	adjustment file
`.map`	zobrazení názvů TEXových metrik na názvy písem `ADOBE`	
`.dvi`	Popis stránky	device independent
`.cpy`	`dvi` soubor — výstup `dvicopy`	
`.pfb`	`ADOBE` Type-1 font binárně	
`.pfa`	`ADOBE` Type-1 font v ascii zápise	

Petr Sojka
Ústav výpočetní techniky,
Masarykova Universita, Brno
`<sojka@muni.cz>`

* Některé změny ve zdrojových METAFONTových souborech způsobují nepatrně odlišné chování originálních Knuthových Computer Modern a našich $\mathcal{CS}$fontů (např. vyrovnání dvojice Av, teček ....). (*Pozn. red.*)

Postscriptová písma v TEXu

Tomáš Mráz

Tento článek by měl pojednávat o mých zkušenostech s použitím postscriptových písem v TEXu. Zmíním se zejména o postupech jejich převodu do podoby použitelné s TEXem a problémech, se kterými jsem se při něm setkal. Vůbec se nebudu zabývat problémy použití postscriptových písem v matematické sazbě, neboť jsem zatím neměl příležitost setkat se s postscriptovými fonty vhodnými pro tento druh sazby, zejména co se týče matematických symbolů.

Při konverzi postscriptových písem existuje několik možných cest, po kterých je možno postupovat. Zmíním se zde o některých z nich.

Konverze metrických informací

Cesta první — česká písma

Podaří-li se nám sehnat někde kvalitní česká písma, která obsahují české znaky, pak máme za předpokladu, že jsou skutečně kvalitní, již zčásti vyhráno. Problémem však zůstává, kde a za jakou cenu lze tato písma sehnat. Z mých zkušeností s písmy vyrobenými či lépe řečeno počeštěnými českými firmami vyplývá, že jsou většinou horší kvality a že způsob jejich počeštění je značně nekompatibilní se standardními postscriptovými fonty. Například názvy jednotlivých českých písmen neodpovídají tomu, co se pod nimi ve skutečnosti skrývá. Třeba pod názvem *ydieresis* se schovává písmeno *ř*. Další vadou těchto fontů bývá vypuštění ligatur *fi* a *fl*, malý počet kerningových párů a nedostatečné nebo přímo chybné hinty. Naopak je jasné, že tyto fonty mají obrovskou výhodu v ceně oproti písmům firem Adobe a URW, což jsou jediná zahraniční písma, o kterých vím, že se u nás prodávají a obsahují české znaky.

Přesto pokud se nám podaří tato písma sehnat, přichází další problém. Zejména u méně kvalitních fontů nenajdeme soubory s příponou `.afm`, což jsou soubory Adobe Font Metrics, obsahující metrické informace k fontům. Vzhledem k tomu, že tato písma jsou určena k použití pod MS Windows, jsou k nim dodávány soubory `.pfm`. Získat soubory

`.afm` k počeštěným fontům je možné, pokud je mi známo, snadno pouze pomocí komerčních programů, jako je například Fontographer.[1]

Existuje ještě jedna cesta, kterou však získáme pouze informace o velikosti jednotlivých písmen. Pomocí chytře napsaného postscriptového makra můžeme přinutit tiskárnu, nebo lépe nějaký interpret PostScriptu — typicky Ghostscript, aby si jednotlivá písmena vykreslil a pak nám sdělil, jak jsou velká. Bohužel tak nemůžeme dostat informace o kerningových párech. Tyto bychom naopak mohli vyextrahovat ze souborů `.pfm`, ale o žádném programu, který by to uměl, nevím. Na druhou stranu, pokud fonty zakoupíme přímo u firmy, která je vyrobila, můžeme požádat o vygenerování souborů `.afm` přímo ji.

Další úkol, který před námi stojí, je zjistit kódování fontu s názvy jednotlivých písmen, což je opět nejjednodušší pomocí programu Fontographer. Pokud známe anglické názvy akcentů, které používá firma Adobe, a odpovídají-li názvy písmen jejich tvarům, pak se lze jednoduše podívat do souboru `.afm` a názvy jednotlivých písmen si opsat. Protože kódování zcela jistě nebude odpovídat tomu, které bychom potřebovali pro práci s TEXem, musíme provést překódování fontu, ke kterému máme dvě příležitosti. Jednak můžeme použít PostScript a jeho kódový vektor, a nebo zajistit změnu kódů jednotlivých písmen pomocí virtuálních fontů. Jednodušší se mi zdá být cesta použití virtuálních fontů, ačkoliv předpokládá, že máme *dvi* ovladače, které si s virtuálními fonty rozumějí. Naštěstí ovladače Eberharda Mattese mezi ně patří, jakož i program DVIPS od Tomase Rokického.

Teď již přistoupíme k vlastní konverzi metrik. Vytvoříme si soubor `czencode.enc`, do kterého zapíšeme něco takového:

```
/CzEncoding [
 /Gamma /Delta /Theta ...
 %další řecká písmena pokud nějaká v našem fontu jsou
 /Phi /Psi /Omega /ff /fi /fl /ffi /ffl /dotlessi /dotlessj /grave
 /acute /caron /breve /macron /ring /cedilla /germandbls /ae /oe
 /oslash /AE /OE /Oslash /.notdef /exclam /quotedblright
 /numbersign /dollar /percent /ampersand /quoteright /parenleft
 /parenright /asterisk /plus /comma /hyphen /period /slash /zero
```

[1] Nebo FontMonger (oba programy existují pro MS Windows). Nedávno se objevil ve volném prodeji CD-ROM s českými verzemi písem z programu Corel 3.0, na němž najdeme i odpovídající `afm` soubory. Na rozdíl od jiného balíku těchto fontů však zcela chybějí některé standardní znaky jako ligatury apod. Pokud je mi známo, jsou fonty pro tento program počešťovány automaticky (asi podobně jako když použijeme `accents`), tuto verzi jsem však ještě nestihl prozkoumat. (*Pozn. red.*)

```
 /one /two /three /four /five /six /seven /eight /nine /colon
 /semicolon /guilsinglleft /equal /guilsinglright /question
 /at      /A       /B       /C       /D       /E       /F       /G
 /H       /I       /J       /K       /L       /M       /N       /O
 /P       /Q       /R       /S       /T       /U       /V       /W
 /X       /Y       /Z
 /bracketleft /quotedblleft /bracketright /circumflex /dotaccent
 /quoteleft /a     /b       /c       /d       /e       /f       /g
 /h       /i       /j       /k       /l       /m       /n       /o
 /p       /q       /r       /s       /t       /u       /v       /w
 /x       /y       /z
 /endash  /emdash  /hungarumlaut /tilde /dieresis
 /.notdef /.notdef /.notdef /.notdef /.notdef /.notdef /.notdef
 /.notdef /.notdef /.notdef /.notdef /.notdef /.notdef /perthousand
 /.notdef /.notdef /.notdef /.notdef /.notdef /.notdef /.notdef
 /.notdef /.notdef /.notdef /Agrave  /.notdef /.notdef /.notdef
 /hyphen /ogonek /guillemotleft /quillemotright /.notdef /.notdef
 /.notdef /.notdef /.notdef /Lcaron /.notdef /.notdef
 % nejsem si jist jestli se Ľ jmenuje Lcaron, podobně ľ, ť, ď
 /.notdef /Scaron /.notdef /Tcaron /.notdef /.notdef /Zcaron /.notdef
 /.notdef /.notdef /.notdef /.notdef /.notdef /lcaron /.notdef
 /.notdef /agrave /scaron /.notdef /tcaron /.notdef /.notdef /zcaron
 /.notdef /Racute /Aacute /.notdef /.notdef /Adieresis /Lacute
 /.notdef /.notdef /Ccaron /Eacute /.notdef /.notdef /Ecaron /Iacute
 /.notdef /Dcaron /.notdef /.notdef /Ncaron /Oacute /Ocircumflex
 /.notdef /Odieresis /.notdef /Rcaron /Uring /Uacute /.notdef
 /Udieresis /Yacute /.notdef /.notdef /racute /aacute /.notdef
 /.notdef /adieresis /lacute /.notdef /.notdef /ccaron /eacute
 /.notdef /.notdef /ecaron /iacute /.notdef /dcaron /.notdef /.notdef
 /ncaron /oacute /ocircumflex /.notdef /odieresis /.notdef /rcaron
 /uring /uacute /.notdef /udieresis /yacute /quotedblbase
 /quotedblleft ]
% LIGKERN hyphen hyphen =: endash ; endash hyphen =: emdash ;
% LIGKERN quoteleft quoteleft =: quotedblleft ;
% LIGKERN quoteright quoteright =: quotedblright ;
% LIGKERN guilsinglleft guilsinglleft =: guillemotleft ;
% LIGKERN guilsinglright guilsinglright =: guillemotright ;
% frenchdblquotes
% LIGKERN comma comma =: quotedblbase ;
% czquoteleft
% LIGKERN space {} * ; * {} space ; zero {} * ; * {} zero ;
% LIGKERN one {} * ; * {} one ; two {} * ; * {} two ;
% LIGKERN three {} * ; * {} three ; four {} * ; * {} four ;
% LIGKERN five {} * ; * {} five ; six {} * ; * {} six ;
% LIGKERN seven {} * ; * {} seven ; eight {} * ; * {} eight ;
% LIGKERN nine {} * ; * {} nine ;
% LIGKERN question {} quoteleft ; exclam {} quoteleft ;
% obrácený otazník a vykřičník nejsou pro běžné použití třeba,
% lomené závorky se hodí víc
```

S největší pravděpodobností bude třeba v závislosti na konkrétním fontu něco změnit.

Pak již můžeme pomocí programu AFM2TFM zkonvertovat soubor `.afm` na soubory `.vf` a `.tfm`. Použijeme příkaz

```
afm2tfm <type1font> -o czencode.enc -v <virtfont> <rawfont>.
```

Přípony není třeba uvádět. Program vytvoří dva soubory: 1. *raw tfm* — soubor, který obsahuje pouze metriky fontu v původním kódování bez ligatur a kerningů. 2. *virtual property list* `.vpl` — obsahuje metriky fontu v našem kódování včetně ligatur a kerningů a definici virtuálního fontu pomocí fontu *raw*. Pomocí programu VPTOVF překonvertujeme soubor `.vpl` do souborů čitelných TEXem — vzniknou soubory `.vf` pro příslušný ovladač a `.tfm` pro TEX. Nyní je již stačí umístit do správných adresářů a můžeme náš nový font používat při sazbě. Používáme vždy virtuální soubor `.tfm`.*

Cesta druhá — písma v kódování Adobe Standard Encoding

Tyto fonty můžeme sehnat pomocí internetovské služby *ftp*, ale většinou se jedná o nepříliš kvalitní písma. Pokud bychom chtěli jejich kvalitnější verzi zakoupit, většinou se nám vyplatí koupit českou verzi. Jediný důvod by mohl být ten, že chceme právě a jedině font, který v české verzi ani od firmy Adobe, ani od URW, případně od jiných neexistuje.

