CSTUG

ZPRAVODAJ

Československého sdružení uživatelů TEXu

1

2013

ISSN 1211-6661
Tištěná verze

ISSN 1213-8185
Online verze

Ročník 23

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/`.

Zpravodaj je zařazen do Seznamu recenzovaných neimpaktovaných periodik vydávaných v České republice, viz `http://www.vyzkum.cz/`.

Své příspěvky do Zpravodaje můžete zasílat v elektronické podobě, nejlépe jako jeden archivní soubor (`.zip, .arj, .tar.gz`). Postupujte podle instrukcí, které najdete na stránce `http://bulletin.cstug.cz/`. Pokud nemáte přístup na Internet, můžete zaslat příspěvek na disketě, CD, či DVD na adresu:

Zdeněk Wagner
Vinohradská 114
130 00 Praha 3

`zpravodaj@cstug.cz`

Nezapomeňte přiložit všechny soubory, které dokument načítá (s výjimkou standardních součástí TEX Live), zejména v případě, kdy vás nelze kontaktovat e-mailem.

ISSN 1211-6661 (tištěná verze)
ISSN 1213-8185 (online verze)

Úvodník

Zdeněk Wagner

Milé čtenářky, vážení čtenáři,

právě otvíráte první číslo Zpravodaje ročníku 2013. Zpravodaj vychází po delší pauze a podařilo se jej sestavit zejména díky aktivnímu přístupu Tomáše Hály, který tomu věnoval jistě hodně práce i svého času. Patří mu za to velké poděkování.

V tomto čísle najdete tři články, které se věnují grafice v TEXu, každý z trochu jiného úhlu pohledu autorů Petra Olšáka, Romana Plcha a Zdeňka Wagnera. Jsou doplněny texty Tomáše Fábryho o fontu Comenia a Petra Olšáka o záložkách. Poslední článek, kde Tomáš Hála popisuje své zkušenosti s ConTEXtem, vhodně doplňuje články Víta Zýky, které vyšly ve Zpravodaji č. 4 z roku 2008.

V redakci se již chystají další čísla Zpravodaje. Redaktoři oslovili autory přednášek z konferencí TEXperience 2012 a 2013, aby zaslali ke zveřejnění plné texty. Redaktoři věří, že autoři najdou dostatek času i sil, překonají aktivační bariéru, své texty skutečně napíší a pošlou, abyste další čísla dostali co nejdříve. Informace, které zatím zazněly jen ústně na konferencích, se tak stanou dostupnými i pro ty, kdo na TEXové akce přijet nemohli.

Připravuje se i jedno zvláštní číslo, které ale má zatím zůstat překvapením. Aby to bylo skutečně překvapení, není možno prozradit, co v tomto čísle bude. Není ani možné prozradit jméno autora. Zatím prozradíme jen tolik, že autora dobře znáte. Je to autor, od něhož jste si již mohli přečíst řadu článků, a ti z vás, kdo jezdí na konference TEXperience, si mohli vyslechnout několik jeho přednášek. Redakce věří, že to bude překvapení příjemné.

Při vydávání Zpravodaje se nabízí možnost přebírání článků ze zahraničních zdrojů, ať už v originálu, nebo v českém či slovenském překladu. Redakce zahraničních TEXových časopisů nám obvykle vycházejí vstříc a přetisk i překlad povolují, ale myslíme si, že to není správná cesta. Přebírání nejzajímavějších článků ze zahraničních zdrojů se nebráníme, ale stále klademe důraz na to, aby v tomto časopise byl vytvořen prostor zejména pro české a slovenské autory. Články se nemusejí zabývat jen specifiky české a slovenské sazby, protože i zde řada uživatelů TEXu připravuje sazbu v cizích jazycích, případně sazbu vícejazyčnou, podílejí se na mezinárodních projektech při tvorbě makrobalíčků apod.

Zpravodaji bych na závěr chtěl popřát, aby měl dostatek autorů, kteří budou posílat zajímavé články, činorodé redaktory, kteří články zpracují a sestaví do jednotlivých čísel, a časopis tím získal mnoho spokojených čtenářů a čtenářek.

Zdeněk Wagner
šéfredaktor

Fonty Comenia pre systém TEX

Tomáš Fábry

Tento článok popisuje podporu fontov Comenia pre systém TEX, ktorá vznikla ako súčasť mojej diplomovej práce [1]. V úvode je načrtnutá motivácia, ktorá viedla k vzniku tejto podpory. Následne sa čitateľ môže dozvedieť niekoľko informácií o jednotlivých rodinách Comenia. V ďalších kapitolách je popísaný proces tvorby podpory pre OpenType fonty a tiež návod na inštaláciu a použitie balíčka s podporou.

Kľúčové slová:

Comenia, fontspec, OpenType fonty

1. Úvod

Fonty Comenia vznikli v rámci projektu, ktorý je reakciou na pozvoľné znižovanie kvality typografickej úpravy školských tlačovín v Českej republike. Tento úpadok je výsledkom snahy mnohých vydavateľstiev, ktoré sa usilujú minimalizovať výrobné náklady, a to často aj veľmi nevhodným spôsobom (napr. znižovaním počtu stránok pomocou zmenšovania či zužovania písma, šetrením na grafickej úprave či väzbe...) [2].

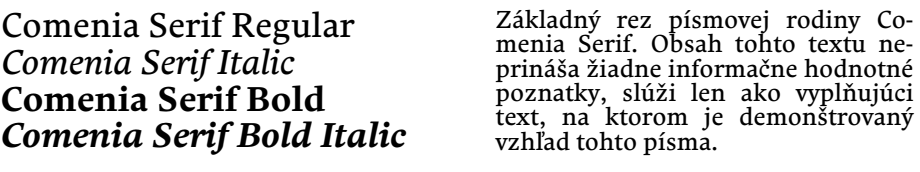
Medzi hlavné nedostatky grafickej úpravy učebných materiálov patrí nevhodne zvolené písmo. Autori často používajú písma, ktoré boli pôvodne navrhnuté na úplne iné účely (napr. pre noviny). Takéto písma môžu znížiť pozornosť čitateľa, čo následne vedie k horšiemu porozumeniu samotného textu.

Písma Comenia boli špeciálne navrhnuté na použitie v školstve. Ich tvary, proporcie a estetické kvality sú upravené tak, aby eliminovali únavu očí a viedli k lepšiemu vzhľadu školských materiálov [2].

Fonty Comenia sú šírené tak, ako väčšina súčasných fontov, vo formáte OpenType. Tento formát avšak nie je priamo podporovaný v systéme TEX[1], ktorý sa často používa na prípravu učebných materiálov na akademickej pôde. Táto skutočnosť bola hlavnou motiváciou, ktorá viedla k vzniku TEXovej podpory pre fonty Comenia, o ktorej pojednáva tento článok.

[1] S výnimkou najnovších TEXových formátov XƎTEX a LuaTEX.

 doi: 10.5300/2013-1/2



Obr. 1: Písmová rodina Comenia Serif

2. Predstavenie fontov Comenia

2.1. Písmová rodina Comenia Serif

Autorom rodiny Comenia Serif je František Štorm, popredný český typograf, návrhár písma a zakladateľ spoločnosti „Střešovická písmolijna“.

Táto rodina je určená na sadzbu dlhých textov učebníc, diplomových a vedeckých prác alebo na sadzbu šlabikárov. Diakritika ladí s kresbou malej aj veľkej abecedy a plne rešpektuje zvyklosti stredoeurópskych jazykov [2]. Comenia Serif ponúka jeden základný a tri vyznačovacie rezy písma, ktoré môžeme vidieť na obrázku 1.

2.2. Písmová rodina Comenia Sans

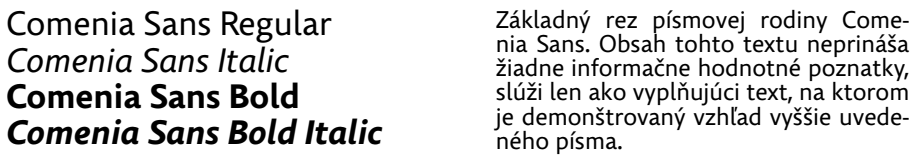
Autorom rodiny Comenia Sans je Tomáš Brousil, český grafik, návrhár písma a zakladateľ spoločnosti „Suitcase Type Foundry“.

Comenia Sans bola vytvorená ako bezpätkový variant písmovej rodiny Comenia Serif. Tieto písmové rodiny majú mnoho spoločných charakteristík ako veľkosť verzálok a mínusok, dĺžku doťahov, ale aj váhu. Vďaka tomu je možné kombinovať použitie fontov oboch rodín aj v rámci toho istého riadku.

Písmová rodina Comenia Sans však na rozdiel od Comenia Serif neobsahuje žiadne okrasné elementy a tieňovanie písmových ťahov. Tieto prvky dodávajú dlhým textom svieži vzhľad, no v textoch s kratším rozsahom nie sú potrebné [2].

Fonty tejto rodiny boli špeciálne navrhované na použitie na obrazovkách alebo iných zariadeniach s nízkym rozlíšením. Dobrá čitateľnosť je zaručená aj vo veľkosti pod 10 bodov, a to vďaka jednoduchým tvarom a dostatočne veľkým okám.

Comenia Sans obsahuje dvanásť rezov písma, z toho je šesť rezov zúžených, tie sú vhodné v situáciach, keď sme priestorovo limitovaní alebo je potrebné prekvapiť (napr. na plagátoch). Ukážku základných rezov môžeme vidieť na obrázku 2.



Obr. 2: Písmová rodina Comenia Sans

Obr. 3: Písmové rodiny Comenia Script

2.3. Písmové rodiny Comenia Script

Autorkou je Radana Lencová, česká výtvarníčka a grafická návrhárka.

Comenia Script je praktické písané písmo pre deti. Vyznačuje sa tým, že je jednoduché, moderné a súčasné. Jeho úlohou je slúžiť ako základný tvar písaných znakov, ktorý bude obohatený o individuálnu tendenciu každého pisára.

Písmo Comenia Script je tvorené tromi rodinami, ktoré obsahujú vždy len základný rez písma. Jednotlivé rodiny sa vyznačujú nasledujúcimi charakteristikami:

- **Comenia Script A** – rozvinutejšia pätková forma písma, ktorá pracuje s výbehmi spojovacích ťahov; uplatnenie má v bežnej písomnej komunikácii, ale aj v rôznych výtvarných vyjadreniach,
- **Comenia Script B** – jednoduchšia bezpätková forma písma, ktorá nemá spojovacie ťahy; bola vypracovaná na špeciálne účely, napr. pre dysgrafikov, mentálne či telesne hendikepovaných; uplatnenie však nachádza aj v nadpisoch alebo ako technické písmo v matematike či geometrii; vzhľadom pripomína tlačenú formu, je však určená na bežnú písomnú komunikáciu,
- **Comenia Script Pro** – táto rodina je určená pre počítače, môže slúžiť na sadzbu šlabikárov alebo detskej literatúry.

Ukážky vyššie predstavených rodín Comenia Script sú na obrázku 3.

3. Tvorba podpory pre OpenType fonty

Základný problém, na ktorý narazíme pri tvorbe podpory OpenType fontov do systému TEX, spočíva v kódovaní. Systém TEX pracuje s 8-bitovými kódovaniami[2], ktoré umožňujú v jednom okamihu používať najviac 256 znakov, pričom OpenType fonty môžu vďaka kódovaniu Unicode obsahovať teoreticky až 65 tisíc znakov, medzi inými aj rôzne neštandardné znaky ako starobylé číslice, zlomky, ozdobné písmená a iné. Prakticky nie je možné vytvoriť plnohodnotnú podporu, ktorá by umožnila rozumným spôsobom používať všetky znaky obsiahnuté v OpenType fontoch.

Z tohto dôvodu podporu vytvárame len pre znaky, ktoré bežne používame. Na základe týchto znakov vyberáme aj kódovania, ktoré použijeme pri tvorbe podpory. Primeranú podporu znakov používaných v našej oblasti poskytujú kódovania XL2 a T1 (Cork), ktoré boli použité aj pri vytváraní balíčka podpory pre fonty Comenia.

Keď už máme zvolené vstupné kódovanie, musíme fonty Comenia alebo iné OpenType fonty previesť do podoby, ktorému TEX bude rozumieť. Typicky bude potrebné vytvoriť celý rad súborov[3]:

- **`enc`** – súbory kódovania; slúžia na prekódovanie kódovania fontu do kódovania metrík,
- **`fd`** – definičné súbory pre LATEXovskú schému NFSS; obsahujú väzby medzi súbormi metrík a parametrami NFSS (kódovanie, rodina, váha, variant a veľkosť),
- **`map`** – mapovacie súbory; obsahujú všetky potrebné informácie na korektné zavedenie fontov DVI ovládačmi,
- **`pfb/ttf`** – súbory s kresbami znakov; obsahujú kresby znakov vo formáte, ktorému rozumie DVI ovládač; všetky bežne používané DVI ovládače podporujú postscriptový formát `pfb` a niektoré z nich (napr. pdfTEX) aj TrueType formát `ttf`,
- **`tfm`** – súbory s metrickými informáciami; metrické informácie fontu využíva systém TEX pri rozmiestňovaní jednotlivých znakov do zrkadla sadzby,
- **`vf`** – súbory virtuálnych fontov; majú široké využitie (napr. tvorba kompozitných znakov, transformácie znakov, substitúcia znakov...).

Niektoré z týchto súborov môžeme získať pomocou špecializovaných nástrojov, iné budeme musieť vytvoriť ručne.

Pri tvorbe podpory pre fonty Comenia som používal nástroje z balíčka LCDF Typetools od Eddieho Kohlera, ktoré sú súčasťou inštalácie TEX Live. Základným nástrojom tohto balíčka je nástroj **`otftotfm`**[4], ktorý umožňuje vygenerovať súbory

[2] S výnimkou najnovších TEXových formátov XeTEX a LuaTEX, ktoré pracujú s kódovaniami založenými na Unicode.

[3] Súbory `enc`, `fd` a `vf` nie sú potrebné vždy.

[4] Návod na použitie nástroja `otftotfm` je možné nájsť v [3].

`tfm`, `enc`, `pfb` a `vf`. Naviac na štandardnom výstupe tohto nástroja je možné nájsť riadky, ktoré je potrebné pridať do mapovacieho súboru `map`.

