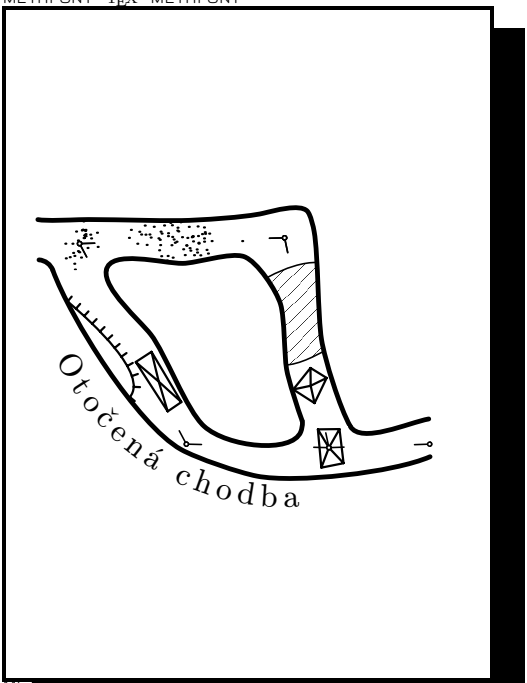
TEX METAFONT



4

ZPRAVODAJ 99

ČESKOSLOVENSKÉHO SDRUŽENÍ UŽIVATELŮ TEXU

ISSN 1211-6661 Ročník 9

OBSAH

Zpravodaj Československého sdružení uživatelů TEXu je vydáván v tištěné podobě a distribuován zdarma členům sdružení. Po uplynutí dvanácti měsíců od tištěného vydání je poskytován v elektronické podobě (PDF) ve veřejně přístupném archívu dostupném přes `http://www.cstug.cz`.

Své příspěvky do Zpravodaje můžete zasílat v elektronické podobě anonymním ftp na `ftp.icpf.cas.cz` do adresáře `/wagner/incoming/`, nejlépe jako jeden archivní soubor (`.zip, .arj, .tar.gz`). Současně zašlete elektronickou poštou upozornění na `mailto:bulletin@cstug.cz`. Uvedený adresář je pro vás „write/only“. Pokud nemáte přístup na Internet, můžete zaslat příspěvek na disketě na adresu:

Zdeněk Wagner
Vinohradská 114
130 00 Praha 3

Disketu formátujte nejlépe pro DOS, formát Macintosh 1.44 MB je též přijatelný. Nezapomeňte přiložit všechny soubory, které dokument načítá (s výjimkou standardních součástí $\mathcal{CS}$TEXu), zejména v případě, kdy vás nelze kontaktovat e-mailem.

ISSN 1211-6661

$\mathcal{NTS}$ a interaktivní dokumenty na Fakultě informatiky

David Antoš

V sobotu 11. prosince 1999 pořádalo Československé sdružení uživatelů TEXu CSTUG a Fakulta informatiky Masarykovy univerzity v Brně v prostorách fakulty přednášky o systému $\mathcal{NTS}$. $\mathcal{NTS}$ (New Typesetting System) je připravovaný následník TEXu.

Přednášejícími byli Hans Hagen, vedoucí a koordinátor projektu $\mathcal{NTS}$ a Karel Skoupý, programátor, který se v současné době práci na $\mathcal{NTS}$ plně věnuje.

V prvním bloku sobotního dopoledne informoval Karel Skoupý o postupech, stavu a problémech implementace $\mathcal{NTS}$. Idea vznikla před třemi lety a s vlastní implementací se začalo před rokem. Není bez zajímavosti, že Karel Skoupý je jediným programátorem projektu. První fází, kterou autoři zamýšlejí dokončit do poloviny roku 2000, je systém plně kompatibilní s TEXem.

Pro implementaci byl zvolen jazyk Java. Patří totiž mezi nejlépe přenositelné programovací jazyky vůbec a navíc se realizačnímu týmu zalíbil javovský systém balíčků objektů. TEX je program více než dvacet let starý a ačkoli byl vytvářen podle nejlepších postupů své doby, je to rozsáhlý monolit strukturovaného kódu.

TEX byl tedy „rozebrán“ na části a ty jsou postupně přepisovány v Javě přísně objektově. Za hlavní problém označil Karel Skoupý existenci velkého množství závislostí mezi jednotlivými částmi TEXu, které jsou přirozeným důsledkem jeho architektury, zato se však nesmírně obtížně modelují pomocí objektů. Ačkoli TEX patří mezi nejlépe dokumentované programy, TEXbook popisuje jeho činnost pouze ve standardních situacích. Pro dosažení úplné kompatibility je třeba provádět mnoho pokusů, které odhalují jeho chování v neobvyklých podmínkách.

$\mathcal{NTS}$ v současné době zvládá expanzi maker, odstavcový zlom a výstup, což dostačuje k tomu, aby nám byla jeho funkce demonstrována. Karel Skoupý pomocí $\mathcal{NTS}$ zalomil tři odstavečky textu, pak totéž provedl TEXem a výsledky porovnal. Nejen že byly stejné napohled, ale výsledné `dvi` soubory se shodovaly do posledního bitu.

Určitě čtenáře napadá otázka, zda takové úsilí stojí za to, aby vznikl „TEX II“, jenže objektový, v Javě a, jak jsme viděli, také mnohokrát pomalejší. Odpověď poskytla přednáška Hanse Hagena pod názvem

What things do we want $\mathcal{NTS}$ to do?

$\mathcal{NTS}$ není prvním následníkem TEXu. Jmenujme třeba ε-TEX, nebo známější pdfTEX. Všechny dosavadní deriváty TEXu stojí na původním Knuthově kódu, ve kterém je část přepsána či upravena. To není jednoduché, zdrojový kód obsahuje mnoho velmi účinných optimalizací, které ovšem vedou k naprosté nepřehlednosti. Počet lidí na světě, kteří se v `tex.web` skutečně vyznají, se odhaduje na několik desítek.

Cílem zvolené architektury $\mathcal{NTS}$ je maximální modularita. Tím se autoři projektu snaží dát prostor programátorům, kteří by chtěli přepracovat a vylepšit část systému. Počítá se s tím, že se objeví nové výstupní formáty, jako se to třeba nedávno stalo s PDF, takže by mělo být umožněno zpracovávat jiné vstupy, než ty, na které jsou TEXisté zvyklí, například SGML. Ačkoli určitě není pochyb o tom, že kupříkladu řádkový zlom TEXu patří k nejlepším, musíme si také přiznat, že i TEX má své slabiny. Sazba na rejstřík, vícesloupcová sazba se zarovnáním nebo třeba obtékání obrázků jsou sice úkoly řešitelné, ale nepříliš pohodlně, přesněji řečeno velmi obtížně.

Tyto problémy by postupně měly řešit další verze $\mathcal{NTS}$, také iniciativě programátorů mimo zatím poněkud uzavřený tým se meze nekladou. Kód bude volně dostupný a $\mathcal{NTS}$ bude šířen podobně jako TEX. Předpokládá se, že jeho jednotlivé moduly budou moci být využity i samostatně, například váš WWW browser si přečte zdrojový kód dokumentu třeba v LATEXu a zobrazí jej pomocí $\mathcal{NTS}$ jako typograficky krásné dílo.

Mezi další kroky patří integrace $\mathcal{NTS}$ a METAFONTu. Vychází se z toho, že TEX dnes vůbec nebere v úvahu tvary znaků, které sází. Prostě jen skládá boxy a „neví“, jestli znak třeba box nepřesahuje. Autoři rovněž uvažují o vytvoření interaktivního prostředí, ve kterém by bylo možno (slovy Hanse Hagena) „sedět s rukama na řídících pákách“, měnit parametry zlomu a sledovat, co se s dokumentem děje.

$\mathcal{NTS}$ je velmi zajímavý projekt zaměřený do blízké i vzdálenější budoucnosti. Nám tedy nezbývá, než celému týmu popřát mnoho úspěchů.

Tvorba interaktivních dokumentů

Hans Hagen vedl na půdě Fakulty informatiky v pozdním odpoledni 13. 12. 1999 opět pod záštitou CSTUGu a Fakulty informatiky čtyřhodinový tutoriál s názvem Advanced Interactive Documents. Jak již pravila pozvánka, TEX je jeden z mála programů schopných vytvářet sofistikované dokumenty v PDF. Chtělo by se dodat, že zvláště tehdy, když se TEX a PDF spojí s Hansem Hagenem, což dokazovaly jeho lehce avatgardně navržené a zpracované prezentace.

Pan Hagen rozebral v úvodu výhody a nevýhody formátu PDF jako prostředku pro přenos informací a dále se věnoval problému, kdy (ne)použít TEX.

TEX se ukazuje jako velmi vhodný prostředek pro kvalitní typografické zpracování dokumentů pravidelné struktury, jejichž sazbu lze dobře algoritmizovat.

Dále byly rozebrány základní charakteristiky (před několika lety se u nás říkalo „špecifiká“) interaktivních dokumentů. Je třeba počítat s omezeními, která přináší použitá technologie a na druhé straně je škoda nevyužít možností. Pak lze těžko rozlišit, co je ještě dokument a co je program. Představte si zkoušku z angličtiny, při které vám PDFko nabízí možnosti doplňování slov, a když slovo vyberete, je do dokumentu nejen vloženo (to by nebylo nic moc), ale odstavec je vysazen, takže vypadá zcela dokonale, i když jste vybrali nejméně dobrou ze všech možností.

Zbytek tutoriálu byl věnován tématům důležitosti strukturovaného a flexibilního kódování dokumentů a tomu, jak si můžeme takové dokumenty navrhnout a vytvořit sami.

Hans Hagen si vzal za cíl touto přednáškou ukázat, že uživatelé TEXu se mohou dostat až na hranice možností tvorby dokumentů. Myslím, že se mu to podařilo, stejně jako se podařilo CSTUGu připravit dva dny plné zajímavých informací.

TEXLive4 pod Windows so slovenčinou a češtinou

Janka Chlebíková

V Spravodaji CSTUGu 1-2/99 [2] bol uvedený kompletný preklad dokumentácie k CD-ROMu TEXLive4. Používatelia platformy Windows boli však pravdepodobne zaskočení rozdielom medzi dokumentáciou a skutočným obsahom CD-ROMu . Zmena bola totiž robená na poslednú chvíľu a informáciu o nej sme sa dozvedeli až z Tugboatu 20(1), ktoré došlo spolu s CD-ROMom! Ospravedlňujeme sa čitateľom za tento nesúlad a uverejňujeme plné znenie prekladu manuálu z [1] a to časti týkajúcej sa Windows. Tento preklad je rozšírený o praktické rady, ako odstrániť niektoré známe chyby inštalácie, resp. ako doinštalovať CSTEX, $\mathcal{AMS}$-TEX a iné veci hlavne v prostredí programu `WinEdt`. Záver patrí základným informáciam o národných štýloch `slovak.sty` a `czech.sty`.

Začíname teda prekladom oficiálneho manuálu doplneného o drobné poznámky.

Inštalácia a používanie pod Windows

Táto časť sa vzťahuje iba na systémy s Windows 9x alebo NT. Ak používate Windows 3.1, je potrebné si nainštalovať ručne `emtex` z hlavného adresára `systems`.

Je tiež dôležité mať Windows nastavené tak, že používajú 'Microsoft Joliet extensions' na čítanie z CD-ROM mechaník. Na overenie si prezrite pomocou programu `Prieskumník` (*Windows-Explorer*) CD-ROM, či zobrazuje dlhé názvy súborov s malými a veľkými písmenami zároveň. Ak nie, nemôžte spúšťať samostatne bežiace programy z CD-ROMu.

Čo je fpTEX?

Systém pre Windows nachádzajúci sa na CD-ROMe je fpTEX od Fabrice Popineau. Tento vznikol konverziou známej distribúcie teTEXu z Unixu pre systémy Win9x a NT — označované ako Win32. Presnejšie, vzhľadom na rozdiely medzi platformami Unix a Win32 niektoré funkcie sa správajú rozdielne pod fpTEXom, niektoré ešte chýbajú, ďalšie sú úplne rozdielne. Vo väčšine prípadov sa však správajú rovnako ako pod Unixom.

Čo obsahuje táto konverzia?

Verzia pod Win32 na TEXLive CD-ROMe zahŕňa rovnaké programy ako verzia pre Unix spolu s niektorými ďalšími programami, ktoré môžu byť nainštalované.

Medzi programami sa nachádza niekoľko DLL súborov:

- súbory začínajúce na msvc sú knižnice pre Microsoft C, ktoré používajú viacvláknové (*multi-thread*) aplikácie,
- dynamicky linkovaná knižnica Kpathsea,
- `zlib.dll` (knižnica pre kompresiu) a `libpng.dll` (*Portable network graphics*) pre pdfTEX a niekoľko ďalších programov,
- `tex.dll`, `pdftex.dll` a niekoľko ďalších vzťahujúcich sa k rôznym TEXovým nadstavbám – viď vysvetlenie nižšie.

Všetky rôzne TEXové nadstavby sa distribujú vo forme `.dll` súborov – pre základné funkcie (*core*) a `.exe` pre proces, ktorý komunikuje s používateľom a zároveň odovzdáva jeho požiadavky na spracovanie (*front-end*). Tým je daná odpoveď na problémy linkovania súborov, ktoré neexistujú na Win32 platforme. Napríklad štandardný TEX sa skladá z nasledujúcich súborov:

```
11/19/98 11:07a 217,088 tex.dll
11/19/98 11:07a  16,384 tex.exe
```

`Latex.exe` nie je teda nič iné, len (približná) kópia súboru `tex.exe` používajúca rovnakú knižnicu `tex.dll`. Podobným spôsobom je riešená skupina programov `mktex*.exe`, ktoré používajú knižnicu `mktex.dll`.

Spúšťanie z CD-ROMu
Všetky TEX programy môžte spustiť priamo z CD-ROMu, zároveň máte prístup ku všetkým makrám a fontom, avšak za cenu nižšej rýchlosti ako pri inštalácii na pevný disk. Pred spúšťaním priamo z CD-ROMu musíte pridať cestu k adresáru `bin\win32` na CD-ROMe do premennej `PATH` použitím konfiguračných utilít vo Windows. Teraz môžte spúšťať programy z príkazového riadku alebo použitím sharewarového programu `WinEdt`, ktorý ich umožňuje spúšťať pomocou prehľadnej ponuky.

Inštalácia
Inštalačný program by sa mal spustiť automaticky po vložení CD-ROMu TEXLive do mechaniky. V prípade, že sa tak nestane, spustite program `autorun.exe`. Potom postupujte podľa inštrukcií. Tu sú uvedené základné z nich:

- Vyberte si *hlavný* (*root*) adresár, do ktorého chcete inštalovať TEX, štandardne je nastavený adresár `c:\TeX`. Môžte si ho samozrejme zmeniť, pretože potrebujete veľa priestoru na disku – viac ako 300 MB pre plnú inštaláciu, ale v súvislosti s veľkosťou 'clusterov' na FAT partícií sa môže veľkosť inštalácie ešte zväčšiť.
- Nepoužívajte v názve cesty medzery, TEX ich nemá v obľube. Každopádne `setup.exe` kontroluje ich prítomnosť v špecifikácii cesty.
- Váš *hlavný* (*main*) `texmf` strom bude `<root>/texmf` a označený je premennou `$TEXMFMAIN`.
- Navyše máte možnosť pridať si viac `texmf` stromov:
  – *lokálny* (*local*) `texmf` strom, ktorého umiestnenie je uložené v premennej `$TEXMFLOCAL` a je štandardne nastavený na `<root>/texmf.local`. Predpokladá sa, že v ňom budú uložené všetky lokálne makrá, štýly a tiež lokálne generované fonty. *Ak nešpecifikujete lokálny* `texmf` *strom, budete vyzvaný nastaviť hodnotu premennej* `$VARTEXFONT` *na nejaký adresár, kde budú ukladané lokálne generované fonty. Ak špecifikujete lokálny* `texmf` *strom, tento bude použitý pre generované fonty.*
  – *osobný* (*home*) `texmf` strom, ktorého umiestnenie je uložené v premennej `$HOMETEXMF` a jeho štandardná hodnota je `$HOME/texmf`. Toto nastavenie je zmysluplné len pod Windows NT, kde každý používateľ má nastavenú premennú `$HOME`. Používatelia Windows 9x ju zvyčajne nemajú nastavenú a preto by nemali vypĺňať túto položku.
  Horeuvedené cesty možno tiež nastaviť priamo v súbore `texmf/web2c/texmf.cnf` editovaním hodnôt príslušných premenných.
- Inštalačný program sa vás opýta, či chcete nainštalovať:
  – PK fonty,
  (Poznámka: ak si nezaškrtnete túto voľbu, ušetríte si miesto na disku a PK fonty sa vám budú dogenerovávať podľa použitia.)

– iba voľne šíriteľné balíky programov (v tomto prípade sa neobjavia niektoré balíky zo zoznamu pri ďalšom výbere voliteľnej (*custom*) inštalácie),
– dokumentáciu ku každému balíku, ktorý je nainštalovaný; príručky a všeobecná dokumentácia je vždy inštalovaná, ale nie všetky materiály k nejakým špeciálnym balíkom,
(Poznámka: zaškrtnutím sa neinštaluje rozšírená dokumentácia!)
– možnosť inštalovať zdrojové súbory.
(Poznámka: zaškrtnutím sa neinštalujú!)

- Ďalej si vyberáte typ inštalácie z nasledujúcich možností: *základná* (*basic*), *doporučená* (*recommended*), *plná* (*full*). Ak už viete, ktoré balíky chcete mať nainštalované, môžte si vybrať *voliteľnú* (*custom*) inštaláciu. V tom prípade sa zobrazí zoznam skupín balíkov, ktoré možno nainštalovať. Každý balík možno samostatne odznačiť (resp. označiť). Pri väčšine sa nachádzajú krátke popisy, ktoré sú prevzaté z Web katalógu od Graham Williamsa.
(Poznámka: Po výbere typu inštalácie treba chvíľu počkať, kým sa inicializuje zoznam inštalovaných súborov. Ak si vyberiete *doporučenú* inštaláciu, nenainštalujú sa vám napríklad aj nejaké fonty. Pokiaľ teda radi s TEXom experimentujete, doporučujeme *full* inštaláciu napríklad bez PK fontov.)
- Ignorujte položku 'setup' v zozname programov, ktoré možno nainštalovať. Vzťahuje sa na programy v `setup32` na CD-ROMe a nemala by byť viditeľná.
- Ďalej máte možnosť pridať si k základnej inštalácii niektoré špeciálne programy. Konkrétne `Ghostscript` a `Ghostview` – postsciptový interpreter; prehliadač `ImageMagick`, ktorý umožňuje pracovať s obrázkami a konverziami (využíva `TeX4ht`); `WinEdt`, príjemné, shareware prostredie pre prácu s TEXom podobne ako `texshell`, ktorý má podobné funkcie, naviac je voľne šíriteľný (ale menej populárny).
(Poznámka: Pri výbere `Ghostview` alebo `ImageMagick` inštálacia nemusí vždy prebehnúť, viď sekciu .)
- Môžete si prezrieť nastavenia inštalácie (t.j. čo sa bude inštalovať) a ak je všetko v poriadku, začnú sa kopírovať súbory na disk.
- Po skončení sa spustia inštalačné programy pre `Ghostview` a `WinEdt`, ak ste zvolili ich inštaláciu. Inštalácia a konfigurácia všetkých ostatných vecí je uskutočnená automaticky cez `setup.exe`, vrátane zostavenia `ls-R` súborov.
- Po skončení inštalácie sa zobrazí dokumentácia pomocou Web prehliadača, ktorý je štandardne nastavený. Používatelia Windows 9x musia reštartovať systém pred prvým použitím nainštalovaných programov.
(Poznámka: Rozumnejšie je reštartovať systém aj pod Windows NT!)

V ponuke `Start->Programs->TexLive` sa objavia odkazy na viaceré inštalované programy.

V čom spočíva činnosť setupu?