Pak musíme použít program ACCENTS (L2ACCENT) pro vytvoření českého virtuálního fontu z původního písma. Nejprve ale vytvoříme z původního `.afm` souboru soubor `.tfm`. Tento soubor však nesmí být nijak překódovaný a neměly by v něm chybět kerningy a ligatury, pokud je chceme mít ve výsledném fontu. Nejlepší je proto použít program `aftopl` od pana Clarka pro konverzi souboru `.afm` na soubor `.pl` a program `pltotf` na konverzi souboru `.pl` na `.tfm`. Program `aftopl` jsem našel ve zdrojovém tvaru na `ftp.muni.cz`. Na takto vytvořenou metriku nám již nic nebrání vypustit program ACCENTS — kódování Cork alias DC, nebo L2ACCENT — kódování Latin 2 upravené pro použití se standardní distribucí CSTEXu. Použijeme příkaz `accents <rawtfm.tfm> <virtfont>`

* To ovšem není nutné, zvolíme-li při zadání kódovacího souboru `-T` místo `-o`. Pak budou oba vzniklé `.tfm` soubory kódovány shodně a na identickou permutaci znaků soubor `.vf` přeci nepotřebujeme. Budeme pak samozřejmě zas používat virtuální `.tfm` soubor, jen vymažeme *raw* `.tfm` a `.vf`. Aby to navíc nebylo tak jednoduché, verze `afm2tfm` 7.8 — nejnovější, kterou mám k dispozici — namísto `-o` má `-p|t|T`. (*Pozn. red.*)

`[virtfont.adj]`, kterým vytvoříme metriky použitelné pro naši další práci s TEXem. Pokud by se nám postavení akcentů nad některými písmeny nelíbilo, můžeme jejich polohu upravit pomocí souboru `.adj` se syntaxí stejnou jako u souborů `.pl` a `.vpl`, kde u každého znaku, u kterého chceme upravit postavení akcentů, uvedeme direktivu `RIGHT`, `LEFT` resp. `UP` a `DOWN`. Velikost parametrů doporučuji zjistit experimentálně. Například:

```
(COMMENT Toto je komentář )
(DESIGNUNITS R 10)
(COMMENT Počet jednotek, ve kterých zadáváme parametry,)
(COMMENT na velikost písma)
(COMMENT Teď následují jednotlivé znaky, které chceme měnit)
(CHARACTER O 206 (RIGHT R 1.25)(UP R .5))
(COMMENT Toto je znak s kódem oktalově 206)
```

Pokud metriku vytvořenou pomocí tohoto programu začneme používat, zjistíme, že neobsahuje některé znaky, které se v českých *cmr* fontech vyskytují, patří mezi ně například francouzské a české uvozovky. Tento nedostatek se mi bohužel podařilo odstranit pouze přímou editací vzniklého souboru `.vf` a `.tfm`, samozřejmě po jejich konverzi do čitelného souboru `.vpl`. Zřejmě by bylo vhodné upravit program ACCENTS, aby příslušnou úpravu za nás udělal sám, pokud ovšem neexistuje jiná jednodušší cesta, jak to udělat.

Konverze vlastního fontu — outline křivek

Cesta první — nekonvertujeme

Touto cestou se dáme, máme-li postscriptovou tiskárnu nebo emulátor PostScriptu.

Chceme-li pro výstup na postscriptovou tiskárnu použít program DVIPS od Tomase Rokického musíme uvést do seznamu postscriptových fontů (v souboru `psfonts.map`) něco jako:

```
rptmr Times-Roman <c:\psfonts\tir_____.pfb
```

`rptmr` je název našeho souboru *raw* `.tfm` bez přípony, `Times-Roman` je skutečný název našeho nového fontu — najdeme jej na začátku souboru `.afm` v položce `FontName`, `<c:\psfonts\tir_____.pfb` je cesta k našemu souboru s písmem.

Pak by již mělo být možné soubor pomocí DVIPS vytisknout — DVIPS připojí náš soubor s písmem na začátek výstupního souboru. Pokud neu-

vedeme cestu k fontu, bude DVIPS považovat font za rezidentní v tiskárně a do výstupního souboru jej nepřidá.

Cesta druhá — konverze programem PS2PK

Tato cesta je snad ještě jednodušší. Seženeme si program PS2PK nejlépe v co nejnovější verzi zkompilované pomocí DJGCC. Ve verzi 1.4 se tento program vyskytuje na `ftp.muni.cz`.

Příkazem

```
ps2pk -X<rozlišení v dpi> -P<velikost písma v pt> <font.pfb>
```

vygenerujeme soubory `.pk`. Maximum velikosti písma je v této verzi 60 pt pro rozlišení 1 270 dpi. Výsledný soubor stačí vhodně přejmenovat,* přemístit do patřičného adresáře a vyzkoušet si, že skutečně funguje např. i s ovladačem `dviscr`.

Nejdůležitější oblasti, které jsem čtenáři zamlčel

Jednak jsem se prakticky nezmínil o možnosti měnit kód písma pomocí kódového vektoru PostScriptu, neřekl jsem nic o postupu, jak využít písmena, která se sice v písmu nacházejí, ale nevyskytují se v původním kódovém vektoru, ani jsem se nezabýval podrobnostmi a možnými chybami při práci s uvedenými programy. Možná, že se o to pokusím někdy příště.

Doporučená literatura, z níž jsem čerpal

[1] Dokument `aboutacc.tex` od Petra Sojky
[2] Dokumentace k programu DVIPS a AFM2TFM — `dvips.tex` od Tomase Rokického

Oba dva dokumenty doporučuji přečíst každému vážnějšímu zájemci.

Poděkování

Donaldu E. Knuthovi za jeho skvělý program TEX.
Eberhardu Mattesovi za jeho implementaci TEXu pro počítače PC.
Tomasi Rokickimu za jeho programy DVIPS a AFM2TFM.
Pietu Tutelaersovi za jeho program PS2PK.

* Žádoucí jméno (i s cestou) lze připojit na konec příkazového řádku. (*Pozn. red.*)

Všem členům CSTUGu, kteří se podíleli na tvorbě české distribuce emTEXu — CSTEX.
Jiřímu Zlatuškovi za jeho program ACCENTS.

Tomáš Mráz
xmraz@cslab.felk.cvut.cz

Záludnosti PostScriptu

Zdeněk Wagner

Každý, kdo se vážně zabývá typografií, dostane se dříve nebo později k PostScriptu. PostScript je jazyk pro popis stránek, který poskytuje další možnosti, jež lze TEXem a METAFONTem realizovat jen s obtížemi nebo vůbec. To samozřejmě vyžaduje určité znalosti. Lze se však spokojit s přístupem uživatele, který programem DVIPS vytvoří PostScriptový soubor a ten vytiskne na PostScriptové tiskárně nebo osvitové jednotce. Přesto je vhodné o PostScriptu něco vědět, aby se člověk vyhnul záludnostem, které v sobě skrývá jak PostScript tak další TEXovské pomocné programy.

Murphy by PostScript definoval jako nástroj, s jehož pomocí lze ve velmi krátkém okamžiku znehodnotit značné množství drahého materiálu. Experimentálně bylo totiž ověřeno, že rychlost PostScriptového zařízení je nepřímo úměrná smysluplnosti tištěného textu.

Smyslem tohoto článku je upozornit na hlavní záludnosti. Vše, co bude dále uvedeno, bylo objeveno většinou metodou pokusů a omylů a bohužel i finančně nákladných chyb. Jiná, zřejmě i lepší řešení jsou jistě možná...

Moderní tiskárny mají PostScript level 2, zatímco starší zařízení znají pouze level 1. Nebudeme se zde příliš zabývat rozdíly. Téměř vše, co je v tomto článku uvedeno, platí pro level 1. Odlišnosti budou zdůrazněny pouze tam, kde mohou působit problémy.

PostScript je podobně jako TEX interpretační jazyk. Společným znakem je možnost definice uživatelských maker. Oba jazyky však mají řadu odlišností. Hlavní rozdíl spočívá v tom, že PostScript neumí lámat řádky a odstavce. V PostScriptu je nutno popsat již zlomenou stránku.

TEX čte vstupní proud (input stream[1]), expanduje makra a výsledek expanze vrací zpět do vstupního proudu. Pokud nějaké makro vyžaduje parametry, vezme je TEX ze vstupního proudu z textu, který následuje za příslušným makrem. V PostScriptu probíhá expanze trochu odlišně. Parametry si expandované makro bere ze zásobníku a případný výsledek opět vkládá do zásobníku. Není to tedy expanze ve stejném smyslu, jak ji známe z TEXu. Pokud chceme makrem `moveto` nastavit aktuální pozici na souřadnice x, y, musíme je do zásobníku vložit předem. Celý příkaz pak bude vypadat:

x y `moveto`

PostScript má čtyři zásobníky, z nichž nás budou prozatím zajímat dva. Tím prvním, který jsme již použili, je zásobník operandů (operand stack). Jak název napovídá, předávají se v něm operandy maker (v PostScriptu se makrům obvykle říká procedury) a výsledky jejich expanze. Dále bude užitečný zásobník slovníků (dictionary stack), jehož význam vysvitne později.

Chceme-li v TEXu definovat nové makro, použijeme `\def`. Přitom nás nezajímá, kam si TEX tuto definici uloží. Chceme-li pouze dočasně předefinovat již existující makro, je vhodné otevřít skupinu (group) a definici provést lokálně. Při interpretaci hledá TEX definici makra počínaje pravě otevřenou skupinou a pokračuje do vnějších skupin, dokud definici nenajde. Po uzavření skupiny se zapomenou lokální definice a obnoví se definice původní. PostScript přistupuje k definicím odlišně. Všechny definice se ukládají do slovníků. Slovníky, se kterými se právě pracuje, jsou uloženy (přesněji řečeno jejich adresy) v zásobníku slovníků. PostScriptový interpret prohledává slovníky v zásobníku odshora dolů. V okamžiku startu interpretu obsahuje zásobník dva slovníky: `systemdict`, v němž jsou všechny systémové příkazy, a `userdict` určený pro uživatelské definice. Tyto dva slovníky nelze ze zásobníku odstranit. Chceme-li otevřít nový slovník, řekněme `MyDict`, vložíme jej napřed do zásobníku operandů a poté použijeme příkaz `begin`. Nové definice se potom budou ukládat do slovníku `MyDict`. Příkazem `end` se slovník ze zásobníku slovníků odstraní. Z toho vidíme dvě odlišnosti:

[1] Česká terminologie mi občas činí potíže. Proto budu v závorkách uvádět anglické termíny.

1. Slovník může být v zásobníku slovníků, a tedy v prohledávacím řetězci několikrát.
2. Po odstranění slovníku ze zásobníku slovníků nejsou definice ztraceny, jsou pouze neaktivní.

V TEXu jsme se nestarali o to, zda máme v otevřené skupině dostatek paměti pro definice maker. To si v PostScriptu nemůžeme dovolit. Když vytváříme slovník příkazem `dict`, musíme jako parametr zadat jeho velikost. Zde je první důležitý rozdíl mezi level 1 a level 2! Je-li slovník přeplněn, level 2 si jej zvětší, zatímco level 1 ohlásí chybu `-dictfull-` a ukončí zpracování. Softwarový interpret PostScriptu, program Ghostscript, má sice implementován PostScript level 1, ale vzhledem k rozšiřování přeplněného slovníku se chová jako level 2.