Súbory `fd` je potrebné vytvoriť ručne. Ide o textové súbory, ktoré majú pevne stanovenú štruktúru názvu – prvú časť tvorí skratka kódovania metrík, druhú skratka pre názov písmovej rodiny. Napríklad pre kódovanie T1 a rodinu Comenia Serif vytvoríme súbor s názvom `t1scf.fd` a obsahom:

```
1   \ProvidesFile{t1scf.fd}[2013/05/12 font definitions for T1/scf.]
2
3   \typeout{Comenia\space Serif}
4   \typeout{Created\space by\space Firstname\space Lastname.}
5
6   \DeclareFontFamily{T1}{scf}{}
7
8   \DeclareFontShape{T1}{scf}{m}{n}{<-> scfr8t}{}
9   \DeclareFontShape{T1}{scf}{m}{sc}{<-> scfrc8t}{}
10  \DeclareFontShape{T1}{scf}{m}{it}{<-> scfri8t}{}
11  \DeclareFontShape{T1}{scf}{m}{ic}{<-> scfric8t}{}
12  \DeclareFontShape{T1}{scf}{m}{sl}{<-> ssub * scf/m/it}{}
                              ⋮
35  \endinput
```

Príkaz `\ProvideFile` obsahuje názov súboru a voliteľne aj komentár. Príkaz `\typeout` slúži na výpis informácii pri zavádzaní fontu na terminál. Ďalšie príkazy slúžia na deklaráciu jednotlivých rezov rodiny Comenia Serif v danom kódovaní T1. Najprv deklarujeme skratku pre písmovú rodinu pomocou príkazu `\DeclareFontFamily` a potom previažeme metrické súbory[5] jednotlivých rezov s parametrami NFSS pomocou príkazov `\DeclareFontShape`, pričom môžeme využívať aj substitúciu nedostupných rezov dostupnými (riadok 12 v uvedenom príklade).

Ak plánujeme pri svojej práci voliť fonty pomocou balíčka makier OFS, budeme musieť ručne vytvoriť ďalšie definičné súbory aj pre tento balíček makier. A hoci z pohľadu používateľa poskytuje balíček makier OFS rovnaké rozhranie v plainTEXu aj v LATEXu, z pohľadu implementácie pracuje makro v každom formáte inak. Kvôli tejto skutočnosti budeme musieť pre každý formát TEX vytvoriť samostatný súbor, konkrétne:

- **`sty`** – definičný súbor pre formát LATEX,
- **`tex`** – definičný súbor pre formát plainTEX.

[5] Metrické súbory v uvedenom príklade, ako aj súbory v balíčku podpory, sú pomenované podľa schémy názvov Fontname.

Štruktúra týchto súborov je veľmi dobre popísaná v príručke [4], ktorú Petr Olšák k tomuto balíčku makier vytvoril.

Keď už máme pripravené všetky potrebné súbory, je vhodné zaradiť ich do adresárovej štruktúry v súlade s TDS. Balíček podpory, ktorý zachováva túto štruktúru, totiž výrazne uľahčí budúcemu používateľovi jeho inštaláciu. Časť TDS, ktorá je zaujímavá z nášho pohľadu, uvádza obrázok 3.

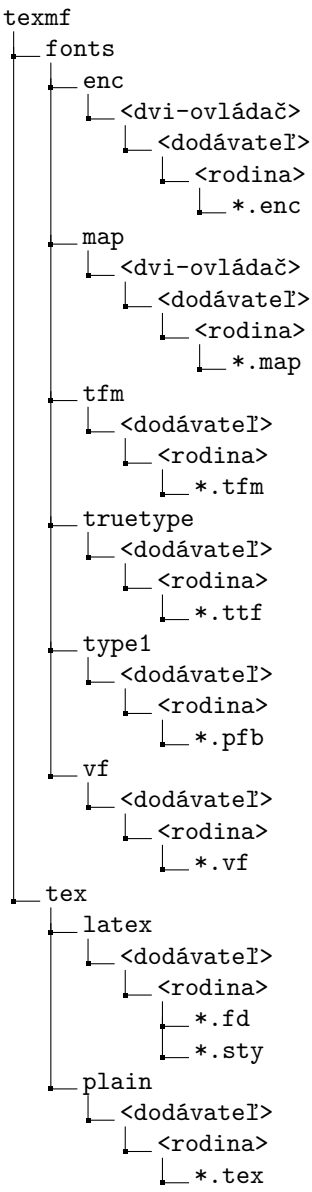
Takto pripravený balíček podpory je zvykom skomprimovať do jedného súboru s príponou `zip`.

Balíček s podporou pre fonty Comenia je obsiahnutý v súbore s názvom `comenia-support.zip`. Tento súbor bol nahraný aj do archívu CTAN, avšak do distribúcie TEX Live 2013 sa už pravdepodobne nestihne dostať.

4. Inštalácia balíčka podpory fontov Comenia

Na úspešnú inštaláciu balíčka podpory fontov Comenia je potrebné vykonať nasledujúce kroky:

- Rozbaliť súbor `comenia-support.zip` do lokálneho `texmf` adresára:
  - v distribúcii TEX Live má tento adresár názov `texmf-local`,
  - v distribúcii MiKTEX bude potrebné pravdepodobne tento adresár najprv vytvoriť.
- Aktualizovať vyhľadávaciu databázu súborov, aby sa systém TEX dozvedel o nových súboroch:
  - v TEX Live použijeme CLI príkaz `mktexlsr`, prípadne jeho alias `texhash` alebo GUI nástroj „TEX Live Manager“, kde zvolíme z menu „Činnosti“ a „Aktualizovať databázu názvov súborov“,
  - v MikTEXu použijeme CLI príkaz `initexmf --update-fndb` alebo GUI nástroj „Settings“, v ktorom na záložke „General“ klikneme na tlačidlo „Refresh FNDB“.
- Nakoniec je nutné oznámiť DVI ovládačom, že pribudli nové mapovacie súbory MAP:
  - v TEX Live sa o všetko potrebné stará nástroj `updmap`, ktorý zavoláme príkazom `updmap --enable map=comenia.map`, následne znovu obnovíme databázu súborov príkazom `mktexlsr` alebo pomocou nástroja „TEX Live Manager“,
  - v MiKTEXu je situácie o niečo zložitejšia:
    1. príkaz `initexmf -edit-config-file=updmap.cfg` vytvorí, prípadne otvorí lokálny súbor `updmap.cfg` aktívneho používateľa v predvolenom textovom editore,



Obr. 4: Časť štruktúry TDS. Uvedená štruktúra je miestami doplnená podľa rozšírených zvyklostí.

2. pomocou editoru pridáme do tohoto súboru riadok:
   `Map comenia.map`,
3. spustíme príkaz `initexmf -u`, ktorý zaktualizuje databázu názvov fontov (ak súbor `updmap.cfg` už existoval tento krok nie je nutný),
4. na záver spustíme príkaz `initexmf --mkmaps`, resp. jeho alias `updmap`.

Po vykonaní týchto krokov sú fonty Comenia pripravené na používanie, a to spôsobom, ktorý je predstavený v nasledujúcom oddieli.

5. Použitie balíčka podpory fontov Comenia

Ku fontom Comenia v systéme TEX môžeme pristupovať viacerými spôsobmi, štyri z nich sú popísané v oddieloch nižšie.

5.1. Príkaz `\font`

Základný TEXový formát plainTEX ponúka na výber fontu príkaz `\font`, ktorého parametrom je názov metriky `tfm`. Tento spôsob výberu fontu však nie je príliš intuitívny, keďže metriky fontov Comenia sú pomenované podľa schémy Fontname.

Napríklad na výber fontu Comenia Sans, rezu zúžená kurzíva a veľkosti 10 pt, použijeme príkaz:

```
\font scsri8tc at 10pt
```

5.2. Schéma NFSS

Schéma NFSS je určená pre TEXový formát LATEX, v súvislosti s fontami Comenia môžu jeho parametre nadobúdať tieto hodnoty:

- `fontencoding`: `T1` (kódovanie Cork), `IL2` (kódovanie ISO 8859-2),
- `fontfamily`: `scf` (Comenia Serif), `scs` (Comenia Sans), `sca` (Comenia Script A), `scb` (Comenia Script B), `sco` (Comenia Script Pro),
- `fontseries`: `m` (štandardný rez), `b` (tučný rez), `sb` (polotučný rez), `c` (zúžený rez), `bc` (tučný zúžený rez), `sbc` (polotučný zúžený rez),
- `fontshape`: `n` (zvislé písmo), `it` (kurzíva), `sc` (kapitálky), `ic` (kurzíva kapitálok),
- `fontsize`: ľubovoľná veľkosť v ľubovoľných jednotkách.

Font z predchádzajúceho príkladu zvolíme pomocou schémy NFSS takto:

```
\fontencoding{T1}
\fontfamily{scs}
\fontseries{c}
\fontshape{it}
\fontsize{10pt}{10pt}
\selectfont
```

5.3. Makro OFS

Makro OFS môžeme v prípade Comenia fontov používať vo všetkých formátoch TEXu, pričom jeho zavedenie v jednotlivých formátoch bude vyzerať nasledovne:

- `\usepackage [scomenia] {ofs}` – pre LaTeX a odvodených formátoch;
- `\input ofs [scomenia]` – pre plainTeX a odvodené formáty.

Zoznam dostupných rodín a rezov získame príkazom `\showfonts`.

Font z predchádzajúceho príkladu zvolíme pomocou makra OFS takto:

```
\setfonts [ComeniaSans/10pt]
\cfit
```

5.4. Priamy prístup k fontom OS

V najnovších formátoch TEXu ako XeTeX a LuaTeX je možné pristupovať aj k fontom, ktoré sú nainštalované priamo v OS.

Ak sa rozhodneme pre tento prístup, stačí fonty Comenia nainštalovať bežným spôsobom do OS[6] a následne spustiť príkaz `fc-cache`, ktorý aktualizuje TEXovú databázu dostupných fontov. Inštalácia balíčka s podporou v tomto prípade nie je potrebná.

Základným príkazom na výber fontu je naďalej príkaz `\font`, jeho možnosti sa však rozšírili, a preto je možné použiť ho aj takto:

```
\font\comserif = "Comenia Serif"
\font\comserifital = "Comenia Serif Italic"
\font\comseriftitle = "Comenia Serif" at 16pt
\font\comserifcaps = "Comenia Serif:+smcp"
```

V podstate ide o deklaráciu nových prepínačov pre uvedené fonty a veľkosti. V rámci týchto príkazov si môžeme explicitne vyžiadať špeciálne sady znakov (napr. pre kapitálky sme vyššie použili voľbu `+smcp`). Použitím deklarovaného prepínaču daný font aktivujeme.

V LaTeXových formátoch XeLaTeX a LuaLaTeX môžeme pristupovať k fontom OS prostredníctvom schémy NFSS a to bez nutnosti vytvárať definičné FD súbory.

[6] Postup inštalácie fontov do OS je závislý na konkrétnom OS.

Namiesto toho stačí zaviesť balíček fontspec v preambule dokumentu príkazom `\usepackage{fontspec}`.

Medzi hlavné príkazy, ktoré tento balíček ponúka, patri nasledujúce:

- `\fontspec[<features>]{font}` – slúži na jednorázový výber fontu,
- `\setmainfont[<features>]{font}` – slúži na výber predvolenej písmovej rodiny dokumentu,
- `\setsansfont[<features>]{font}` – slúži na výber predvolenej bezserifovej písmovej rodiny dokumentu,
- `\setmonofont[<features>]{font}` – slúži na výber predvolenej neproporcionálnej písmovej rodiny dokumentu,
- `\newfontfamily\<switch>[<features>]{font}` – slúži na deklarovanie vlastného prepínača `\<switch>` pre font, ktorý budeme v dokumente používať častejšie.

Parameter `<features>` akceptuje aj zoznam viacerých položiek oddelených čiarkou. Napríklad na zvolenie fontu Comenia Sans so starobylými neproporcionálnymi číslicami a s vypnutými ligatúrami použijeme príkaz:

```
\fontspec[Numbers={OldStyle,Monospaced},
    Ligatures=NoCommon]{Comenia Sans}
```

Viac podrobností je možné nájsť v dokumentácii balíčka `fontspec`.

Literatúra

[1] Fábry, Tomáš. *Comenia fonty pre TEX* [online]. Diplomová práca, školiteľ: Petr Sojka, Masarykova univerzita, Brno, 2013. [cit. 2013-05-26]. Dostupné na: `http://is.muni.cz/th/325338/fi_m/thesis.pdf`.

[2] Brousil, Tomáš; Lencová, Radana; Štorm, František. *Comenia: České školní písmo* [online]. [cit. 2013-05-20]. Dostupné na: `http://www.uhk.cz/cs-cz/o-univerzite/vizualni-styl-univerzity/jednotny-vizualni-styl/Documents/Projekt%20Comenia.pdf`.

[3] Kohler, Eddie. *otftotfm Manual* [online]. [cit. 2013-05-21]. Dostupné na: `http://www.lcdf.org/type/otftotfm.1.html`.

[4] Olšák, Petr. *OFS: Olšákův fontový systém* [online]. 2004. [cit. 2013-05-21]. Dostupné na: `ftp://math.feld.cvut.cz/pub/olsak/ofs/ofsdoc.pdf`.

Summary: Comenia fonts support for TEX

This article describes the Comenia fonts support for the system TEX which was created as a part of my diploma thesis 'Comenia fonts support for TEX' [1]. The introduction outlines the motivation that led to the creation of this support. Subsequently reader can find some information about Comenia fonts. The process of the Comenia fonts support creation is described in the section 3. The installation and using instructions are given at the end of the article.

Key words:

Comenia, fontspec, OpenType fonty

Tomáš Fábry

Jednoduchá grafika PDF-primitivně

Petr Olšák

Představme si, že potřebujeme do dokumentu přidat jednoduchou čáru nebo tvar či vytvořit speciální opakující se symbol. V takovém případě nemusíme volat složitá makra na komplexní grafiku ani vytvářet nový font. Je totiž možné na věc jít přímočaře, a to použitím pdfTEXových primitivních příkazů a elementárních grafických operátorů, kterým rozumí PDF rasterizér. K rozšíření našich možností stačí znát velmi omezenou sadu těchto příkazů

V tomto článku shrneme primitivní příkazy pro tvorbu grafiky a ilustrujeme je na příkladech. Některé věci již uvedli Zýka [1] a Chvála [2]. Příklady v textu, který právě čtete, ukazují navíc možnosti, které v citovaných článcích nebyly zmíněny.

Pochopitelně nelze očekávat, že v následujících příkladech vytvoříme pohodlné uživatelské rozhraní pro „programování“ obrázků. K tomu slouží například velmi propracované makro TikZ [3], které pracuje v LaTeXu i plainTeXu. Někdy je také vhodné vytvořit obrázky v interaktivním editoru a vkládat je do pdfTEXu pomocí `\pdfximage`.