Ak chcete zmeniť nastavenie bez setup programu, v tejto časti sa nachádza detailnejší popis, čo vlastne `setup` robí a čo nie. Zmení sa nastavenie premennej `PATH`, aby bolo možné spúšťať nainštalované programy. Overí sa, či neobsahuje odkazy na staršie verzie fpTEXu alebo TEXLive a v prípade existencie, je odkaz odstránený. Deje sa to pomocou vyhľadávania súboru `kpathsea.dll` cez jednotlivé odkazy v `PATH`.

Ak sa inštaloval `Ghostscript`, do premennej `PATH` sa vloží cesta k adresáru `gs5.50`, pretože súbory `gswin32c.exe` (prostredie `Ghostscriptu`), `gsdll32.dll` (`dll` súbor potrebný pre `Ghostscript`) využívajú viaceré programy TEXLive. `Ghostscript` používa Windows registre (verzia 5.50), preto nie je potrebné nastavovať žiadne iné premenné, aby našiel potrebné súbory. Predchádzajúca alebo voliteľná inštalácia môže vyžadovať nastavenie premennej `GS_LIB`. Viac v príslušnej dokumentácii.

`ImageMagick` je tiež pridaný do `PATH`, ak ste zvolili jeho inštaláciu. Naviac súbor `delegates.mk` je nakopírovaný podľa typu platformy (Windows NT alebo Windows 9x). Viac v príslušnej dokumentácii k programu.

Hlavný `<root>/texmf/web2c/texmf.cnf` súbor sa zmení podľa `texmf` stromov, ktoré ste špecifikovali a miesta pre uloženie lokálne generovaných fontov. Premenné `$TEXMF`, `$TEXMFLOCAL`, `$HOMETEXMF` a `$VARTEXFONTS` sú patrične zmenené.

V súbore `<root>/texmf/web2c/mktex.cnf` je pridaná možnosť *varfonts*, ktorá spôsobí, že každý lokálne generovaný font je uložený v `$VARTEXFONTS`.

Konfigurácia pre `tex4ht` sa vykoná zmenou súboru `<root>/texmf/tex4ht/base`. Ďalej je dôležité spustiť utilitu `convert.exe` z `ImageMagick`, aby sa aktualizovala cesta.

Testovanie inštalácie

Dôležitý nástroj na otestovanie inštalácie je program `kpsewhich`.

Najprv by ste mali overiť, či Web2c správne identifikuje cestu k stromu `texmf`. V príkazovom riadku napíšte:

```
kpsewhich -expand-path=$TEXMF
```

Odpoveďou by mala byť cesta k `texmf` stromu (napr. `c:/TeX/texmf`, ak ste inštalovali súbory ako v predchádzajúcom príklade. Ale drobné upozornenie, odpoveď je v štýle Unixu t.j. MS-DOS štýl \ je nahradený /. Nemusíte sa však obávať zlého nastavenia.) Ak vaša cesta k `texmf` je iná ako odpoveď programu,

pravdepodobne ste zmenili štandardnú adresárovú štruktúru. V takomto prípade zmeňte ručne nastavenie premennej `$TEXMF` na správnu hodnotu.

Ak si chcete byť naozaj istý, napíšte `mktexlsr`, ktorý obnoví `ls-R` databázu, aj keď `ls-R` súbor by mal byť vygenerovaný až po inštalácii.

Sieťová inštalácia a súborové systémy

Všetky dodávané súbory okrem súborov v `bin/win32` sú používané aj inštaláciou na Unixe. Preto možno použiť program `Samba` buď na pripojenie sa k Windows NT serveru z Unixovskej pracovnej stanice alebo naopak. K dispozícii je niekoľko možností:

- Dajte všetky súbory na server. Potom jednoducho pridajte všetky používané súbory pre vybranú platformu a architektúru do adresáru `bin`. To znamená napríklad `bin/win32`, `bin/i386-linux`, atď.
- Nainštalujte lokálnu kópiu binárnych a formátových súborov. V tomto prípade nastavte `$TEXMFMAIN` na hlavný `texmf` strom umiestnený na sieti.

Tento postup by mal zabezpečovať `InstallShield`, ale vyskytlo sa množstvo problémov s ním, preto sa tieto možnosti zastavili v ďalších verziách setupu.

Win32 podporuje niekoľko druhov systémov súborov:

- MS-DOS FAT, 8.3 a veľké písmená pre názvy súborov,
- 'Protected' mód FAT, dlhé názvy súborov, veľkosť písmen nie je podstatná,
- NTFS, dlhé názvy súborov a veľkosť písmen je podstatná,
- ISO9660 CD-ROM, 8.3 a veľké písmená pre názvy súborov.

Naviac vo volaní súborov Win32 nerozlišuje veľkosť písmen a ešte niekoľko ďalších špecifík NTFS, ktoré Win32 zatiaľ nie je schopné využívať. Ďalšia odlišnosť je v používaní iného oddeľovača adresárov / alebo \, ale Win32 akceptuje obidve možnosti.

Teda aké komplikácie môžu nastať?

Väčšinou budete používať štýly, ktoré budú uložené v súboroch s dlhými názvami. Ak používate systém, ktorý ich podporuje[1], nevyskytnú sa žiadne problémy a nemusíte nič špeciálne robiť. V opačnom prípade budete musieť používať aliasy z Kpathsea. Predpokladajme, napríklad, že sa snažíte používať `texmf` na FAT a máte štýl uložený v súbore `longtable.sty`. Názov je skrátený na tvar 8.3 t.j. `longtabl.sty`. V tomto prípade musíte vytvoriť súbor s menom `aliases` na rovnakom mieste, ako je uložený súbor `ls-R` vo vašom `texmf` strome. Súbor by mal obsahovať riadok:

`longtabl.sty longtable.sty`.

Všetky odkazy na `longtable.sty` sa presmerujú na súbor `longtabl.sty` v prípade, že sa nenájde súbor s dlhým názvom.

[1] Napr. NTFS ale nie FAT!

Ak aj naďalej si myslíte, že problémy nastávajú s názvami súborov, skúste ešte overiť nasledujúce:

- cesty v konfiguračných súboroch by mali byť uvedené s / namiesto \,
- odkazy v databáze `ls-R` musia byť malými písmenami, aj keď spúšťate z FAT alebo CD-ROMu,
- použite debugovacie črty Kpathsea a `kpsewhich` na odhalenie problému a pošlite e-mailom na adresu TEXLive4 výsledky vášho skúmania.

Toľko teda preklad oficiálneho manuálu, ktorý vznikol za pomoci Mareka Hyčka, študenta MFF UK v Bratislave. V ďalšej časti nasledujú základné postrehy a rady k inštalácii TEXLive4 pod Win32.

Komentáre k inštalácii TEXLive4 pod Win32

Je nutné podotknúť, že inštalácia sa správa rôzne pod Windows 95, 98, NT, ... Preto, ak sa nejaké chyby u vás neprejavili, nič mimoriadne sa nedeje.

Uvádzame zoznam aspoň základných chýb:

- Inštalácia pod Win98 sa za istých okolností skončila neúspešne pri inštalovaní `ImageMagick` (nutný násilný BREAK celého systému). Prejavilo sa to napríklad pri 'Full' inštalácii s voľbou doinštalovať všetky ponúkané veci typu `Ghostview`, `WinEdt`, `ImageMagick` a `TeXShell`. `ImageMagick` je využívaný okrem iného i programom `TeX4ht` pre generovanie HTML stránok, preto je dôležité jeho korektné fungovanie.
  `ImageMagick` možno doinštalovať (a dobehnúť inštaláciu), ak si vyberiete 'Custom' inštaláciu a pri danej voľbe si nevyberiete žiadny z ponúkaných balíkov. Z ďalšej ponuky si vyberiete doinštalovať `ImageMagick` a ďalšie programy podľa vlastnej potreby.
  Ďalej je nutné skonfigurovať `ImageMagick`:
  – skopírovať súbor z adresára `Imagick\delegates\win95.mgk` do súboru `Imagick\delegates.mgk`
  – ak sa vám v súbore `autoexec.bat` nenainštalovala cesta `gstools\gs5.50` pre program `gswin32`, treba skonfigurovať súbor `delegates.mgk` a nastaviť absolútnu cestu k danému programu
  – v súbore `autoexec.bat` treba nastaviť premennú `delegate_path` na adresár, kde je uložený súbor `delegates.mgk`, napr.:
  `SET delegate_path=d:\tex\Imagick`
- S chybovou hláškou (pod Win95/98) skončila aj inštalácia programu `WinEdt` (hláška: `Can't get winedt location!`), ak bol vybraný pri základnej inštalácii TEXLive4. Pre doinštalovanie je nutné spustiť z CD-ROMu samostatnú inštaláciu: `\setupw32\winedt\setup`.
  Prostredie `WinEdt` editoru nakonfigurujeme jednoducho na TEXLive4 inštaláciu prepísaním súboru `\WinEdt\winedt.ini` súborom `winedt.ini`

z adresára <CDROM>\setupw32\WinEdt\winedt-cfg. (Súbor treba prepísať, aj keď prebehla inštalácia WinEdt automaticky.)

- Ak chcete generovať akékoľvek nové TEXovské formáty (viď nasledujúcu kapitolu), je nutné vytvoriť adresár <root>\texmf-var\web2c.
  Ináč vaše snaženie skončí s hláškou typu:

```
fmtutil: format directory 'D:/TEX/texmf-var/web2c' does not exist.
```

- Niektoré BAT súbory z <root>\tex\bin\win32, napr. csplain.BAT kontrolujú existenciu formátu. Ak aj formát existuje, test (pod Win95/98) sa nikdy neskončil okay a program sa vždy snažil generovať formát.

```
@echo off
kpsewhich cslatex.fmt > nul
if "%ERRORLEVEL%"=="0" goto okay:
fmtutil --byfmt csamstex
:okay
tex -fmt=csplain -translate-file=cp1250cs %1 %2 %3 %4 ...
```

  Ak ste pred spustením takého BAT súboru nevykonali ani predchádzajúci bod, ďaleko sa nedostanete...
  Za predpokladu, že dané formáty sú už vytvorené stačí len zapoznámkovať príslušné riadky:

```
@echo off
rem kpsewhich cslatex.fmt > nul
rem if "%ERRORLEVEL%"=="0" goto okay:
rem fmtutil --byfmt csamstex
rem :okay
tex -fmt=csplain -translate-file=cp1250cs %1 %2 %3 %4 %5 ...
```

- V súbore texmf.cnf bola chyba pri nastavení adresárov pre pdfcs∗ formáty na prehľadávanie – namiesto : má byť ; – správna hodnota premenných má teda vyzerať:

```
TEXINPUTS.pdfcslatex = .;$TEXMF/{pdftex,tex}/{cslatex,csplain,
                       latex,generic,}//
TEXINPUTS.pdfcsplain = .;$TEXMF/{pdftex,tex}/{csplain,plain,
                       generic,}//
```

- Pre korektné fungovanie MetaPostu je dobré nahradiť niektoré súbory v <root>\tex\bin\win32 novšími verziami (tl4-upd-bin-win32.zip), ktoré sú dostupné z http://www.tug.org/texlive.
- Windvi pri generovaní istého druhu PK-fontov (presnejšie tých, pre ktoré má k dispozícii pfb formáty) potrebuje program gsview32c.exe. Preto pokiaľ nemáte v súbore autoexec.bat v premennej PATH nastavený aj adresár <..>\gstools\gs5.50, je potrebné umiestniť súbor gsview32c.exe do nejakého adresáru, v ktorom bude externe volaným programom nájdený, napríklad <..>\gstools\gsview, ktorý býva štandardne nastavený pri inštalácii Ghostview.

- Pokiaľ chcete využívať program `tex4ht`, je treba opraviť konce riadkov v súbore `<root>\tex\bin\win32\ht.bat`. Pre korektné generovanie HTML stránok s akcentovanými znakmi je nutné vytvoriť príslušné cs*.htf súbory (niektoré sú dostupné z www.cstug.sk). Ďalej je treba modifikovať v súbore `<root>\tex\texmf\tex4ht\base\tex4ht.env` volanie parametra transparency s nasledujúcou hodnotou:
  ```
  Gconvert -crop 0x0 -density 110x110 -transparency #FFFFFF tex4ht.ps %%3
  ```
  V uvedenom riadku je tiež volaný (bez absolútnej cesty) program `convert` z balíku ImageMagick. Ak sa chcete vyhnúť zbytočným problémom, je lepšie volanie programu `convert` doplniť s absolútnou cestou. Mimochodom, v Delphi5 sa tiež nachádza program `convert`...
- Štandardný DVI prehliadač `Windvi` sa niekedy správal nekorektne a je vidieť, že je ešte stále vo vývoji (viď tiež [3]). Ostáva tak dúfať, že v nasledujúcej verzii tohto prehliadača budú niektoré závažné chyby odstránené. Možnosťou je nainštalovať a skonfigurovať iný DVI prehliadač, resp. vypomôcť si cez `Ghostview`.

Z používateľského hľadiska podstatný rozdiel medzi dosiaľ rozšírenými verziami TEXu pod Windows (predovšetkým emTeXu) a TEXLive4 je v inej adresárovej štruktúre a hlavne v spôsobe vyhľadávania súborov. Ak v nejakom dokumente inklúdujete iný súbor (napríklad cez `\input ...` alebo `\usepackage{...}`), súbor je hľadaný v adresároch podľa nastavenia v `texmf.cnf` súbore v závislosti od použitého formátu. Pre urýchlenie vyhľadávania je možnosť vytvoriť `ls-R` súbor pre každý `texmf` strom a TEX bude prehľadávať len databázový súbor namiesto príslušného `texmf` stromu na disku. Krátky ilustračný príklad: pokiaľ pridáte nejaký súbor do stromu `$TEXMFMAIN`, tento nebude pri TEXovaní nájdený, pokiaľ nevygenerujete nový `ls-R` súbor!!!. O tom, ako generovať súbory `ls-R` bude krátka informácia neskôr, pre konkrétne nastavenia doporučujeme pozrieť sa do súboru(-ov) `texmf.cnf`.

Ak chcete prestavovať nastavené cesty na hľadanie súborov (súbor `<root>\texmf\web2c\texmf.cnf`), či generovať nové formáty (podľa súboru `<root>\texmf-var\web2c\fmtutil.cnf`), je dobré urobiť si ich lokálne verzie. Kópiu súboru `texmf.cnf` je možné umiestniť do adresára `$TEXMFLOCAL\web2c`[2] a `fmtutil.cnf` do `<root>\texmf-var\web2c` a spustiť pregenerovanie databáz príkazom `mktexlsr` (alebo `Accessories->Rebuild databases` z prostredia `Winedt`). Od tohoto okamihu budú brané do úvahy vaše lokálne nastavenia podľa daných súborov.

[2] Ak ste nemenili štandardné nastavenie lokálneho `texmf` stromu, premenná `$TEXMFLOCAL` je nastavená na hodnotu `<root>\texmf.local`, teda napríklad `c:\TEX\texmf.local`.

Ako pohodlne TEXovať s TEXLive4 pod Win32

Všetky nasledujúce rady budú viazané na prostredie `WinEdt` — shareware pracovný stôl pre TEX. `WinEdt` je pohodlným (programovateľným) otvoreným textovým editorom s veľmi silnou nadstavbou pre TEX. Umožňuje spúšťať priamo z prostredia rôzne aplikácie ako napríklad TEX, prehliadače, či prekladače. Má preddefinovaných niekoľko typov makier uľahčujúcich vkladanie do textu (napríklad pre rôzne prostredia, literatúru, rôzne konštrukcie), prehľadne vizualizované matematické symboly a iné prostriedky urýchľujúce písanie textu, či farebné odlišovanie textu pre riadiace slová. Okrem toho editor vie komunikovať s LOG súborom, umožňuje rýchlu navigáciu po sekciách, referenciách, citáciach, umožňuje 7-bitové načítanie a ukladanie súborov, ... Je vybavený aj kontextovým dopĺňaním slov podľa slovníka a kontrolou pravopisu, pričom pre jednotlivé jazyky je možné pridávať nové slovníky. Každý používateľ od úplneho začiatočníka až po pokročilého znalca editorov má možnosť dostatočne sa vyrealizovať v jeho nastaveniach.

V súčasnosti jeho podstatnou výhodou je fakt, že stačí zmeniť len jediný konfiguračný súbor a TEX spolu s všetkými aplikáciami ako prehliadače, generovanie chýbajúcich fontov, MetaPost a radou iných, ktoré sú dostupné na TEXlive4 sú už nakonfigurované. Toto istotne ocení hlavne široká TEXová verejnosť, ktorá nie vždy má dostatok času a trpezlivosti, aby nejaký ten TEX na svojom počítači skonfigurovala. Navyše, stačí zmeniť len pár súborov a môžte veľmi pohodlným spôsobom TEXovať aj v slovenčine pod Win32! Pravdou je, že pokiaľ chcete zmeniť konfiguráciu štandardne nastavenej tlačiarne pre generovanie fontov, štandardne nastavený DVI prehliadač alebo iné, musíte sa trochu zorientovať v nemalom množstve súborov. TEXLive, fpTEX a `WinEdt` sú však neustále vo vývoji a tak sa nechajme prekvapiť, čo prinesie nová verzia TEXLive5.

Výhodou celého prostredia `WinEdt` je skutočnosť, že je otvoreným systémom a dáva možnosť nastavenia na akékoľvek používateľove aplikácie, či makrá. Uvedené často neplatí v prípade TEXových distribúcií, v ktorých si používateľ síce môže pohodlne nastaviť konfigurácie tlačiarne, vybrať prehliadač, ale celý systém je uzavretý pre nové aplikácie (prehliadače, tlačiarne, ...).

Za pripomenutie však stojí, že `WinEdt` je shareware produkt a ak si nezakúpite licenciu, mesiac po inštalovaní sa k vám začne správať veľmi nepriateľsky (licencia pre 1 osobu stojí 40$, licencia pre školy na 50 počítačov stojí 500$).

Ako TEXovať po slovensky s TEXLive4 pod Win32 a s WinEdt?

V nasledujúcej kapitole popíšeme ako generovať a spúšťať nové formáty a tiež pár všeobecných rád a doporučených zmien pre slovenčinu.[3] Jednotlivé zmeny budú popisované z prostredia `WinEdt`, ale väčšina z nich má širšiu platnosť a líši sa len volaním príslušných programov.

Ako generovať nové formáty

Globálne pravidlá pre vytváranie nových formátov je možné zhrnúť do nasledujúcich bodov: aktualizovať súbor `<root>\texmf-var\web2c\fmtutil.cnf`, inicializačné súbory umiestniť do podadresára `$TEXMFLOCAL\tex\...`, spustiť vytváranie formátov `Accessories->Rebuild formats` a potom pregenerovať databázu príkazom `Accessories->Rebuild databases`. Adresár `<root>\texmf-var\web2c` je totiž prehľadávaný len podľa databázy `ls-R` a nie skutočne celý podstrom, resp. adresár na pevnom disku.

Nastavený spôsob prehľadávania adresárov sa dá zmeniť (najlepšie v lokálnom) v súbore `$TEXMFLOCAL\texmf.cnf` prestavením premennej `TEXMF` na hodnotu:

```
TEXMF={$HOMETEXMF,$TEXMFLOCAL,$VARTEXMF,!!$TEXMFMAIN}
```

a súčasne vyhodením už existujúceho súboru `ls-R` z daného podadresára.

Ako vytvoriť formát $\mathcal{AMS}$-TEX, pdf$\mathcal{AMS}$-TEX

- Vytvoriť v adresári `$TEXMFLOCAL\tex\pdftex` súbor `pdfamstex.ini`, ktorý vytvoríte z `<root>\texmf\pdftex\plain\config\pdftex.ini` pridaním riadku `\input amstex` hneď za riadok `\input bplain`.
- Modifikovať súbor `<root>\texmf\var-web2c\fmtutil.cnf` pridaním riadku
  `pdfamstex pdftex - pdfamstex.ini`
  a odpoznámkovaním riadku:
  `amstex tex - amstex.ini`
  (všetky ostatné riadky môžu zostať zapoznámkované).
- Spustiť vygerenovanie formátov `Accessories -> Rebuild formats` a pregenerovanie databázy `Accessories -> Rebuild databases`.