Tento rozdíl mi způsobil trochu trápení. Soubor, který jsem připravil, se vytiskl bez problémů na tiskárně Hewlett-Packard 4M a dokonce byl dobře zobrazen Ghostscriptem. Na osvitové jednotce se však nevytisklo vůbec nic. Příčinou, jak již možná tušíte, byl přeplněný slovník. Nebylo to ovšem tak jednoduché.

Slovník může obsahovat leccos, například jiné slovníky. Definice fontu je také velmi složitý slovník, v němž kromě řady jiných objektů jsou další slovníky. Jedním z nich je `CharStrings`[2] obsahující vektorové popisy jednotlivých znaků. Ve fontu, který jsem ve zmíněném dokumentu použil, chybělo původně „i bez tečky“ (ı). Přestože nejsem expert na PostScript, nebylo složité převést definici fontu do „lidsky čitelné podoby“, zkopírovat definici písmene „i“, vyhodit příkazy pro nakreslení tečky a výsledek zkompilovat zpět do kompaktního tvaru. Zapomněl jsem však na jednu věc. Do slovníku `CharStrings` jsem přidal další písmeno, ale nezvětšil jsem velikost slovníku. Proto mi osvitová jednotka nic nevytiskla.

Chcete-li vytvořit soubor, který by se měl vytisknout na libovolném zařízení, je vhodné používat pouze příkazy level 1. Program DVIPS to dělá vždy (tedy skoro, viz dále), v jiných programech lze interpretaci zvolit z menu. Je však nutné uvědomit si, že Ghostscript nemůže být měřítkem. V konkrétním případě byl Ghostscript schopen správně zobrazit soubor, který osvitová jednotka nevytiskla. Na druhé straně mi Ghostscript mnohokrát nahlásil nesmyslné chyby v souborech, které osvitovou jednotkou prošly zcela bez problémů.

Tím ovšem PostScriptové patálie nekončí. Nyní si ukážeme příklady, které se mohou často hodit. Budeme již přitom programovat v Post-

[2] Podrobnější popis přesahuje rámec tohoto článku.

Scriptu. Podobně jako v TEXu nejsou mezery mezi objekty významné (tj. stačí jedna). Můžeme psát dlouhé řádky (pokud se nemýlím, maximální povolená délka je 512 znaků). Tak dlouhé řádky jsou však nepřehledné. V příkladech budeme řádky číslovat, abychom se na ně mohli odvolávat, do skutečných PostScriptových programů se však čísla řádků nepíší.

Typickým příkazem, který budeme potřebovat, je zrcadlení stránky. Než přistoupíme k jeho definici, musíme si něco říci o tom, jak vypadá popis stránky generovaný programem DVIPS. Prvním příkazem stránky je `bop`, což je makro definované v souboru `texc.pro`. Za ním následuje vlastní popis stránky a na jejím konci je makro `eop`. Makro `bop` se podívá, zda je ve slovníku `userdict` definován `bop-hook`. V kladném případě je `bop-hook` proveden. Zde je tedy vhodné místo, kam naprogramovat zrcadlení. Lze to provést následujícím programem, který nejprve napíšeme a pak jej vysvětlíme.

```
1 userdict begin
2 /bop-hook
3 {
4   -1 1 scale
5   -595 0 translate
6 }
7 def
8 end
```

Řekli jsme si, že `bop-hook` musí být ve slovníku `userdict`. Nemáme jistotu, který slovník bude na vrcholu zásobníku slovníků v okamžiku zpracovávání naší definice. Proto na řádku 1 otevřeme `userdict`. Řádek 2 obsahuje tzv. „name-literal“, což neumím říci česky. Napíšeme-li `bop-hook`, bude PostScriptový interpret hledat definici a odpovídající příkazy provede. Pokud však zapíšeme jméno s úvodním lomítkem, vloží se do zásobníku jméno objektu, nikoliv objekt. Zápis `/bop-hook` ve spojení s dalšími operacemi říká, že definujeme nový příkaz `bop-hook`.

Definici lze vložit do složených závorek (viz řádky 3 a 6). Pak se do definice uloží vše přesně v tom tvaru, jak je to napsáno, tj. bez expanze. Je to tudíž obdoba TEXového `\def`. Pokud bychom závorky vynechali, vložil by se do definice až výsledek expanze podobně jako v TEXovém `\edef`.

Abychom pochopili další řádky, musíme si opět vysvětlit některé rysy PostScriptu. Bod s nulovými souřadnicemi je totiž v levém dolním rohu

papíru. Zatímco x-ová souřadnice roste směrem doprava, y-ová souřadnice roste ve směru zdola nahoru. V TEXu roste y-ová souřadnice při pohybu shora dolů. Vzdálenost se měří v PostScriptových jednotkách (PostScript unit), jejichž velikost se shoduje s TEXovým „bp“.

Řádek 4 představuje první transformaci souřadnic. Uživatelské souřadnice zde převádíme na souřadnice tiskárny tak, že x vynásobíme hodnotou -1 a y hodnotou 1. Stránka se tím zrcadlí podle levé hrany papíru. Tím ovšem text dostaneme mimo papír a tiskli bychom prázdné stránky, což by byla levnější varianta chyby (ne však na osvitové jednotce, protože i prázdný film se platí). Musíme tedy posunout počátek souřadnic do pravého dolního rohu papíru. Máme-li formát A4, je šířka stránky 210 mm, což je přibližně 595 bp. Protože jsme ale otočili směr osy x, je na řádku 5 záporná hodnota.

Řádek 7 ukončí definici a na řádku 8 odstraníme slovník `userdict` ze zásobníku slovníků, abychom vše vrátili do původního stavu.

Tuto definici lze urychlit. Operátor `def` totiž říká, že PostScriptové příkazy použité v definici se uloží tak, jak jsou, a interpretují se až v okamžiku provádění makra. Pokud bychom na řádku 7 použili `bind def`, vložil by se do definice aktuální význam použitých příkazů `scale` a `translate`. Při provádění příkazu `bop-hook` by se již nehledaly odpovídající definice ve slovnících. To má ještě další výhodu. Příkazy `scale` a `translate` by mohly být později předefinovány, aniž by tím byla narušena funkce makra `bop-hook`!

Zrcadlení stránky A4 jsme tedy vymysleli. Zbývá ještě nalézt vhodné místo, kam tuto definici napsat. V návodu k programu DVIPS se dočteme o příkazu `\special`. Ten umožňuje zapsání PostScriptových příkazů do DVI souborů, které pak DVIPS zkopíruje do PostScriptového souboru. Tímto způsobem lze zapsat i příkaz level 2, což, jak bylo uvedeno dříve, může vést ke katastrofě.

Program pro zrcadlení stránek by měl být na začátku PostScriptového souboru. Proto musí být prvním znakem příkazu `\special` vykřičník. Zcela automaticky napíšeme:

```
 9 \special{!userdict begin /bop-hook {-1 1 scale
10   -595 0 translate} bind def end}
```

Ve většině případů to bude fungovat. Bohužel ne vždycky...

Představte si následující situaci. Soubor nejprve ladíme na vlastní PostScriptové tiskárně nebo pomocí Ghostscriptu. Proto chceme tisknout vše bez transformací. Po odladění přidáme příkaz z řádku 9,

samozřejmě na začátek souboru. Při ladění jsme tiskli celý dokument. Protože první stránka je vakát (nebo obrázek, který bude vytvořen jinou technikou) a film je drahý, použijeme např. program DVI2DVI, jímž z DVI souboru všechny vakáty a stránky pro obrázky vyřadíme a teprve potom vyrobíme PostScriptový soubor programem DVIPS. Výsledek však nebude zrcadlově obrácen. Příkaz `\special` byl totiž na první stránce, kterou jsme vyhodili.

Příkaz `\special` z řádku 9 má ještě jednu nevýhodu. Máme-li hotový dokument a chceme jej pouze zrcadlově obrátit, musíme při tomto přístupu znovu zpracovat dokument TEXem, což vyžaduje určitý čas. Uvedeme tedy dvě řešení, která odstraní oba problémy.

První řešení je použitelné pouze v případě, že tiskneme na vlastní PostScriptové tiskárně, která není připojena v síti. Pak totiž můžeme poslat do tiskárny kratičký soubor obsahující příkazy 1–8 a ihned potom PostScriptový dokument. Je nutno zdůraznit, že příslušný dokument musíme poslat do tiskárny *ihned*. PostScriptová zařízení jsou totiž vybavena dvojsečnou zbraní, kterou je „job timeout“. Tiskárna pracuje, dokud do ní přicházejí data. Je-li tok dat na určitou dobu přerušen, interpret předpokládá, že došlo k chybě. Aby nebyly poškozeny následující soubory, tiskárna se resetuje. Při používání tiskárny na síti je to vlastnost užitečná. Pokud ale používáte trik popsaný v tomto odstavci, musíte být dostatečně rychlí. Nejlepší je, když oba soubory napíšete ve správném pořadí na jeden příkazový řádek.

Zmíněná vlastnost skýtá další záludnost. Může totiž ukrýt chybu, takže na ni přijdete až později. Zpočátku jsem pro zrcadlový tisk používal příkaz `\special`, který jsem vyhazoval programem DVI2DVI, aniž bych si to uvědomil. Tiskl jsem vždy více souborů současně, přičemž, shodou okolností, v prvním souboru byl příkaz `\special` přítomen. Všechny soubory tedy tiskárna považovala za jeden „job“ a zrcadlení fungovalo. Až jednou jsem v tisku udělal přestávku...

Nyní je ale čas pro druhé řešení. První řešení totiž nemůžeme použít, pokud chceme tisknout na síťové tiskárně, nebo v případě, že soubor chceme odnést na osvitovou jednotku. Je sice možné libovolným textovým editorem připsat příkazy pro zrcadlení přímo do PostScriptového souboru, ale DVIPS nabízí ještě efektnější metodu. Vytvoříme soubor pojmenovaný např. `mirror.hdr` obsahující příkazy z řádků 1–8 a DVIPS pak vyvoláme s parametrem `-h mirror.hdr`.

Život však není šedivý. V praxi budeme tisknout i na jiné formáty než A4. Můžeme sice pro každý formát napsat jiná makra, ale kdo se

s tím má dřít? Raději využijeme toho, že DVIPS vezme rozměry z příkazu `\special{papersize=...}` a vloží šířku papíru do konstanty `hsize` a výšku do `vsize`. Program pro zrcadlení pak můžeme napsat obecněji:

```
11 userdict begin
12 /bop-hook
13 {
14   -1 1 scale
15   hsize neg 0 translate
16 }
17 def
18 end
```

Změna je na řádku 15. Místo konkrétního čísla zde použijeme záporně vzatou (`neg`) hodnotu `hsize`. Na řádku 17 nyní nesmíme použít `bind def`, protože v okamžiku definice `bop-hook` není hodnota `hsize` ještě známa.