Klíčová slova

pdfTEX, kód PDF, grafika

1. Shrnutí primitivních příkazů pro grafiku

V pdfTEXu se můžeme setkat s následujícími příkazy, které se týkají vkládání grafiky z externího souboru nebo řízení procesu tvorby grafiky uvnitř dokumentu. Čísla před příkazy odkazují na čísla sekcí v tomto článku, kde jsou příkazy podrobněji vysvětleny.

```
2. \pdfximage, \pdfrefximage, \pdflastximage
                                    % vložení externího obrázku
3. \pdfsave, \pdfrestore, \pdfsetmatrix   % lineární transformace
4. \pdfliteral{⟨pdf kód⟩}           % kresba PDF kódem
5  \pdfsavepos \pdflastxpos \pdflastypos
                                    % souřadnice vzhledem ke straně
6. \pdfxform, \pdfrefxform, \pdflastxform
                                    % podprogramy v PDF kódu (Forms)
```

Další desítky primitivních příkazů pdfTEXu pro nastavování parametrů dokumentu, využití mikrotypografického rozšíření, interní práci s PDF kódem, hyperlinkové odkazy, odkazy na audio a zvuk, sestavení rozbalovacího klikacího obsahu v prohlížeči (outlines) a mnoho dalších nejsou obsahem tohoto článku. Najdete je například v [2] nebo v dokumentaci PDFTEXu [4].

doi: 10.5300/2013-1/13

2. Vložení externího obrázku

OPmac [5] pro tuto činnost nabízí makro `\inspic` ⟨*filename*⟩, viz kapitolu 12 v [6]. Zde rozebereme primitivní pohled na vkládání obrázku. Je možné vkládat obrázky ve formátu `png`, `jpg`, `jbig2` a `pdf`. Posledně jmenovaný umožňuje vkládat i vektorovou grafiku a vytáhnout z vícestránkového PDF dokumentu požadovanou stránku jako jeden vkládaný obrázek.

Na pdfTEX-primitivní úrovni má vložení obrázku dvě fáze. Nejprve je obrázek načten a vložen do výstupního PDF pomocí `\pdfximage`. Takto vložený obrázek se ještě nezobrazí. Místo, kde se má zobrazit, je v PDF kódu řešeno odkazem. Až tedy budeme vědět, kam obrázek chceme do dokumentu umístit, v tomto místě použijeme příkaz `\pdfrefximage`⟨*číslo odkazu*⟩. Důvod tohoto rozfázování spočívá v možnosti odkazovat na jednou načtený obrázek na více místech v dokumentu. Obrázek se opakovaně objeví, ale do PDF souboru je vložen jen jednou.

Uvedu příklad vložení obrázku:

```
\pdfximage width4cm {obrazek.jpg}  % vložení bitmapové grafiky
zde: \pdfrefximage\pdflastximage   % na toto místo.
```

Příkaz `\pdfximage` obsahuje následující parametry (povinný je jenom parametr ⟨*filename*⟩).

`\pdfximage width`⟨*dimen*⟩ `height`⟨*dimen*⟩ `page`⟨*num*⟩ `{`⟨*filename*⟩`}`

Tento příkaz vloží do PDF výstupu obrázek z ⟨*filename*⟩, připraví ho ve velikosti podle `width` a `height`. Jsou-li uvedeny oba parametry, bude pravděpodobně obrázek deformován, při jednom parametru se druhý rozměr dopočítá tak, aby k deformaci poměru šířka:výška nedošlo. Parametr `page` je možno použít jen při čtení z PDF souboru, přitom parametr určuje stránku, která má být přečtena.

Po provedení příkazu `\pdfximage` se v registru `\pdflastximage` dozvíme číslo odkazující na načtená data, které je potřeba použít jako parametr příkazu `\pdfrefximage`⟨*číslo odkazu*⟩. Následuje příklad, ve kterém chceme opakovat obrázek `logo.pdf` na každé straně v záhlaví dokumentu.

Je nutné si číslo odkazu zapamatovat, protože mezitím může být použit příkaz `\pdfximage` pro další obrázky.

```
\newcount\reflogo
\pdfximage width12mm {logo.jpg} \reflogo=\pdflastximage
\headline={\pdfrefximage\reflogo \hfil text}
```

Příkaz `\pdfrefximage`⟨*číslo odkazu*⟩ vloží do sazby (vertikálního nebo horizontálního módu) box o výšce a šířce odpovídající obrázku. V příkazu `\pdfximage` je možno také specifikovat parametrem `depth` hloubku tohoto boxu.

Podle toho se přizpůsobí sazba, která bezprostředně předchází nebo následuje za `\pdfrefximage`⟨*číslo odkazu*⟩.

Další příklad ukazuje možnost přečtení celého PDF dokumentu a jeho připojení do stávajícího dokumentu. Využívá jednak parametru `page`, jednak registru `\pdflastximagepages`, ve kterém je celkový počet stránek PDF dokumentu, jehož aspoň jedna stránka byla naposledy přečtena příkazem `\pdfximage`.

```
\nopagenumbers
\hoffset=-1in \voffset=-1in
\newcount\tmpnum
\def\add#1{%
   \pdfximage width\pdfpagewidth page 1 {#1}
   \vbox to0pt{\pdfrefximage\pdflastximage\vss}\vfil\break
   \tmpnum=1
   \loop
     \ifnum\tmpnum<\pdflastximagepages
        \advance\tmpnum by1
         \pdfximage width\pdfpagewidth page \tmpnum {#1}
         \vbox to0pt{\pdfrefximage\pdflastximage\vss}\vfil\break
         \repeat
}
\add{document1.pdf} \add{document2.pdf}
```

Uvedená ukázka přečte dva dokumenty `document1.pdf` a `document2.pdf` a spojí je do jediného výstupního dokumentu. Tento kód se také osvědčil, když člověk obdrží PDF dokument v jiném formátu než A4 a tiskárna ho nedokáže správně vytisknout.

3. Lineární transformace

O lineárních transformacích pojednává kapitola 13 v dokumentaci k OPmac [6] a přidává navíc popis makra `\rotate`. Zde jen stručně shrneme odpovídající primitivní příkazy pdfTEXu. Transformační matici lze pozměnit příkazem `\pdfsetmatrix{`⟨*a*⟩ ⟨*b*⟩ ⟨*c*⟩ ⟨*d*⟩`}`. To zahrnuje všechny možnosti lineární transformace (tedy otočení, deformace ve směrech, zkosení). Jakákoli sazba (text, obrázky) následovaná za příkazem `\pdfsetmatrix` bude odpovídajícím způsobem transformována. Sazba musí být obklopena příkazy `\pdfsave` a `\pdfrestore`, celá se musí odehrát na jedné stránce a aktuální bod sazby se musí vrátit do původního místa před příkaz `\pdfrestore`, který uzavře nastavenou transformaci, tj. za tímto příkazem se sazba vrací k normálu.

Příkaz `\pdfsave` si zapamatuje grafický stav včetně prováděné lineární transformace a polohy aktuálního bodu sazby. Příkaz `\pdfrestore` se pak vrací k zapamatovanému nastavení. Uvidíme v další sekci, že tyto příkazy se podobají příkazům `\pdfliteral{q}` a `\pdfliteral{Q}` jen s tím rozdílem, že příkazy

`\pdfliteral` nekontrolují, zda se sazba po návratu k původnímu grafickému stavu vrátila do stejného bodu, v jakém začala. Pokud se to nestane, TEX pak předpokládá, že má aktuální bod sazby jinde, než co předpokládá PDF rasterizér, což může vést k nepředvídatelným událostem.

4. Kresba PDF kódem

Budeme-li chtít kreslit přímo PDF kódem, tj. použít `\pdfliteral{`⟨*pdf kód*⟩`}`, nemusíme hned studovat sedmisetstránkovou PDF specifikaci [8]. Je ale dobré znát následující užitečné příkazy:

```
q           % zahájení skupiny pro nastavení grafického stavu
Q           % ukončení skupiny, návrat k původnímu grafickému stavu
⟨num⟩ g     % (Gray) nastavení stupně šedi pro plochy
⟨num⟩ G     % (Gray) nastavení stupně šedi pro tahy
⟨r⟩ ⟨g⟩ ⟨b⟩ rg      % nastavení barvy v RGB pro plochy
⟨r⟩ ⟨g⟩ ⟨b⟩ RG      % nastavení barvy v RGB pro tahy
⟨c⟩ ⟨m⟩ ⟨y⟩ ⟨k⟩ k   % (cmyK) nastavení barvy v CMYK pro plochy
⟨c⟩ ⟨m⟩ ⟨y⟩ ⟨k⟩ K   % (cmyK) nastavení barvy v CMYK pro tahy
⟨width⟩ w   % (Width) nastavení šířky čáry
⟨typ⟩ j  % typ lámání čáry, 0 s hranami, 1 kulatě, 2 s ořezem
⟨typ⟩ J  % typ zakončení čáry, 0 hranatý, 1 kulatý, 2 s přesahem
⟨a⟩ ⟨b⟩ ⟨c⟩ ⟨d⟩ ⟨e⟩ ⟨f⟩ cm % (Current Matrix)
              % pronásobení transformační matice
⟨x⟩ ⟨y⟩ m    % (Moveto) nastavení polohy kreslícího bodu
⟨dx⟩ ⟨dy⟩ l  % (Lineto) přidání úsečky
⟨x1⟩ ⟨y1⟩ ⟨x2⟩ ⟨y2⟩ ⟨x3⟩ ⟨y3⟩ c  % (Curveto) přidání Bézierovy křivky
⟨x⟩ ⟨y⟩ ⟨dx⟩ ⟨dy⟩ re            % (Rectangle) příprava obdélníku
h    % uzavření postupně budované křivky
S    % (Stroke) vykreslení připravené křivky čárou
s    % stejné jako h S
f    % (Fill) vyplnění oblasti dané uzavřenou připravenou křivkou
B    % (Fill and Storke = Both) vyplnění oblasti a její obtažení čarou
W n  % nastavení připravené uzavřené křivky jako omezující (clipping)
```

Jednotlivé PDF elementární příkazy si dále ukážeme v příkladech podrobněji.

Příkaz `\pdfliteral{`⟨*pdf kód*⟩`}` z hlediska TEXu neudělá se sazbou nic. Následující sazba tedy pokračuje tam, kde předchozí skončila. Přitom ⟨*pdf kód*⟩ může obsahovat elementární příkazy pro PDF rasterizér například na změnu grafického stavu nebo na vykreslení nějaké grafiky.

4.1 Nastavení barev

Nejprve vysvětlíme a na příkladech ukážeme příkazy na změnu barvy `g`, `G` (šedá), `rg`, `RG` (RGB), `k` a `K` (CMYK). Jsou zde dvě varianty (malá a velká písmena) pro každý barevný prostor. Příkaz s malými písmeny ovlivní použití barvy při vyplňování uzavřených křivek (Fill) a při sazbě textu. Je to pochopitelné, protože kresba jednotlivých písmen v textu probíhá také vyplňováním uzavřených křivek. Příkaz pro změnu barvy s velkými písmeny ovlivní barvu při kresbě podél křivek (Stroke). Tato dvě nastavení jsou na sobě nezávislá, lze tedy nastavit vyplňování zelené a obtahování červené. Je třeba také vědět, že pdfTEX řeší vykreslení `\vrule` a `\hrule` pomocí Stroke, je-li objekt tenčí nebo roven 1 bp. A vyplní obdélník pomocí Fill, má-li oba rozměry větší než 1 bp. Z tohoto pohledu se nastavení barvy pomocí malých písmen týká „tlustých“ `\vrule` a `\hrule`, zatímco nastavení barvy pomocí velkých písmen „tenkých“ `\vrule` a `\hrule`.

Poznamenejme, že nastavování barev v OPmac je vyloženo v sekci 8 manuálu [6] a že OPmac nabízí rozlišení těchto dvou „barevných druhů“ pomocí prefixu `\linecolor`.

Příklad přepnutí do červené sazby může vypadat takto:

```
Tady je černý text.
\pdfliteral{1 0 0 rg}Tady je červený.\pdfliteral{0 0 0 rg}
Tu je zase černý.
\pdfliteral{0 1 1 0 k}Tu znovu červený.\pdfliteral{0 g}
A zpátky černý.
```

Dostaneme tento výsledek: Tady je černý text. Tady je červený. Tu je zase černý. Tu znovu červený. A zpátky černý.

Argumenty příkazů pro nastavování barvy jsou obecně desetinná čísla (s desetinou tečkou) v rozsahu od nuly do jedné. Například `\pdfliteral{0.7 g}` znamená třicetiprocentní šedou. Nebo třeba `\pdfliteral{0.25 0.3 0.75 rg}` nastaví barvu smíchanou z 25% červené, 30% zelené a 75% modré v aditivním barevném prostoru RGB.

Na příkladu vidíme, že je v podstatě jedno, jaký barevný prostor použijeme. Barvy ovšem nejsou totožné, protože CMYK prochází korekcemi vhodnými pro tisk. Dále je možné uzavřít nastavení barvy do skupiny

```
"\pdfliteral{q}...\pdfliteral{Q}".
```

Po uzavření skupiny se sazba vrátí k původní barvě. Stejně tak se vrátí i sazba do původního bodu, což často není žádoucí. Proto je lepší ukončit sazbu v barvě příkazem `\pdfliteral{0 g}`, který jednoduše vrátí barvu černou.

S nastavováním barev souvisí poměrně komplikovaný problém, který se neprojeví, je-li barva nastavena lokálně pro objekt, který nikdy nepřekročí hranici stránky. Změna barvy je totiž pokyn pro PDF rasterizér, o kterém TEX neví.

Máte-li změnu barvy na první stránce a návrat k černé na některé další, PDF rasterizér změní barvu od místa změny po konec stránky. Pro každou stranu zakládá PDF rasterizér novou skupinu, takže na konci stránky barva mizí. Přitom se obarví i patička, tedy stránková číslice. Na další stránce rasterizér zakládá novou skupinu a vůbec neví, že má pokračovat ve speciální barvě, a pokračuje tam barvou černou.

Tento problém řeší pdfTEXové primitivní příkazy pro tzv. colorstack. Ty ale nejsou bohužel dokumentovány, nicméně jsou použity např. v LATEXovém balíku `color.sty`. Makro OPmac je nepoužívá právě proto, že nejsou dokumentovány, a řeší uvedený problém ve vlastní režii jen na úrovni maker.

4.2 Kresba křivek

Křivku je potřeba pomocí PDF elementárních příkazů nejprve připravit (Moveto, Lineto, Curveto) a poté podél připravené křivky můžeme vést čáru (Stroke) nebo, je-li uzavřena, můžeme vyplnit vnitřek křivky (Fill).

Argumenty příkazů pro přípravu křivky jsou desetinná čísla v jednotkách, které jsou implicitně nastaveny na bp (typografický bod, 1/72 palce), a souřadnicový systém implicitně prochází bodem aktuálního bodu sazby, tj. místem, kde je použit příslušný příkaz `\pdfliteral`. První souřadnicová osa x směřuje doprava a druhá y nahoru. Toto implicitní chování je možné změnit změnou transformační matice, o čemž pojednáme později.