Ako vytvoriť $\mathcal{CSAMS}$-TEX, pdf$\mathcal{CSAMS}$-TEX

- Vytvoriť v adresári `$TEXMFLOCAL\tex\csamstex` súbor `csamstex.ini`,

[3] Až na pár výnimiek, ktoré sú jasné z kontextu, všetko ostáva v platnosti ak sa slovíčko slovenčina nahradí češtinou.

ktorý vytvoríte z `csplain.ini` pridaním riadku `\input amstex` hneď za riadok `\input plain`.

- Modifikovať súbor `<root>\texmf-var\web2c\fmtutil.cnf` pridaním riadkov
  ```
  csamstex   tex -       csamstex.ini
  pdfcsamstex   pdftex   -    csamstex.ini
  ```
  (všetky ostatné riadky môžu zostať zapoznámkované).
- Spustiť vygerenovanie formátov `Accessories -> Rebuild formats` a pregenerovanie databázy `Accessories -> Rebuild databases`.

Formáty $\mathcal{CS}$plainu, $\mathcal{CS}$LaTeXu a ich pdf-verzií sú už priamo vytvorené na CD--ROMe. Podľa horeuvedeného algoritmu sa však dajú vytvárať rôzne iné formáty, napr. $\mathcal{CS}$plain bez preddefinovanej veľkosti strany A4, ...

Navyše pre každý nový formát (alebo aj starý existujúci) je možné nastaviť (alebo prestaviť) cestu, kde sa majú hľadať súbory. Pre nové formáty to znamená pridanie nasledujúcich riadkov do súboru `$TEXMFLOCAL\web2c\texmf.cnf`

```
TEXINPUTS.csamstex = .;$TEXMF/tex/{csamstex,csplain,
                     plain,generic,}//
TEXINPUTS.pdfcsamstex = .;$TEXMF/{pdftex,tex}/{csamstex,csplain,
                     amstex,plain,generic,}//
```

Ako spustiť nové formáty cez WinEdt?

Najskôr treba vytvoriť príslušné BAT súbory, ktoré budú volať dané formáty. Existuje niekoľko možností, jedna z nich je vytvoriť príslušné BAT súbory v adresári `<root>\bin\win32`, pričom BAT súbory pre $\mathcal{CS}$plain, $\mathcal{CS}$LaTeX sú už vytvorené. BAT súbory pre verzie $\mathcal{AMS}$-TeXu sa dajú vytvoriť jednoduchou modifikáciou uvedených súborov.

Teraz už stačí len pridať volanie príslušných BAT súborov do menu `WinEdt` cez `Options -> Menu Setup -> Accessories`. (Najlepšie nastaviť sa na nejakú položku v 'Menu items', pravým tlačidlom myši skopírovať obsah položky, ktorý potom treba zmodifikovať).

Pár doporučených zmien pre slovenčinu

Pokiaľ chcete byť dôsledný a mať aj vlastné malé $\mathcal{CS}$-ikony, tu je krátky návod, ktoré súbory treba modifikovať:

- Naeditovať ikony pre jednotlivé varianty $\mathcal{CS}$TeXu (pdf.$\mathcal{AMS}$-TeXu) v adresári `\Winedt\Bitmaps\Buttons` a `\Winedt\Bitmaps\Images`. V súbore `\Winedt\Winedt.btn` podľa vlastného uváženia nahradiť nepotrebné čísla ikoniek ikonkami $\mathcal{CS}$TeXu, resp. pridať malé ikonky do súboru `\Winedt\Winedt.img`. (Všetky spomínané modifikované súbory a ikony možno nájsť na www.cstug.sk).

- Ikony potom pridávajte cez `Options -> Menu Setup -> Accessories` kliknutím na niektorú z ikoniek v položke 'Images and Hint'.

Pokiaľ sa vám nezobrazujú vo `WinEdt` korektne akcentované znaky, je treba prestaviť nasledujúce:

- Prepnúť kódovanie v `Options->Preferences->Font->Script` na 'Central European'.
- Zapnúť v `Options->Settings->Language->Options` voľbu '16-bit (Wide) Characters Enabled' pre korektné zobrazovanie akcentovaných znakov pri zapnutej kontrole pravopisu.

Ak chcete používať slovenský slovník na kontrolu gramatiky, je možné si ho stiahnuť z adresy www.cstug.sk. (Nie je dokonalý, ale určite lepšie ako žiadny.) Nový slovník sa pridáva sa cez `Options->Dictionary` a potom pravým tlačidlom myši cez položku 'Insert' máte možnosť pridať nový slovník (položkou 'Load' je možné realizovať jeho okamžité načítanie) a cez okno 'Enable' rôzne kombinovať slovníky. Viac možno nájsť v dokumentácii k editoru.

Čo je ešte dobré vedieť?

- Vo WinEdt je lepšie vypnúť 'Wrap' options v `Options->Preferences->Default`, pretože ináč je ignorovaný 'Enter' ako prechod do nového riadku.
- Ak chcete používať pri preklade `dvips` poscriptové fonty za predpokladu, že nepoužívate iné ako štandardné cs-cm fonty stačí odpoznámkovať nasledujúce riadky:
  ```
  p +csfontd.mapfile
  p +bsr.map
  ```
  v kópii súboru `<root>\texmf\dvips\config\config.ps`, ktorú je najvýhodnejšie umiestniť do `$TEXMFLOCAL\dvips\config\config.ps`. Pokiaľ budete používať `Ghostview` výhradne aj ako DVI prehliadač, takýmto postupom sa vám nebudú na pevnom disku zbytočne generovať rôzne bitmapové zväčšenie fontov. Predtým je však lepšie oboznámiť sa s komentármi o postscriptových fontoch z modifikovaného súboru `config.ps`.
  Poznámka: príkazom typu 'p +bsr.map' nesmie predchádzať v riadku žiadna medzera.
- Pokiaľ chcete nahradiť cm∗ fonty cs∗ fontami globálne pri použití nejakého balíka makier, napríklad v prípade štýlu `amsppt.sty` pri použití `csamstexu`, stačí pred volaním makier zavolať
  `\input csfonts`
  a po skončení makier vrátiť pôvodný význam príkazom:
  `\restorefont`
  Jedná sa o balík makier od P. Olšáka, ktorý predefinuje v prípadoch, kde to má zmysel volanie cm∗ fontov cs∗ fontami.

- Pokiaľ máte umiestnené nejaké makrá pod `tex.local` v korektných adresároch podľa aktuálneho `texmf.cnf` a TEX ich napriek tomu nevie nájsť, skúste vyhodiť databázový súbor `ls-R`, pokiaľ ho tam máte vyrobený. Aj keď by podľa nastavenia TEX mal predhľadávať podstrom `$TEXMFLOCAL` na disku, existencia súboru `ls-R` mu v tom zabraňuje (aspoň pod Win98 z prostredia `WinEdt`).
- Pokiaľ máte `Ghostview` nainštalovaný na inom pevnom disku ako c, treba upraviť cestu cez `Options->MenuSetup->Accessories`: v položke 'Menu items' treba vybrať 'GSView' a v okne 'Utility' zmeniť označenie pevného disku.

Ako je to so štýlom slovak.sty a czech.sty

Hneď v úvode treba konštatovať, že existuje niekoľko úplne odlišných štýlov `slovak.sty` nachádzajúcich sa na rôznych servroch (a obdobná situácia je aj v prípade `czech.sty`). Spomeňme aspoň tri najrozšírenejšie: prvý má pôvod z Babelu, druhý s hlavičkou '1998/03/18 v2.2 CSTeX slovak style' je z dielne J. Šnajdr, Z. Wagner, J. Zlatuška a tretí pochádza niekedy z roku 1992 od O. Ulrycha (a veľkého kolektívu) s mojimi drobnými zmenami pre slovenčinu. Ak sa uspokojíme s tvrdením, že druhá verzia je v podstate vylepšením tretej verzie, ešte stále nám ostávajú dve odlišné verzie národných štýlov. Jedným z cieľov výboru je zjednotiť tieto štýly, čo je však do značnej miery závislé od postoja pána J. L. Braamsa, autora Babelu.

Ak berieme do úvahy túto realitu, potom niekoľko základných poznatkov pri slovenskom TEXovaní s TEXLive4:

- Pokiaľ budete používať len `csplain/csamstex` (resp. ich pdf-varianty), akcentované písmená sa budú zobrazovať korektne. Pokiaľ chcete používať vymoženosti štýlu `slovak.sty` ako ste naň zvyknutý (rozdeľovanie slov, úvodzovky, ...), treba na začiatok dokumentu napísať
  `\input slovak.sty`
  Ak ste náhodou zabudli modifikovať cesty pre CSAMS-TEX v súbore `texmf.cfg`, `tex.exe` nájde BABELovský `slovak.sty` a skončí s hláškou:
  ```
  ...
  (D:/TEX/texmf/tex/generic/babel/slovak.ldf
  ! Undefined control sequence.
  1.20 \ProvidesFile
                     {slovak.ldf}
  ```
- V prípade CSLATEXu je nutné inklúdovať
  `\usepackage{slovak}`
  už aj pre zobrazovanie akcentovaných písmen. Pri konfigurácii z TEXlive4 sa bude inklúdovať `slovak.sty` rovnaký ako v prípade CSplainu (či

$\mathcal{CS}\mathcal{AMS}$-TEXu), ale pre volanie v LATEXu niektoré jeho príkazy (našťastie nie príliš podstatné) budú pre TEX neznáme (napr. `\originalTeX`, ...). Ďalší z problémov, ktorý ostáva doriešiť.

- Slovenské rozdeľovanie však aj tak nebude fungovať tak, ako bolo navrhnuté a ako fungovalo v starších poslovenčených verziách TEXu, ktoré nevychádzali z $\mathcal{CS}$TEXu. Rozdiel je dosť podstatný. Kým v $\mathcal{CS}$TEXu dĺžka poslednej slabiky pri rozdelení slova musí byť aspoň 3, v starších verziách mohla byť aj 2. Pre prestavenie tohto stačí použiť príkaz `\righthyphenmin=2`, ktorý je najlepšie volať na začiatku vlastného dokumentu po inklúdovaní všetkých makier v dokumente. Iná možnosť je prestaviť túto hodnotu na 2 priamo v súbore `<root>\texmf\tex\csplain\hyphen.lan` a vygenerovať všetky formáty $\mathcal{CS}$TEXu znova. (Pre kompatibilitu $\mathcal{CS}$TEXu sa však doporučuje premenovať všetky menené súbory!) Teraz už dostávame podstatne slušnejšie slovenské delenie slov. Snahou výboru bude dosiahnuť dohodu aj v tomto bode a zjednotiť túto hodnotu pre ďalšie distribúcie $\mathcal{CS}$TEXu.

Slovo na záver

Samozrejme, že by sa patrilo zmieniť viac o rôznych rozšíreniach, ktoré sú z TEXLive4 priamo dostupné, ako aj o inštalácii TEXLive4 pod nejakým voľne šíriteľným editorom. O tom snáď niekedy nabudúce!

Ak sa vám podarilo inštaláciu zrealizovať a TEX ste naučili po slovensky – gratulujeme!!! Teraz je už len čas na dobrú kávu ...

Aktuálnejšie a podrobnejšie informácie k chybám inštalácie TEXLive4 (špeciálne pod Win32), ako i doinštalovania slovenčiny a češtiny všeobecne i pod `WinEdt` je možné nájsť na stránke prístupnej z www.cstug.sk, kde je tiež možné stiahnuť modifikované súbory. Nachádza sa tam aj súbor winedt.sk, ktorý obsahuje skonfigurované prostredie editoru. Tí odvážnejší ním môžu skúsiť prepísať `\WinEdt\winedt.ini` a časť práce si (možno) ušetria.

Odkazy

[1] TEXLive CD-ROM, TUGboat, Volume 20(1999), No. 1, 20–44.

[2] Príručka TEXLive, štvrté vydanie, Zpravodaj 1-2, 1999, 1–77.

[3] frTEX: A teTEX-based Distribution for Windows, TUGboat, Volume 20(1999), No. 3, 290–297.

Janka Chlebíková,
MFF UK, Bratislava

Vytváření záložek operátorem PDFmark za asistence Acrobat Distilleru

Martin Černý

Tento článek má upozornit na možnost vytváření záložek pro PDF dokumenty získané destilací postscriptového souboru programem Acrobat Distiller nebo nějakým jeho ekvivalentem. Jde zejména o vytváření obsahu, seznamu tabulek, obrázků apod.

Co je PDF

PDF (Portable Document Format) je formát souborů používaných pro zobrazení dokumentu nezávisle na aplikačním software, hardware a operačním systému, které byly použity při jeho vytváření. Vychází z modelu postscriptového jazyka, ale přidává možnosti interaktivního zobrazení, definuje strukturovanější formát a obsahuje objekty jako anotace a hypertextové odkazy. PDF soubor je sestaven z pořadově číslovaných objektů: text, grafika, obrázky, zvuky a video.

Operátor PDFmark

Acrobat Distiller převádí postscriptový soubor na formát PDF. PDF soubory mohou mít zvláštní vlastnosti jako poznámky (notes), odkazy (links), záložky (bookmarks), články (articles), položky slovníku (info dictionary entries) a zastřižení stránky (page cropping). Tyto informace obyčejně nejsou v postscriptovém souboru přítomny, a proto nemohou být připojeny destilací (distilling process) do PDF souboru.

Operátor PDFmark reprezentuje výše uvedené vlastnosti PDF v postscriptovém kódu. Lze tedy do postscriptového souboru začlenit definice PDFmark, které budou interpretovány při destilaci do PDF.

Vlastnost bookmarks

Bookmarks (záložky) mohou tvořit například obsah, nebo odkazy na význačná místa v dokumentu, jsou zobrazovaná AcrobatReaderem v samostatném rámu a aktivují se klávesou F5, menu Window — Show Bookmarks nebo ikonou Show/Hide Navigation Panel. Většinou tvoří stromovou strukturu s položkami,

které po kliknutí zobrazí dokument na straně a pozici referované zvolenou záložkou.

Syntaxe bookmarks

Záložka je specifikována použitím operátoru `pdfmark` ve spojení se jménem `OUT`. Záložka může odkazovat na určitou pozici v dokumentu, jiný PDF soubor, soubor jiného formátu nebo URL. V dalším textu se zaměříme na vytváření odkazů do aktuálního dokumentu. Pracovně si definujeme následující syntaxi:

```
[/Count c /Page p /View v~/Title t /OUT pdfmark
```

`[`	uvozující znak
`/Count c`	následujících `c` položek jsou podpoložky aktuální položky
`/Page p`	odkaz na stránku s pořadovým číslem `p`
`/View v`	`v` definuje způsob zobrazení odkazované stránky
`/Title t`	`t` je zobrazovaný text záložky
`/OUT pdfmark`	uzavření definice

`c` je celé číslo, které definuje počet podpoložek, záporná hodnota způsobí zavření dané větve, `v` je název typu zobrazení s příslušnými parametry:

`/Fit`	velikost na celé okno
`/FitB`	bounding box stránky na celé okno
`/FitH top`	horizontální rozměr na celé okno, `top` určuje vertikální vzdálenost od počátku
`/FitBH top`	podobně jako `FitH`, ale pouze bounding box
`/FitR` x_1 y_1 x_2 y_2	daný čtverec na celé okno
`/FitV left`	vertikální rozměr na celé okno, `left` určuje horizontální vzdálenost od počátku
`/FitBV left`	podobně jako `FitV`, ale pouze boundig box
`[/XYZ left top zoom]`	`left` a `top` určují vzdálenost od počátku stránky; `zoom` určuje zvětšení; hodnoty `null` kopírují aktuální hodnoty.

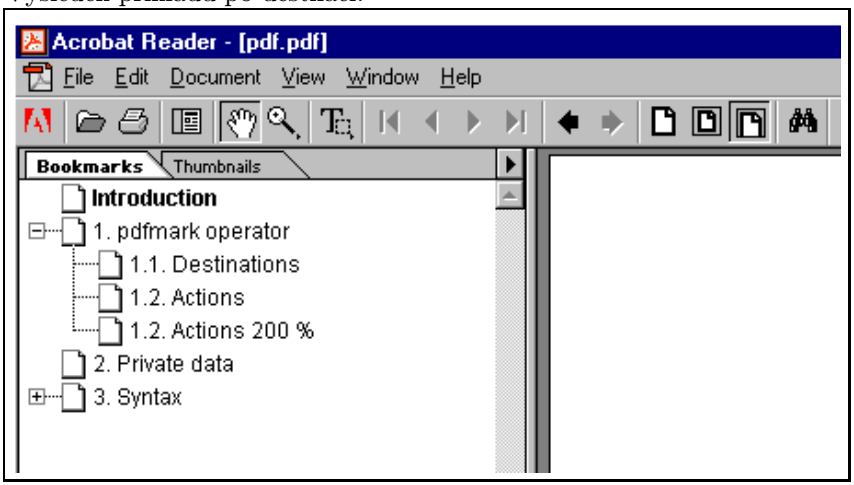
Číslování stránek je pořadové a nemusí odpovídat číslování v dokumentu – první stránka dokumentu má číslo jedna, druhá dvě, ...

Příklad:

```
[/Page 1 /View[/XYZ 10 800 1.0] /Title(Introduction) /OUT pdfmark
[/Count 3 /Page 2 /View[/Fit] /Title(1. pdfmark operator)
/OUT pdfmark
[/Page 4 /View[/Fit] /Title(1.1. Destinations ) /OUT pdfmark
[/Page 5 /View[/FitB] /Title(1.2. Actions) /OUT pdfmark
```

```
[/Page 5 /View[/XYZ 100 800 2.0] /Title(1.2. Actions 200 %)
/OUT pdfmark
[/Page 6 /View[/FitH 50] /Title(2. Private data) /OUT pdfmark
[/Count -2 /Page 7 /View[/FitV 100] /Title(3. Syntax) /OUT pdfmark
[/Count -2 /Page 7 /View[/Fit] /Title(3.1. Annotations)
/OUT pdfmark
[/Page 9 /View[/Fit] /Title(3.1.1. Notes) /OUT pdfmark
[/Page 10 /View[/Fit] /Title(3.1.2. Links) /OUT pdfmark
[/Page 11 /View[/XYZ null null 0] /Title(3.2. Bookmarks)
/OUT pdfmark
```

Výsledek příkladu po destilaci:



PDF, PS a TEX

PDF je vhodný a rozšířený způsob šíření dokumentů. Prohlížeč PDF dokumentů je volně šiřitelný a lze ho stáhnout z `www.adobe.com`. Verze 4 pro OS Windows má řadu vylepšení oproti verzím předcházejícím, např. umožňuje tisk sudých a lichých stránek.

PostScript můžeme chápat jako předstupeň k vytvoření PDF dokumentu. Prostředí TEXu jako soubor prostředků vytváření dokumentů nám nabízí řadu způsobů získání PDF.

Pokud bude náš dokument „prošpikovaný“ hypertextovými odkazy nebo je formát PDF velmi často cílovým formátem sázeného textu, bude patrně nejlepší použití PDFTEXu, jehož výstupem je místo DVI souboru PDF. Překonání instalačních potíží a čas strávený získáváním nových znalostí se jistě brzy zúročí.

Druhou cestou je konverze DVI souboru do PDF pomocí nějakého převodníku, například Dvipdfm, který využívá TEXovského příkazu `\special` k vložení vlastností PDF dokumentů.

Třetí způsob využívá možností programu Dvips a Acrobat Distilleru. Dvips vytvoří z DVI postscriptový soubor, který Acrobat Distiller konvertuje na PDF. V tomto případě lze ručně nebo programově obohatit postscriptový soubor o kód reprezentující vlastnost bookmarks.