Uvedli jsme, že osa y má v PostScriptu opačný směr než v TEXu. Proto musíme v příkazu `\special{papersize=...}` uvést skutečné rozměry papíru. Jinak nebude text správně umístěn. Výše uvedené příkazy tedy nebudou fungovat v případě, kdy tiskneme v orientaci „landscape“. Rotace o 90° totiž způsobí, že se prohodí výška a šířka. Místo řádku 15 bychom tedy měli psát

```
19   vsize neg 0 translate
```

Nechceme však mít dva různé soubory pro zrcadlení, protože pak bychom v tom měli zmatek. Orientace je programem DVIPS vložena do konstanty `isls`, což zřejmě značí „is landscape“. Zcela obecně pak napíšeme:

```
20 userdict begin
21 /bop-hook
22 {
23   -1 1 scale
24   isls
25     { vsize }
26     { hsize }
27   ifelse
28   neg 0 translate
29 }
```

```
30 def
31 end
```

Řádky 24–27 tvoří podmíněný příkaz. Je-li hodnota `isls` pravdivá, provede se příkaz v prvním páru složených závorek, tj. na řádku 25 se do zásobníku operandů vloží hodnota `vsize`. V opačném případě se na řádku 26 vloží do zásobníku operandů hodnota `hsize`. Zbytek je již stejný jako dříve.

Nyní přistoupíme k poslednímu, komplikovanějšímu příkladu. Zde budeme demonstrovat, co vše lze s PostScriptem provést. Současně však důrazně varujeme před používáním takových praktik. Přesto však některé obraty mohou být užitečné.

Často chceme tisknout na papír formátu A4 dvě stránky A5 vedle sebe. Lze to provést na úrovni DVI souboru např. programem DVI2DVI. Můžeme to ovšem naprogramovat i v PostScriptu. Použijeme k tomu následující příkazy, které si opět uložíme do souboru `twoup.hdr`, abychom pak mohli volat DVIPS s parametrem `-h twoup.hdr`.

```
32 userdict begin
33 userdict /ZWM known {} { /ZWM false def } ifelse
34 userdict /ZW known {} { /ZW 2 def } ifelse
35 /start-hook { @landscape } def
36 /bop-hook {
37   gsave
38   ZWM { -1 1 scale vsize neg 0 translate } if
39   dup ZW mod
40   dup hsize ZW div neg mul 0 exch translate
41   1 add ZW eq
42   { userdict begin
43       /eop-hook { grestore } def
44       /flushpage {} def
45     end
46   }
47   { userdict begin
48       /eop-hook { end grestore } def
49       /flushpage { showpage } bind def
50     end
51     ZWdict begin
52   }
53   ifelse
```

```
54 } def
55 /flushpage {} def
56 /end-hook { flushpage } def
57 end
58 /ZWdict 5 dict def
59 ZWdict begin /showpage {} def end
```

Na řádku 32 opět otevřeme slovník `userdict`, kam budeme vkládat část definic. Řádky 33 a 34 jsou zde kvůli větší obecnosti. Operátorem `known` testujeme, zda je daný symbol v zadaném slovníku definován. Pokud se příslušný symbol najde, nebudeme dělat nic, proto jsou na obou řádcích první složené závorky prázdné. V druhých závorkách máme standardní hodnotu. Všimněte si, že v těchto definicích není hodnota konstanty v závorkách. Konstantu `ZWM` použijeme k rozhodování, zda se má stránka zrcadlově otočit, `ZW` definuje počet stránek, které se mají tisknout vedle sebe.

Dále potřebujeme zajistit, aby se text vždy tiskl v orientaci „landscape“. DVIPS to zajišťuje příkazem `@landscape`. Protože nemáme jistotu, že uživatel nastaví takové parametry, aby DVIPS tento příkaz vyslal, zařídíme to sami. Ještě před první stránku DVIPS umístí příkaz `@start`. Ten potom zavolá z `userdict` makro `start-hook`. Řádek 35 tedy zajistí tisk v orientaci „landscape“.

Řádky 36 až 54 přinášejí jádro řešení. Před vysvětlením příkazu z řádku 37 však musíme na okamžik odbočit. Na začátku článku jsme psali o zásobnících. Zde přidáme další, a to zásobník grafických stavů. Příkaz `gsave` totiž uschová grafický stav do zásobníku grafických stavů, příkazem `grestore` původní stav obnovíme. Mezi `gsave` a `grestore` obvykle uzavíráme všechny dočasné změny grafického stavu. Může to být např. změna transformace souřadnic. To je i náš případ.

Řádek 38 provede zrcadlení. Zrcadlíme podél svislé osy, abychom nenarušili činnost maker skládajících stránky.

Nyní musíme naprogramovat makra pro skládání stránek. K tomu potřebujeme znát číslo stránky, kterou právě tiskneme. Naštěstí `bop-hook` dostává v zásobníku dva parametry: číslo stránky dosazené TEXem a pořadové číslo stránky (číslované od nuly). Parametry můžeme použít, ale musíme je ponechat v zásobníku. Nám bude stačit pouze pořadové číslo, které máme právě na vrcholu zásobníku. Na řádku 39 jej tedy nejprve zkopírujeme instrukcí `dup` a potom vypočteme zbytek po dělení hodnotou `ZW`.

Po otočení o 90° do orientace „landscape“ bude pravý dolní roh tištěné stránky v levém horním rohu papíru. Pro nultou stránku je to správně, ostatní musíme posunout dolů. Výška papíru, což v orientaci „landscape“ je nyní šířka, je uložena v konstantě `hsize`. Musíme tedy stránku posunout o `hsize/ZW` vynásobenou zbytkem po dělení čísla stránky hodnotou `ZW`. To se právě zařídí na řádku 40. Zbytek po dělení čísla stránky hodnotou `ZW` budeme ještě potřebovat, proto si nejprve uděláme jeho kopii. Potom vypočteme hodnotu svislého posunu a přidáme do zásobníku nulu pro vodorovný posun. Protože parametry jsou nyní v zásobníku v opačném pořadí, než vyžaduje `translate`, musíme je prohodit instrukcí `exch`.

Místo řádku 40 jsme mohli použít příkazy:

```
60 dup /zbytek exch def
61 0
62 hsize ZW div
63 zbytek neg mul
64 translate
```

Příkazy jsme pro větší přehlednost ještě více rozčlenili. Neobvykle vypadá řádek 60. Jeho smyslem je uložení objektu na vrcholu zásobníku do pojmenované konstanty `zbytek`. Nejprve totiž vytvoříme kopii objektu na vrcholu zásobníku instrukcí `dup`. Poté vložíme na vrchol zásobníku „name-literal“ `/zbytek`. Instrukcí `exch` pořadí nejvyšších dvou objektů zaměníme, takže, označíme-li výsledek výpočtu z řádku 40 symbolicky $\langle zbytek \rangle$, mají instrukce

```
dup /zbytek exch
```

stejnou funkci jako

`/zbytek` $\langle zbytek \rangle$

Příkaz `def` tedy dostane vše tak, jak je vyžadováno.

Ostatní řádky jsou již obvyklé. Na řádku 61 vložíme do zásobníku nulovou hodnotu vodorovného posunu. Na řádku 62 vydělíme `hsize` hodnotou `ZW` a podíl vynásobíme na řádku 63 záporně vzatou hodnotou uschovaného zbytku. Na řádku 64 pak zavoláme transformaci `translate`.

Řádky 41–53 tvoří podmíněný příkaz. V něm nejprve zbytek po dělení čísla stránky hodnotou `ZW` zvětšíme o jednotku a výsledek porovnáme se

`ZW`. V případě rovnosti jsme skoro hotovi. Provedou se příkazy na řádcích 42–46. Zde se nadefinuje `eop-hook` tak, aby se instrukcí `grestore` obnovil grafický stav a `flushpage` se nadefinuje jako prázdná operace. Protože nevíme, jaký slovník bude na vrcholu zásobníku slovníků v okamžiku vykonávání `bop-hook`, otevřeme si explicitně `userdict`.

V případě nerovnosti se provedou příkazy na řádcích 47–52. Zde se definuje `eop-hook`, kde nejprve odstraníme aktuální slovník z vrcholu zásobníku slovníků a obnovíme grafický stav. Makro `flushpage` bude ekvivalentní standardnímu PostScriptovému příkazu `showpage`, který tiskne stránku. Na řádku 51 pak vložíme do zásobníku slovníků vlastní slovník `ZWdict`. Právě ten odstraňuje námi definovaný `eop-hook`.

Řádek 55 definuje počáteční význam procedury `flushpage`.

Příkaz na řádku 56 je velmi důležitý. Nemáme totiž jistotu, že počet stránek dokumentu bude celočíselným násobkem `ZW`. Využijeme tedy toho, že na konci dokumentu se volá `end-hook` a nadefinujeme jej tak, aby se volalo naše makro `flushpage`. To jsme během zpracování souboru stále předefinovávali podle toho, zda byla již složena celá stránka.

Na řádku 58 vytvoříme operátorem `dict` slovník `ZWdict`, do kterého se vejde 5 definic. Na následujícím řádku do něj vložíme definici `showpage`, která nebude dělat nic. Na řádku 51 jsme vložili slovník s touto definicí na vrchol slovníku zásobníků. Protože na konci stránky se vždy volá `showpage`, máme tím zajištěno, že se nevytisknou neúplné stránky.

Pokud se po odladění dokumentu rozhodneme, že jej chceme zrcadlově převrátit, stačí použít navíc soubor s příkazem:

```
65 userdict /ZWM true def end
```

Podobně tisk tří stran vedle sebe způsobíme souborem s příkazem:

```
66 userdict /ZW 3 def end
```

Zbývá ještě vysvětlení, proč jsme před používáním tohoto příkladu varovali. Souvisí to s požadavky na členění PostScriptových souborů. Podle standardu Adobe má mít PostScriptový soubor tzv. prolog a skript. Prolog obsahuje definice vyžadované v celém dokumentu, download nerezidentních fontů, nastavení specifických konstant a podobně. Skript obsahuje popis jednotlivých stránek a na jeho konci je „trailer“, který vše uklidí. Stránky musí být samostatné. Žádná stránka nesmí ovlivňovat stránky následující. Právě to jsme v našem příkladu porušili. Existují totiž programy, které dokážou z PostScriptového souboru vybrat některé stránky a poslat je do PostScriptového zařízení. Kdyby stránky

byly na sobě závislé, nedopadlo by to dobře. Příklad, který jsme zde uvedli, pamatuje skoro na vše. Vybereme-li souvislý úsek, vytiskne se vše dobře. Program, který stránky vybírá, musí totiž vždy zkopírovat prolog i trailer. Tím se vše, co jsme napáchali, zase napraví. Pokud ale vybereme několik náhodných stránek, nebude podmíněný příkaz na řádcích 41–53 fungovat správně a výsledkem bude zmatek.

V tomto příspěvku nebylo možné podat vysvětlení PostScriptu. Není zde dostatek prostoru. Jeho cílem bylo pouze upozornění na některé nové možnosti a záludnosti. Pro toho, kdo má zájem dozvědět se o PostScriptu více, lze doporučit např. knížku Davida A. Holzganga: Understanding PostScript (SYBEX, San Francisco), která poslouží jako vhodná základní učebnice pro začátečníky.