Následuje příklad, který vytvoří zelený trojúhelník a modrý půldisk.

```
\pdfliteral{q          % uchování grafického stavu
  0 1 0 RG 0 0 1 rg    % nastavení barvy pro čáry (zelená)
                       %       a pro výplně (modrá)
  3.2 w                % (Width) šířka čáry bude 1.2 bp
  0 0 m                % (Moveto) pero položíme do počátku
  30 30 l              % (Lineto) přidáme úsečku z 0 0 do 30 30
  30 0 l               % (Lineto) přidáme úsečku a 30 30 do 30 0
  h                    % uzavření křivky,
  S                    % (Stroke) kresba křivky čárou v dané
                       %       šířce a barvě
  50 0 m               % (Moveto) nastavení pera do bodu 50 0
  50 10 60 20 70 20 c  % (Curveto) první čtvrtina disku
  80 20 90 10 90 0 c   % (Curveto) druhá čtvrtina disku
  h                    % uzavření křivky
  f                    % vyplnění uzavřené křivky barvou
```

```
  Q                    % návrat k původním hodnotám grafického stavu
}
```

Kresba se zjeví v aktuálním bodě sazby a nebude zabírat žádné místo. Abychom tímto obrázkem nepřekreslili předchozí text, je potřeba připravit místo pro obrázek manuálně. V tomto konkrétním příkladě jsem rozměry pro obrázek odhadl a napsal:

```
\nobreak\vskip2cm%
   \centerline{\hss\pdfliteral{⟨předchozí kód⟩}\hskip4cm\hss}
```

Při kresbě tímto způsobem je nutné mít na paměti následující pravidla:

- Nastavení barvy a tloušťky čáry je vhodné dělat mezi `q` a `Q`.
- Před vykreslením pomocí `S`, `f` nebo `B` je nutné připravit křivku.
- Příprava křivky musí začínat příkazem `m`, jenž nastavuje aktuální bod kresby.
- Křivka se připravuje příkazy `l`, `c`, které budují křivku postupně z částí. Každý další příkaz `l` nebo `c` připojí další úsek křivky k již sestavované a posune na její konec aktuální bod kresby.
- Je možné během přípravy křivky použít další příkaz `m`, čímž vznikne křivka nesouvislá.
- Příprava křivky ještě neznamená její vykreslení. To je možné provést pomocí `S` nebo `f` nebo `B`.
- Po vykreslení křivky její data z paměti zmizí.
- Neuzavřou-li se souvislé části křivky pomocí `h` a použije-li se příkaz `f` nebo `B`, provede rasterizér uzavření každé jednotlivé části (oddělené příkazy `m`) samostatně.

Bézierova křivka tvořená pomocí ⟨*x1*⟩ ⟨*y1*⟩ ⟨*x2*⟩ ⟨*y2*⟩ ⟨*x3*⟩ ⟨*y3*⟩ `c` je určena počátečním bodem ⟨*x0*⟩ ⟨*y0*⟩, který je roven aktuálnímu bodu kresby, dále koncovým bodem ⟨*x3*⟩ ⟨*y3*⟩ a dvěma kontrolními body ⟨*x1*⟩ ⟨*y1*⟩ a ⟨*x2*⟩ ⟨*y2*⟩. Jak vypadá chování takové křivky, lze zjistit v nějakém interaktivním editoru pro vektorovou grafiku. Je vhodné vědět, že spojnice ⟨*x0*⟩ ⟨*y0*⟩ `--` ⟨*x1*⟩ ⟨*y1*⟩ je tečnou křivky v počátečním bodě a stejně tak spojnice ⟨*x2*⟩ ⟨*y2*⟩ `--` ⟨*x3*⟩ ⟨*y3*⟩ je tečnou křivky v koncovém bodě. Počátečním a koncovým bodem křivka prochází a kontrolními body obvykle neprochází (ty křivku jen „přitahují“). Matematicky je výše uvedenými podmínkami určena jednoznačně kubika (graf polynomu třetího stupně), která přesně definuje příslušnou Bézierovu křivku.

PDF rasterizér disponuje jedním složeným příkazem ⟨*x*⟩ ⟨*y*⟩ ⟨*dx*⟩ ⟨*dy*⟩ `re`, který je zkratkou za

⟨*x*⟩ ⟨*y*⟩ `m` ⟨*x+dx*⟩ ⟨*y*⟩ `l` ⟨*x+dx*⟩ ⟨*y+dy*⟩ `l` ⟨*x*⟩ ⟨*y+dy*⟩ `l h`

a používá se k přípravě obdélníka.

4.3 Transformační matice

O transformační matici byla již zmínka v souvislosti s příkazem `\pdfsetmatrix` v sekci 3. Tento příkaz pracuje s maticí 2×2 a dovoluje jen lineární transformace. Na druhé straně elementární operátor

⟨*a*⟩ ⟨*b*⟩ ⟨*c*⟩ ⟨*d*⟩ ⟨*e*⟩ ⟨*f*⟩ `cm`

pracuje s maticí 3×3 a umožňuje lineární transformace a posunutí. Matice se doplní na třetím řádku čísly `0 0 1`. Bod se souřadnicemi (x, y) se transformuje na bod se souřadnicemi (x', y') podle následujícího maticového násobení:

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} a & c & e \\ b & d & f \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

Vidíme, že údaje a, b, c, d realizují lineární transformaci; dále se provede posunutí o vektor (e, f). Následující matice provádějí jednoduché transformace:

```
1 0 0 1 ⟨e⟩ ⟨f⟩ cm % posunutí o vektor (⟨e⟩, ⟨f⟩)
0 1 -1 0 0 0 cm    % rotace o 90 stupňů v kladném směru
⟨a⟩ 0 0 ⟨d⟩ 0 0 cm % škálování ⟨a⟩ krát ve směru x a ⟨d⟩ krát ve směru y
-1 0 0 1 0 0 cm    % zrcadlení podle osy y
1 0 0 -1 0 0 cm    % zrcadlení podle osy x
⟨cos α⟩ ⟨sin α⟩ -⟨sin α⟩ ⟨cos α⟩ 0 0 cm  % rotace o úhel α.
```

PDF rasterizér udržuje v paměti aktuální transformační matici a každá další aplikace operátoru `cm` způsobí pronásobení aktuální matice maticí sestavenou z parametrů operátoru `cm` zleva. To odpovídá skládání jednotlivých zobrazení.

Existují dva pohledy na aplikaci transformační matice. Podle prvního z nich každý bod s danými souřadnicemi transformujeme podle výše uvedeného maticového násobení a dostáváme souřadnice, kam máme bod nakreslit. Druhý pohled interpretuje transformaci jako změnu souřadnicového systému. Matice aplikovaná pomocí `cm` změní souřadnicový systém následovně: v prvních dvou sloupcích matice čteme směrové vektory nových os x' a y' (jednotky na těchto osách odpovídají velikosti směrových vektorů) a v posledním sloupci přečteme souřadnice nového počátku. Dále si představíme nový souřadnicový systém a veškeré údaje příkazů `m`, `l`, `c` nyní vztahujeme k tomuto novému souřadnicovému systému. Oba pohledy si osvětlíme na matici

```
72 0 0 -72 0 0 cm
```

První pohled: Například bod o souřadnicích $(2, 3)$ se transformuje na bod o souřadnicích $(144, -216)$. Druhý pohled: Původní souřadný systém měl jednotku $1/72$ palce. Nový souřadný systém má jeden směrový vektor $(72, 0)$ a ten tvoří novou jednotku v nové ose x. Ta má stejný směr jako původní osa x, tedy

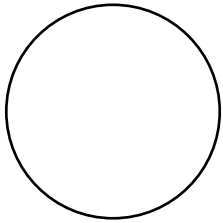
doprava. Druhý směr je $(0, -72)$, takže nová osa y' má stejnou jednotku, ale je orientovaná nikoli nahoru ale dolů. Počátek souřadného systému zůstává na stejném místě. Máme-li nyní nakreslit bod o souřadnicích $(2, 3)$, provedeme to přímočaře v novém souřadném systému, v nových jednotkách a směrech, tedy v palcích: dva palce doprava a tři dolů. Oba pohledy samozřejmě vedou ke stejnému výsledku.

Jako příklad uvedeme možnost změnit souřadný systém pro kresbu z původních jednotek `bp` na častěji používané jednotky `pt`. Vytvoříme ukázkové makro `\koso{`⟨*velikost*⟩`}`, které vytvoří čtverec o straně ⟨*velikost*⟩ otočený o 45 stupňů. Pochopitelně to lze udělat jednoduše pomocí `\vrule`, ale zapomeňme na chvíli na tento primitivní příkaz, jelikož chceme ukázat, jak mohou TEXové jednotky spolupracovat s jednotkami PDF rasterizéru. Uživatel může použít makro `\koso{2mm}` nebo `\koso{\parindent}` a nemůžeme ho nutit, aby nám své rozměry přepočítával do jednotek `bp`. O převod se musí postarat makro. Jakýkoli rozměr v TEXu můžeme uložit třeba do `\dimen0` a pak jej pomocí `\the\dimen0` vypsat. To nám TEX ochotně udělá v jednotkách `pt` a tuto jednotku navíc připojí. My potřebujeme symbol `pt` jednak odpojit a jednak nastavit transformační matici tak, aby odpovídající rozměry bylo možno zadávat přímo v `pt`.

```
\def\koso#1{\dimen0=#1\relax
   \hbox to1.4142\dimen0{\hss\vbox to1.4142\dimen0{\vss \kosoA}\hss}}
\def\kosoA{\pdfliteral{q                % pdfsave
  0.996264 0 0 0.996264 0 0 cm          % přechod z bp na pt
  0.7071 0.7071 -0.7071 0.7071 0 0 cm   % otočení o 45 stupňů
  0 0 \nopt\dimen0, \nopt\dimen0, re f % nakreslení obdélníka
  Q                                     % pdfrestore
}}
\def\nopt#1,{\expandafter\ignorept\the#1 }
{\lccode`\?=`\p \lccode`\!=`\t  \lowercase{\gdef\ignorept#1?!{#1}}}
```

Vidíme, že převod souřadnic probíhá na úrovni příkazu `cm`, přitom číslo 0,996264 je přibližně rovno zlomku 72/72.27. To souvisí s tím, že `pt` má rozměr 1/72.27 palce a `bp` má rozměr 1/72 palce. Zbylý kód makra obsahuje již jen drobné triky. Především v makru `\koso` je třeba TEXovsky vytvořit box potřebné šířky a výšky, čehož dosáhneme pomocí vnořených `\hbox` a `\vbox`, které mají výšku i šířku $\sqrt{2}$krát větší než zadaný rozměr strany čtverce. Vlastní kresba se pak odehrává dole uprostřed tohoto boxu jako makro `\kosoA`. V registru `\dimen0` je uložen požadovaný rozměr. Je-li tímto rozměrem třeba `2mm`,TEX pomocí `\the\dimen0` vypíše `5.69054pt`. My ale potřebujeme odstranit písmena `pt`, která by v PDF kódu překážela, a vložit jen `5.69054`. K tomu slouží makro `\ignorept` (jeho definice je převzata z OPmac). Makro `\nopt`, které je nakonec v PDF kódu použito, vezme registr typu ⟨*dimen*⟩ až po čárku a odebere mu jednotku pt. Mezera za čárkou už je platná a odděluje parametry v PDF kódu.◆

V dalším příkladu vytvoříme kružnici. Kresba kružnic nebo jejich částí není podpořena přímo PDF elementárním příkazem a je nutno ji nahradit pro každou čtvrtinu kružnice příkazem `c` (curveto) s vhodnou polohou kontrolních bodů. Matematicky samozřejmě není možno dosáhnout přesné kružnice, protože pomocí Bézierovy kubiky (polynomu třetího stupně) nelze zapsat odmocninu. Přesto je následující aproximace tak dokonalá, že to oko nevidí:



```
\def\circle{.5 0 m
.5 .27615 .27615 .5 0 .5 c
-.27615 .5 -.5 .27615 -.5 0 c
-.5 -.27615 -.27615 -.5 0 -.5 c
.27615 -.5 .5 -.27615 .5 0 c
}
\pdfliteral{q 80 0 0 80 0 0 cm .0125 w \circle S Q}
```

V makru `\circle` je připravena kružnice o průměru 1. Před jejím použitím je pomocí transformační matice realizováno zvětšení, takže kružnice bude mít průměr 80 bp. Aby měla čáru tloušťky 1 bp, musíme zadat pro příkaz `w` převrácenou hodnotu zvětšení. Nakreslit kružnici libovolného průměru TEXovým makrem je dále jednoduchým cvičením pro zručného TEXistu.

4.4 Rámeček s oblými kouty

Pro nakreslení rámečku s oblými kouty by se hodilo, kdybychom mohli argumenty příkazů lineto a curveto zapisovat relativně k aktuálnímu bodu kresby, nikoli k počátku. Tedy například místo ⟨*x*⟩ ⟨*y*⟩ `l` by byl užitečnější příkaz ⟨*dx*⟩ ⟨*dy*⟩ `dl`, který by vedl úsečku z bodu ⟨*x0*⟩ ⟨*y0*⟩ (aktuálního bodu kresby) do bodu ⟨*x0+dx*⟩ ⟨*y0+dy*⟩. To ale PDF rasterizér neumí. O přepočet do absolutních souřadnic se tedy musí postarat TEX. Připravíme si makra

```
\dmoveto ⟨x⟩,⟨y⟩,  % nastavení aktuálního bodu kresby
\dlineto ⟨dx⟩,⟨dy⟩,% úsečka relativně k aktuálnímu bodu kresby
\dcurveto ⟨dx1⟩,⟨dy1⟩,⟨dx2⟩,⟨dy2⟩,⟨dx3⟩,⟨dy3⟩,
   % křivka relativně k aktuálnímu bodu
```

Makra předpokládají, že rasterizér pracuje v souřadnicích s jednotkou `pt`, takže je potřeba předřadit příslušnou matici transformace (z `bp` do `pt`) a využít makra `\nopt` z předchozího příkladu.

```
\newdimen \cpX \newdimen\cpY % souřadnice aktuálního bodu kresby
\def\dmoveto #1,#2,{\cpX=#1\cpY=#2\pdfliteral{\nopt\cpX,\nopt\cpY,m}}
\def\dlineto #1,#2,{\advance\cpX by#1\advance\cpY by#2%
   \pdfliteral{\nopt\cpX,\nopt\cpY,l}}
\def\dcurveto #1,#2,#3,#4,#5,#6,{%
   {\advance\cpX#1\advance\cpY#2\pdfliteral{\nopt\cpX,\nopt\cpY,}}%
   {\advance\cpX#3\advance\cpY#4\pdfliteral{\nopt\cpX,\nopt\cpY,}}%
    \advance\cpX#5\advance\cpY#6\pdfliteral{\nopt\cpX,\nopt\cpY,c}}
```

Údaje pro kontrolní body při `\dcurveto` jsou přepočítány uvnitř skupiny, takže jsou všechny relativní k počátku křivky, nikoli relativní jeden k druhému.