Jak obohatit postscriptový soubor o záložky

Záložky v dokumentu si mohou vystačit pouze s určením odkazované strany. Mnohem příjemnější by samozřemě bylo, kdyby byla známa i vertikální a horizontální souřadnice význačného místa na odkazované straně, ale to by celý problém dělalo složitější a vedlo by nakonec k použití primitivu `\special` nebo jiných výše jmenovaných prostředků. Zaměříme se na skoro automatické generování bookmarks z existujícího obsahu dokumentu bez ohledu na to, zda je využito LATEXovského souboru *.toc nebo našeho vlastního obsahu. Předpokládá se, že tento soubor obsahuje název význačného místa (kapitola, tabulka, obrázek) a číslo strany kde se nachází. Tento soubor lze převést na postscriptový kód, který bude před destilací vložen do postscriptového souboru získaného programem Dvips. Pro začátek si můžeme vystačit s textovým editorem, který umí provádět náhrady a definovat uživatelská makra, například TextPad. Ilustrující může být následující příklad:

```
\chap{Introduction}{1}
\chap{1. pdfmark operator}{2}
\sec{1.1. Destinations}{4}
\sec{1.2. Actions}{5}
\chap{2. Private data}{6}
\chap{3. Syntax}{7}
\sec{3.1. Annotations}{7}
\subsec{3.1.1. Notes}{9}
\subsec{3.1.2. Links}{10}
\sec{3.2. Bookmarks}{11}
```

```
[/Page 1 /View [/Fit] /Title (Introduction) /OUT pdfmark
[/Count -2 /Page 2 /View [/Fit] /Title (1. pdfmark operator)
/OUT pdfmark
[/Page 4 /View[/Fit] /Title(1.1. Destinations) /OUT pdfmark
[/Page 5 /View[/Fit] /Title(1.2. Actions) /OUT pdfmark
[/Page 6 /View[/Fit] /Title(2. Private data) /OUT pdfmark
[/Count -2 /Page 7 /View[/Fit] /Title(3. Syntax) /OUT pdfmark
[/Count -2 /Page 7 /View[/Fit] /Title(3.1. Annotations)
```

```
/OUT pdfmark
[/Page 9 /View[/Fit] /Title(3.1.1. Notes) /OUT pdfmark
[/Page 10 /View[/Fit] /Title(3.1.2. Links) /OUT pdfmark
[/Page 11 /View[/Fit] /Title(3.2. Bookmarks)
/OUT pdfmark
```

Jediné co budeme muset provést ručně, bude doplnění čísla následujícího za parametrem `/Count`. Celý problém vytvoření doplňkového kódu lze řešit algoritmicky, ale to předpokládá jednotný formát TEXem generovaného obsahu. Lepší způsob je spolu se souborem `*.toc` získat další soubor, který bude přímo obsahovat postscriptový kód. Výhodou bude nejen rychlejší modifikace, ale i sekvenční počítání odkazovaných stran, které je pro PDF nezbytné.

Samotný kód můžeme vložit do postscriptového souboru kamkoli. Jisté je, že pokud ho dáme na konec, tak zjistíme chybu až po předestilování celého souboru, což v případě objemných dokumentů může trvat delší čas. Sám jsem testoval doplňující kód samostatně a po odladění ho vložil do sekce `%%Trailer`. Aktivace navigačního panelu je možná po otevření v prohlížeči nebo vložením následujícího kódu do sekce `%%BeginSetup`:

```
% OPEN INFO
[ /PageMode /UseOutlines
/Page 1 /View [/XYZ null null null]
/DOCVIEW
pdfmark
```

Hodnoty parametru `/PageMode` jsou `/UseNone`, `/UseOutlines`, `/UseThumbs` a `/FullScreen`, význam je zřejmý z jejich názvu. Parametry `/Page` a `/View` mají stejné možnosti jako při definování záložek.

Závěr

Použití této metody není příliš elegantní ani rychlé a ani si nemyslím, že dojde k jejímu masovému nasazení. Může se však stát (jako mně), že náhlá nebo ojedinělá potřeba vytvoření záložek bude vyřešena bez peripetií spojených s instalací nebo ovládnutím náležitého nástroje.

Tipy

1. Při ukládání editovaného postscriptového souboru je nutné vypnout volbu Word-Wrap (lámání textu). Zapnutá volba může způsobit chybu při převodu do PDF.
2. Před destilací je vhodné se přesvědčit, zda nastavení stránky odpovídá našemu formátu postscriptového souboru.

3. Při ladění není možné mít otevřený destilovaný soubor v prohlížeči, protože dochází ke kolizi při zápisu.
4. Vkládáme-li do dokumentu EPS soubory, je lepší ASCII formát oproti binárnímu z hlediska editace postscriptového souboru.

Literatura

1. Adobe Developer Support *pdfmark Reference Manual — Technical Note #5150.* Dokumentace k programu Acrobat Distiller.
2. Adobe Systems Incorporated *Portable Document Format Reference Manual — Version 1.3.* ftp.adobe.com.

Martin Černý
cernym@fcmail.com

METAPOST nielen na nakreslenie loga

Martin Budaj

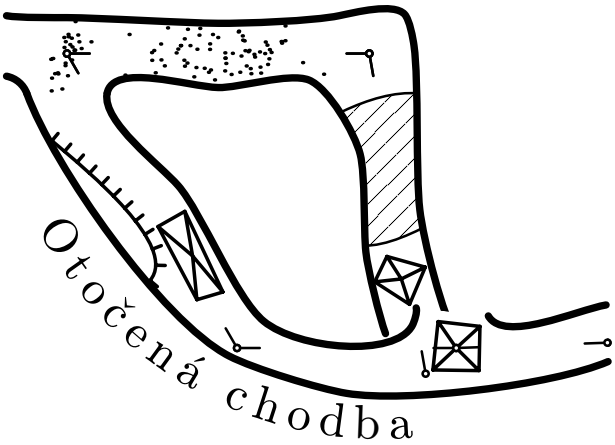
Doposiaľ bolo publikovaných niekoľko článkov o METAPOSTe a jeho využití pri kreslení grafov či technických ilustrácií – viď napr. [1]. Jeho charakter (textový súbor ako vstup, veľká presnosť technického kreslenia, otvorenosť celého systému a dobrá spolupráca s inými programami) ho však predurčuje aj pre použitie v omnoho rozsiahlejších projektoch.

Príkladom aj inšpiráciou pre podobné projekty môže byť *θηρίον*[1] (autori S. Mudrák a M. Budaj) [2], kolekcia programov na spracovanie meračských dát z jaskýň. Jedným z výstupov programu je 2D mapa generovaná pomocou METAPOSTu. Základným problémom bola požiadavka, aby väčšina dát bola viazaná relatívne k meračským bodom. Sú to jediné body, ktorých súradnice (odhliadnúc od chyby merania) poznáme. Poloha stien, kameňov a všetkých ostatných objektov je známa, až keď je známa poloha meračských bodov. Relatívna väzba sa stráca okamžite po nakreslení mapy rukou alebo (bežne používaným) naklikaním v niektorom grafickom programe – pre porovnanie skúste nadefinovať relatívne súradnice v programoch ako Illustrator alebo AutoCAD. Pri relatívnej väzbe dát k meračským bodom je možné bez námahy vygenerovať novú mapu, ak sa

[1] čítaj *thērion*; slovo pochádza z gréčtiny

upresnia polohy meračských bodov. Pri rozsiahlych jaskynných systémoch môže zmena polohy dosiahnuť rádovo metre. Priložené obrázky názorne ilustrujú flexibilnosť tohto prístupu – zdrojové texty sa líšia v jedinom príkaze, ktorý spojí dva meračské body (body sú označené krúžkom s čiarkami v smere merania) do jedného.

Samozrejme, možné uplatnenie nového prístupu nie je len v speleokartografii. Čo napríklad 2D animácie – môžeme vygenerovať sériu obrázkov so zmenou niekoľkých súradníc bodov a parametrov, pričom uvedieme do pohybu komplikované objekty…

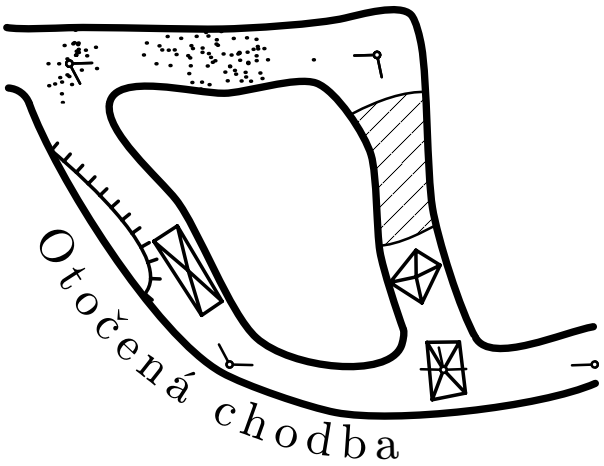


Princíp činnosti programu je naznačený v niekoľkých bodoch:

1. vstupom je textový súbor, v ktorom sú dáta viazané relatívne
2. dáta sú spracované do súboru s absolútnymi súradnicami pomocou programov v jazykoch C a Perl
3. z tohto súboru sú rôznymi filtrami generované vstupné súbory pre METAPOST, 3D vizualizačný program, TEX či SQL databázu
4. ďalším skriptom sú výstupné obrázky METAPOSTu modifikované do výslednej podoby.

METAPOST prináša aj ďalšie META-vlastnosti: nezávislosť na značkovom kľúči (je to vec definícií METAPOSTu) ako aj možnosť nelineárnej zmeny veľkosti písma, hrúbky čiar či hustoty šrafovania pri lineárnej zmene mierky.

Rozsiahly projekt prináša aj rozsiahle problémy. V nasledovných odsekoch je naznačená možnosť riešenia niektorých z nich.



Veľké čísla v METAPOSTe

Limitácia číselných premenných hodnotou 4096 známa z METAFONTu je odstránená po nastavení `warningcheck:=0`; vtedy je limitnou hodnotou 32768.

Ak používame v EPS obrázkoch veľké súradnice, hoci obrázky majú bežnú veľkosť, je potrebné upraviť makrá na vkladanie EPS súborov do TEXu (bežne `epsf.tex`) tak, aby na uloženie súradníc BoundingBoxu používali registre typu ⟨*count*⟩ a až rozdiel súradníc, teda rozmery obrázku, vložili do registra typu ⟨*dimen*⟩. Veľmi jednoduchý príklad takého makra je možné nájsť v [2].

Veľké obrázky

Mapy často dosahujú rozmery niekoľko metrov a obsahujú tisíce čiar a bodov. Kapacita pamäte METAPOSTu neumožňuje priame spracovanie tak veľkých obrázkov. Meračské dáta sú teda už vo vstupnom súbore rozdelené na bloky; každý s takým množstvom dát, aby bol METAPOST schopný spracovať jeden blok ako jeden obrázok. Výstupom je teda množstvo čiastkových, chaoticky sa prekrývajúcich EPS obrázkov. Tieto sú načítané Perlovským skriptom, ktorý podľa nastavenej veľkosti výstupného mapového listu vyberie pre každý mapový list EPS súbory, ktoré doň zasahujú (BBox informácia), dekóduje informácie o fontoch použitých v každom súbore, spojí PostScriptové dáta do jedného súboru a vygeneruje novú hlavičku. Ak blok (tj. EPS súbor z METAPOSTu) zasahuje do viacerých mapových listov, dáta sa opakujú; avšak pri rozumnej voľbe blokov a mapových listov je to zanedbateľné duplikovanie informácií.

Fonty v EPS obrázkoch

METAPOST umožňuje generovať dva typy EPS súborov – štruktúrované (použiteľné aj v iných grafických programoch, vyžaduje nastavenie `prologues:=1`) a neštruktúrované (`prologues:=0`, default). Pri nastavení `prologues:=0` zapisuje METAPOST do hlavičky obrázku, ktoré znaky z ktorého fontu sa v obrázku používajú. Tento typ EPS súborov je použiteľný v TEXovských dokumentoch pri použití `dvips` a následne prehliadača PostScriptu. Je možné použiť bitmapové (.pk) aj PostScriptové fonty. Pri pokuse zobraziť takýto obrázok priamo – bez prekladu cez `dvips` – vyhlási interpréter PostScriptu chybu.

Ak použijeme `prologues:=1`, dostávame obrázok priamo zobraziteľný prehliadačom PostScriptu. Vyžaduje však používanie jedine PostScriptových fontov. (Fonty sa do obrázku nevkladajú, je ich treba mať nainštalované v systéme. Ak potrebujeme nutne obrázok, ktorý by obsahoval fonty, je možné vložiť súbor s PostScriptovým fontom – ručne v textovom editore alebo na tento účel napísaným skriptom – do hlavičky EPS súboru; interpréter PostScriptu takýto font takisto načíta.) V tomto prípade METAPOST nezapisuje do hlavičky informácie o používaných znakoch, takže pri spracovaní cez `dvips` je potrebné nastaviť, aby bol do výsledného súboru vložený celý font.

Prístup k niektorým funkciám PostScriptu

METAPOST neumožňuje napríklad používanie vzoriek (*patterns*, slúžia na vyplnenie ohraničenej plochy pravidalne sa opakujúcim vzorom). Nie je to neriešiteľný problém; možný prístup ukazuje balík `mpattern` P. Boleka dostupný v archíve CTAN, ktorý bol použitý aj na šrafovanie jazierok v predchádzajúcich obrázkoch.

Rozšírenia jazyka

Od verzie 0.60 umožňuje METAPOST priamu komunikáciu so súbormi pomocou operátora `readfrom` *filename* a príkazu `write` *string* `to` *filename*. Ďalšie užitočné vylepšenia je možné nájsť v [3].

Rozbor jedného z makier

Z množstva makier, ktoré bolo pre kreslenie máp potrebné napísať, má širšiu oblasť použitia napríklad makro pre písanie textov pozdĺž ľubovoľnej krivky.

Nehodí sa síce pre použitie v texte knihy či časopisu, o to skôr však pri kreslení obrázkov alebo návrhu pečiatky či loga.

```
picture pict[];
pair zz[];
string ch, Strutstring;

Strutstring = "(Č";

def Freetext(expr String, Path, Font, Scale) =
    l := length String;                                  % počet znakov reťazca
    pict0 := String infont Font scaled Scale;
    strl := xpart(lrcorner pict0- llcorner pict0);       % skutočná dĺžka reťazca
    pathl := arclength Path;
    delta := (pathl- strl)/(l-1);                        % rozostupy medzi znakmi
    cas := 0;
    for i = 0 upto (l-1):
        ch := substring(i, i+1) of String;
        pict0 := ch infont Font scaled Scale;
        charwidth := xpart(lrcorner pict0- llcorner pict0);      % šírka znaku
        if ASCII(ch) ≠ 32:
            zz1 := lrcorner pict0;
            addto pict0 also Strutstring infont Font scaled Scale;
            zz2 := llcorner pict0;
            zz3 := ulcorner pict0;
            pict1 := ch infont Font scaled Scale;
            setbounds pict1to
                zz2 -- (xpart zz1, ypart zz2) --
                (xpart zz1, ypart zz3) -- zz3 -- cycle;
            cas := cas + charwidth/2;
            t := arctime cas of Path;
            label(pict1 rotated (angle direction t of Path),
                point t of Path);                        % vysádzanie znaku
        else:
            cas := cas + charwidth/2;                    % medzeru preskakujeme
        fi;
            cas := cas + charwidth/2 + delta;
    endfor;
enddef;
```

Makro musí riešiť dva hlavné problémy: pohyb pozdĺž krivky a sádzanie jednotlivých znakov. Oproti METAFONTu situáciu veľmi uľahčujú nové operátory

`arclength` a `arctime`, ktoré počítajú so skutočnou dĺžkou krivky. (Ako komplikovane bolo treba riešiť pohyb pozdĺž krivky v METAFONTe, je vidieť napríklad v [4, str. 49–51].)

Najprv je teda potrebné zistiť počet znakov reťazca, dĺžku krivky a dĺžku, na akú by bol reťazec vysádzaný štandardnou metódou. Rozdiel týchto dĺžok sa rovnomerne rozdelí medzi znaky. Avšak miesta, na ktorých budú znaky umiestnené, nemôžeme po krivke rozmiestniť v pravidelných intervaloch, pretože každý znak môže mať inú šírku.

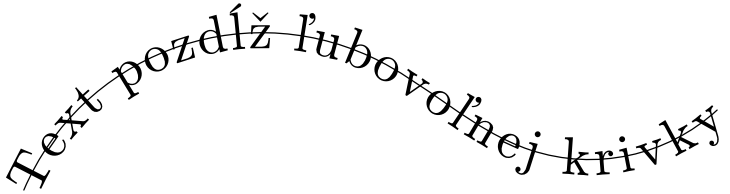
Dostávame sa k druhému problému: sadzba jednotlivých znakov (pomocou príkazu `label`). Pretože každý znak je otočený v smere dotyčnice ku krivke v bode, na ktorom znak leží, je nevyhnutné spomedzi všetkých typov zarovnania, ktoré nám `label` poskytuje, vybrať zarovnanie na stred. Znaky však majú rôznu výšku a toto centrovanie by ich rôzne poposúvalo. Je teda potrebné, aby všetky znaky mali zdanlivo rovnakú výšku. To umožňuje malý trik, keď sa do pomocnej premennej uloží znak aj reťazec, ktorý obsahuje znaky zasahujúce najvyššie a najhlbšie od účiaria. Je to niečo ako `\strut` v TEXu s tým rozdielom, že nemá nulovú šírku. Pre fonty v kódovaní Latin 2 je použitý reťazec `"(Č"`. Do ďalšieho pomocného obrázka sa opäť uloží sádzaný znak; ponechá sa mu jeho šírka, avšak pomocou `setbounds` sa zmení jeho výška na výšku predchádzajúceho pomocného obrázka s reťazcom `"(Č"`.

Teraz ostáva už iba nájsť vhodné miesto krivky a pomocou `label` znaky (každý z nich už s jednotnou výškou a správne otočený) vysádzať.

Použitie je veľmi jednoduché. Po zadefinovaní makra stačí napísať nasledovných niekoľko riadkov

```
beginfig(1);
path P;
P = (20, 20) .. (150, 70) .. (210, 35) .. (280, 50);
Freetext("Text pozdĺž ľubovoľnej krivky", P, "csr10", 2);
draw P withcolor 0.5*black;                    % znázorníme aj krivku
endfig;
```

aby sme dostali efektný výsledok:



Odkazy

[1] L. Dobiáš: *Začínáme s METAPOSTem aneb Udělejte si vlastní logo.* Zpravodaj Československého sdružení uživatelů TEXu, **8** (3–4), 167–175 (1998), R. Špalek: *METAPOST.* Zpravodaj Československého sdružení uživatelů TEXu, **8** (3–4), 175–218 (1998).
[2] `http://therion.homepage.com`
[3] `http://plan9.bell-labs.com/cm/cs/who/hobby/mphist.html`
[4] P. Šedivý, M. Brož, J. Gřondilová, M. Píše, K. Houfek: *Kreslíme METAFONTem.* Zpravodaj Československého sdružení uživatelů TEXu, **8** (1), 1–63 (1998).

Martin Budaj
`Martin.Budaj@st.fmph.uniba.sk`

Písma v PostScriptu

Pavel Janík ml.

V tomto článku bude nastíněn význam jazyka PostScript pro počítačovou sazbu a formáty písem použitelné v PostScriptu. Popíšeme si také reprezentaci fontu v postscriptovém interpretu a odlišné položky ve slovníku fontu pro různé formáty písem. Hlavní důraz bude kladen na formáty Type 3 a Type 1.

Jazyk PostScript

PostScript je jazyk pro popis stránky. Je to interpretovaný programovací jazyk, který obsahuje dostatečné prostředky pro kompletní popis tiskového materiálu, použitých barev a fontů. Vznikl v roce 1985, kdy společnost Adobe Incorporated tento jazyk otevřela i pro odbornou veřejnost (některé prameny uvádějí vznik v roce 1982 [2]). V té době bylo velmi obtížné sdílet dokumenty případně připravovat podklady pro tiskárny. Každá tiskárna totiž používala proprietární formát a dokonce i fonty specifické pro daný typ tiskárny. V těchto podmínkách bylo téměř nemožné vyměňovat dokumenty, a tak byl formát PostScript, který je nezávislý na rozlišení výstupního zařízení, přijat přívětivě i odbornou veřejností. V roce 1985 byly uvedeny i první produkty obsahující tzv. *postscriptový interpret*. Tiskárny Apple LaserWriter obsahovaly vestavěný interpret jazyka PostScript.