Zdeněk Wagner
<wagner@icpf.cas.cz>

Kreslení obrázků v TEXu pomocí `mujmfpic`

Jaromír Kuben

V několika posledních číslech TEX bulletinu se objevila řada příspěvků týkajících se otázky, jak vyrobit v TEXovském dokumentu obrázky nejrůznějšího druhu. Chtěl bych se se zájemci o tuto problematiku podělit o své zkušenosti s balíkem `mfpic`, který využívá pro tyto účely program METAFONT, aniž je třeba znát jeho jazyk (což skalní zastánci tohoto skvělého programu odsuzují).

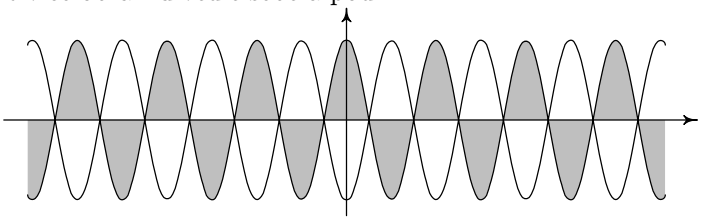
S tímto balíkem ve verzi 0.2 jsem se poprvé setkal před prázdninami 1993. Vzorové obrázky vypadaly pěkně, dobře dopadly i první experimenty, ale současně se objevily i jisté potíže. Protože jsem tehdy měl za sebou určité, sice nevelké ale úspěšné, pokusy s METAFONTem (namaloval jsem obrázky do dvojích skript), pokusil jsem se problémy odstranit. Stálo mne to značné úsilí, ale podařilo se. Přitom mne napadlo, že když už jsem věnoval tolik času na proniknutí do zdrojového kódu, nebylo by od věci rozšířit škálu obrázků, které lze pomocí `mfpic` namalovat. Zejména jsem postrádal křivky dané parametricky nebo v polárních sou-

řadnicích, vadilo mi, že uzavřené křivky, které se měly nějak „vyplnit“, se nesměly protínat, nerespektoval se přesah popisů vlevo, nahoru a dolů mimo plochu obrázku a pod.

Když jsem vše dokončil, dozvěděl jsem se, že autoři uveřejnili další verzi 0.25, která je opravená. „Naštěstí“ šlo skutečně jen o opravu některých chyb a vylepšení průběžných hlášení při překladu, takže moje práce s rozšířením nebyla zbytečná. Svůj „výtvor“ (vycházející z verze 0.2) jsem nazval `mujmfpic` a zaslal ho autorům. Ti se o něm vyjádřili pochvalně a přislíbili, že většinu mých doplňků zahrnou do nové vylepšené verze `mfpic`, která by se podle poslední informace snad mohla v dohledné době objevit (pracuje se na ní).

A nyní konečně něco k praktickému použití. Jde o makra, která lze použít ve formátech `plain`, $\mathcal{AMS}$-TEX i LATEX. Přitom obrázky nakreslí METAFONT a případný popis provede TEX. Kompletní zdrojový kód pro obrázky se napíše do zdrojového TEXovského souboru. Při překladu TEXem je vygenerován pomocný soubor, který obsahuje zdrojový kód pro METAFONT. Tento soubor je třeba přeložit METAFONTem a vzniklý generický font převést na `pk` font. Po dalším překladu TEXem a zavolání prohlížečky se objeví obrázky na obrazovce.

Nejprve je třeba načíst příslušné definice příkazem `\input mujmfpic` pro `plain` resp. `\input mlamfpic` pro LATEX (Pozor! Pokud používáte instalaci $\mathcal{CS}$TEXu a styl `czech`, nesmí být v tomto okamžiku aktivní příkaz `\csprimeson`). V nové verzi už nebude speciální soubor pro LATEX a nebude tudíž speciální LATEXovské okolí pro obrázky — viz níže. Dále příkazem `\opengraphsfile{jmeno}` určíme, jak se bude jmenovat pomocný soubor pro METAFONT. Kód pro jednotlivé obrázky je obklopen dvojicí příkazů `\picture` a `\endpicture` pro `plain` resp. `\begin{mfpic}` a `\end{mfpic}` pro LATEX. Někam za poslední obrázek je třeba umístit příkaz `\closegraphsfile`. Celý obrázek bude vysázen jako `\hbox` na místě odpovídajícím umístění zdrojového kódu. Chová se tedy jako jeden velký symbol a je proto možné jej obtékat, dát do plovoucího materiálu, umístit více obrázků vedle sebe a pod.



Předchozí obrázek byl vytvořen příkazy

```
\opengraphsfile{obr}
\begin{mfpic}[3]{-43}{44}{-12}{14}
\axes
\shadefcn{-40,40}{10*cosd(32*x)}{0}
\function{-40,40,1,10*cosd(32*x)}
\function{-40,40,1,-10*cosd(32*x)}
\end{mfpic}
\closegraphsfile
```

Všechny rozměry jsou zadávány v relativních jednotkách. Parametr `[4]` udává, kolik bodů (pt) má tato jednotka. Ve verzi 0.2 a v `mujmfpic` toto číslo musí být celé, ve verzi 0.25 už tomu tak není. Dále mohou být odlišné jednotky v horizontálním a vertikálním směru, pak se zadá např. `[3][2]`. Další parametry udávají horizontální a vertikální rozměry kreslící plochy jako rozsah souřadnic na osách x a y. Skutečná velikost je tedy v našem případě 261 pt × 78 pt. Další kód je celkem názorný. U příkazu `\shadefcn` jsou zadány dvě funkce (zde $y = 10\cos 32x$ a $y = 0$), mezi nimiž se plocha „vyšedí“. Funkce `cosd` je kosinus s argumentem ve stupních.

Po překladu TEXem vznikne mimo jiné soubor `obr.mf`. Aby byl překlad úspěšný, je třeba umístit soubory `mlamfpic.tex` a `mujmfpic.tex` do některého adresáře, kde TEX hledá vstup. Dále je třeba umístit soubor `mgraphba.mf` do adresáře, kde hledá vstup METAFONT. Pak se přeloží soubor `obr.mf` METAFONTem, což se provede např. příkazem `mf obr`. Výstup bude určen pro zařízení odpovídající volbě `mode=localfont`. Pokud má být pro jiné zařízení, např. tiskárnu Epson, bude vstup např. `\mode=epsonfx;input obr`. Je-li konkrétně `localfont=hplaser`, vzniknou soubory `obr.tfm` a `obr.300`. Číslo 300 znamená, že rozlišitelnost, pro niž byl tento font vygenerován, je 300 dpi. Nyní se zavolá program `gftopk`, který převede tzv. generický font `obr.300` na `pk` font `obr.pk`. Tyto soubory budou v aktuálním adresáři. Proto je třeba zajistit, aby TEX hledal metriky i zde. Např. pro `tex386` je nutné nastavit proměnnou `TEXTFM`. Konečně je třeba zajistit, aby příslušné ovladače (pro obrazovku, tiskárnu a pod.) hledaly bitové mapy fontů také aktuálním v adresáři. Při dalším překladu TEXem bude obrázek zařazen na své místo.

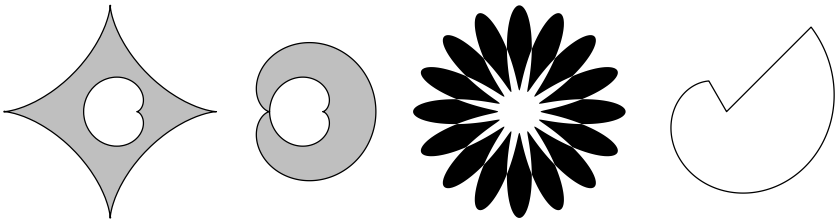
Nyní je na čase přiznat se k malému podvodu. Předchozí obrázek je vycentrován. Ukázalo se ale, že LATEXovské okolí `center` dělá jisté problémy (související zřejmě s přechodem z vertikálního do horizontálního

modu). Ty se projeví tím, že obrázek není přesně uprostřed, popisy jsou zcela odtrženy mimo obrázek a pod. Nepomohl ani příkaz `\leavevmode`. Nakonec jsem problém vyřešil následujícím způsobem. Dodefinoval jsem příkaz `\putintobox`, který má jeden parametr, což musí být malá římská číslice od jedné do pěti, tj. `i` až `v`. Pak příkaz `\putintobox{i}` umístěný kdekoli mezi `\begin{mfpic}` a `\end{mfpic}` způsobí, že obrázek nebude vysazen, ale pouze umístěn do boxu se jménem `\boxi`. Kód pro předchozí obrázek obsahoval právě tento příkaz a vlastní obrázek byl pak umístěn takto:

```
\begin{center}
\leavevmode
\box\boxi
\end{center}
```

Elegantnější by bylo zjistit, co je skutečnou příčinou nežádoucích efektů, ale to se mi nepodařilo. Třeba mi některý zkušenější TEXpert poradí, a pak bude možné tento obrat vypustit.*

Předvedeme si nyní několik ukázek toho, co lze pomocí `mujmfpic` nakreslit. Výběr je volen víceméně náhodně.



Zdrojový kód vypadá takto:

```
\begin{mfpic}[2]{-20}{136}{-20}{20}
\paradiscshade[o]{0,360,6,0,360,6}%
{(20*((cosd(t))**3),20*((sind(t))**3))}%
{(-5*(1+cosd(t))*cosd(t)+5,-5*(1+cosd(t))*sind(t))}
\polardiscshade[o]{(30,0),0,360,(40,0),0,360}%
```

* V plainu funguje osvědčené `\noindent\hfil` (za předpokladu, že není změněna hodnota `\parfillskip`, nebo jiného důležitého parametru). (*Pozn. red.*)

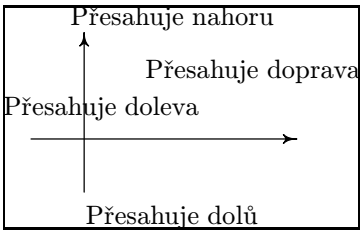
```
{10*(1+cosd(t))}{-5*(1+cosd(t))}
{\shadespace=0pt
\polardiscshade{(77,0),0,360,(77,0),0,360}%
{12+8*cosd(8t)}{12-8*cosd(8t)}}
\polarwedge{(116,0),120,405,t/18}
\end{mfpic}
```

V prvním obrázku jsou obě křivky (asteroida a kardioida) zadány parametricky. Ve druhém obrázku jsou obě kardioidy dány v polárních souřadnicích, Stejně tak je tomu ve třetím obrázku, z něhož je vidět, že křivky se mohou protínat a že místo „vyšedění“ je možné obrázek „vyčernit“ (příkaz `\shadespace=0pt`). Ve čtvrtém obrázku je Archimedova spirála dána v polárních souřadnicích. Tato křivka není uzavřená a příkaz `\polarwedge` spojí její konce se středem souřadnic.