Vlastní rámeček se zaoblenými kouty je dán následujícími parametry, které může uživatel měnit:

```
\newdimen\rfR \rfR=5pt % poloměr zaoblených rohů
\newdimen\rfM \rfM=1pt % okraje mezi boxem a čárou rámečku
\def\rfType{1 1 0 rg 1 0 0 RG 1 w} % barvy plochy a čáry a šířka čáry
```

Následuje kód makra `\roundedframe{`⟨*text*⟩`}`, který vykreslí rámeček. Kresba rámečku je zahájena levým horním rohem (jeho spodní částí). K tomu účelu musíme umístit počáteční bod na souřadnice `0` ⟨*výška*⟩, kde tato ⟨*výška*⟩ je rovna výšce boxu plus velikost okraje `\rfM` mínus poloměr zaoblení `\rfR`. Parametr ⟨*výška*⟩ je připraven v `\dimen2`. Podobně jsou v `\dimen1` a `\dimen3` předpočítány další parametry kresby.

```
\def\roundedframe#1{\setbox0=\hbox{\strut#1}%
   \hbox{\drawroundedframe \kern\rfM \box0 \kern\rfM}}
\def\drawroundedframe{\dimen0=\rfR \advance\dimen0 by-\rfM
   \dimen1=\wd0 \advance\dimen1 by-2\dimen0 % délka vodorovné linky
   \dimen2=\ht0 \advance\dimen2 by-\dimen0  % výška počátečního bodu
   \dimen3=\dp0 \advance\dimen3 by-\dimen0
                \advance\dimen3 by\dimen2   % délka svislé linky
   \pdfliteral{q 0.996264 0 0 0.996264 0 0 cm \rfType}%  parametry
   \dmoveto 0pt,\dimen2,                    % výchozí bod
   \dcurveto 0pt,.5\rfR, .5\rfR,\rfR, \rfR,\rfR,
                                            % levý horní roh
   \dlineto \dimen1,0pt,                    % vodorovná linka
   \dcurveto .5\rfR,0pt, \rfR,-.5\rfR, \rfR,-\rfR,
                                            % pravý horní roh
   \dlineto 0pt,-\dimen3,                   % svislá linka
   \dcurveto 0pt,-.5\rfR, -.5\rfR,-\rfR, -\rfR,-\rfR,
                                            % pravý dolní roh
   \dlineto -\dimen1,0pt,                   % vodorovná linka
   \dcurveto -.5\rfR,0pt, -\rfR,.5\rfR, -\rfR,\rfR,
                                            % levý dolní roh
   \pdfliteral{h B Q}}                      % close fill + stroke
```

Rámeček `\roundedframe{`⟨*text*⟩`}` se z hlediska TEXu chová jako `\hbox{`⟨*text*⟩`}`, takže jej lze použít třeba pro vyznačení tlačítka v textu odstavce. Chceme-li do rámečku schovat celý `\vbox`, je třeba psát `\roundedframe{\vbox{`⟨*text*⟩`}}`.

5. Souřadnicový systém vzhledem k počátku strany

Příkaz `\pdfliteral` má ještě jednu možnost použití s parametrem `page`. Tedy:

`\pdfliteral page {`⟨*pdf kód*⟩`}`

Toto pracuje zcela stejně jako obvyklý `\pdfliteral` jen s tím rozdílem, že výchozí souřadnicový systém neprochází aktuálním bodem sazby, ale dolním a levým okrajem papíru. Přitom vlevo dole v rožku má svůj počátek. Je tedy možno tímto způsobem nakreslit obrázek, který je ukotven ke stránce, nikoli k sazbě. Je ovšem potřeba mít v ⟨*pdf kódu*⟩ správně spárovány operátory `q` a `Q`, protože celý kód je rovněž vložen do skupiny související s popisem strany. Tím se tento `\pdfilteral` liší o běžného, kde není nutno mít spárovány `q` a `Q` a ⟨*pdf kód*⟩ více příkazů `\pdfliteral` může být kombinováno s běžnými TEXovými příkazy.

Ukážeme si příklad, v němž jsou spojeny čárou dvě místa v textu, která musejí být na stejné straně. V prvním místě napíše uživatel `\startsipky` a v druhém místě `\stopsipky`. Dále před takto označeným textem (na stejné straně) napíše `\kreslisipku`. PdfTEX disponuje příkazem `\pdfsavepos`, který se uloží jako bezrozměrná značka a aktivuje se v době činnosti `\shipout`. V takovém okamžiku uloží do registrů `\pdflastxpos` a `\pdflastypos` polohu značky v jednotkách `sp` (bez připojené jednotky), přičemž tato poloha se počítá od dolního levého rohu papíru. Dříve než v `\shipout` se polohu bodu v sazbě TEX z principu nemůže dozvědět. Je tedy potřeba použít externí soubor a polohu bodu z něj zpětně přečíst. V ukázce používáme externí REF soubor, se kterým pracuje i OPmac. Založit externí soubor můžeme manuálně, ale v ukázce využíváme zavedení makra OPmac pomocí `\input opmac`. Do REF souboru se uloží příkazy `\XpdfposA{`⟨*x*⟩`}{`⟨*y*⟩`}` a `\XpdfposB{`⟨*x*⟩`}{`⟨*y*⟩`}` (pro začátek a konec čáry). Ještě před `\input opmac` je tedy potřeba mít tyto definice:

```
\newdimen\sipkaAx \newdimen\sipkaAy
\newdimen\sipkaBx \newdimen\sipkaBy
\def\XpdfposA#1#2{\sipkaAx=#1sp \sipkaAy=#2sp \relax}
\def\XpdfposB#1#2{\sipkaBx=#1sp \sipkaBy=#2sp
   \relax}
```

Definice maker `\startsipky`, `\stopsipky` a `\kreslisipku` vypadá takto:

```
\def\startsipky{\pdfsavepos
   \wref\XpdfposA{{\the\pdflastxpos}{\the\pdflastypos}}}
\def\stopsipky {\pdfsavepos
   \wref\XpdfposB{{\the\pdflastxpos}{\the\pdflastypos}}}
\def\kreslisipku{\openref \dimen0=\sipkaAx \dimen1=\sipkaAy
   \advance\dimen0 by-\sipkaBx
   \advance\dimen1 by-\sipkaBy \dimen2=-\dimen1
   \pdfliteral page {q
   0.996264 0 0 0.996264 0 0 cm       % transformace z bp do pt
   \nopt\dimen0, \nopt\dimen1, \nopt\dimen2, \nopt\dimen0,
   \nopt\sipkaBx, \nopt\sipkaBy, cm   % souřadnice v protisměru šipky
   .04 w 1 J .8 G          % tloušťka čáry, konce oblé, barva šedá
```

```
   0 0 m   % začátek šipky
   1 0 l   % konec šipky
   0 0 m .1 0 .2 .02 .3 .1 c 0 0 m 0.1 0 .2 -.02 .3 -.1 c S % hrot
   Q
}}
```

Podobný příklad je možné najít v [1]. V řešení jsme použili transformaci souřadnic do souřadnic, kde první osa je v protisměru šipky a druhá je na ni kolmá. Vykreslení šipky pak odpovídá příkazu `0 1` (lineto) následovanému křivkami pro hrot (curveto). Makro `\nopt` je stejné jako v předchozích příkladech a makra `\wref`, `\openref` obsluhující REF soubor jsou z OPmac.

6. Forms – podprogramy v PDF kódu

Příkaz `\pdfxform` umožňuje vytvořit v PDF kódu proceduru a propojit ji s boxem v TEXu. Lze to použít na opakované volání stejné kresby. V takovém případě se do PDF souboru nevkládá kresba opakovaně znovu, ale vloží se tam jednou a v místě použití se na ni vytvoří jen odkaz. To ostatně už známe z používání primitivního příkazu `\pdfximage`. Použití příkazu `\pdfxform` si ukážeme na následujících příkladech.

6.1 Opakovaný znak kreslený křivkami

Dejme tomu, že chceme vyznačovat odrážky v seznamech pomocí následujícího znaku:

```
\def\bulletdraw{\pdfliteral{q
   0 1 1 0 k  % červená
   1 1 m 6 1 l 6 6 l 1 6 l 3 3.5 l h f % praporec
Q}}
```

Jak to vypadá, se může čtenář podívat na výčet na straně 8 v tomto článku. Také je potřeba kresbu usadit do boxu, což by mohlo vypadat takto:

```
\def\normalitem{\hbox to12pt{\vbox to 10pt{\vss\bulletdraw}\hss}}
```

Je zřejmé, že takový „znak“ se bude používat v TEXu opakovaně, a je tedy účelné pro něj zavést proceduru. Místo přímého volání `\pdfliteral` opakovaně vytvoříme pomocí `\setbox` jednou box, který může obsahovat `\pdfliteral` a ten ztotožníme z procedurou Form pomocí `\pdfxform` takto:

```
\newcount\mybullet
\setbox 0 = \hbox to12pt{\vbox to 10pt{\vss\bulletdraw}\hss}
\pdfxform 0 \mybullet=\pdflastxform
\def\normalitem{\pdfrefxform\mybullet}
```

Příkaz `\pdfxform`⟨*číslo boxu*⟩ načte obsah boxu ⟨*číslo boxu*⟩ a uloží jej do PDF souboru jako proceduru (v PDF kódu se nazývá Form). Odkazovat na tuto proceduru je možné později pomocí `\pdfrefxform`⟨*číslo formu*⟩. Argument ⟨*číslo formu*⟩ se dozvíme po vykonání příkazu `\pdfxform` z registru `\pdflastxform`. Samotný odkaz zabere v PDF kódu jen cca 6 bytů, což je podstatně méně než opakované překreslování celé procedury. Box ⟨*číslo boxu*⟩ se při činnosti příkazu `\pdfxform` vyprázdní a z hlediska sazby je příkaz `\pdfrefxform`⟨*číslo formu*⟩ shodný se sazbou uvedeného boxu.

Uvedený postup má ještě jeden důležitý rys. Ačkoli nakreslíte uvnitř boxu pomocí `\pdfliteral` obrázek libovolných rozměrů, nakonec je oříznut do hranice boxu ⟨*číslo boxu*⟩. Je tedy bezpodmínečně nutné, aby tento box měl nenulové rozměry, jinak neuvidíte nic.

6.2 Duhy, barevné přechody

Příkaz `\pdfxform` má možnost nepovinného parametru `resources {`⟨*další pdf kód*⟩`}`, který je možné vložit ještě před ⟨*číslo boxu*⟩, takže syntaxe je následující:

`\pdfxform resources {`⟨*další pdf kód*⟩`}` ⟨*číslo boxu*⟩

Uvedený ⟨*další pdf kód*⟩ může obsahovat slovníkové údaje, se kterými lokálně pracuje uvedená procedura (Form). Prakticky se to používá pro barevné přechody, kdy je ve slovníku `/Shading` definováno `/Sh`, ke kterému jsou přiřazeny odpovídající funkce, jež barevný přechod realizují. Je-li toto všechno připraveno, je možno vykreslit duhu pomocí `\pdfliteral{/Sh sh}`. Tento příkaz musí být součástí boxu ⟨*číslo boxu*⟩.

Následující příklad implementuje barevný přechod na úsečce AB. Ve vrcholu A začíná Barva1 a ve vrcholu B končí Barva5. Na cestě podél úsečky od A do B se Barva1 promění v Barvu2, dále v Barvu3, Barvu4 až v Barvu5. Umístění Barev2 až 4 je na úsečce vymezeno pomocí procent délky celé úsečky. Vrstvy duhy se stejným barevným provedením jsou kolmé na směr úsečky AB a duha je omezena v boxu, se kterým pracuje `\pdfxform`. Pro tvůrce maker je připraveno toto rozhraní:

```
\def\colorOne{1 1 1} \def\colorTwo{1 0 0} \def\colorThree{0 1 0}
\def\colorFour{0 0 1} \def\colorFive{0 0 0} % Barva1 až Barva5 v RGB
\def\shadeWidth{300} \def\shadeHeight{20}   % rozměry boxu v pt
\def\shadeFrom{0 0}
\def\shadeTo{\shadeWidth\space0}  % vymezení bodů A,B úsečky
\def\shadeTriple{25 50 75}        % polohy Barvy2 až 4 (procenta)
\newcount\myshade                 % ⟨číslo formu⟩
\createshade\myshade              % příprava duhy
box: \pdfrefxform\myshade         % použití duhy (možno opakovaně)
```

Příklad vytvoří box:

Makro `\createshade` využívá primitivní příkaz `\pdfxform resources` takto:

```
\def\createshade#1{% #1 is a counter declared by \newcount
  \setbox0=\hbox to\shadeWidth pt{\vbox to\shadeHeight pt{\vss
     \pdfliteral{q 0.996264 0 0 0.996264 0 0 cm % bt to pt conversion
        /Sh sh Q}}\hss}%
  \pdfxform resources {/Shading <<
    /Sh << /ShadingType 2 /ColorSpace /DeviceRGB /Domain [0 100]
       /Coords [\shadeFrom\space\shadeTo] /Function
    << /FunctionType 3 /Domain [0 100] /Functions
     [ << /FunctionType 2 /Domain [0 1] /C0 [\colorOne] /C1
            [\colorTwo] /N 1 >>
       << /FunctionType 2 /Domain [0 1] /C0 [\colorTwo] /C1
            [\colorThree] /N 1 >>
       << /FunctionType 2 /Domain [0 1] /C0 [\colorThree] /C1
            [\colorFour] /N 1 >>
       << /FunctionType 2 /Domain [0 1] /C0 [\colorFour] /C1
            [\colorFive] /N 1 >>
     ] /Bounds [\shadeTriple] /Encode [0 1 0 1 0 1 0 1]
    >>  /Extend [false false]
       >> >>} 0     % \pdfxform převezme \box0
  #1=\pdflastxform
}
```

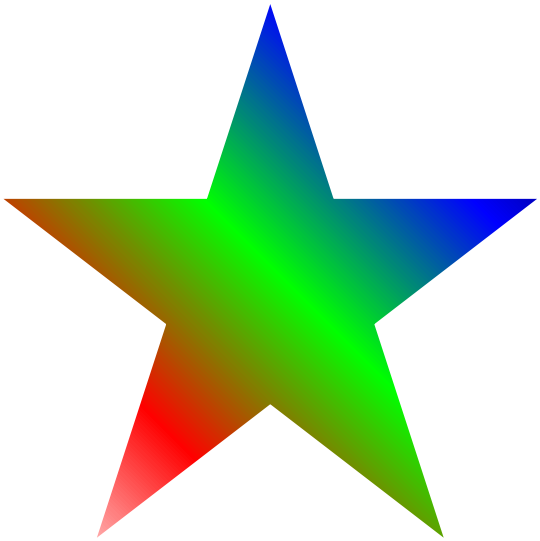
Pokud slovník naplníme jinak, dosáhneme duhy jiných vlastností. Význam jednotlivých údajů ve slovníku přesahuje rámec tohoto článku a zájemce odkazujeme na PDF referenci [8].

7. Použití ořezové křivky

Na závěr článku ukážeme kombinaci grafiky s ořezovou křivkou. Ořezovou křivku nastavíme do tvaru hvězdy a následně vykreslíme duhu. Dostáváme duhovou hvězdu.

```
\newcount\shade
\def\shadeFrom{0 0} \def\shadeTo{\shadeWidth\space\shadeHeight}
\def\shadeWidth{201} \def\shadeHeight{201}
\createshade\shade

\centerline{\pdfliteral{q                          % pdfsave
  35 0 m 100 200 l 165 0 l 0 127 l 200 127 l % hvezda
  h W n                                        % uzavrit a orez
  }\rlap{\pdfrefxform\shade}%                  % duha
  \pdfliteral{Q}\hskip201pt}                   % pdfrestore
```



8. Využití Inkscape pro přípravu kresby

Při tvorbě šablony CUstyle [7] jsem potřeboval do sazby vkládat jednoduché piktogramy. Existují dva způsoby jak toho dosáhnout:

- Nakreslit piktogramy nějakým grafickým editorem a vložit je do sazby pomocí `\pdfximage`.
- Použít TikZ [3] nebo něco podobného a obrázky naprogramovat v makrech TEXu.