Kompletní specifikace jazyka PostScript je vydávána knižně [6]. Jednotlivé verze jazyka jsou číslovány pomocí tzv. *Levelu*. První verze jazyka je nazývána PostScript Level 1. Druhá verze je Level 2 a aktuální verze jazyka PostScript je Level 3. Specifikace PostScript Level 3 je k dispozici i na webové stránce společnosti Adobe určené partnerům (`http://partners.adobe.com/`). Referenční postscriptový interpret společnost Adobe neuvolnila, ale společnost Aladdin Enterprises (`http://www.ghostscript.com/`) pravidelně uvolňuje nové verze svého interpretu Ghostscript pod licencí GNU GPL.

Formáty písem v PostScriptu

Protože je jazyk PostScript určen především pro popis tiskové strany, musí mít optimalizovánu práci s texty a s jednotlivými znaky. V této kapitole se zmíníme o tom, jak jsou jednotlivá písma v PostScriptu organizována, jak přidáme nové písmo do našeho dokumentu a také jak nové písmo v PostScriptu použijeme.

Ještě než se zmíníme o podrobnostech práce s fonty v PostScriptu, budeme si muset osvětlit používanou terminologii. Pod pojmem *znak* budeme rozumět symbol, tedy nikoli jeho vlastní tvar, sklon či jiné charakteristiky, ale spíše jeho abstraktní reprezentaci. Znakem rozumíme jednotlivá písmena abecedy: A, B, C apod. Reprezentaci znaku v daném fontu nazýváme *písmeno* (*glyph*). Písmenem tedy jsou např. A, B, C, tedy grafické reprezentace znaků A, B a C z fontu Helvetica. Pojmem *font* budeme chápat množinu písmen reprezentujících skupinu znaků ve specifické reprezentaci. Font je možné chápat jako program, který je nutno před jeho použitím provést. Každé písmeno je v tomto programu reprezentováno podprogramem.

Poté, co je font reprezentovaný programem, spuštěn, vytvoří si postscriptový interpret tzv. *slovník fontu*, který je možné posléze využívat pro získání grafické reprezentace jednotlivých písmen.

Samotné rastrování písmen již probíhá v režii postscriptového interpretu. Uživatel pouze specifikuje, který znak z kterého fontu (slovníku) chce použít a postscriptový interpret si pomocí slovníku příslušného fontu najde proceduru, která příslušné písmeno vytvoří.

PostScript podporuje několik typů fontů, které rozlišuje podle položky **FontType** ve slovníku fontu. Hodnotou této položky je nezáporné celé číslo. Proto hovoříme o fontech Type 0, Type 1, Type 42 apod. Každý typ fontu má jasně danou strukturu. V této práci se zmíníme pouze o fontech Type 1, Type 3 a krátce také o Type 42. Fonty Type 1, 2, 3, 14 a 42 jsou nazývány *základními fonty*.

Fonty Type 0 jsou tzv. *složené fonty*. Je možné je přirovnat k virtuálním fontům v TEXu. Libovolné písmeno tohoto fontu může být převzato z jiného fontu a výsledný font tak bude „složený“ z původních fontů. Fonty Type 1 mají jasně danou strukturu, která je popsána v knize Adobe Type 1 Font Format [9].

Většina latinkových fontů společnosti Adobe je dostupná právě v této podobě. Fonty ve formátu Type 1 používají omezenou sadu postscriptových operátorů, což zajišťuje jednodušší zpracování a vyšší rychlost vykreslování písmen. Formát Type 2 se nazývá Compact Font Format (CFF). Bližší informace o tomto formátu je možné nalézt v knize PostScript Language Reference Manual [6] a ve specifikaci formátu CFF [14] dostupné na serverech společnosti Adobe. Fonty ve formátu Type 3 obsahují pro každý znak kompletní postscriptový program definující vzhled vyrastrovaného písmene. Při jeho vytváření je možné použít jakéhokoli postscriptového operátoru.

Formáty Type 4 a Type 5 slouží pro trvalé uchovávání fontů v paměti tiskárny. PostScript Level 2 totéž umožňuje i pro fonty Type 1, a proto je formát Type 4 již zastaralý. Formát Type 5 se liší od Type 1 pouze strukturou souboru, nikoli jeho obsahem, který je stejný jako u Type 1. Oba tyto formáty jsou proprietární, a proto jsou i nedokumentované. Formát Type 42 je založen na formátu True Type. Je to pouze jakási obálka nad vlastním True Type fontem. Další typy fontů (Type 9, 10, 11, 14 a 32) jsou popsány v knize PostScript Language Reference Manual [6] a na webových stránkách společnosti Adobe (`http://partners.adobe.com/supportservice/devrelations/japan/typeforum/ftypes.html`).

Položky ve slovníku fontu

Slovník fontu je reprezentací programu fontu v postscriptovém interpretu. Jeho jednotlivé položky obsahují informace specifické pro daný typ fontu. Existuje však množina vlastností, které jsou společné pro všechny typy fontů a mohou tedy být uvedeny ve slovníku každého fontu. Tyto vlastnosti si popíšeme v tabulce 1.

Základní fonty mohou navíc kromě těchto položek ve slovníku fontu obsahovat i další položky, které jsou přehledně shrnuty v tabulce 2. Přesný význam jednotlivých položek bude vysvětlen na straně 211 na příkladu fontu ITC Anna.

Type 3 fonty

Type 3 fonty jsou z hlediska jazyka PostScript soubory několika postscriptových podprogramů. Ostatní typy fontů použitelné v PostScriptu jsou vytvořeny zcela nezávisle na jazyce PostScript a podléhají jiným specifikacím (např. Adobe Type 1 Font Format [9]) částečně či úplně nezávislým na jazyce PostScriptu.

V tabulce 3 si uvedeme dvě další položky, které mohou být obsaženy ve slovníku fontu Type 3. Jsou to procedury **BuildGlyph** a **BuildChar**, které slouží pro vlastní vykreslení písmene.

Položka	Typ	Význam hodnoty
FontType	celé číslo	Typ fontu. Je určující hodnotou pro strukturu fontu a pro reprezentaci jednotlivých písmen.
FontMatrix	pole	Matice přechodu od báze souřadného systému písmene k uživatelskému souřadnému systému.
FontName	řetězec	Jméno fontu, které není postscriptovým interpretem k ničemu použito. Má pouze informační hodnotu.
FontInfo	slovník	Slovník fontu obsahující další informace, ke kterým nepřistupuje postscriptový interpret přímo.
LanguageLevel	celé číslo	Minimální požadovaná verze postscriptového interpretu pro korektní chování fontu. Standardní hodnotou je 1.
WMode	celé číslo	Směr písma daného fontu. Tato hodnota určuje, který ze dvou směrů písma (horizontální, vertikální) bude použit. Definováno v PostScriptu Level 2. Standardní hodnota je 0 (horizontální písmo).
FID	fontID	Speciální objekt typu fontID. Jeho hodnota je definována po prvním použití operátoru **definefont**.

Tabulka 1: Položky slovníku fontu společné pro všechny typy fontů

Rozdíl mezi těmito procedurami je pouze v tom, že procedura **BuildChar** očekává na zásobníku kromě slovníku fontu ještě kód znaku, zatímco procedura **BuildGlyph** očekává kromě slovníku fontu jméno znaku. Procedura **BuildChar** tedy ještě musí uvnitř svého kódu zjistit, jak se jmenuje procedura vykreslující písmeno pro znak daného kódu. Nově vytvářené fonty ve formátu Type 3 by proto měly obsahovat definici procedury **BuildChar**, která zajišťuje plnou zpětnou kompatibilitu.

Informace, které jsme si výše popsali, si nyní demonstrujeme na velmi jednoduchém fontu. Tento font obsahuje pouze jedno jediné písmeno, které reprezentuje znak I.

```
8 dict begin
% Volitelná položka ve slovníku fontu
```

Položka	Typ	Význam hodnoty
Encoding	pole	Pole jmen znaků, které je použito při zobrazení znaku zadaného kódem.
FontBBox	pole	Pole čtyř čísel definující bounding box fontu.
UniqueID	celé číslo	Nezáporné celé číslo menší než $2^{24} - 1$ jednoznačně identifikující font.
XUID	pole	Pole celých čísel jednoznačně identifikující daný font včetně jeho varianty. Tato položka je definována v PostScript Level 2.

Tabulka 2: Položky slovníku fontu společné pro základní typy fontů

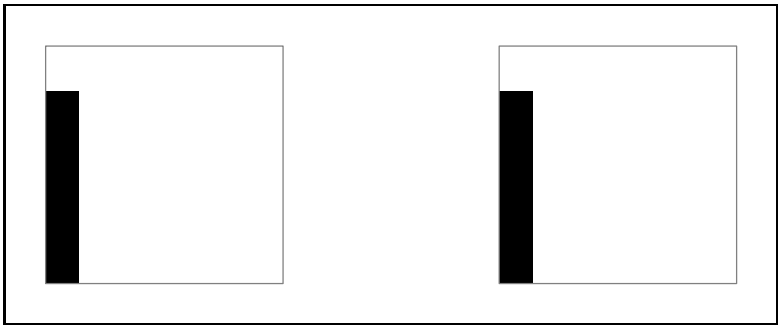
Položka	Typ	Význam hodnoty
BuildGlyph	procedura	Procedura, která vyrastruje písmeno pro požadovaný znak. Při volání této procedury již zásobník musí obsahovat slovník fontu a jméno znaku, který má procedura vyrastrovat. Tato procedura je volitelná a je definována až v PostScript Level 2.
BuildChar	procedura	Procedura, která vyrastruje písmeno pro požadovaný znak. Tato procedura musí být obsažena ve fontech určených pro PostScript Level 1 a pokud chybí i procedura **BuildGlyph** v PostScript Level 2 nebo 3, tak i zde. Při volání této procedury již zásobník musí obsahovat slovník fontu a kód znaku, který má procedura vyrastrovat.

Tabulka 3: Další položky slovníku fontu pro fonty Type 3

```
/FontName (MyTypeThreeFont) def
% Nutné položky ve slovníku fontu
/FontType 3 def
/FontMatrix [.001 0 0 .001 0 0] def
/FontBBox [0 0 750 750] def
% Definice kódovacího vektoru
/Encoding 256 array def
```

```
   0 1 255 {Encoding exch /.notdef put} for
   Encoding 73 /myonlychar put
   % Vedlejší slovník obsahující popisy jednotlivých písmen
   /CharProcs 2 dict def
      CharProcs begin
         /.notdef {} def
         % Definice vlastního znaku
         /myonlychar
         {
            % Definice cesty
            0 0 moveto
            100 0 lineto
            100 800 lineto
            0 800 lineto
            closepath
            % Vyplnění definované cesty
            fill
         } bind def
      end
   % Procedura BuildGlyph
   /BuildGlyph
   {
      % Nastavení výstupního a vyrovnávacího zařízení
      1000 0
      0 0 750 750
      setcachedevice
      % Vyzvednutí jména procedury ze slovníku CharProcs
      exch /CharProcs get exch
      2 copy known not
         { pop /.notdef }
         if
      % Provedení procedury
      get exec
   } bind def
   % Procedura BuildChar pro zpětnou kompatibilitu
   /BuildChar
   {
      1 index /Encoding get exch get
      1 index /BuildGlyph get exec
   } bind def
   currentdict
end
/MyTypeThreeFont exch definefont pop

% Nyní definovaný font použijeme
/MyTypeThreeFont findfont 100 scalefont setfont
0 0 moveto
(IAI) show
```

Pokud tento program v jazyce PostScript předložíme postscriptovému interpretu, dostaneme výstup podobný obrázku 1.



Obrázek 1: Výstup z námi definovaného fontu ve formátu Type 3

Mezi dvěma čtverci, které pouze zvýrazňují souřadný prostor písmen a nejsou výsledkem našeho programu, je volné místo o šířce 750 jednotek v souřadném systému písmene, protože jsme v příkladu použili znak A, jehož písmeno není v našem fontu definováno.

Formát Type 1

Jazyk PostScript tedy umožňuje definovat vlastní fonty pomocí formátu Type 3. Obrovskou nevýhodou tohoto formátu je nutnost implementace kompletního postscriptového interpretu. Navíc fonty ve formátu Type 3 mohou být velmi složité, jejich jednotlivé kontury se mohou vypočítávat případně mohou být i rekurzivně definovány, což je pro implementaci a větší rozšíření velmi nevýhodné. Společnost Adobe se proto pokusila navrhnout další formát, který by tyto nevýhody odstranil. A proto vznikl i dokument Adobe Type 1 Font Format [9], specifikace písem ve formátu Type 1. Formát Type 1 je mezinárodním standardem organizace ISO registrovaným pod číslem 9541 [5].

Formát Type 1 je stejně jako formát Type 3 založen na postscriptových příkazech, ale jejich syntaxe je velmi výrazně zjednodušena. Stejně tak je zúžena i množina použitelných příkazů. Postscriptový interpret pro rastrování fontů ve formátu Type 1 používá jiné algoritmy, které zajišťují kvalitnější výstup (zejména při extrémně malých velikostech písma). Další podstatnou výhodou fontů ve formátu Type 1 jsou i tzv. *hinty*, které charakterizují významné tahy či hranice jednotlivých písmen fontu. Jazyk PostScript žádnou podobnou vlastnost nenabízí a tudíž ji fonty ve formátu Type 3 nemohou využít. Fonty ve formátu Type 3 však mohou využít kompletní příkazovou sadu jazyka PostScript, zatímco formát Type 1 disponuje pouze omezenou sadou příkazů.

Formát Type 1 pracuje na podobném principu jako formát Type 3 pouze s tím rozdílem, že žádný font neobsahuje proceduru **BuildChar**. Tato procedura je totiž implementována přímo v postscriptovém interpretu. Pokud bychom se podívali např. na implementaci této procedury v postscriptovém interpretu Ghostscript, našli bychom podobný kód jako u fontů ve formátu Type 3:

```
% Here are the BuildChar and BuildGlyph implementation for Type 1 fonts.
% The names %Type1BuildChar and %Type1BuildGlyph are known \
  to the interpreter.
% The real work is done in an operator:
%       <font> <code||name> <name> <charstring> .type1execchar -

(%Type1BuildChar)  cvn {        % <font> <code> %Type1BuildChar -
  1 index /Encoding get 1 index get .type1build .type1execchar
} bind def
(%Type1BuildGlyph) cvn {        % <font> <name> %Type1BuildGlyph -
  dup .type1build .type1execchar
} bind def
```

Tento kód je obsažen v souboru `gs_type1.ps` v adresáři `lib` příslušné distribuce postscriptového interpretu Ghostscript. Procedura **BuildChar** u Type 1 fontů tedy očekává na zásobníku index do pole **Encoding**, které obsahuje seznam jmen procedur vykreslujících písmena pro jednotlivé znaky (indexy). Tato vlastnost se s výhodou dá využít k tzv. *překódování* fontu, kdy pouhou změnou pole **Encoding** dochází ke změně použité procedury pro rastrování znaku a tím i k rozdílnému výstupu bez nutnosti změny vlastního fontu (což je většinou znemožněno autorskými právy svázanými s daným fontem).

Jméno procedury, které **BuildChar** získá z pole **Encoding** je poté použito opět jako index do slovníku **CharStrings**, který je obsažen ve fontu. Slovník **CharStrings** obsahuje binárně zakódované instrukce definující vlastní proceduru, jež obsahuje definici jednotlivých kontur písmene. Tato procedura je poté provedena a jí definovaná kontura je buď vyplněna nebo pouze vykreslena podle toho, jakou hodnotu obsahuje položka **PaintType** ve slovníku fontu. Uživatel tedy může rozhodnout o tom, zda bude font a jeho písmena vyplněná či budou vykresleny pouze jejich obrysy pouhou změnou jedné položky ve slovníku fontu. To je jednak výhodou proti fontům ve formátu Type 3, protože tato úprava by znamenala přepsání definice samotného písmene a také je to nevýhodné pro tvůrce fontů Type 1, kteří musí brát v potaz obě varianty sazby a musí upravit definici kontur písmene.

Slovník Type 1 fontu

Stejně jako u Type 3 fontů jsou i Type 1 fonty pro postscriptový interpret reprezentovány slovníkem, který vznikne provedením samotného Type 1 fontu. Položky slovníku fontu společné všem typům fontů jsou uvedeny v tabulce 1.

Tabulka 4 obsahuje další položky, které mohou být součástí slovníku fontu ve formátu Type 1.

Slovník Type 1 fontu může stejně jako slovník kteréhokoli jiného typu fontu obsahovat slovník **FontInfo**, který například u fontu ITC Anna obsahuje následující informace:

```
/FontInfo 10 dict dup begin
/version (001.000) readonly def
/Notice (Copyright (c) 1993 Adobe Systems Incorporated. \
         All Rights Reserved. \
         ITC Anna is a~trademark of International Typeface Corporation.
        ) readonly def
/FullName (ITC Anna) readonly def
/FamilyName (ITC Anna) readonly def
/Weight (Regular) readonly def
/isFixedPitch false def
/ItalicAngle 0 def
/UnderlinePosition -100 def
/UnderlineThickness 50 def
end readonly def
```

První a poslední řádek výpisu jsou operátory jazyka PostScript, které uvozují (resp. ukončují) definici slovníku **FontInfo**. Jednotlivé řádky uvnitř definice slovníku definují verzi fontu (položka **version**), poznámku o autorských právech či ochranných známkách svázaných s fontem (položka **Notice**), plné jméno fontu (položka **FullName**), rodinu písma (položka **FamilyName**) a duktus (**Weight**). Položka **IsFixedPitch** slouží k identifikaci neproporcionálních písem, položka **ItalicAngle** udává sklon vertikálních tahů od vertikální osy. Zbylé položky (**UnderlinePosition** resp. **UnderlineThickness**) definují doporučené umístění resp. šířku podtrhovací linky. Podrobnější informace o jednotlivých položkách slovníku **FontInfo** jsou uvedeny v tabulce 5.

Jazyk PostScript nestanoví přesná pravidla, která by měly splňovat jednotlivé položky slovníku fontu, ale postupně se vytvořily jisté konvence, které jsou předpokládány některými aplikačními programy. Jméno fontu (položka **FontName**) je zkrácenou podobou plného jména fontu (**FullName**). Protože je hodnota této položky použita pro postscriptový operátor **definefont**, je zvykem vypustit mezery z plného jména fontu, případně změnit za znak minus. Např. font ITC Zapf Chancery Medium Italic má v položce **FontName** slovníku **FontInfo** hodnotu ZapfChancery-MediumItalic. Některými aplikacemi je rodina písma (položka **FontFamily**) použita pro hierarchické zařazení fontu. Proto jsou např. fonty Times Bold, Times Italic a Times Roman součástí větve Times v hierarchii fontů. Tato konvence je např. aplikována v programu Adobe Type Reunion [8]. Plné jméno fontu (položka **FullName**) většinou začíná jménem rodiny (**FontFamily**) a je následováno seznamem slov popisujícím kresbu písma. Jednotlivá slova jsou oddělena mezerami. Již zmíněný font ITC Zapf

Položka	Typ	Význam hodnoty
PaintType	celé číslo	Hodnotou této položky je dán typ kresby písmen. Pokud je hodnotou položky **PaintType** číslo 0, budou kontury písmen vyplněné. Pokud zde bude hodnota 2, budou pouze vykresleny kontury a nebudou vyplněny. Tato položka je povinná.
StrokeWidth	číslo	Šířka kontury. Tato hodnota je rovna šířce kontur, pokud nejsou vyplněny (tj. pokud je hodnota položky **PaintType** rovna 2). Pokud není definována, je její hodnota 0. Standardně distribuované fonty tuto položku neobsahují, je ji proto nutné při definici nového fontu, který bude bez vyplněných kontur, přidat do slovníku fontu s nenulovou hodnotou. Jedná se tedy o nepovinnou položku.
Metrics	slovník	Slovník **Metrics** obsahuje metrické informace pro horizontální směr písma. Tento slovník je nepovinný a v původních fontech bývá nedefinován. Používá se pouze při změně metrických informací písmen.
Metrics2	slovník	Jazyk PostScript Level 2 umožňuje specifikovat pomocí slovníku **Metrics2** metrické informace i pro vertikální směry písma.
CDevProc	procedura	Tato procedura souvisí s kompozitními fonty, nemusí být nutně definována a je ignorována postscriptovými interprety Level 1, které nepodporují kompozitní znaky.
CharStrings	slovník	Tento slovník přiřazuje jménům znaků z pole **Encoding** vlastní procedury pro vykreslení příslušných písmen.
Private	slovník	Slovník **Private** obsahuje informace, které by měly být k dispozici pouze samotnému fontu a nikoli uživateli.