Obrázky je rovněž možné popisovat. K tomu slouží příkaz `\label` (není to standardní příkaz LaTeXu, který platí mimo okolí `mfpic`). Následující obrázek byl vytvořen takto:

```
\begin{mfpic}[2]{-10}{40}{-10}{20}
\axes
\label[bl]{12}{12}{Přesahuje doprava}
\label[br]{22}{5}{Přesahuje doleva}
\label[bc]{17}{22}{Přesahuje nahoru}
\label[tc]{17}{-12}{Přesahuje dolů}
\putintobox{i}
\end{mfpic}
\fboxsep=0pt
\fbox{\box\boxi}
```

Text bude vysázen do boxu, jenž bude vzhledem k referenčnímu bodu umístěn horizontálně vlevo, centrovaně nebo vpravo a vertikálně nahoru, centrovaně nebo dolů. Přesah popisů je brán v úvahu při určování velikosti výsledného boxu (obsahujícího obrázek a popisy). Totéž platí pro případný popis pod obrázkem vytvořený příkazem `\caption` (i tento příkaz je odlišný od standardního příkazu LaTeXu,



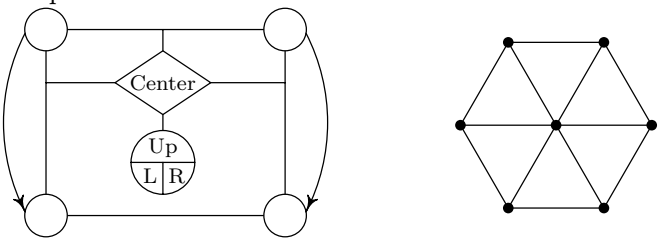
který platí mimo okolí `mfpic`). Současně je vidět, že není problém obrázek obtéct textem.

Poslední ukázka je především pro zájemce o teorii grafů. Celkem jednoduše lze nakreslit i komplikované grafy. Výhodou je, že METAFONT umí „přemazávat“ čáry, takže je možné nejprve nakreslit hrany, a pak teprve uzly (např. jako kroužky). Kdo umí trochu METAFONT, může souřadnice vrcholů zadávat pomocí funkcí, které jsou k dispozici. Proto je snadné vytvořit např. pravidelný šestiúhelník. Zdrojový kód vypadá takto:

```
\footnotesize
\begin{mfpic}[2]{-4}{119}{0}{43}
\polycurve{(4,4),(49,4),(49,39),(4,39),(4,4)}
\line{(26,14),(26,39)}\line{(4,29),(49,29)}
\dkruh{6,(26,14)}\kruhy{4,(4,4),(49,4),(4,39),(49,39)}
\rozhod{(26,29),9,6}
\setlength{\headlen}{6pt}
\arrowcorr{-3}
\curvedarrow{(4,39)+4(cosd(-170),sind(-170)),%
(-4,21.5),(4,4)+4(cosd(170),sind(170))}
\arrowcorr{3}
\curvedarrow{(49,39)+4(cosd(-10),sind(-10)),%
(57,21.5),(49,4)+4(cosd(10),sind(10))}
\label[bc]{26}{15.5}{Up}\label[tr]{25}{13}{L}
\label[tl]{27}{13}{R}\label[cc]{26}{29}{Center}
\polygon{18(cosd(0),sind(0))+(100,21),%
18(cosd(60),sind(60))+(100,21),%
18(cosd(120),sind(120))+(100,21),%
18(cosd(180),sind(180))+(100,21),%
18(cosd(240),sind(240))+(100,21),%
18(cosd(300),sind(300))+(100,21)}
\line{18(cosd(0),sind(0))+(100,21),%
18(cosd(180),sind(180))+(100,21)}
\line{18(cosd(60),sind(60))+(100,21),%
18(cosd(240),sind(240))+(100,21)}
\line{18(cosd(120),sind(120))+(100,21),%
18(cosd(300),sind(300))+(100,21)}
\disky{1,(100,21),18(cosd(0),sind(0))+(100,21),%
18(cosd(60),sind(60))+(100,21),%
```

```
18(cosd(120),sind(120))+(100,21),%
18(cosd(180),sind(180))+(100,21),%
18(cosd(240),sind(240))+(100,21),%
18(cosd(300),sind(300))+(100,21)}
\putintobox{i}
\end{mfpic}
\begin{center}
\leavevmode
\box\boxi
\end{center}
```

A takhle dopadne obrázek:



Nejprve se příkazy `\polycurve` a `\line` nakreslí hrany. Pak se příkazy `\kruhy`, `\dkruh` („dělený“ kruh) a `\rozhod` vytvoří uzly. Příkazy `\curvedarrow` udělají oblouky se šipkami. Implicitně jsou osy šipek tečnami k těmto obloukům. Protože se mi zdálo někdy hezčí (v případě velké křivosti oblouku) šipku pootočit, zavedl jsem příkaz `\arrowcorr`, který říká, o kolik se má šipka pootočit od tečné polohy. Podobně je vytvořen šestiúhelník. Kód je poněkud dlouhý, ale to je dáno způsobem zadání souřadnic vrcholů pomocí funkcí `cosd` a `sind`. Díky tomu lze snadno celý šestiúhelník posunout nebo změnit jeho poloměr.

Na závěr několik poznámek. Součástí balíku `mujmfpic.zip` je dokumentace, kde lze najít popis dalších příkazů, a soubor s ukázkami. Celý balík je k dispozici např. na anonymním ftp `ftp.muni.cz` v adresáři `pub/tex/teware`.

Zejména při tvorbě „vyšeděných“ obrázků je třeba počítat s tím, že překlad METAFONTem může trvat hodně dlouho (i s `mf386` až několik minut; např. zpracování obrázků k tomuto článku na PC486 trvalo asi 65s). Dále pokud je „vyšeděná“ plocha příliš velká (čtverec $10\,\mathrm{cm} \times 10\,\mathrm{cm}$ je už hrozně moc), METAFONT zhavaruje (bohužel i `mf386`, nemám zkušenosti s UNIXem). Podobně se můžeme setkat s potížemi při převodu

generického fontu na `pk` font pomocí programu `gftopk`. Pod DOSem lze ve WEBovském zdroji, který mám k dispozici, nastavit jakýsi parametr maximálně na 65 534 (vím, že pod UNIXem toto omezení neplatí). Poněkud větší verzi mi poskytl kolega Horák, ale i ta zhavaruje např. pro čtverec se svislým šrafováním 1 mm od sebe o velikosti asi 7 cm × 7 cm. Často ale pomůže rozdělit obrázek do několika, použít umístění do boxu (příkaz `\putintobox`) a boxy přes sebe překrýt, což TEX zcela přesně dokáže (viz též následující ukázka).

A na úplný konec „lahůdka“ pro ty, kteří mají k dispozici program `dvips` a `ghostview` (nebo aspoň `ghostscript`) a barevný monitor. Snad vás výsledek následujícího zdrojového textu potěší.

```
\documentclass{report}
\usepackage{colordvi}
\input mlamfpic
\textwidth400pt\pagestyle{empty}
\begin{document}
\opengraphsfile{kvet}
\begin{mfpic}[9]{-22}{22}{-22}{22}
\shadespace=0pt\cdisk{(0,0),12}\putintobox{i}
\end{mfpic}
\begin{mfpic}[9]{-22}{22}{-22}{22}
\shadespace=0pt\cdisk{(0,0),4}\putintobox{ii}
\end{mfpic}
\begin{mfpic}[9]{-22}{22}{-22}{22}
\shadespace=0pt
\polardiscshade{(0,0),0,360,(0,0),0,360}{12+8*cosd(8t)}%
{12-8*cosd(8t)}\putintobox{iii}
\end{mfpic}
\begin{mfpic}[9]{-22}{22}{-22}{22}%
\shadespace=0pt\polarcurve{(0,0),0,360,12+8*cosd(8t)}
\polarcurve{(0,0),0,360,12-8*cosd(8t)}\putintobox{iv}
\end{mfpic}
\wd\boxi=0pt\wd\boxii=0pt\wd\boxiii=0pt
\centerline{\Green{\box\boxi}\Red{\box\boxii}%
\Yellow{\box\boxiii}\OrangeRed{\box\boxiv}}
\closegraphsfile
\end{document}
```

Pokud by program `dvips` zhavaroval (verze `dvips32` pro PC386 vyžadující koprocesor to zvládne), zmenšete velikost měřítka 9 na 3. Současně je vidět, že `mujmfpic` se snáší i s LaTeXem 2_ε.

Rychlejší tisk na devítijehličkových tiskárnách

Jan Bryscejn

Tento krátký článek se zabývá možnostmi tisku `.dvi` souborů na devítijehličkových (či 24jehličkových) tiskárnách s použitím Mattesova ovladače a několika dalších utilit.

Nejprve se podívejme na situaci, kterou chceme řešit. Je jasné, že pro výstup z TeXu, který má vůbec někdo číst, se hodí laserová nebo alespoň inkoustová tiskárna s minimálně 300 dpi. Aby byl výstup opravdu kvalitní, je dokonce i 600 dpi někdy málo. Na druhé straně příznivci TeXu v tomto programu vyrábějí kdeco, takže vzniká několik situací, kdy je vhodný tisk na jehličkové tiskárně.

Zejména se může stát, že není kvalitní tiskárna dostupná. Podle dalších okolností může být výtisk na jehličkové tiskárně již konečný, nebo požadujeme pouze koncept kvůli nastavení tiskárny; často nás zajímá celkový layout vytištěné strany a nikoliv podrobnosti. Konečně je možné, že dokument je pro naši soukromou potřebu a obětujeme kvalitu tisku rychlosti; to může nastat zejména u různých `.doc` souborů dodávaných s TeXem.

Pokud máme k dispozici 24jehlovou tiskárnu, nenastává žádný problém, neboť při rozlišení 180×180 dpi tiskne celou řádku na jeden průchod. Proto se tímto případem nebudeme dále zabývat. Problémy však nastanou při použití devítijehlové tiskárny, kdy se každý vodorovný proužek dat tiskne na tři, nebo dokonce na šest průběhů s mikrořádkováním mezi jednotlivými průběhy. To je jednak časově zcela neúnosné a kromě toho kvalita bývá horší než při 180×180 dpi 24jehličkové tiskárny, přestože rozlišení je 240×216 dpi.

Jak tedy tento problém vyřešit? Pokud máme k dispozici Mattesův ovladač `dvidot` a utilitu `makedot`, můžeme ovladač přinutit, aby tiskl

každý řádek na jeden či na dva průchody. Dostaneme tak výsledné rozlišení buď 240×72 dpi či 240×144 dpi. (Samozřejmě můžeme snížit i rozlišení ve vodorovném směru — např. 120×144 dpi — ale to je naprosto zbytečné.) Tím se sice zhorší kvalita už tak dost nekvalitního výstupu, ale získáme čitelný náhled dokumentu v tištěné podobě a v rozumném čase. Při rozlišení 240×72 dpi (jeden průchod) je ještě čitelná sazba písmem `cmr10`. Písma menší či bezpatková už představují problém.

Informace dále uvedené pocházejí z mého experimentování a čtení helpů a nemusí tedy být stoprocentně správné — předem se omlouvám za nepřesnosti. Konečné řešení je ovšem spolehlivě ověřeno v praxi.