Nevýhodou prvního z nich je, že vzniká sada externích soborů, se kterými je třeba při sazbě nějak manipulovat: umístit je na potřebné místo, kde je pdfTEX najde, archivovat je společně s makry atd.

Nevýhodou druhého způsobu je, že programování složitějších obrázků je poněkud šílené, mnohdy až nemožné. Zejména pokud si člověk uvědomí, že v grafickém editoru má totéž za pár minut.

Rozhodl jsem se tedy pro postup třetí, který vylučuje nevýhody obou předchozích postupů a spojuje jejich výhody. Požádal jsem mladšího syna Radka, ať mi potřebný piktogram nakreslí v interaktivním grafickém editoru Inkscape. To mu zabralo opravdu jen pár minut. Pak provedl export do `*.eps`. Když jsem tento EPS soubor otevřel textovým editorem, shledal jsem, že tam jsou nejprve PostScriptové definice typu `/s {curveto} def /l {lineto} def` atd. a dále je celý obrácek nakreslen klasickým PDF kódem. Stačilo tedy vyhledat první `q` a jemu odpovídající poslední `Q` a tento blok přesunout do argumentu `\pdfliteral` v makrech, která se starají o tyto piktogrmy. A je vymalováno. Doslova. Žádné načítání složitých maker typu TikZ, žádné „programování“ obrázků, žádné starosti s externími obrázky. TEXová makra řeší piktogramy ve vlastní režii.

9. Literatura

[1] Zýka, Vít. Používáme pdfTEX I–V. *Zpravodaj CSTUG*, 4/2001 (doi: 10.5300/2001-4/181), 1/2002 (doi: 10.5300/2002-1/13), 2/2002 (doi: 10.5300/2002-2/47), 2–3/2002 (doi: 10.5300/2002-3-4/140), 2/2004 (doi: 10.5300/2004-2/47), 1/2005 (doi: 10.5300/2005-1/90), 2/2007 (doi: 10.5300/2007-2/67).

[2] Chvála, František. O možnostech pdfTEXu. *Zpravodaj CSTUG*, 1/2005 (doi: 10.5300/2005-1/2).

[3] Tantau, Till. *TikZ & PGF*: manual. Soubor `pgfmanual.pdf` v distribucích TEXu. Dostupné na: `http://sourceforge.net/projects/pgf/`.

[4] Thành, Hàn Thế et al. *The pdfTEX user manual*. Dostupné na: `http://www.tug.org/applications/pdftex/`.

[5] Olšák, Petr. *OPmac – rozšiřující makra plainTEXu*. Dostupné na: `http://petr.olsak.net/opmac.html`.

[6] Olšák, Petr. *Uživatelská dokumentace k OPmac*. Dostupné na: `http://petr.olsak.net/ftp/olsak/opmac/opmac-u.pdf`.

[7] Olšák, Petr. *CUstyle – Šablona v plainTEXu pro sazbu studentských závěrečných prací na Univerzitě Karlově*. Dostupné na: `http://petr.olsak.net/custyle.html`.

[8] PDF reference. Dostupné na: `http://www.adobe.com/devnet/pdf/pdf_reference.html`.

Summary: Simple Graphics with PDF-primitives

When an user inserts a simple graphics (a few lines, for example) to the document then he doesn't need to use a complicated macro package or a software in order to programme or generate such graphics. The usage of low level PDF code is sufficient and more straightforward. We need to know only a small set of PDF operators to do interesting results.

This article summarizes the basic primitive commands of PDF language and of the pdfTEX. They are illustrated in many examples here. Some similar techniques were mentioned in the articles [1–2] but more original examples are presented in this article.

We have to distinguish between creating a simple graphics presented in this article and programming more complex pictures by programming language at higher level like TikZ [3].

Key words

pdfTEX, PDF code, graphics

Interaktivní 3D grafika v PDF dokumentech

Roman Plch

Příspěvek navazuje na článek [8] a všímá si zejména změn, které v problematice vkládání interaktivní 3D grafiky do PDF dokumentů nastaly od doby jeho publikování. Balíček `movie15` byl nahrazen balíčkem `media9`, který začleňování 3D grafiky do PDF dokumentů významným způsobem zjednodušuje. Druhá část příspěvku popisuje současné možnosti tvorby matematických 3D objektů a jejich konverzi do formátů PRC a U3D a zaměřuje se zejména na využití open source produktů.

Klíčová slova

interaktivní 3D grafika, U3D, PRC, PDF, pdfLaTeX

Úvod

Formát PDF (Portable Document Format) od firmy Adobe se během uplynulých let stal de facto standardem pro výměnu dokumentů nezávislou na platformě. Tento formát zajišťuje, že libovolný dokument bude na všech zařízeních zobrazen stejně. Jednou z „nových“ vlastností tohoto formátu je možnost začleňování 3D objektů (od verze PDF-1.6).

Začleněním podpory 3D do formátu PDF získává tento formát nový rozměr. Již není nutné se při zobrazování 3D dat omezovat pouze na dvojrozměrné exporty a projekce. Nyní je možné do PDF začlenit skutečně interaktivní 3D materiál. Interaktivita 3D objektů v PDF spočívá v možnosti rotace, posunu, zvětšení, změny osvětlení, změny projekce, rozložení 3D objektu a zobrazení průřezů. S pomocí JavaScriptu jsou možné i animace. Můžeme takto šířit 3D modely s jistotou, že si příjemce tento objekt prohlédne, aniž by vlastnil aplikaci, ve které byl model vytvořen.

Vkládání 3D objektů do PDF dokumentu jsme se poprvé věnovali ve Zpravodaji v roce 2008 ([8]), tento článek téma dále rozšiřuje a shrnuje vývoj, který od té doby proběhl.

Interaktivní grafika v PDF dokumentu

V současné době je možné do PDF dokumentů vkládat 3D objekty ve dvou formátech – U3D a PRC.

doi: 10.5300/2013-1/31

Universal 3D (U3D) je otevřený souborový formát sloužící pro ukládání a přenos 3D dat. Byl navržen konsorciem 3DIF (3D Industry Forum), v srpnu 2005 byl standardizován společností Ecma a slouží jako univerzální standard reprezentace 3D dat všeho druhu, umožňující přenositelnost mezi různými platformami. Specifikaci lze nalézt na [3], v současné době již ve čtvrté edici (z června 2007).

Na formát U3D bylo kladeno několik zásadních požadavků. Aby byl použitelný opravdu univerzálně, musí být jeho vnitřní datová reprezentace dostatečně jednoduchá, neobsahuje tedy žádné složité geometrické struktury, základním prvkem všech objektů je nejjednodušší objekt – trojúhelník. Dále se v tomto formátu nevyskytují spliny nebo spline povrchy, žádná pole vektorů atd. U3D umožňuje také kompresi, která znatelně snižuje objem dat 3D scény oproti formátům používaným v CAD systémech. Kompresní poměr může být až 1 : 30. Tento formát začala podporovat i firma Adobe, a to od specifikace PDF-1.6 (Adobe Acrobat 7.0 a Adobe Reader 7.0).

Přes svou otevřenost se formát U3D příliš nerozšířil, proto Adobe umožnilo vkládání 3D objektů i ve formátu PRC (Adobe Acrobat 8.1(3D) a Adobe Reader 8.1). Formát PRC (Product Representation Compact, [1]) je ISO standardem pro ukládání informací o 3D objektu (struktura, geometrie, materiál, ...). PRC poskytuje především vysoké kompresní poměry (až stokrát menší objem oproti původnímu CAD souboru) a schopnost zachování detailů použitelných dále v CAD, CAM a CAE aplikacích. Rovněž garantuje vysokou rychlost načítání i zpracování modelů v prostředí Adobe Readeru.

Vložení grafického objektu ve formátu U3D (PRC) do PDF dokumentu

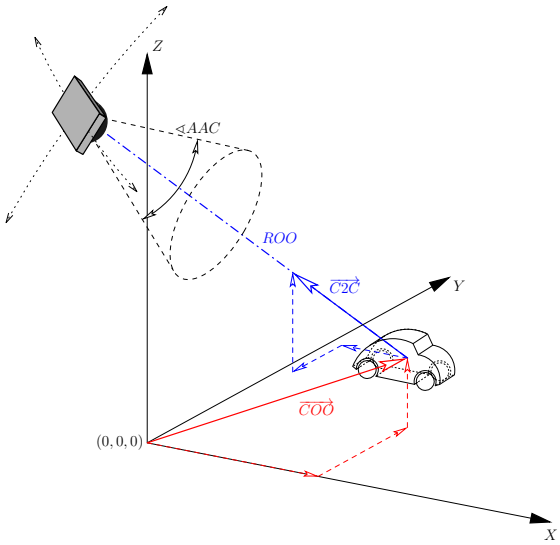
3D grafický objekt vložíme do PDF dokumentu buď pomocí komerčního produktu Adobe Acrobat v kombinaci s pluginem 3D PDF Converter (dříve 3D Reviewer) od společnosti tetra4D (`http://www.tetra4d.com`), nebo použijeme sázecí systém TEX a balíček `media9` ([6]). Dále se věnujeme pouze „nekomerční“ cestě, tj. využití TEXu a balíčku `media9`. Pro přímý výstup do PDF dokumentu použijeme pdfTEX (požadována verze nejméně 1.30). Pro korektní zobrazení výsledného dokumentu musíme použít Adobe Reader verze 9 (a vyšší) a v nastavení Adobe Readeru zvolit oboustranné zobrazení grafického objektu (Preferences – 3D & Multimedia – Enable double-sided rendering).

Balíček načteme ve zdrojovém souboru příkazem

```
\usepackage{media9}
```

a vlastní začlenění interaktivní grafiky provedeme příkazem `\includemedia`, jehož syntaxe je:

```
\includemedia[volby]{text}{soubor.u3d}.
```

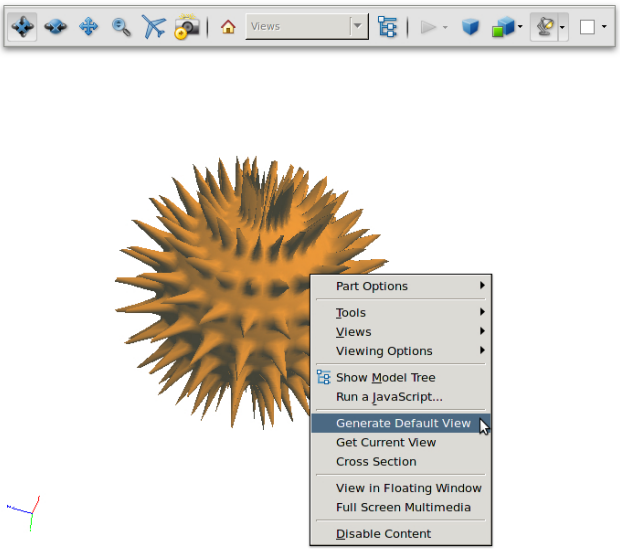


Obrázek 1: Umístění objektu na scéně, převzato z [6]

Podrobný popis všech voleb pro začleňování 3D grafických objektů najdeme v manuálu k balíčku `media9` ([6]). Zmíníme jen některé z nich:

- Volbou `3Dlights=<lighting scheme>` nastavíme osvětlení objektu, například `3Dlights=Day` nastaví denní barvy. Implicitně se používá osvětlení specifikované v 3D modelu.
- Volba `3Dbg=<r> <g> <b>` definuje barvu pozadí. Hodnoty je možné zadávat jako čísla v pohyblivé desetinné čárce v rozsahu od 0 do 1.
- Volbou `3Drender=<render mode>` určíme zobrazovací mód, například zobrazení drátěného modelu nastavíme pomocí `3Drender=Wireframe`.
- Volbou `3Dmenu` přidáme do menu ovládacího panelu položky „Generate Default View“, „Get Current View“ a „Cross Section“.
- Volba `activate=onclick | pageopen | pagevisible` definuje způsob aktivace objektu (média). Pokud objekt není aktivní, zobrazuje se obsah nastavený parametrem `text`.
- Volbou `3Dtoolbar` zobrazujeme ovládací panel (umístěním kurzoru myši na obrázek).

Za nejdůležitější volby považujeme ty, kterými ovlivňujeme umístění objektu na scéně, případně různé pohledy na objekt. Budeme se jim proto věnovat podrobněji.



Obrázek 2: Umístění objektu na scéně – nastavení parametrů

Optimální umístění objektu na scéně

Umístění objektu na scéně (obr. 1) je specifikováno pomocí vektoru $\overrightarrow{COO}$ směřujícího z počátku soustavy souřadnic do středu objektu (volba `3Dcoo`), pomocí vektoru $\overrightarrow{C2C}$ směřujícího ze středu objektu do virtuální kamery (volba `3Dc2c`) a pomocí vzdálenosti *ROO* virtuální kamery od objektu (volba `3Droo`). Kromě toho lze volbou `3Daac` nastavit průzorový úhel (aperture angel) kamery a volbou `3Droll` otočení kamery o daný úhel kolem optické osy.

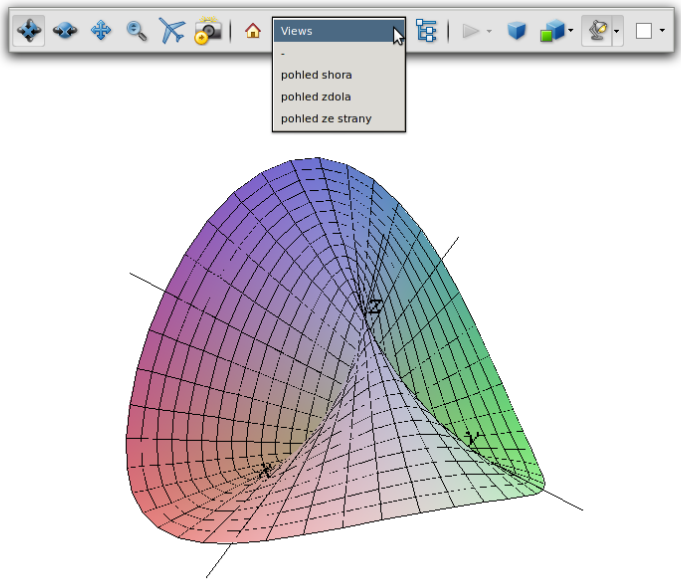
Pokud nejsou tyto parametry nastaveny, je virtuální kamera umístěna na pozici $(0, 0, 0)$ a dále `3Droo=0`, `3Dcoo=0 0 0`, `3Dc2c=0 -1 0`, `3Daac=30`, `3Droll=0`.