Tabulka 4: Další položky slovníku fontu pro fonty Type 1

Položka	Typ	Význam hodnoty
version	řetězec	Hodnotou této položky je číslo verze Type 1 fontu.
Notice	řetězec	Pokud se k fontu vztahují ochranné známky či autorská práva, měla by být tato skutečnost zmíněna v položce **Notice**.
FullName	řetězec	Řetězec jednoznačně identifikující jméno fontu.
FamilyName	řetězec	Jméno rodiny fontů, do které font patří.
Weight	řetězec	Duktus fontu.
IsFixedPitch	boolean	Identifikátor neproporcionálních písem.
ItalicAngle	číslo	Úhel, který svírají dominantní vertikální tahy se svislou osou. Úhel je specifikován ve stupních.
UnderlinePosition	číslo	Doporučená vzdálenost podtrhovací linky od účaří.
UnderlineThickness	číslo	Doporučená šířka podtrhovací linky.

Tabulka 5: Položky slovníku **FontInfo**

Chancery Medium Italic je členem rodiny písem ITC Zapf Chancery. Slova Medium a Italic tedy specifikují kresbu písma a jeho řez.

Tato pravidla však nejsou pro tvůrce fontů závazná, společnost Adobe pouze doporučuje jejich dodržování a všechny její aplikace a fonty tato pravidla také dodržují. Tvůrci fontu také musí dodržovat jisté elementární předpoklady, aby jejich font mohly používat i aplikace, které neobsahují kompletní postscriptový interpret. Takovým softwarem je např. i Adobe Type Manager [9, Adobe Type Manager Compatibility] umožňující práci s postscriptovými fonty na operačních systémech společnosti Microsoft, které pro práci s fonty interně používají formát True Type.

Příklad Type 1 fontu

V tomto odstavci si uvedeme zdrojový kód fontu ITC Anna, který budeme v dalším textu používat pro demonstraci vlastností formátu Type 1.

```
%!PS-AdobeFont-1.0: Anna 001.000
%%CreationDate: Tue Jul 13 10:29:56 1993
%%VMusage: 20678 27570
%% ITC Anna is a trademark of International Typeface Corporation.
11 dict begin
```

```
/FontInfo 10 dict dup begin
/version (001.000) readonly def
/Notice (Copyright (c) 1993 Adobe Systems Incorporated. \
         All Rights Reserved. \
         ITC Anna is a~trademark of International Typeface Corporation.
        ) readonly def
/FullName (ITC Anna) readonly def
/FamilyName (ITC Anna) readonly def
/Weight (Regular) readonly def
/isFixedPitch false def
/ItalicAngle 0 def
/UnderlinePosition -100 def
/UnderlineThickness 50 def
end readonly def
/FontName /Anna def
/Encoding StandardEncoding def
/PaintType 0 def
/FontType 1 def
/FontMatrix [0.001 0 0 0.001 0 0] readonly def
/UniqueID 41116 def
/FontBBox{-81 -250 932 856}readonly def
currentdict end
currentfile eexec
cf0eeff329fe1db159a37891f19957002e7d435f06065e5e0a71393ede47
7128a76e9ca6da8df0c353e1352afd7fb01e92cbd565fc568222229162e2
... zkráceno
d54edd725c881f49e22678743289b3e9ef9f262271cab2be4a8872ca6c8b
9624a36451f78916e9db508a003da4b339ece06582ef04f84a
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
... zkráceno
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
cleartomark
```

Fonty ve formátu Type 1 jsou programy v jazyce PostScript, a proto by měly stejně jako ostatní programy v PostScriptu začínat dvojicí znaků `%!` . Některé operační systémy tuto dvojici znaků potřebují k identifikaci dokumentů ve formátu PostScript například pro určení vhodného filtru pro tisk. Zbytek řádku identifikuje formát fontu, jeho jméno a verzi:

```
%!PS-AdobeFont-1.0: Anna 001.000
```

Řetězec `PS-AdobeFont-1.0:` je používán fonty od společnosti Adobe. Některé fonty mohou používat i odlišnou specifikaci:

```
%!FontType1-1.0: Anna 001.000
```

V obou případech řetězec `1.0` identifikuje verzi specifikace formátu Adobe Type 1. V současné době je aktuální verzí specifikace verze 1.1 [9]. Řetězec `Anna` je jméno fontu a `001.000` je číslo verze fontu.

Řádek obsahující řetězec `CreationDate: Tue Jul 13 10:29:56 1993` udává čas, kdy byl font vytvořen.

```
%%VMusage: 20678 27570
```

Tento komentář je užitečný spíše pro aplikační programy než přímo pro postscriptový interpret. Udává totiž množství paměti, které font spotřebuje v postscriptovém interpretu. Aplikační program se proto ještě před jeho zavedením může ujistit, že postscriptový interpret disponuje dostatečným množstvím paměti. Druhé číslo (v našem případě 27570) vznikne rozdílem obsazení paměti po a před prvním zavedením fontu do paměti. První číslo vznikne odečtením obsazené paměti po a před druhým zavedením. První číslo je vždy menší nebo rovno druhému číslu, protože postscriptové interprety umožňují sdílení řetězců mezi fonty. Pro přibližné zjištění těchto čísel je možno použít i postscriptového interpretu Ghostscript. Následující postscriptový program zjistí paměťové nároky fontu:

```
vmstatus pop /number1 exch def pop
(Anna.pfa) run
vmstatus pop /number2 exch def pop
(Anna.pfa) run
vmstatus pop /number3 exch def pop

(%%VMusage is: ) =

number3 number2 sub =
number2 number1 sub =

quit
```

Příkladem jeho výstupu pro náš vzorový font ITC Anna je:

```
%%VMusage is:
23153
27073
```

Pokud bychom tedy font ITC Anna vytvářeli, uložili bychom do jeho definičního souboru řádek

```
%%VMusage: 23153 27073
```

Poslední komentářový řádek je pouze vyjádřením společnosti International Typeface Corporation, která vlastní ochrannou známku ITC Anna.

Následuje definice slovníku **FontInfo**, která je popsána výše. Poté je nutné definovat další položky slovníku fontu. Položka **FontName** obsahuje řetězec `/Anna`, proto bude možné font používat pod jménem `/Anna`.

Položka **Encoding** definuje kódovací vektor fontu. Místo vlastního pole je použito jedno z předdefinovaných polí. Kódovací vektor skrývající se pod jménem `StandardEncoding` je definován ve specifikaci jazyka PostScript

[6, příloha E.6]. Tato specifikace obsahuje definice dalších vektorů (`Expert`, `ISOLatin1Encoding`, `Symbol`). To ovšem neznamená, že by tvůrce fontu byl omezen jen těmito kódovacími vektory. Ani uživatel fontu není omezen těmito vektory a může si stejně jako tvůrce fontu definovat vlastní. Potom by definice kódovacího vektoru mohla vypadat například takto:

```
/Encoding 256 array
0 1 255 {1 index exch /.notdef put} for
dup 32 /space put
dup 33 /exclam put
dup 34 /universal put
dup 35 /numbersign put
dup 36 /existential put
dup 37 /percent put
... zkráceno
dup 250 /bracketrightex put
dup 251 /bracketrightbt put
dup 252 /bracerighttp put
dup 253 /bracerightmid put
dup 254 /bracerightbt put
readonly def
```

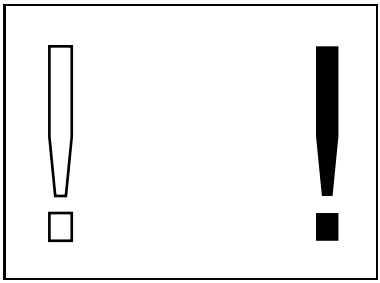
Tento kód v jazyce PostScript nejprve definuje pole **Encoding** o 256 prvcích, přiřadí každému prvku hodnotu `/.notdef` a teprve poté definuje význam jednotlivých položek v poli, kterým přiřazuje názvy procedur pro vykreslení jednotlivých písmen. Příklad je převzat z fontu Symbol. Pokud bude uživatel tohoto fontu chtít vykreslit znak s kódem 33, převezme procedura **BuildChar** z kódovacího vektoru název procedury `/exclam`, zavolá ji, a znak bude poté vykreslen, případně budou vykresleny pouze jeho kontury v závislosti na hodnotě položky **PaintType**.

```
/PaintType 0 def
```

Ta je v našem příkladu rovna 0, a proto bude znak vykřičník i vyplněn. Pokud by byla hodnota položky **PaintType** rovna 2, byl by vykřičník pouze vykreslen:

Další dvě položky jsou v téměř všech fontech ve formátu Type 1 stejné. Položka **FontType** musí obsahovat hodnotu 1 a hodnotou položky **FontMatrix** je matice přechodu od báze souřadného systému písmene k uživatelskému souřadnému systému. Jinými slovy je tak definován souřadný systém, ve kterém jsou prováděny procedury vykreslující jednotlivá písmena. Většina fontů ve formátu Type 1 používá souřadný systém o 1000 jednotkách, stejně jako náš vzorový font ITC Anna.

Položka **FontBBox** definuje obdélník, který vznikne opsáním útvaru vzniklého po vyrastrování všech znaků ve společném počátku. Tento údaj je využíván zejména postscriptovým interpretem k efektivnější práci s vyrovnávací pamětí a také pro ořezávání.



Obrázek 2: Vliv položky **PaintType** na vzhled písmen

Poslední čitelnou položkou fontu ITC Anna je **UniqueID**. Tato položka slouží k uchovávání vyrovnávací paměti s vyrastrovanými písmeny mezi jednotlivými úlohami postscriptového interpretu. Pokud font tuto položku neobsahuje, je tato paměť po skončení aktuální úlohy vymazána. Pokud však je položka **UniqueID** ve slovníku fontu obsažena, je vyrovnávací paměť uchována a při příštím použití tohoto fontu (resp. fontu se stejnou hodnotou **UniqueID**) již není proces volání jednotlivých procedur prováděn, vše je zjednodušeno vzhledem k obsahu vyrovnávací paměti. Tento proces však předpokládá jednoznačné přiřazení hodnot položky **UniqueID** jednotlivým fontům. Společnost Adobe proto drží databázi již přiřazených hodnot a tvůrci nového fontu ji mohou požádat o přidělení nového čísla. Přesný postup pro podání žádosti je uveden ve specifikaci formátu Type 1 [9, kapitola 2.5]. Hodnotou položky může být celé číslo menší než 16 777 215 ($2^{24} - 1$).

Zbylá část fontu je zašifrována, aby nebylo možné jednoduše „přečíst“ procedury vykreslující jednotlivá písmena. Metoda použitá pro šifrování je ale přehledně popsána ve specifikaci formátu Type 1 [9, kapitola 7]. Ostatně existuje i několik Open Source implementací této metody (např. knihovna t1lib, případně balík utilit t1utils).

Šifrovaná část obsahuje dvě zbylé položky slovníku fontu. Jsou to slovníky **Private** a **CharStrings**.

Slovník Private

Slovník **Private** o bsahuje informace, které se týkají každého písmene fontu. Jsou v něm obsaženy procedury, které jsou sdíleny podprogramy pro kresbu jednotlivých znaků, jsou zde uvedeny také hintovací informace pro celý font apod. Slovník **Private** je obsažen v zašifrované části definičního souboru fontu. Do čitelné podoby je možné jej převést např. programem `t1disasm` z balíku t1utils. Slovník **Private** našeho vzorového fontu obsahuje následující informace:

```
dup /Private 16 dict dup begin
/-||{string currentfile exch readstring pop}executeonly def
/||-{noaccess def}executeonly def
/||{noaccess put}executeonly def
/BlueValues [ -25 0 704 722 436 436 728 741 ] ||-
/OtherBlues [ 281 281 -49 -49 ] ||-
/MinFeature{16 16} ||-
/StdHW [77] ||-
/StdVW [77] ||-
/ForceBold false def
/password 5839 def
/UniqueID 41116 def
/Subrs 170 array
dup 0 {
   3 0 callothersubr
   pop
   pop
   setcurrentpoint
   return
   } ||
... zkráceno
dup 169 {
   169 68 96 138 0 171 rrcurveto
   0 171 -96 137 -169 68 rrcurveto
   return
   } ||
||-
```

Slovník **Private** obsahuje definici tří procedur, které budou ve vlastním slovníku často používány. Další položky slovníku **Private** již definují informace podstatné pro postscriptový interpret resp. pro funkci **BuildChar**. Hodnotou pole **BlueValues** je několik dvojic, které definují tzv. *písmovou osnovu* [1, strana 7]. Jsou to oblasti, do kterých zasahují jednotlivé tahy písmen. První číslo z každé dvojice udává spodní a druhé číslo udává horní hranici každé oblasti. První dvojice (pokud je přítomna) vždy udává oblast pod účařím (tj. první číslo je y-ová souřadnice tzv. *dolního přetahu* a druhé číslo je y-ová souřadnice *účaří*). Druhá dvojice v našem příkladu (704, 722) udává *verzálkovou dotažnici*. Třetí dvojice udává *střední dotažnici*. Poslední dvojice z našeho příkladu definuje *horní dotažnici*. Podobný význam mají hodnoty v položce **OtherBlues**.

Položky **StdHW** a **StdVW** definují standardní šířku horizontálních a vertikálních tahů. Tyto informace používá postscriptový interpret zejména při malých velikostech písma, kdy hraje roli každý bod. Postscriptový interpret pomocí těchto hodnot zajišťuje konzistentní vzhled písmen. Při větších velikostech písma nejsou rozdíly několika málo bodů viditelné a proto tyto hodnoty nejsou využity. Podobný význam mají i položky **StemSnapH** a **StemSnapV**, které ale nejsou ve vzorovém fontu použity.

Položka **ForceBold** má hodnotu `false`. Tato položka bude použita, pokud budou písmena fontu vykreslována v malých velikostech. V těchto velikostech se totiž ztrácejí rozdíly mezi normálním a tučným písmem. Právě proto je možno použít položku **ForceBold**, která informuje postscriptový interpret o tom, jak se má chovat k tučným písmenům. Pokud je hodnotou položky **ForceBold** řetězec `true`, postscriptový interpret rozšíří všechny tahy tučných písmen.

Položky **MinFeature** a **password** musí být ve slovníku **Private** přítomny, ale jejich význam je bohužel nedokumentován.

Přesný význam vysvětlených i dalších položek ve slovníku **Private** je definován ve specifikaci formátu Type 1 [9]. Bohužel ne všechny položky jsou popsány přesně a dostatečně, a proto je velmi obtížné se hlavně ve složitých fontech ve formátu Type 1 orientovat.

Slovník **Private** dále obsahuje i pole **Subrs**, které obsahuje často se opakující procedury. Tyto procedury mohou například vykreslovat akcenty, často se opakující vertikální či horizontální tahy, definovat hinty apod. V našem fontu pole **Subrs** obsahuje 170 procedur.

Slovník CharStrings

Slovník **CharStrings** obsahuje vlastní procedury pro vykreslování jednotlivých písmen a definice hintovací informace pro jednotlivá písmena. Procedury jsou indexovány svým jménem. Pokud tedy chceme vykreslit nějaké písmeno, musíme znát jméno znaku, který reprezentuje. Pokud známe kód znaku, můžeme jméno příslušné procedury (a tedy index do pole **CharStrings**) získat z pole **Encoding**. Definice slovníku **CharStrings** v našem fontu ITC Anna obsahuje následující text:

```
% Definice slovníku CharStrings
2 index /CharStrings 229 dict dup begin
% Písmeno mezera
/space {
   0 185 hsbw
   endchar
   } |-
% Písmeno vykřičník
/exclam {
   54 185 hsbw
   0 77 vstem
   0 77 hstem
   710 -20 hstem
   77 hmoveto
   137 callsubr
   77 148 rmoveto
   562 vlineto
   -77 hlineto
   -562 vlineto
```

```
   closepath
   endchar
   } |-
... zkráceno
/ecircumflex {
   -25 333 hsbw
   148 callsubr
   127 574 rmoveto
   86 callsubr
   } |-
% Nedefinovaný znak
/.notdef {
   0 185 hsbw
   endchar
   } |-
% Konec definice slovníku
end
```

Tento text je samozřejmě ve fontu kódován jednoduchým algoritmem [9, strana 47] a poté zašifrován pomocí metody eexec [9, strana 64]. Důvodem pro kódování je hlavně ušetření místa, protože jednotlivé příkazy procedur jsou kódovány do jednoho či maximálně dvou bajtů, zatímco jejich jména jsou mnohem delší. Důvody pro šifrování jsou spíše „politické“ či komerční. Společnost Adobe si nepřála, aby mohl kdokoli jednoduše kopírovat její písma, ale tato ochrana ani neklade příliš velké překážky. Existují dokonce volně šiřitelné implementace rozkrývající tuto jednoduchou ochranu.

Vlastní procedury ve slovníku **CharStrings** mohou obsahovat instrukce pěti typů:

- příkazy začínající nebo končící kontury písmen
- příkazy, které sestavují jednotlivé kontury
- příkazy pro práci s podprocedurami
- příkazy definující hintovací informace pro jednotlivé znaky
- aritmetické příkazy

Každá procedura ve slovníku **CharStrings** má následující strukturu:

```
% Definice procedury
% Jméno procedury
/jméno {
   % Left sidebearing point
   x y hsbw
   ...
   % Konec definice znaku
   endchar
   } |-
```

Tato sekvence příkazů definuje proceduru **jméno**. Každá procedura ve slovníku fontu začíná příkazem **hsbw** (případně jeho variantou **sbw**). Příkaz **hsbw**

očekává na zásobníku dvě hodnoty (x a y). Hodnota x udává x-ovou souřadnici levého okraje bounding boxu písmene (tzv. *left sidebearing point*). Druhá hodnota (y) udává šířku znaku. Tento příkaz také nastaví aktuální bod na souřadnice $(x, 0)$. K aktuálnímu bodu se vztahuje většina příkazů, které vytvářejí kontury písmene. Aktuální bod ale není součástí definice vlastní kontury, a proto je možné použít dále zmíněné příkazy pro nastavení počátečního bodu kontury.

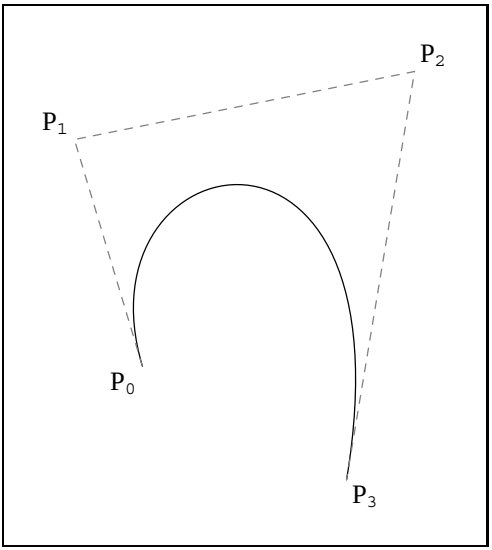
Každá procedura ve slovníku **CharStrings** musí končit příkazem **endchar**, který ukončuje definici kontur písmene a předává řízení proceduře **BuildChar**. Tato procedura volá speciálně upravené verze postscriptových procedur **stroke** a **fill**, které definované kontury vykreslí (v případě, že hodnotou položky **PaintType** ve slovníku fontu je 2) nebo vyplní (pokud je hodnota položky **PaintType** rovna 0).