Program `dvidot` potřebuje ke své činnosti parametrický soubor s příponou `.dot`, který do značné míry řídí jeho chování. V instalaci CSTEXu či emTEXu najdete těchto `.dot` souborů několik — pro různé jehličkové tiskárny v různých rozlišeních a s různými šířkami válce, ale také pro „tisk“ do grafických souborů typu `PCX` či `BMP`. Pro devítijehličkovou epsonku jsou zde však pouze dva soubory pro různou šířku válce. V instalaci CSTEXu jde o základní soubor `dot240.dot`, který pracuje v rozlišení 240×216 dpi, tedy tři průběhy v rozlišení 240×72 dpi na každý řádek.

Bohužel, když si tyto soubory prohlédnete, zjistíte, že jsou binární a tudíž lidsky nečitelné. K převedení těchto souborů do čitelné podoby a zpět slouží program `makedot` s příslušnými řádkovými přepínači.

Program `dvidot` ovšem zpracovává i další soubory: vstupní `.dvi` soubor, výstupní soubor (tiskárna = `PRN`, soubor pro pozdější kopii na tiskárnu, či grafický soubor `PCX`, `BMP`) a soubor s parametry. Ten se připojuje pomocí znaku `@` následovaného jménem souboru. V něm jsou na jednotlivých řádcích uvedeny údaje např. o tom, které stránky se mají tisknout, cesty k fontům, pozice tisku na stránce atd. Informace tohoto typu lze ovšem programu `dvidot` sdělit i jinými způsoby. Buď jako parametry na příkazové řádce, nebo v interaktivním módu z klávesnice — do interaktivního módu se dostaneme přepínačem `//` uvedeném buď v konfiguračním souboru, či na příkazové řádce. Kromě toho všeho ovlivňují chování `dvidot` i proměnné DOSu, nastavené pomocí příkazu `SET`. Obecně tedy může vypadat volání ovladače `dvidot` takto:

```
DVIDOT DOT240.DOT myfile.dvi @myconf{\tt .cfg} /t2cm //
```

Zde jsme nastavili 2 cm shora a vyvolali interaktivní režim, v němž je možné příkazem `?` či `??` zjistit nastavení všech parametrů a případně provést změny.

Jak vidíme, je celé řízení programu `dvidot` velmi komplikované. I když s tím běžný uživatel téměř nepřijde do styku, doporučuji vřele alespoň si prohlédnout `help` soubor `dvidot.doc`. Upozorňuji, že má asi 180 kB (více než program).

Naším úkolem nyní bude připravit soubory `.dot` a `.cnf` pro práci v rozlišení 240×72 dpi a 240×144 dpi, tedy pro tisk na devítijehličkové tiskárně v jednom či dvou průchodech. Jako výchozí nám poslouží parametrický soubor `dot240.dot` a konfigurační soubor pro tisk na této tiskárně, který by se měl jmenovat `prn240.cnf`. V konkrétní instalaci TEXu se tyto soubory mohou jmenovat jinak a zjistíte to z `.bat` souboru, který vyvolává tisk. Dále budeme potřebovat utilitu `makedot.exe` a doporučuji k prohlédnutí i `makedot.doc` a `dvidot.doc`.

Nejprve je třeba zčitelnit parametrický soubor `dot240.dot`. To provedeme příkazem

```
MAKEDOT -d DOT240.DOT DOT240.TXT
```

Upozorňuji, že pokud neuvedete výstupní soubor, ukládá se výsledek do `dot240.dot`. Abychom věc neprodlužovali, uvedeme rovnou příklad úpravy textového tvaru řídícího souboru. Pro tisk na jeden průchod v rozlišení 240×72 dpi může vypadat takto:

```
%
% dotx72.txt
%
COMMENT=EPSON FX-80 (240x72 dpi)
TYPE=DOT
ENV_NAME=DVIFX
LOG_NAME=dvidot.dlg
FONT_PATH=\texfonts\pixel.fx\@rdpi\@f{.pk,.pxl}
VF_PATH=
PAGE_WIDTH=8in
PAGE_HEIGHT=11in
FORM_LENGTH=
RESOLUTION=240 72
COLUMNS=1920
ONE_LINE_FEED=20
BLANK_WIDTH=24
METHOD=1 1
```

```
MAX_WIDTH=1920
PINS=7 0
MAX_LF=255
FF_METHOD=FF
S_OPTION=DOUBLE_STRIKE
INIT1=ESC '@ ESC '! 0 ESC '3 BYTE (one_lf*3)
INIT2=ESC '@ ESC '! 0 ESC '3 BYTE (one_lf*3)
EXIT=ESC '@
GRAPH_MODE=ESC 'Z WORD_LH pixels
GRAPH_END=
LINE_FEED=ESC 'J BYTE (line_feed*3)
FORM_FEED=FF
POS_X=
DOUBLE_SIDED=
```

Podstatné jsou vlastně jen změny v nastavení `RESOLUTION`, `METHOD`, `INIT1`, `INIT2` a `LINE_FEED`, kde je po řadě místo `240 216`, `1 1`, `one_lf`, `one_lf` a `line_feed` postupně `240 72`, `1 1`, `(one_lf*3)`, `(one_lf*3)` a `(line_feed*3)`. Všimněme si ještě řádku `S_OPTION=DOUBLE_STRIKE`, který říká ovladači, že zapnutím přepínače `s` se bude každý řádek tisknout dvakrát.

Nyní stačí zpětně přeložit upravený soubor (nazvěme jej např. `dotx72.txt`) do binárního tvaru pomocí:

```
MAKEDOT -c DOTX72.TXT DOTX72.DOT
```

Tím jsme v podstatě hotovi. Nyní je třeba provést úpravu v konfiguračním `.cfg` souboru. Můžeme použít buď zmenšené fonty, kdy zmenšení ve svislém směru na jednu třetinu provede přímo `dvidot`, nebo vygenerovat nové fonty METAFONTem, kde by však bylo třeba zavést nové výstupní zařízení jednak do `mfjob` definic a jednak do `local.mf` souboru metafontu a vytvořit znova bázové soubory. V případě rozlišení 240×72 dpi druhou možnost rozhodně nedoporučuji (první, s čím se setkáte, budou hlášení METAFONTu `strange path...`). Použijeme tedy první možnost. Nejprve zkopírujeme starý konfigurační soubor `prn240.cnf` na nový, např. `prnx72.cnf` a zavedeme do něj tyto změny:

```
\rx 240
\ry 72
```

```
\fsy 3
```

Ty způsobí nastavení správného rozlišení v obou směrech a zmenšování fontů svisle na třetinu výšky. Pokud se nechcete zabývat změnami ve svém menu systému, kterým spouštíte práci s TEXem z DOSu a vyvoláváte z něj TEXovou smyčku a tisk atd., můžete vytvořit jednoduchý batch soubor pro tisk v tomto rozlišení, např.:

```
dvidrv dvidot dotx72.dot @prnx72.cfg %MAIN% //
```

Tento soubor nazvěte třeba `prnx72.bat` a umístěte do některého v DOSu přístupného adresáře. Důležité je, že jej nesmíte volat rovnou z DOSu, ale až po nastavení správných environment proměnných, tedy např. příkazem DOS z TEXového menu.

Situace při použití rozlišení 240×72 dpi je zcela bez problému, protože fonty mají svislé rozlišení 216 dpi, což je přesně trojnásobek základního rozlišení devítijehlové tiskárny (72 dpi). Pokud však chceme tisknout ve střední kvalitě, tj. na dva průchody v celkovém rozlišení 240×144 dpi, nastanou podstatné problémy. Díky způsobu, jakým přistupuje `dvidot` k řádkování, dojde prakticky vždy k zaokrouhlovacím chybám (písmena se buď budou roztrhávat nebo stlačovat). Tím se změní i celková délka stránky, přičemž na A4 straně může chyba činit něco okolo 1 cm. Zde jsou klíčové změny v `.dot` souboru:

```
%
% dotx144.txt
%
COMMENT=EPSON FX-80 (240x144 DPI)
RESOLUTION=240 144
METHOD=1 2
INIT1=ESC '@ ESC '! 0 ESC '3 BYTE ((one_lf*3)/2)
INIT2=ESC '@ ESC '! 0 ESC '3 BYTE ((one_lf*3)/2)
LINE_FEED=ESC 'J BYTE ((line_feed*3)/2)
```

Ještě větším problémem jsou fonty. Pokud nebudeme generovat nové pomocí METAFONTu, máme možnost zmenšení stávajících pouze na polovinu, nebo na třetinu příkazy `\fsy 2` resp. `\fsy 3` v konfiguračním souboru. Přitom bychom potřebovali zmenšení na 2/3. Dochází samozřejmě

ke zkreslení (čitelnost naštěstí příliš netrpí). Pokud se rozhodneme nagenerovat nové fonty pro rozlišení 240×144 METAFONTem, musíme zavést nové výstupní zařízení apod. a nové fonty zaberou místo na disku. Také nesmíme zapomenout rozlišit cestu k těmto fontům v konfiguračním souboru. To vše lze udělat, jen je to poněkud pracné.

Z výše uvedeného vyplývá, že pokud chcete např. pro tisk manuálů použít devítijehlovou tiskárnu, stačí PlainTEX s fontem `csr10` a upravený `.dot` a `.cfg` soubor pro tisk na jeden průběh v rozlišení 240×72 dpi.

Na závěr znovu doporučuji, pokud máte možnost, prostudovat soubor `dvidot.doc` — `dvidot` je rozsáhlý a zajímavý program se spoustou možností a pan Mattes nám ho dal zadarmo; využijme tedy všechny jeho možnosti.

Recenze: 4allTEX

Pavel Rychetský

Na červnovém setkání CSTUGu se objevil compact disk nazvaný 4allTEX, presentovaný jako sbírka TEXovského softwaru a nabídnutý k recenzi, ke které jsem se odhodlal. Nakonec se ukázalo, že vše je trochu jinak, 4allTEX si klade za cíl býti něčím trochu jiným než jen TEXovským archívem a já jsem si uvědomil, že místo recenze je jedině možné – a doufám že i účelnější – pokusit se jen o informaci „what is what“ doplněnou nesouvislými, ale o to subjektivnějšími postřehy.