Vzhledem k tomu, že výpočet optimálního nastavení těchto parametrů je poměrně obtížný, je výhodné použití volby `3Dmenu`:

```
\includemedia[
  width=0.6\linewidth, height=0.6\linewidth,
  activate=pageopen,
  3Dmenu
]{}{jezura1.u3d}
```

Grafický objekt nastavíme do námi požadované polohy a zvolíme vhodnou velikost (např. pomocí menu „Part Options“ – „Fit Visible“, které aktivu-

doi: 10.5300/2013-1/34



Obrázek 3: 3DToolbar a ukázka použití pojmenovaných pohledů

jeme kliknutím pravým tlačítkem myši na objekt). Poté pomocí položky menu „Generate Default View“ (obr. 2) otevřeme okénko, kde jsou uvedeny aktuální hodnoty parametrů pro umístění objektu. Získané hodnoty zkopírujeme do zdrojového textu a znovu přeložíme.

Zobrazení různých pohledů na scénu

V Adobe Readeru nastavíme objekt do námi požadované polohy, pomocí položek „Part Options“ a „Viewing Options“ 3D kontextového menu nastavíme viditelnost částí objektu, barvu pozadí atd. a nakonec pomocí volby menu „Get Current View“ odečteme parametry a tyto uložíme do externího textového souboru `<views file>` (v příkladu níže je to soubor `pohledy.vws`). Každému pohledu přiřadíme nějaký název.

Pomocí volby `3Dviews=<views file>` příkazu `\includemovie` specifikujeme tento externí soubor s přednastavenými pohledy. Ve výsledném PDF dokumentu se tyto pojmenované pohledy zobrazí v Toolbaru (obr. 3) a je možno z nich vybírat.

```
\includemedia[
  label=obr2,
  width=0.6\linewidth,height=0.6\linewidth,
  activate=pageopen, 3Dmenu,
  3Dcoo=0.42 0.43 0.18, 3Droo=6,
  3Dc2c=0.5 0.3 0.8, 3Droll=116,3Dviews=pohledy.vws,
  3Dlights=Headlamp
]{}{zprav.prc}
```

Obsah souboru `pohledy.vws`:

```
VIEW=pohled shora
  COO=-0.000000007450580597 0 0.000000014901161194
  C2C=0.05584815889596939 0.9959907531738281 0.0698816552758216
  ROO=6.000000130885302
  ROLL=136.75371971291514
  BGCOLOR=1. 1. 1.
  LIGHTS=Headlamp
  RENDERMODE=Solid
END

VIEW=pohled zdola
  COO=0.000000044703483582 -0.000000238418579102 -0.000000014901
  C2C=0.10392342507839203 -0.9910069704055786 -0.084291554987430
  ROO=6.000000120371679
  ROLL=-112.8719769403439
  BGCOLOR=1. 1. 1.
  LIGHTS=Headlamp
  RENDERMODE=Solid
END
```

Odkazy na předdefinované pohledy vytváříme pomocí příkazu `\mediabutton` s volbou `3Dgotoview` (od `media9` verze 0.22).

```
Podívejte se na danou funkci z~různých pohledů --
\mediabutton[
  3Dgotoview=obr2:0
]{\textcolor{red}{shora}},
\mediabutton[
  3Dgotoview=obr2:1
]{\textcolor{red}{zdola}}
a~\mediabutton[
  3Dgotoview=obr2:2
]{\textcolor{red}{ze strany}}.
```

kde `obr2` je odkaz na značku vytvořenou příkazem `\includemedia` a 0 (resp. 1, 2) je číslo pojmenovaného pohledu.

3D objekty v PDF dokumentech lze ovládat i pomocí JavaScriptu. Skript se specifikuje volbou `add3Djscript=<JavaScript file>` a spustí se aktivováním 3D objektu v dokumentu. Při práci s Adobe Readerem na linuxových systémech je třeba nastavit systémovou proměnnou `export LC_NUMERIC="C"`, jinak se vložený JavaScript neprovede. Více o použití JavaScriptu v PDF dokumentech v [2].

Vytváření a konverze 3D objektů

K vytváření „matematických“ 3D objektů je možno využívat velké množství specializovaných či obecných matematických programů. Některým z nich se věnujeme podrobněji dále. Získaný 3D objekt je následně nutno převést do formátu PRC (U3D). Pro konverzi můžeme použít řadu komerčních a volně šířených programů:
Komerční produkty:

- Deep Exploration `http://www.righthemisphere.com/products/dexp/de_std.html`
- 3D PDF Converter (dříve 3DReviewer, součást Acrobatu 3D) `http://www.tetra4d.com/3dpdf`
- PDF3D ReportGen (k dispozici i linuxová verze) `http://www.pdf3d.com/products.php`
- Okino Universal-3D Geometry Export Converter `http://www.okino.com/conv/exp_u3d.htm`

„Nekomerční“ produkty:

- Meshlab `http://meshlab.sourceforge.net/`
- Jreality `http://www3.math.tu-berlin.de/jreality/`
- IDTFConverter `http://sourceforge.net/projects/u3d/`

Ukažme si nyní na konkrétních příkladech možnosti (a úskalí) tvorby a konverze matematických 3D objektů.

Maxima

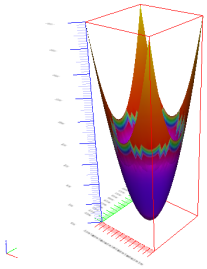
Maxima je svobodný, komplexní systém počítačové algebry (`http://maxima.sourceforge.net/`).

Graf funkce dvou proměnných vytvoříme příkazem `draw3d` z balíčku `draw`. Po nastavení terminálu pro vykreslování grafiky na VTK (`http://riotorto.users.sourceforge.net/vtk/index.html`, místo implicitního gnuplot) můžeme 3D grafiku uložit ve formátu VRML[1].

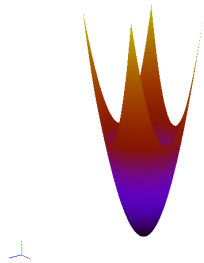
[1] Virtual Reality Modeling Language, založený na deklarativním programovacím jazyce, navržený především pro popis trojrozměrných scén, více např. na `http://cs.wikipedia.org/wiki/VRML`

```
load(draw);
draw_renderer : 'vtk $
draw3d(
    axis_3d   =true,
    file_name = "ukazka",
    terminal= vrml,
    enhanced3d = true,
    explicit(sin(x^2+y^2)/5, x, -2, 2, y, -2, 2) )$
```

Na obrázcích 4 a 5 vidíme rozdíly v konverzi 3D objektu ve formátu VRML získaného v Maximě, při použití komerčního PDF3D ReportGen a nekomerčního Meshlabu. Oba programy nastavují stejné měřítko na osách, Meshlab ale neumožňuje konverzi (zobrazení) os.



Obrázek 4: Graf vytvořený v Maximě, konverze do PRC pomocí PDF3DReportGen



Obrázek 5: Graf vytvořený v Maximě, konverze do U3D pomocí Meshlabu

Sage

Sage (`http://www.sagemath.org`) je dalším ze svobodných systémů počítačové algebry. Zajímavostí je využití prostředí internetového prohlížeče pro grafické uživatelské rozhraní.

Graf funkce dvou proměnných vytvoříme pomocí následujících příkazů

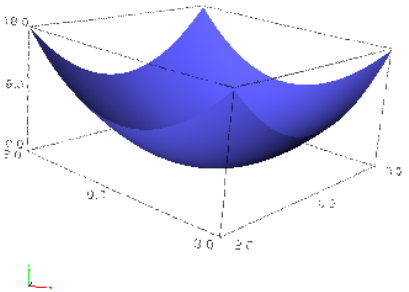
```
var('x y'); plot3d(y^2-x^2,(x,-2,2),(y,-1,1))
```

Získaný graf uložíme ve formátu JMOL (Toggle Advanced Controls – Download this view). Následně tento soubor otevřeme v programu Jmol (`http://jmol.sourceforge.net/`) a exportujeme do formátu IDTF[2]. Exportovaný soubor poté převedeme do U3D pomocí programu `IDTFconverter` příkazem

[2]Intermediate Data Text Format, `http://wiki.jmol.org/index.php/File_formats/3D_Objects`

```
IDTFconverter -input soubor.idtf -output soubor.u3d
```

Výsledek (po vložení do PDF dokumentu) vidíme na obrázku 6. I když popsaný postup vypadá na první pohled trochu komplikovaně, jedná se (zatím) o jedinou čistě nekomerční cestu k začlenění matematické 3D grafiky vytvořené CAS systémem (i s osami a popisem os) do PDF dokumentu.



Obrázek 6: Graf vytvořený pomocí CAS Sage

Maple

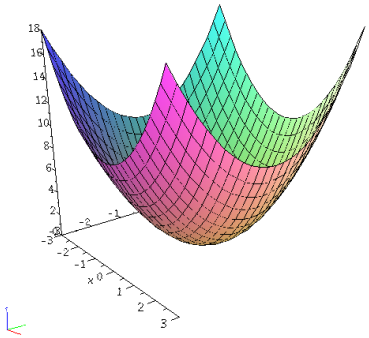
Komerční systém počítačové algebry Maple (`http://www.maplesoft.com`) a tvorbu grafu funkce dvou proměnných jsme popsali již v [8], [9]. Novější verze Maplu (od verze 13) umožňují navíc export 3D grafiky do formátu X3D[3] jednoduše kliknutím pravým tlačítkem myši na obrázek a volbou Export – Extensible 3D. Pro následnou konverzi do formátu PRC je možné použít program `maplex3d2prc`, který umí exportovat i osy s popisem a zachovává nastavená měřítka na osách a barevné schéma (obr. 7). Program se spouští s jediným argumentem – jménem souboru X3D exportovaným z Maplu. Na výstupu dostáváme PRC soubor, PDF soubor s vloženou 3D grafikou a JavaScript, který zajišťuje správnou orientaci popisu os a musí být připojen při vkládání objektu do PDF souboru.

Tento převodník zatím není k dispozici volně ke stažení, autor Michail Vidiassov jej však na vyžádání zašle na testování.

Matlab, R

Na závěr ještě stručně zmíníme tvorbu 3D objektů ve dvou programech na zpracování dat, v komerčním Matlabu a volně šířeném R:

[3] Extensible 3D, XML formát na ukládání 3D scén, ideový nástupce VRML, `http://www.web3d.org/x3d/`



Obrázek 7: Graf vytvořený v Maplu a převedený do PRC pomocí `maplex3d2prc`

- Matlab – export do VRML pomocí příkazu `vrml` nebo pomocí balíčku `fig2u3d` (`http://www.mathworks.com/matlabcentral/fileexchange/37640`)
- R – použití balíčku `misc3d` (`http://cran.r-project.org/web/packages/misc3d/misc3d.pdf`) a příkazu

  ```
  exportScene(scene, filename, format=c("OFF", "IDTF", "ASY"))
  ```

Obdobně můžeme postupovat s libovolným matematickým programem, který umožňuje export v některém z 3D formátů. Pro následnou konverzi do PRC (U3D) bez nutnosti použití komerčních produktů se jako nejvýhodnější jeví formát IDTF (např. Sage, R). Pro ostatní formáty je nutno použít Meshlab[4] (bohužel bez možnosti exportu i s osami a popisem) nebo zakoupit některý z komerčních programů. Do budoucna se plánuje možnost načítání 3D formátů do programu Asymptote (viz následující kapitola) či přímý PRC výstup z programu R.

Přímá tvorba 3D objektu ve formátu PRC – Asymptote

Asymptote (`http://asymptote.sourceforge.net/`) je interpretovaný programovací jazyk se syntaxí podobnou C++ určený pro generování grafiky. Mezi jeho výhody patří zejména tyto možnosti:

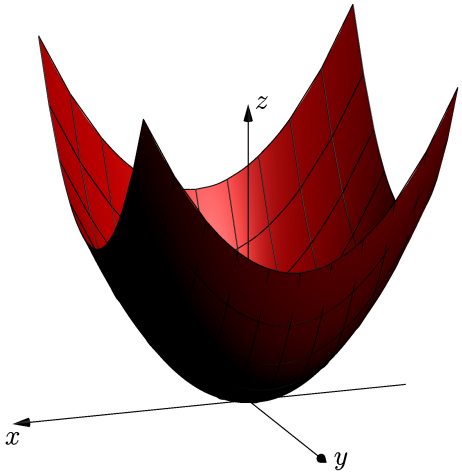
- na výstupu můžeme získat 3D grafiku přímo ve formátu PRC;
- pro popis obrázků využívá TEX;
- existuje LATEXový balíček, který umožňuje vkládat kód Asymptote přímo do zdrojového dokumentu.

[4] Meshlab aktuálně pracuje s formáty PLY, STL, OFF, OBJ, 3DS, COLLADA, PTX, V3D, PTS, APTS, XYZ, GTS, TRI, ASC, X3D, X3DV, VRML a ALN.

Není tedy problém vytvářet interaktivní grafiku i s popisem, jak je vidět v následující ukázce (obr. 8), kterou uvádíme pro ilustraci i se zdrojovým kódem. Celou řadu dalších ilustrativních ukázek najdeme na webu Asymptote.

V preambuli dokumentu načteme balíček
`\usepackage[inline]{asymptote}`

Nyní můžeme použít prostředí `asy` a do něj umístit zdrojový kód obrázku. Druhou možností je načtení kódu ze samostatného souboru pomocí příkazu `\asyinclude`.



Obrázek 8: Graf vytvořený pomocí programu Asymptote

```
\begin{asy}
import graph3;

size (200 ,200 , keepAspect = false );
currentprojection = orthographic (3 ,9 ,5);

real f( pair z){
real x=z.x,y=z.y;
return x^2+y^2;
}

draw ( surface (f ,( -2 , -2) ,(2 ,2) ,xsplinetype = Spline ),
red , meshpen = black +0.5) ;

xaxis3 ("$x$ " ,-2,3, Arrow3 );
```

```
yaxis3 ("$y$ " ,-2,3, Arrow3 );
zaxis3 ("$z$ " ,0,7, Arrow3 );
\end{asy}
\caption{Graf vytvořený pomocí programu Asymptote}\label{asym}
\end{figure}
```

Pro kompilaci dokumentu s grafikou pak použijeme

```
pdflatex dokument.tex
asy dokument-*.asy
pdflatex dokument.tex
```

Závěr

Začlenění podpory pro vkládání interaktivních 3D objektů do formátu PDF výrazným způsobem ovlivnilo možnosti vytváření elektronických matematických publikací. Interaktivní matematickou grafiku můžeme nyní vkládat do textů vysoké typografické kvality, připravovaných systémem TEX. Ve výsledném dokumentu přitom zůstává zachována možnost interakce uživatele s 3D objektem bez nutnosti instalovat dodatečný software.

Nový balíček `media9` významným způsobem zjednodušuje proces tvorby PDF dokumentu s vloženou 3D grafikou. Také exportní možnosti matematických programů a nabídka programů pro následnou konverzi do PRC (U3D) se neustále zlepšují. Můžeme tedy očekávat, že vkládání 3D objektů do PDF dokumentů bude v brzké budoucnosti stejně přirozené, jako je v současnosti vkládání obrázků.