Specifikace formátu Type 1 [9, strana 52] definuje tři příkazy pro změnu aktuálního bodu. Jsou to **rmoveto**, **hmoveto** a **vmoveto**. Příkaz **rmoveto** očekává na zásobníku dvě čísla (dx a dy) a posouvá aktuální bod o vektor (dx, dy). Pokud je tedy aktuální bod před voláním tohoto příkazu (x, y), bude po provedení tohoto příkazu aktuální bod umístěn na souřadnicích $(x + dx, y + dy)$. Tento příkaz tedy funguje naprosto stejně jako stejnojmenný příkaz jazyka PostScript [6, strana 483]. Formát Type 1 také definuje dvě jednoduché modifikace příkazu **rmoveto**. Příkazy **hmoveto** resp. **vmoveto** očekávají na zásobníku pouze jedno číslo (dx resp. dy) a jsou ekvivalentní příkazům `dx 0` **rmoveto** resp. `0 dy` **rmoveto**. Slouží tedy k horizontálním či vertikálním posunům aktuálního bodu.

Další skupina příkazů slouží k doplnění úsečky do kontury písmene. Stejně jako u příkazů pro změnu aktuálního bodu existuje jeden obecný příkaz a z něj odvozené speciální příkazy, existuje příkaz **rlineto**, který očekává na zásobníku dva argumenty (dx a dy). Pokud je aktuální bod před provedením tohoto příkazu umístěn na souřadnicích (x, y), bude do aktuální cesty (kontury) přidána úsečka s krajními body (x, y) a $(x + dx, y + dy)$. Příkaz tedy funguje stejně jako příkaz **rlineto** jazyka PostScript [6, strana 482]. Aktuálním bodem po provedení tohoto příkazu bude koncový bod přidané úsečky. Existují i speciální příkazy **hlineto** a **vlineto**, které očekávají na zásobníku pouze jeden argument a přidávají do cesty horizontální resp. vertikální úsečku. Příkaz `dx` **hlineto** je tedy ekvivalentní příkazu `dx 0` **rlineto** a `dy` **vlineto** má stejný význam jako `0 dy` **rlineto**.

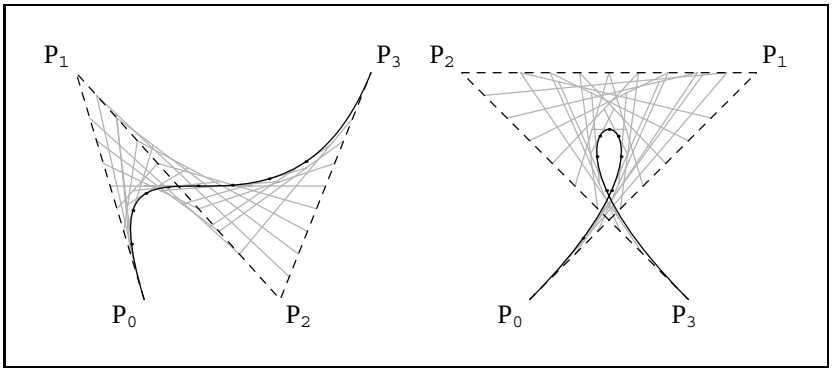
Poslední skupinu příkazů pro modifikaci kontur tvoří příkazy, které přidávají do aktuální cesty tzv. *Bézierovy křivky*. Tyto aproximační křivky nezávisle na sobě v letech 1952–62 používali P. de Casteljau a P. E. Béziere [13, strana 178]. Bézierovy křivky jsou zadány svým počátečním (P_0) a koncovým bodem (P_3) a dvojicí *kontrolních bodů* (P_1, P_2), které ovlivňují výsledný tvar křivky.

Na obrázku 3 je znázorněn „přirozený“ tvar Bézierovy křivky včetně kontrolních bodů a tzv. *kontrolního polygonu*. Pokud budou kontrolní body umístěny v rovině jiným způsobem, bude také tvar křivky odlišný (viz obrázek 4). Bézierova křivka se dokonce může sama protínat. Speciální volbou řídících bodů je



Obrázek 3: Bézierova křivka

možné dosáhnout i toho, že Bézierova křivka se transformuje v úsečku (pokud budou všechny body kolineární) případně i v bod (pokud všechny body splynou). Využívat těchto vlastností ve fontech Type 1 je zbytečné, neboť pro vykreslení úsečky existuje optimalizovaná funkce **rlineto** a vykreslení bodu nemá žádný význam.



Obrázek 4: Bézierovy křivky s jinými kontrolními body

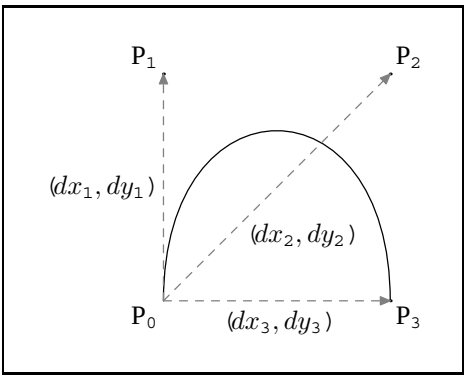
Matematický popis Bézierových křivek je dán následujícími vztahy:

$$P(t) = P_0 B_0^3(t) + P_1 B_1^3(t) + P_2 B_2^3(t) + P_3 B_3^3(t)$$

$$B_i^n(t) = \binom{n}{i} t^i (1-t)^{n-i}$$

kde P_0, P_1, P_2 a P_3 jsou jednotlivé zadané body, $t \in [0,1]$; B_0^3, B_1^3, B_2^3, B_3^3 jsou tzv. *Bernsteinovy kubické polynomy* (Bernsteinovy polynomy třetího stupně), které jsou definovány výše uvedeným obecným vztahem (v našem případě se jedná o polynomy třetího stupně, tj. $n = 3$).

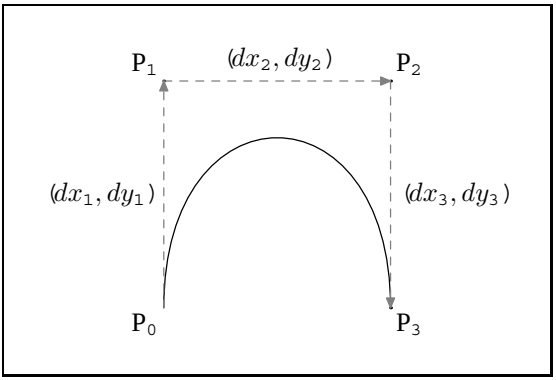
Bézierovy křivky jsou v jazyce PostScript implementovány příkazem **curveto** [6, strana 393], který do aktuální cesty vloží Bézierovu křivku. Tento příkaz má pouze šest parametrů (x-ové a y-ové souřadnice bodů P_1, P_2, P_3). Počáteční bod (P_0) je dán aktuálním bodem. Relativní obdoba tohoto příkazu (příkaz **rcurveto**) očekává na zásobníku opět šestici parametrů. První dvojice (dx_1, dy_1) udává relativní umístění druhého bodu od prvního. Druhá dvojice (dx_2, dy_2) udává umístění třetího bodu vůči prvnímu a poslední dvojice (dx_3, dy_3) reprezentuje vektor z počátečního do koncového bodu. Význam jednotlivých parametrů příkazu **rcurveto** je znázorněn na obrázku 5.



Obrázek 5: Argumenty postscriptového příkazu **rcurveto**

Formát Type 1 používá mírnou modifikaci tohoto příkazu, která očekává také šest argumentů: `dx1 dy1 dx2 dy2 dx3 dy3` **rrcurveto**. Tento příkaz přidá do aktuální cesty Bézierovu křivku, jejíž počáteční bod bude shodný s aktuálním bodem. První kontrolní bod křivky bude posunut o vektor (dx_1, dy_1) stejně jako u postscriptového příkazu **rcurveto**, ale druhý kontrolní bod bude posunut o vektor (dx_2, dy_2) od prvního kontrolního bodu (narozdíl od příkazu **rcurveto**

jazyka PostScript, kdy je druhý kontrolní bod posunut o daný vektor od počátku křivky). Koncový bod křivky bude posunut o vektor (dx_3, dy_3) od druhého kontrolního bodu. Grafická reprezentace jednotlivých parametrů je zřejmá z obrázku 6.



Obrázek 6: Argumenty příkazu **rrcurveto** ve formátu Type 1

Stejně jako u příkazů pro přidání úsečky do aktuální kontury existují i zjednodušené verze příkazu **rlineto**, je možné ve formátu Type 1 použít zjednodušenou podobu příkazů pro Bézierovy křivky. Příkaz `dx1 dx2 dy2 dy3` **hvcurveto** je ekvivalentní příkazu `dx1 0 dx2 dy2 0 dy3` **rrcurveto** a příkaz `dy1 dx2 dy2 dx3` **vhcurveto** je ekvivalentní příkazu `0 dy1 dx2 dy2 dx3 0` **rrcurveto**. Obě zjednodušené verze eliminují dva argumenty, protože předpokládají horizontální či vertikální směr tečných vektorů v krajních bodech křivky.

Posledním příkazem ovlivňujícím kontury písmene je příkaz **closepath**, který uzavírá aktuální konturu. Každá kontura by měla být uzavřena pomocí tohoto příkazu. Pokud by tomu tak nebylo, mohlo by dojít k nepříjemným efektům špatného vyplnění kontury písmene či přesahů při vykreslování kontur. Jazyk PostScript má podobný příkaz [6, strana 375], pouze s tím rozdílem, že ve formátu Type 1 zůstává aktuální bod nezměněn, zatímco u postscriptové varianty je aktuální bod přesunut na konec cesty a dojde tedy k jeho posunu.

Třetí skupinu příkazů formátu Type 1 tvoří příkazy pro práci s procedurami. Stejně jako u jiných programovacích jazyků slouží procedury pro častěji se opakující posloupnosti příkazů. Specifikace formátu Type 1 doporučuje používat procedury např. pro specifikace hintovacích informací (viz dále) nebo pro sekvence vytvářející častěji se opakující části kontur jednotlivých písmen. Procedury jsou uloženy v poli **Subrs** ve slovníku **Private**. Každá procedura v tomto poli má následující strukturu:

```
% Definice procedury v poli Subrs
% Procudera bude uložena pod indexem číslo_procedury
dup číslo_procedury {

   % Tělo procedury
   seznam příkazů...

   % Ukončení procedury a návrat do volajícího kódu
   return
}
```

Příkaz jazyka PostScript **dup** [6, strana 404] zduplikuje v poli **Subrs** odkaz proceduru se zadaným pořadovým číslem. Příkaz **return** vrací řízení zpět a procedura je tím ukončena. Definované procedury je možné volat pomocí dalších příkazů formátu Type 1. Mezi tyto příkazy patří **callsubr** a **callothersubr**. První z těchto příkazů slouží k volání procedur definovaných v poli **Subrs**, druhý k volání procedur z pole **OtherSubrs**. Některá písmena mohou být definována pouze pomocí volání procedur. Příkladem takové definice je například velké písmeno I s čárkou z fontu ITC Anna:

```
% Definice procedury /Iacute
/Iacute {
   % Left sidebearing point
   2 185 hsbw
   % Vertikální dřík
   153 callsubr
   % Umístění akcentu
   130 84 rmoveto
   % Vlastní akcent
   95 callsubr
} |-
```

Obě kontury písmene jsou definovány pomocí procedur a je tudíž velmi jednoduché tyto části (svislý dřík znaku I i akcent) použít k definici jiného písmene.

Při podrobnějším studiu Type 1 fontů se můžeme setkat i s příkazy **pop** a **setcurrentpoint**, které se používají v souvislosti s voláním procedur v poli **OtherSubrs**.

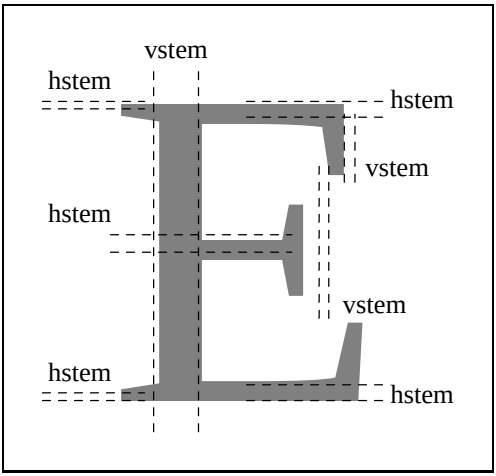
Formát Type 1 dále podporuje jediný příkaz sloužící k aritmetickým operacím. Příkaz **div** funguje stejně jako stejnojmenný příkaz jazyka PostScript [6, strana 404]. Na zásobníku očekává dva operandy, první z nich vydělí druhým a výsledek uloží na zásobník. Tento příkaz není ve většině fontů použit, neboť je poměrně pomalý a zejména na pomalejších počítačích by mohlo jeho použití výrazně zpomalit vykreslování.

Hintovací informace

Tvůrci formátu Type 1 při návrhu formátu vzali v úvahu i možnost jeho použití při nízkých rozlišeních, kdy často závisí na každém bodu. I proto umožňuje

formát Type 1 specifikovat tzv. *hintovací informace*, které upravují chování interpretu Type 1 fontů ve speciálních situacích. Každý *hint* (horizontální i vertikální) je deklarován dvěma souřadnicemi. Význam jednotlivých hintů je patrný z obrázku 7.

Definice písmene E z fontu Times Roman obsahuje celkem osm hintů (pět horizontálních a tři vertikální). S hintovacími informacemi pracuje přímo procedura **BuildChar** vestavěná v postscriptovém interpretu.

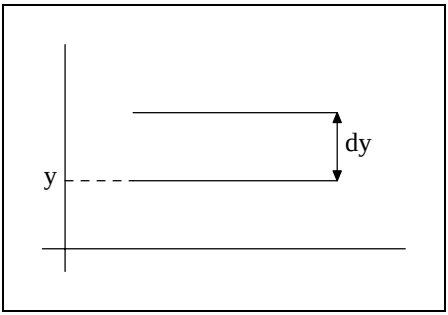


Obrázek 7: Hintovací informace písmene E z fontu Times Roman

Formát Type 1 obsahuje několik příkazů, které specifikují hinty. Horizontální hinty je možné specifikovat pomocí příkazů **hstem** a **hstem3**. Oba tyto příkazy používají pro specifikaci hintu dvou argumentů. První z nich je y-ová souřadnice spodní části hintu a druhý argument je výška hintu (viz obrázek 8).

Příkaz **hstem** očekává dva argumenty a jeho syntaxe je tedy `y dy` **hstem**. Příkaz **hstem3** umožňuje najednou specifikovat tři hinty a tudíž očekává šest argumentů zadaných podobně jako u příkazu **hstem**: `y0 dy0 y1 dy1 y2 dy2` **hstem3**. Podmínkou použití příkazu **hstem3** je stejná šířka všech hintů a umístění prostředního hintu přesně uprostřed mezi oběma krajními hinty. Tento příkaz je tedy možné použít např. pro specifikaci hintů symbolu matematické ekvivalence ($\equiv$).

Vertikální hinty se specifikují obdobně. Příkaz **vstem** očekává dva argumenty: `x dx` **vstem** a definuje vertikální hint s x-ovou souřadnicí mezi x a $x+dx$. Příkaz **vstem3** má podobnou syntaxi jako **hstem3** a používá se zejména pro specifikaci hintů písmene m a dalších se třemi vertikálními tahy.



Obrázek 8: Horizontální hinty

Formát Type 1 obsahuje ještě jeden příkaz, který informuje postscriptový interpret o specifické vlastnosti právě vytvářené kontury. Tento příkaz se jmenuje **dotsection** a používá se v páru. Jeho význam bude zřejmý z výpisu části definice písmene j:

```
   dotsection
   21 35 17 37 hvcurveto
   31 -26 26 -30 vhcurveto
   -30 -25 -25 -32 hvcurveto
   -38 34 -16 21 vhcurveto
   closepath
dotsection
```

Příkaz **dotsection** zde ohraničuje definici tečky, která je použita např. v písmenech reprezentujících znaky j nebo vykřičník. Tento příkaz je vhodný spíše pro použití se staršími interprety, které špatně interpretovaly samostatné kontury (obrysy tečky) a někdy došlo i k jejich slití s hlavní konturou písmene, což je nepřípustné.

Příklad Type 1 fontu

Stejně jako u formátu Type 3 jsme si definovali vlastní font s jedním znakem, vytvoříme nyní font ve formátu Type 1. Náš font bude obsahovat pouze jediné písmeno, které reprezentuje znak I. Samozřejmě nebudeme vytvářet přímo podobu vhodnou pro začlenění do postscriptového souboru, ale využijeme balíku t1utils, který zjednodušuje práci s jednotlivými podobami Type 1 fontů.

```
%!PS-AdobeFont-1.0: MyTypeOne 001.000
%%CreationDate: Sun Feb 13 15:32:57 2000
%%VMusage: 4031 7971
11 dict begin
```

```
/FontInfo 7 dict dup begin
/version (001.000) readonly def
/Notice (This is an example of Type 1 font.) readonly def
/FullName (My Type One) readonly def
/FamilyName (My Type One) readonly def
/Weight (Roman) readonly def
/isFixedPitch false def
/ItalicAngle 0 def
end readonly def
/FontName /MyTypeOne def
/Encoding StandardEncoding def
/PaintType 0 def
/FontType 1 def
/FontMatrix [0.001 0 0 0.001 0 0] readonly def
/FontBBox{0 0 800 800}readonly def
currentdict end
currentfile eexec
dup /Private 11 dict dup begin
/-||{string currentfile exch readstring pop}executeonly def
/Subrs 0 array def
2 index /CharStrings 2 dict dup begin
/I {
   0 800 hsbw
   100 0 rlineto
   0 800 rlineto
   -100 0 rlineto
   endchar
} def
/.notdef {
   0 800 hsbw
   endchar
} def
end
end
readonly put
noaccess put
dup/FontName get exch definefont pop
mark currentfile closefile
```

Výše uvedený text uložíme do souboru se jménem `MyTypeOne.disas` a převedeme jej do podoby PFB souboru (viz dále) pomocí programů `t1asm` a `t1binary` z balíku t1utils:

```
t1asm MyTypeOne.disas >MyTypeOne.pfa
t1binary MyTypeOne.pfa >MyTypeOne.pfb
```

Font je nyní připraven k použití postscriptovým interpretem. Tento fakt si můžeme ověřit prostým zavedením fontu do interpretu a jeho použitím. Na podporovaných operačních systémech (např. Linux) je možné využít např. interpretu Ghostscript:

```
(MyTypeOne.pfa) run
/MyTypeOne findfont 100 scalefont setfont
0 0 moveto
(IAI) show
```

Font je tedy možné využít v postscriptovém interpretu, ale některé další programy jej využít nemohou, protože nemáme k dispozici metrické informace. Vytvoříme tedy ještě soubor s těmito informacemi a ověříme jejich správnost např. pomocí programu pdfTEX. Metrické informace pro postscriptové fonty jsou většinou distribuovány v podobě AFM souboru.

```
StartFontMetrics 2.0
FontName MyTypeOne
EncodingScheme AdobeStandardEncoding
FullName My Type One
FamilyName My Type One
Version 001.000
FontBBox 0 0 800 800
StartCharMetrics 1
C 73 ; WX 800 ; N myonechar ; B 0 0 800 800 ;
EndFontMetrics
```

AFM soubor obsahuje minimum informací o fontu samotném (jeho jméno, číslo verze a bounding box). Jsou zde také uvedeny metrické informace našeho písmene I. Sázecí systém TEX potřebuje k použití tohoto fontu metrické informace ve své nativní podobě (v souboru ve formátu TFM) a tudíž jej budeme muset z AFM souboru vytvořit. Použijeme k tomu program `afm2tfm`, který je standardní součástí kterékoli distribuce TEXu:

```
afm2tfm MyTypeOne.afm
```

Abychom mohli použít náš font v systému pdfTEX, musíme jej ještě informovat o tom, kde má hledat vlastní font. pdfTEX totiž na rozdíl od TEXu potřebuje vědět, kde se font v našem souborovém systému nalézá, neboť jej musí vložit do výsledného souboru ve formátu PDF [12]. TEX do výsledného souboru ve formátu DVI vkládá pouze odkaz na font a nikoli font samotný. pdfTEX proto používá soubor `pdftex.map` [4], který naplníme následujícím řádkem:

```
MyTypeOne MyTypeOne <<MyTypeOne.pfb
```

Nyní je náš font připraven pro použití v systému pdfTEX.