Co nalezneme

Omezím se jen na stručný výčet obsahu CD, tak aby si uživatel mohl vytvořit přibližný obrázek, co zde asi nalezne:

TEX: emTEX verze 3.1415 [3c–beta 11] – `tex86, tex186, tex386` (podpora `emx` i `rsx`), `btex86, btex386`;

METAFONT: Mattesovy `mf86, mf186, mf386, bmf86, bmf186` ve verzi 2.71 [3c–beta1];

TEXovské formáty: plainTEX, LATEX, $\mathcal{AMS}$-TEX, $\mathcal{AMS}$LATEX, LATEX2$_\varepsilon$;

ovladače zařízení: Mattesův `dvidrv` verze 1.5a; další viz *Postscript*;

fonty: Computer Modern, Washington Cyrillic CM, fonty z $\mathcal{AMS}$-TEXu, DC fonty; většina ve formě zdrojových textů pro METAFONT i již přeložené jako `.pk` soubory, resp. zkompilované do `.fli` knihoven;

editory: Qedit (verze 2.15 a 3.0) + TEXovská makra pro něj, REdit (editor německé provenience speciálně napsaný pro podporou TEXu), Emacs, MultiEdit;

PostScript: `.tfm` soubory pro standardní PS fonty, NFSS pro PostScript, makra podporující využití možností PostScriptu v TEXovských dokumentech, `dvips` verze 5.54, Ghostscript + fonty, `afm2tfm`, několik desítek PS fontů ve formě `.pfb` nebo `.pfa`, utility pro práci s PS fonty i `.ps` soubory;

národní a jazykové mutace: francouzský balík PC-GUT, polský MeX a LaMeX, ArabTEX od K. Lagallyho, Dryllerakisův GreeKTEX, Russian TEX, Babel;

„netextový" TEX: MusicTEX, ChemTEX (sazba chemických vzorců), makra pro sazbu šachu, go, bridge;

utility: podpora práce s `webem`, BibTEX + `.bib` soubory obsahující TEXovskou bibliografii, MakeIdx, utility pro další práci s `.dvi` soubory, konverzní utility z formátů různých textových editorů do TEXu, `bm2font` (konverze obrázků do TEXovských fontů), Graphic Workshop (umožňuje konverzi mezi grafickými formáty);

texty: více než 10 publikací zabývajících se TEXem (ze známějších např. A Gentle Introduction to TEX od M. Dooba nebo Partlův LATEX Kurzbeschreibung) včetně elektronické podoby TEXbooku a METAFONTbooku, oblíbené `.faq` soubory (Frequently Asked Questions), záznamy některých TEXovských konferencí (asi nejznámější je `TeXhax`).

Výčet není v žádném případě kompletní, šlo mi spíše jen o ilustrativní pohled a snahu ukázat, co vše se vejde na jeden compact disk (a to není jeho potenciální kapacita využita zdaleka plně).* Je vidět, že CD 4allTEX není sice zdaleka zrcadlovým obrazem některého známého TEXovského archívu, ale na druhé straně obsahuje téměř vše (respektive v mnoha oblastech mnohem více), co potřebuje průměrně náročný uživatel TEXu ke své práci včetně některých specialit.

Na první pohled mi chybí snad jen větší zastoupení maker a fontů pro sazbu textů psaných jinak než latinkou a TEX–XET (umožňuje kombi-

* Ta je využita u nového vydání, které se objevilo v září během EuroTEXu a jež by mělo být už brzy k dispozici i u nás. (*Pozn. red.*)

novat psaní zleva doprava a zprava doleva). Absence tohoto je možná způsobena tím, že dosud není součástí emTEXu.

4allTEX

Jádrem compact disku je stejnojmenný balík TEXovského a paraTEXovského softwaru, vytvořený a distribuovaný prostřednictvím *NTG* (nizozemský TUG). Balík je určen pro DOS resp. OS/2 a je postavený na emTEXu. Na disku se vyskytuje ve dvou verzích. Za prvé je to kopie distribuce na disketách (30 disket 3,5″ HD). Touto jsem se příliš dlouho nezabýval, jen jsem konstatoval, že její instalace je o poznání méně přívětivá, než jak ji známe z idiotenfest CSTEXu.

Za druhé je 4allTEX přítomen již na compactu rozbalený a připravený k použití – díky tomu, že jsou zde fonty ve všech běžně používaných rozlišeních, obsahuje adresář `EMTEX` přibližně 270 MB dat (na celém CD jde o cca 470 MB v téměř 20 000 souborech). Stačí jen spustit krátkou instalační dávku, která vytvoří v námi zvoleném – ale již existujícím – adresáři na pevném disku cca 8 podadresářů s několika málo soubory. Z nich jsou de facto podstatné jen velmi dobře dokumentovaný `texuser.set` a dávkový soubor `4tex.bat`.[1] Po jeho spuštění se ocitneme v prostředí, které nám umožňuje spouštět editor, TEX, prohlížeč `.dvi` souborů, ovladače tiskáren atd. Jde tedy o obdobu MNU, TEXShellu apod. Toto prostředí je stabilní a představuje možnou alternativu k výše zmíněným. Kromě několika drobností (při tisku na 24-jehličkové tiskárně se nabízelo spuštění METAFONTu pro vygenerování fontu v rozlišení 518,4 dpi, když ovladač nalezl jen tento v resoluci 518 dpi; tisk přes Ghostscript neproběhl na laserové ani na jehličkové tiskárně korektně) jsem mohl konstatovat, že vše funguje v zásadě tak, jak si to uživatel asi představuje. Za zvlášť zdařilý považuji připravený výběr z TEXovských formátů (plain, LATEX, GreekTEX a Cyrillic TEX v různých kombinacích s NFSS a Babelem; LATEX2ε; verze „normální“ i big), ovladačů tiskáren (asi vše, co nabízí

[1] Zde je vhodné poznamenat něco o názvech. 4TEX je v dokumentaci název používaný v užším slova smyslu pro skupinu dávkových souborů, které vytvářejí uživatelsky přítulnější (nikoli nutně vždy produktivnější) prostředí pro TEXování, než je příkazová řádka. Šířeji je výraz 4TEX používán jako synonymum pro 4allTEX, tj. celý distribuční balík. Číslice 4 v názvu pochází z `4dos.com`, což je sharewarový program, který nabízí – podle mně dostupných informací – více možností než dosovský `command.com`. Program `4dos.com` je na CD také přítomen a je 4TEXem požadován.

Mattesův `dvidrv` ve verzi 1.5a a DVIPS – včetně přípravy souboru pro osvit) a konverzních utilit (pro text i grafiku).

TEXování s CD

Zde se dostávám již do obecnější roviny, než je informace o 4allTEXu. Nejprve musím mluvit o faktu, který mne při psaní tohoto článku nejvíce překvapil – a to velmi nepříjemně. Jde o to, že CD mechanika s přenosovou rychlostí 150 kB/s (single speed) je nevhodná pro práci takového typu, kterou při TEXování běžně provádíme (smyčka editor – TEX – prohlížeč). Rychlost přístupu a přenosu dat je na takovéto mechanice příliš nízká, než aby bylo možné mluvit o produktivní práci. Bohužel jsem neměl příležitost vyzkoušet práci s CD mechanikou typu double speed nebo ještě rychlejší, ale předpokládám, že výsledný dojem by byl zásadně odlišný; nicméně ani takové mechaniky nemohou dosud v rychlosti přenosu dat a zvláště v době přístupu soutěžit s jen trochu rozumným pevným diskem.

Chceme-li tedy využívat CD typu 4allTEX při běžné práci (nejen jako softwarový archív), ukazuje se jako rozumné nainstalovat často užívané programy a data (TEXovské formáty, které běžně užíváme; fonty pro prohlížeč `.dvi` souborů) na pevný disk a na CD zanechat soubory méně frekventované (fonty pro tisk – zvláště jestliže střídáme výstup na tiskárny s různým rozlišením, METAFONT, konverzní utility apod.). Při takovém uspořádání nás nebude CD-ROM omezovat z hlediska rychlosti a přitom budeme mít možnost využívat všechna dobrodiní vyplývající z používání této technologie.

Tím se dostávám k tomu, co nám spolupráce s CD přináší. Krátce řečeno, největší výhodou je kapacita tohoto média (řádově 650 MB), která umožňuje, aby obsahovalo takové množství TEXovských a paraTEXovských programů[2] a dat – to se týká zejména fontů pro různá výstupní zařízení od jehličkových tiskáren po osvitové jednotky.[3] Jestliže k tomu připočteme finanční dostupnost CD mechanik (dnes i mechaniku typu double speed lze pořídit okolo 5 000,– Kč) a fakt, že komerční soft-

[2] Včetně různých verzí téhož. Jako příklad z vlastní zkušenosti mohu uvést program DVIPS, který mám v současnosti na disku a na disketách ve 4 nebo 5 verzích (starší osvědčené – aktuálně používané – nejnovější opatrně testované). Obdobných příkladů by se dalo najít asi i více – např. beta verze emTEXu

[3] Možnost generovat potřebné fonty on line prostřednictvím `MFjobu` je velká výhoda, ale ne vždy to pravé ořechové.

ware je ve stále větší míře dodáván v CD verzích (které jsou navíc obvykle výrazně levnější – o 10 až 20 % – než varianty na disketách), plyne z toho závěr, ke kterému se snažím dostat.

Domnívám se, že nastala doba, kdy bychom se měli i v rámci CSTUGu snažit o to, aby CSTEX – již třeba verze 1995 – byl dostupný na CD. Je samozřejmé, že snaha o dosažení tohoto výsledku přinese obrovské množství práce – hledáním v archívech počínaje a konče vytvořením vhodné struktury dat (aby byla využita potenciální kapacita média a přitom nedošlo k zahlcení a následnému znechucení uživatele). Nicméně si myslím, že doba je zralá a není důvod se tímto problém nezačít zabývat.

Závěr

V polovině července, když píšu tento článek, je již možná k dispozici (podle dohody na setkání) 7 kusů CD s 4allTEXem členům CSTUGu v obvyklých TEXovských centrech (kolektivní členové objednávali pro svou potřebu asi 10 exemplářů). Cena jednoho kusu je 35 USD, při větších objednávkách 25 USD.

Protože jsem během prázdnin nebyl příliš komunikativní, odložil jsem nakonec vyřízení objednávky na zářijový EuroTEX, neboť bylo známo, že tam bude disk s 4allTEXem součástí nabídky. Ukázalo se, že je na světě už další vydání, takže jsem se promptně rozhodl pro odběr 20 dalších disků, na které mi holandští přátelé vystavili potřebnou fakturu. Ale teprve během listopadu se nám podařilo překonat obtíže s její úhradou, takže podle dohody s vydavateli by už pomalu měly zaplacené CD ROMy dorazit. V době, kdy čtete tuto zprávu, by už snad mohly být i na cestě k těm, kdož o ně na červnové valné hromadě projevili zájem.

Karel Horák

Z obsahu příštího čísla

Philip Taylor: Pragmatický prístup k odstavcom
Karel Horák: Ghostscript versus ps2pk
Ohlasy EuroTEXu

Výroční cena CSTUGu

za rok 1993

byla na základě návrhů z řad členů
přiznána

Jiřímu Veselému
za jeho podíl na šíření TEXu a na vzniku CSTUGu

a **tvůrcům balíku CSTEXu 1993**,
jehož hlavou byl a doposud je Petr Olšák.

Dle rozhodnutí výboru bude cena rozdělena
v poměru 1 : 4.

Vydalo:	Československé sdružení uživatelů TEXu vlastním nákladem jako interní publikaci
Obálka:	Bohumil Bednář
Počet výtisků:	650
Tisk a distribuce:	KOPP, Máchova 16, 370 01 České Budějovice
Adresa:	CSTUG, MÚ UK, Sokolovská 83, 186 00 Praha 8

Podávání novinových zásilek povoleno Oblastní správou pošt
v Českých Budějovicích, j.zn. P 3.202/94 ze dne 19. července 1994