Formáty souborů fontů Type 1

Výše popsaný formát souborů je možné poslat přímo postscriptové tiskárně, jejíž postscriptový RIP dokáže tento sedmibitový proud dat interpretovat [7, Font File Formats]. Totéž umí i softwarové interprety jazyka PostScript. Fonty ve

formátu Type 1 jsou v tomto formátu ukládány na discích počítačů v souborech s příponou `.pfa` a jsou přímo připraveny pro zavedení do postscriptového interpretu.

Tento formát však není vhodný pro ukládání souborů na disku, protože je poměrně náročný na diskový prostor. Hlavička fontu s definicí slovníků je v čitelné podobě a zakódovaná část je reprezentována řetězcem hexadecimálních číslic:

```
currentfile eexec
CF0EEFF329FE1DB159A37891F19957002E7D435F06065E5E0A71393EDE477128
A76E9CA6DA8DF0C353E1352AFD7FB01E92CBD565FC568222229162E248445364
52FEC4350886980508A70995587C8A9C47468FF87BA0F9AFD847390FD5215BAB
... zkráceno
9E5236D615703C5E421C4B129A95EA38D54EDD725C881F49E22678743289B3E9
EF9F262271CAB2BE4A8872CA6C8B9624A36451F78916E9DB508A003DA4B339EC
E06582EF04F84A
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
... zkráceno
0000000000000000000000000000000000000000000000000000000000000000
cleartomark
```

Je zřejmé, že uložení dat v tomto formátu je velmi neefektivní. Proto společnost Adobe vytvořila i binární formát určený pro reprezentaci fontů ve formátu Type 1 a právě tento formát je použit pro ukládání i distribuci fontů pro platformu PC. Příslušné soubory potom mají příponu `.pfb`.

Pro zavedení do postscriptového interpretu je nutné tento formát konvertovat zpět do textové podoby. Tato činnost je u většiny aplikací s postscriptovým výstupem automatizována a nemusíme jí věnovat pozornost. Pokud bychom vytvářeli soubor v PostScriptu ručně, je možné použít utilitu `pfb2pfa` z TEXové distribuce teTEX, program `t1ascii` z balíku t1utils nebo skript `pfbtopfa`, který je součástí postscriptového interpretu Ghostscript.

Formát souborů `.pfb` je velmi jednoduše navržen. Celý soubor je rozdělen do několika segmentů, které obsahují záhlaví a vlastní tělo. Segment může být jeden ze tří typů:

- text složený ze znaků ASCII tabulky (typ `0x01`),
- binární data (typ `0x02`),
- indikátor konce souboru (typ `0x03`).

První typ segmentu je použit pro hlavičku formátu Type 1, která je také v čitelné podobě. Segmenty prvního typu je tedy možné při překódování do formátu `.pfa` pro zavedení do postscriptového interpretu jenom překopírovat bez jakéhokoli zásahu (kromě konvertování znaků konce řádku). Binární data (druhý typ segmentu) by při převodu do formátu `.pfa` měla být konvertována do hexadecimální podoby. Segment posledního typu musí být posledním segmentem v souboru a měl by být v souboru obsažen právě jednou.

Hlavička každého segmentu obsahuje nejméně dva bajty. První z nich zahajuje segment a má vždy hodnotu `0x80` (127 v desítkové soustavě). Druhý bajt obsahuje typ segmentu (viz výše). Segment indikující konec souboru tedy obsahuje bajty `0x80 0x03`. Bajt `0x03` by měl být posledním bajtem souboru.

Datové segmenty mají hlavičku rozšířenu o čtyři bajty specifikující délku datové sekce sektoru, přičemž bajty jsou uspořádány podle rostoucí významnosti, první bajt je tedy nejméně významný. Ihned za těmito bajty následuje datová sekce sektoru.

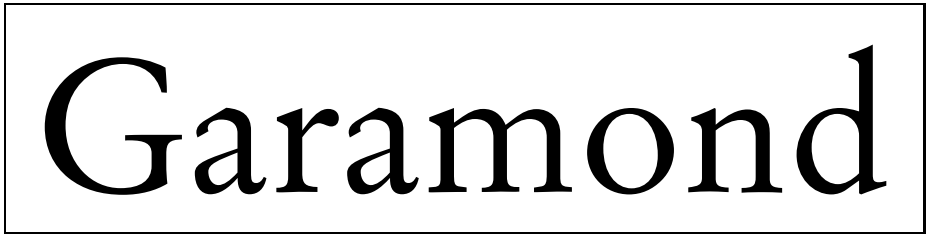
Výše uvedený text by měl po převodu do binární podoby (`.pfb`) obsahovat následující sekvence bajtů:

```
0x80 0x01 0xxx 0x00 0x00 0x00 ... následuje 0xxx znaků hlavičky fontu
0x80 0x02 0xxx 0x00 0x00 0x00 ... následuje 0xxx binárních bajtů

0xCF 0x0E 0xEF 0xF3 0x29 0xFE ... zkráceno

0x80 0x01 0x14 0x02 0x00 0x00 ... následuje 0x0214 znaků '0'
0x80 0x03
```

Soubor ve formátu `.pfb` může obsahovat libovolný počet datových segmentů, které mohou být libovolně kombinovány.

Písma ve formátu Multiple Master

V dobách minulých, kdy ještě nebyly k dispozici počítače a o fontech se tedy mohlo typografům a sazečům pouze zdát, vytvářely písmo umělci (*písmolijci*). Příkladem může být písmo Garamond původně navržené Claudem Garamondem v letech 1530–1540 [3].



Obrázek 9: Font Garamond

Ačkoli původní autor navrhl pouze několik velikostí a variací písma (základní písmo, italika, kapitálky), máme nyní k dispozici i tučné písmo a můžeme si vytvořit písmo zúžené apod. Technologie vektorových písem navíc umožňují

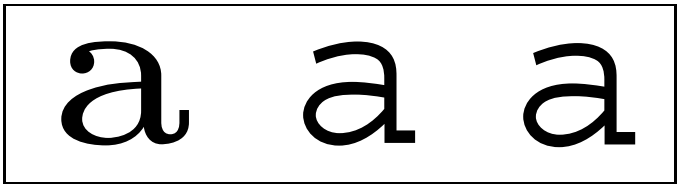
změnit velikost písma bez toho, abychom museli měnit vlastní řez písma. Máme tedy obrovskou výhodu před sazeči minulých dob, kteří museli požádat písmolijce o vytvoření další velikosti písma. Totéž platí i pro písma různých duktů. Dávní písmolijci téměř nevytvářeli písma tučná a nyní máme u většiny písem k dispozici celou škálu od velmi jemného písma přes polotučné až k tučným řezům. Na obrázku 10 je znázorněno písmo Humanist521BT v různých duktech.

Obrázek 10: Písmo různých duktů

Nevýhodou dnešního přístupu k písmům je stejný vzhled písma v malých i velkých velikostech. Dříve byla písma stejného řezu různých velikostí mírně odlišná (opticky škálovaná), nyní jsou písma ve větších velikostech pouze zvětšeninami písma v menších velikostech. Tento nedostatek je možné řešit tzv. *parametrizací písem* [3]. Tento mechanismus umožňuje písma popsaná pomocí kontur upravovat pouhou změnou několika (i mnoha) parametrů. Původcem tohoto mechanismu je pravděpodobně Donald E. KnuthRejstříkDonald E.Knuth, který jej implementoval v systému METAFONT.

Na obrázku 11 je mechanismus optického škálování znázorněn na písmenech a z fontů cmr5, cmr10 a cmbx10 z rodiny Computer Modern ve velikosti 3 cm. Rozdíly v kresbě jednotlivých písmen jsou patrné zejména z chvostu a paty písmene. Zrno je zejména u písmene a z tučného fontu cmbx10 nevýrazné, zatímco u fontů cmr5 a cmr10 je zřetelný přechod mezi zrnem chvostu a horní částí písmene. Pata je u tučného písmene a z fontu cmbx10 redukována.

Podobný mechanismus se pokusila do svého formátu Type 1 [9] implementovat i společnost Adobe, která v lednu roku 1994 zveřejnila dodatek ke specifikaci formátu Type 1 [10]. Tento dodatek doplňuje specifikaci fontů Type 1 o tzv. *Multiple Master fonty*. Multiple Master fonty byly společností Adobe používány již v roce 1992, tedy ještě před zveřejněním jejich specifikace.



Obrázek 11: Princip optického škálování

Obrázek 12: Multiple Master font s jednou osou

Type 1 font obsahuje definici jednoho řezu písma a postscriptový interpret může tento řez zvětšit, či zmenšit, vyplnit či jen vykreslit apod. Multiple Master font může obsahovat až osm párů těchto řezů, které jsou nazývány *hlavními* (master design). Každá dvojice definuje jednu „souřadnou osu“ a všechny hlavní řezy společně definují bázi prostoru fontu. V nejjednodušším případě bude Multiple Master font obsahovat pouze dva hlavní řezy, které budou definovat souřadnou osu jedné charakteristické vlastnosti písma (např. duktu). Multiple Master font znázorněný na obrázku 12 obsahuje dva hlavní řezy. První je znázorněn vlevo od souřadné osy (základní písmo). Druhý řez je tučná varianta téhož písma. Souřadná osa (design axis) tedy definuje duktus fontu. Výsledný font je funkcí parametru α, který specifikuje uživatel fontu. Parametr α musí být reálné číslo z intervalu $[0, 1]$. Pokud uživatel specifikuje hodnotu α rovnou 0 nebo 1, získá původní hlavní řez. Pokud specifikuje hodnotu z vnitřku intervalu, výsledný font vznikne lineární interpolací mezi hlavními řezy.

Multiple Master font může obsahovat více těchto os (maximálně však čtyři), a proto bude výsledný font lineární interpolací hlavních řezů závislou na vektoru parametrů:

$$(\alpha_1, \alpha_2, \alpha_3, \alpha_4) \in [0, 1]^4$$

V Multiple Master fontu mohou být kromě hlavních řezů, které specifikují souřadnou osu, ještě další hlavní řezy zjemňující souřadnou osu (intermediate

designs). Celkový počet hlavních řezů však stále musí být menší nebo roven 16. Pokud budeme chtít použít v Multiple Master fontu čtyři osy, nebudeme moci použít zjemňující hlavní řez. To je obrovská nevýhoda, protože je velmi obtížné lineární interpolací krajních hlavních řezů dosáhnout kvalitního písma. Je také otázkou, zda je možné vzdálenosti v typografii interpolovat lineárně. Yannis Haralambous se v článku Parametrization of PostScript fonts through METAFONT—an alternative to Adobe Multiple Master fonts [3] domnívá, že kvadratická aproximace je pro tento účel vhodnější. Dalším důležitým aspektem zdůrazňujícím nedokonalost designu specifikace Multiple Master fontů je jiné rozmístění či počet kontrolních bodů pro definici jiných řezů písma. Tento aspekt není vůbec v Multiple Master fontech zohledněn.

Společnost Adobe však na počátku roku 2000 oznámila, že ukončuje podporu fontů Multiple Master a tak potvrdila faktický stav podpory Multiple Master fontů, jejichž specifikace je sice veřejně k dispozici, ale neobsahuje přesné a výstižné informace, nejsou k dispozici vzorové fonty, komerční Multiple Master fonty jsou příliš drahé, neexistují (až na velmi drahé komerční nástroje) utility pro jejich tvorbu a podpora ze strany aplikací je stále zanedbána. Opět se tak prokázala neúspěšnost uzavřených standardů a specifikací.

Písma ve formátu Type 42

Jazyk PostScript je určen pro popis tiskové strany a je používán v mnoha různých typech tiskáren. Postscriptové tiskárny (tedy tiskárny vybavené přímo postscriptovým interpretem či kartou implementující jazyk PostScript) jsou používány pro tisk i z operačních systémů používajících pro reprezentaci písem fonty, které nejsou přímo podporované jazykem PostScript. Např. operační systémy Windows společnosti Microsoft používají v grafickém uživatelském rozhraní technologii TrueType a pro tisk dokumentů obsahující tato písma je nutné vlastní kontury písmen konvertovat buď přímo do formátu Type 1 nativně podporovaného postscriptovým interpretem tiskárny nebo do bitmap.

Nevýhodou prvního přístupu (tedy konverze do Type 1) je ztráta hintovacích informací, které jsou ve formátu TrueType podstatně výkonnější, a také nepřesné kontury, neboť ve formátu TrueType jsou použity kvadratické B-spliny místo kubických Bézierových křivek formátu Type 1. Naproti tomu druhý přístup (konverze do bitmap) je naprosto nevyhovující pro velikost bitmap, které jsou zejména při velkých rozlišeních objemné.

Výše popsaná situace je tedy (resp. byla) zdánlivě neřešitelná. Společnost Adobe však do svého postscriptového interpretu zabudovala rasterizér fontů ve formátu TrueType a nová verze specifikace jazyka PostScript Level 3 již tyto fonty podporuje v podobě fontů ve formátu Type 42, který tvoří jakousi

postscriptovou obálku nad formátem TrueType. Specifikace formátu Type 42 [11] je opět veřejná a je k dispozici na serverech společnosti Adobe.

Písma ve formátu Type 42 mají podobnou strukturu jako Type 1. Definují slovník fontu o téměř stejném obsahuje. Slovník fontu Type 42 obsahuje navíc položku **sfnts**, která reprezentuje vlastní TrueType font v mírně pozměněné podobě. Položka **FontType** musí obsahovat hodnotu 42. Stejně jako u Type 1 fontů je předepsána (doporučena) i podoba první řádky souboru obsahujícího TrueType font:

```
%!PS-TrueTypeFont-TTVerze-VerzeFontu
```

Číslo *TTVerze* definuje číslo specifikace formátu TrueType (viz kapitola TrueType fonty) a číslo *VerzeFontu* obsahuje číslo verze vlastního fontu. Na následujícím výpise si uvedeme příklad fontu Verdana, který je distribuován společností Microsoft v operačních systémech Windows:

```
%!PS-TrueTypeFont-XXX-65536
%%VMusage: 0 0
11 dict begin
  /FontType 42 def
  /FontMatrix [ 1.0 0.0 0.0 1.0 0.0 0.0 ] def
  /FontName /Verdana def
  /FontInfo 8 dict dup begin
    /Notice (Typeface and data (c) 1996 Microsoft Corporation.
             All Rights Reserved) readonly def
    /UnderlineThickness 0.0585938 def
    /isFixedPitch false def
    /Version (Version 2.10) readonly def
    /UnderlinePosition -0.0878906 def
    /FullName (Verdana) readonly def
    /ItalicAngle 0.0 def
    /FamilyName (Verdana) readonly def
  end readonly def
  /Encoding 256 array
    dup 0 /.notdef put
    dup 32 /space put
    dup 33 /exclam put
    ... zkráceno
    dup 254 /thorn put
    dup 255 /ydieresis put
  readonly def
  /FontBBox [ -0.0498047 -0.206543 1.44678 1.03369 ] def
  /PaintType 0 def
  /sfnts  [
00010000001201000004002040C5453487930F0C20000012C000002474F532F32267590C3
0000037400000056564444D5874F17C6D000003CC000005E0636D617037C4A855000009AC
0000059C637674204CA740E300000F4800000198667067 6DEE371553000010E000000138
... zkráceno
23442B732B2B2B2B2B2B2B2B2B2B2B2B2B2B2B2B2B2B2B737373737373737373732B732B732B2B2B
2B2B73732B2B2B2B732B2B2B2B00752B2B2B2B2B2B2B2B2B2B2B2B2B2B2B2B2B2B2B2B2B2B2B73
747373742B73742B00>] def
```

```
  /CharStrings 579 dict dup begin
    /onehalf 242 def
    /Igrave 206 def
    ... zkráceno
    /q 84 def
    /kgreenlandic 358 def
  end readonly def
FontName currentdict end definefont pop
```

V první části výpisu je definován typ fontu a matice přechodu od báze souřadného systému písmene k uživatelskému souřadnému systému. Tato matice definuje identickou transformaci na rozdíl od matice přechodu obvyklé u Type 1 fontů. Následuje definice jména fontu a slovník **FontInfo**, kódovací vektor a samotný TrueType font. V závěru je již definován pouze slovník **CharStrings**, který obsahuje pouze odkazy na jednotlivé procedury definované uvnitř TrueType fontu.

Bibliografie

[1] Vydavatelství Úřadu pro normalizaci a měření. *Polygrafické názvosloví: Písmo, písmařství a písmolijectví.* Listopad 1974.

[2] Lindy Amato. *PostScript, Pre-Press a barva.* Computer Press, a. s., 1. edition, 1996.

[3] Yannis Haralambous. *Parametrization of PostScript fonts through METAFONT—an alternative to Adobe Multiple Master fonts.* Electronic Publishing Vol 6(3), Září 1993.

[4] Hans Hagen Thành, Sebastian Rahtz. *The pdfTEX user manual.* Září 1999.

[5] Adobe Systems Incorporated. *Adobe Type 1 Fonts–Communication Handbook.*

[6] Adobe Systems Incorporated. *PostScript Language Reference Manual.* Addison-Wesley Publishing Company, Inc., Reading, MA, USA, 2. edition, 1990.

[7] Adobe Systems Incorporated. *Supporting Downloadable PostScript Language Fonts (Technical Note Nr. 5040).* Březen 1992.

[8] Adobe Systems Incorporated. *Supporting Font in the PostScript Language Environment (Technical Note Nr. 5075).* Březen 1992.

[9] Adobe Systems Incorporated. *Adobe Type 1 Font Format.* Addison-Wesley Publishing Company, Inc., Reading, MA, USA, 3. edition, 1993.

[10] Adobe Systems Incorporated. *Type 1 Font Format Supplement (Technical Specification Nr. 5015).* Leden 1994.

[11] Adobe Systems Incorporated. *The Type 42 Font Format Specification (Technical Note Nr. 5012).* Leden 1994.

[12] Adobe Systems Incorporated. *Portable Document Format Reference Manual Version 1.3*. 1. edition, Březen 1999.
[13] Jiří Žára a kolektiv. *Počítačová grafika – principy a algoritmy*. GRADA, a. s., 1. edition, 1992.
[14] Adobe Developer Support. *The Compact Font Format Specification*. Adobe Systems Incorporated., 1. edition, Prosinec 1996.

Pavel Janík ml.
<Pavel.Janik@linux.cz>

Výroční cena CSTUGu

Výbor Československého sdružení uživatelů TEXu udělil výroční cenu

Štěpánu Potockému

za digitalizaci starších čísel Zpravodaje (do roku 1995 včetně). Práce byla oceněna finanční odměnou ve výši 5000 Kč.

Zpravodaj Československého sdružení uživatelů TEXu
ISSN 1211-6661

Vydalo:	Československé sdružení uživatelů TEXu vlastním nákladem jako interní publikaci
Obálka:	Bohumil Bednář
Počet výtisků:	800
Uzávěrka:	6. června 2000
Odpovědný redaktor:	Zdeněk Wagner
Tisk a distribuce:	KONVOJ, spol. s r. o., Berkova 22, 612 00 Brno, tel. 05-740233
Adresa:	CSTUG, c/o FI MU, Botanická 68a, 602 00 Brno
fax:	05–412 125 68
e-mail:	cstug@cstug.cz

Zřízené poštovní aliasy sdružení CSTUG:

`bulletin@cstug.cz`, `zpravodaj@cstug.cz`
korespondence ohledně Zpravodaje sdružení

`board@cstug.cz`
korespondence členům výboru

`cstug@cstug.cz`, `president@cstug.cz`
korespondence předsedovi sdružení

`cstug-members@cstug.cz`
korespondence členům sdružení

`cstug-faq@cstug.cz`
řešené otázky s odpověďmi navrhované k zařazení do dokumentu CSFAQ

`secretary@cstug.cz`, `orders@cstug.cz`
korespondence administrativní síle sdružení, objednávky CD-ROM

`bookorders@cstug.cz`
objednávky tištěné TEXové literatury na dobírku

ftp server sdružení:
`ftp://ftp.cstug.cz/`

www server sdružení:
`http://www.cstug.cz/`

Podávání novinových zásilek povoleno Českou poštou, s. p. OZJM Ředitelství v Brně č. j. P/2–1183/97 ze dne 11. 3. 1997.