Reference

[1] Adobe Systems Inc. *PRC Format Specification*, dostupné na `http://livedocs.adobe.com/acrobat_sdk/10/Acrobat10_HTMLHelp/API_References/PRCReference/PRC_Format_Specification/index.html`

[2] Adobe Systems Inc. *JavaScript for Acrobat 3D Annotations* [online], duben 2007. Dostupné na www: `http://www.adobe.com/devnet/acrobat/pdfs/js_3d_api_reference.pdf`

[3] ECMA International *Universal 3D File Format (ECMA-363), 4th Edition*, 2007, dostupné na `http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-363%204th%20Edition.pdf`

[4] PDF documents with integrated 3D interactive models: IDTF to U3D. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001 [cit. 2014-02-10]. Dostupné na: `http://wiki.jmol.org/index.php/File_formats/3D_PDF`

[5] Graf, Norman A. 3DPDF: Open Source Solutions for Incorporating 3D Information in PDF Files. In: *IEEE 2012 Nuclear Science Symposium, Medical*

Imaging Conference [online]. 2012 [cit. 2013-05-23]. Dostupné na: http://www.slac.stanford.edu/cgi-wrap/getdoc/slac-pub-15295.pdf

[6] Grahn, Alexander. *The media9 package* [online], leden 2008. Dostupné na www: http://mirrors.ctan.org/macros/latex/contrib/media9/doc/media9.pdf

[7] Kutal, Ondřej. *Tvorba matematické grafiky pomocí programu Asymptote.* 2012, Brno. Dostupné na: http://www.math.muni.cz/~plch/diplomky/asymptote.pdf. Diplomová práce. Masarykova univerzita.

[8] Plch, Roman; Šarmanová, Petra. Interaktivní 3D grafika v HTML a PDF dokumentech. *Zpravodaj CSTUG.* 2008, roč. 18, 1–2, s. 76–92. (doi: 10.5300/2008-1-2/76.) Dostupné též na: http://bulletin.cstug.cz/pdf/bul_0812.pdf

[9] Plch, Roman; Šarmanová, Petra. An Interactive Presentation of Maple 3D Graphics in PDF Documents. *The Electronic Journal of Mathematics and Technology* [online]. 2008, roč. 2, č. 3, s. 281–290 [cit. 2012-04-27]. Dostupné na: https://php.radford.edu/~ejmt/deliveryBoy.php?paper=eJMT_v2n3n1

[10] Plch, Roman; Šarmanová, Petra. Interaktivní prezentace matematické grafiky na webu a v PDF dokumentech. In: *Sborník semináře Technologie pro e-vzdělávání.* Praha: ČVUT FEL, 2007, s. 31–38. ISBN 978-80-01-03756-0. Dostupné na: http://homel.vsb.cz/~s1a64/publikace/clzivy.pdf

Summary: Interactive 3D Graphics in PDF Documents

The paper presents the authors' experience with including interactive 3D objects into PDF documents by using pdfTeX and the media9 package. This procedure preserves the possibility of the user's interaction with 3D objects even in the final PDF document without the necessity of the local installation of original graphical programs. In the second part, creation of 3D graphics in several mathematics programs and its conversion to PRC (U3D) with the use of commercial and open source software has been described.

Key words

interactive 3D graphics, U3D, PRC, PDF, pdfLaTeX

Roman Plch, plch@math.muni.cz

Tisk obrysu písma v PDF

Zdeněk Wagner

Článek ukazuje možnost vykreslení obrysu písma přímým využitím operátorů PDF. Je vysvětlen způsob použití v ovladačích rodiny (x)dvipdfm(x) i v pdfTEXu. Jsou zmíněny možné komplikace a je uveden případ, kdy nelze metodu použít.

Klíčová slova

PDF, obrysy písma, výplň písma, iniciály, Open Type, Type 1

1. Úvod

Tiskoviny některých typů, zejména reklamní, ale i beletrie, občas vyžadují, aby části textu byly zvýrazněny méně obvyklým způsobem. Příkladem takového zvýraznění je iniciála v úvodu tohoto odstavce. Není vytištěna speciálním fontem, jedná se o znak volně šiřitelného písma TEX Gyre Adventor. Rozdíl spočívá v tom, že jsme znak nevytiskli běžným způsobem, ale nechali jsme v PDF zobrazit obrys, který jsme vyplnili jinou barvou. To je umožněno tím, že fonty ve formátech Type 1 a OpenType definují znaky pomocí obrysů, které jsou při normálním tisku vyplněny zvolenou barvou. My jsme pouze použili jiný způsob vykreslení, který si detailně popíšeme v následujícím textu.

2. Zobrazení obrysu písma v PDF

Při zobrazení obrysu písma využijeme toho, že pomocí speciálních příkazů lze do výstupu přímo vkládat operátory PDF. Stejně jako v PostScriptu jsou parametry uváděny před jménem operátoru.

V dokumentaci PDF[1] najdeme operátor `Tr`, jehož popis není zrovna srozumitelný. A právě tento operátor určuje, jakým způsobem je písmo zobrazeno. Obvyklá hodnota 0 způsobí klasické vyplnění tak, jak to vidíte prakticky ve všech textech. Nastavením hodnoty 1 zobrazíme obrys bez výplně, hodnota 2 umožňuje zobrazení obrysu v jedné barvě a výplně v jiné. Tloušťku linky nastavíme operátorem `w`, přičemž jednotkou je *big point*. Barvu nastavíme buď operátorem `g` a `G` (šedá, operátor má jeden parametr), nebo `rg` a `RG` (barevný prostor RGB, tři parametry), nebo `k` a `K` (barevný prostor CMYK, čtyři parametry). Operátory zapsané malými písmeny nastavují barvu výplně, operátory psané velkými písmeny určují barvu linky. Iniciála v úvodním odstavci má

 doi: 10.5300/2013-1/44

tedy nastavení `2 Tr .8 w 0 .1 1 .1 k .8 1 0 0 K`. Před tímto nastavením je uschován grafický stav operátorem `q` a po tisku iniciály je grafický stav obnoven operátorem `Q`. Vše vypadá velmi jednoduše. Stačí tedy vysvětlit, jak zmíněné příkazy dostaneme do PDF.

2.1. Zobrazení obrysu v ovladačích (x)dvipdfm(x)

Začneme jednodušším případem, kterým je rodina ovladačů (x)dvipdfm(x) [2]. Efekt lze tedy využít i v XeTeXu. Následující ukázku

Několik slov si vytiskneme v obrysech.

jsme vytvořili pomocí příkazů

```
{\centering \Large \fontfamily{qag}\bfseries
Několik\special{pdf: code q 2 Tr 0.7 w 0 .5 .5 .05 k 1 1 0 .1 K}
slov\special{pdf: code 0 0 1 0 k 1 0 1 .1 K}
si vytiskneme\special{pdf: code Q} v~obrysech.\par}
```

Ve starších verzích ovladačů (x)dvipdfm(x) není v příkazu `\special` dostupná funkce `code`, ale pouze `put`, která však vkládá obsah mezi `q` a `Q`, čímž je požadovaný efekt ztracen. Lze to napravit jednoduchým trikem, kdy tyto automaticky vložené operátory eliminujeme párovým operátorem. V takovém případě místo `\special{pdf: code q 2 Tr}` napíšeme upravený kód s přidanými operátory `\special{pdf: put Q q 2 Tr q}`. Příkaz `\special{pdf: put Q}` žádnou modifikaci nepotřebuje.

2.2. Zobrazení obrysu v pdfTeXu

Inspirováni předchozí ukázkou napíšeme kód mechanicky s použitím primitivu `\pdfliteral` [3] takto:

```
{\centering \Large \fontfamily{qag}\bfseries
Jedno\pdfliteral{q 2 Tr 0.7 w 0 0 1 0 k~1 0 1 .1 K}
slovo\pdfliteral{Q} jinak.\par}
```

Výsledek však nesplní naše očekávání. Slovo *jinak* bude přetištěno přes *slovo* a navíc se výrazně rozhází sazba zbytku článku. Z tohoto důvodu zde ukázka není, vyzkoušejte si to sami.

Kde se stala chyba? Rozdíl spočívá v tom, že pdfTeX opakovaně nastavuje *current transformation matrix*, ale úschovou a obnovením grafického stavu tuto snahu zničíme. Musíme tedy pdfTeXu říci, aby dočasně od tohoto nastavování upustil, čehož dosáhneme vložením klíčového slova `page`. Zde je fungující kód:

```
{\centering \Large \fontfamily{qag}\bfseries
Jedno\pdfliteral page {q 2 Tr 0.7 w 0 0 1 0 k 1 0 1 .1 K}
slovo\pdfliteral page {Q} jinak.\par}
```

3. Kdy nelze obrysové písmo použít?

Popisovaná metoda předpokládá, že použitý font má formát OpenType nebo Type 1. I tehdy se ale může stát, že tento postup nelze použít. To nastává tehdy, když se znaky v písmu spojují, jako je tomu v písmu arabském a v písmech indických odvozených z písma bráhmí. Na obrázku, obsahujícím přání k muslimskému svátku *íd* v hindštině a urdštině, je zřejmé, že je zobrazen způsob, jak je text ze znaků složen, nikoliv obrys výsledného textu.

Reference

[1] *PDF Reference, sixth edition: Adobe Portable Document Format version 1.7.* Adobe Systems Incorporated, November 2006.

[2] Wicks, Mark A. *Dvipdfm User's Manual.* Version 0.12.4b. September 19, 1999.

[3] Thành, Hàn Thế; Rahtz, Sebastian; Hagen, Hans; Henkel, Hartmut; Jackowski,Paweł; Schröder, Martin. *The pdfTEX user manual.* Rev. 655. November 23, 2010.

Summary: Printing the font outline in PDF

The article presents the possibility of printing the script outline by direct usage of PDF operators. Method of use in the (x)dvipdfm(m) drivers family as well as in pdfTEX is explained. Possible complications are mentioned and an example is shown where the method is inapplicable.

Key words

PDF, font outline and fill, inicial letters, Open Type, Type 1

Zdeněk Wagner
Ice Bear Soft
http://icebearsoft.euweb.cz

PDFuni – akcenty v PDF záložkách

Petr Olšák

Záložky v PDF dokumentu (outlines, bookmarks) jsou texty, které za jistých okolností zobrazují PDF prohlížeče a které jsou klikací: rozvírají se a nabízejí stromovou strukturu. A hlavně klik na text v záložce způsobí skok na tomu odpovídající místo v dokumentu. Záložky se nikdy netisknou. Mnozí uživatelé pdfTEXu si jistě všimli, že český nebo slovenský text má někdy sklon ke znehodnocení akcentovaných znaků, pokud jej posíláme do záložek. V tomto článku si vysvětlíme pozadí těchto problémů a popíšeme makro PDFuni pro plainTEX, které problém s českými a slovenskými texty v záložkách a dalších podobných místech řeší.

Klíčová slova

PDF, outlines, kódování, TEX, balíček maker, PDFuni

1. Kódovací standardy pro PDF stringy

PDF stringy jsou jednak texty v záložkách a jednak texty v informaci o PDF dokumentu vložené příkazem `\pdfinfo` o autorovi, názvu díla, klíčových slovech a podobně. Údaje z `\pdfinfo` lze z dokumentu získat programem `pdfinfo` na příkazovém řádku a některé PDF prohlížeče je také dokáží zobrazit.

PDF stringy nejsou součástí hlavního textu dokumentu, takže pro ně není použit font vybraný autorem dokumentu. Místo toho si PDF prohlížeč k zobrazení PDF stringů zavolá font, který je zrovna v době prohlížení po ruce a který mu nabídne operační systém.

Specifikace PDF formátu [1] vymezuje (v dodatku D) pro PDF stringy dvě možná kódování. První je jednobytové (tj. jeden znak je reprezentován jedním bytem) a jmenuje se `PDFDocEncoding`. Toto kódování se shoduje s viditelnými znaky z ISO-8859-1, takže Západoevropané s ním nemají problémy. Nevýhodou tohoto kódování je, že v něm nenajdeme většinu akcentovaných znaků české a slovenské abecedy. Nestaráme-li se o nic jiného, je uložený string interpretován PDF prohlížečem jako kódovaný podle `PDFDocEncoding`, což nám zákonitě poničí české texty.

Kvůli tomu jsem se rozhodl do makra OPmac [2] zařadit konverzní proces, který veškeré české a slovenské akcentované znaky převede do jejich ASCII podobných znaků bez akcentu a teprve takto převedený text uloží jako PDF string. V záložkách pak sice nevidíme akcenty, ale nevidíme tam ani nesmysly, ke kterým jsou záložky náchylné. I kdyby byly PDF stringy správně kódované pro češtinu (viz dále), může se stát, že se v prohlížeči nezobrazí správně, protože správné zobrazení závisí navíc na správném nastavení locales a na dalších systémových

doi: 10.5300/2013-1/47

parametrech. Rozhodl jsem se tedy pro robustní řešení: PDF stringy generované makrem OPmac budou jenom v ASCII. A nehodlám to měnit.

Nicméně uživatelé například balíčku `hyperref` si jistě všimli, že za jistých příznivých okolností (tj. `hyperref` pracuje v UNICODE a PDF prohlížeč má správně nastaveny veškeré systémové parametry) dokáží vygenerovat korektní české texty i v záložkách. Je to tím, že PDF specifikace nabízí druhou (a poslední) možnost kódování PDF stringů, kterou nazývá `UTF-16BE Unicode` (dále jen UNICODE). Kódování PDF stringu v tomto režimu se pozná tak, že první dva byty stringu mají hodnotu 254, 255. Je to prefix pro přepnutí do tohoto kódování. Podrobněji viz kapitolu 3.8.1 (String Types) v manuálu [1].

Nechceme-li ukládat do PDF stringu jednotlivé byty nativně, máme možnost je popsat pomocí backslashe následovaném třemi oktalovými číslicemi vyjadřujícími hodnotu bytu. String „Cvičení je zátěž“ tedy může v PDF stringu být zapsaný takto:

```
\376\377\000C\000v\000i\001\015\000e\000n\000\355\000
\040\000j\000e\000\040\000z\000\341\000t\001\033\001\176
```

V uvedeném příkladu vidíme nejprve byty 254, 255 zapsané oktalově, dále je oktalově zapsaná nula následovaná písmenem C (tyto dva byty reprezentují písmeno C v UTF-16 kódování). Podobně jistě dokážete dešifrovat písmena „v“ a „i“. Dále `\001\015` je 01,0D hexadecimálně, což reprezentuje znak „č“ v UNICODE. Dál si ten string můžete dočíst sami...

Právě takové stringy musíme generovat TEXem, pokud chceme vidět v záložkách správně napsané české a slovenské znaky. Je asi nemyslitelné, abychom psali ručně do dokumentu něco jako:

```
\pdfoutline goto name{cviceni} count0 {% záložka
\string\376\string\377\string\000C\string\000v\string\000i%
\string\001\string\015\string\000e\string\000n\string\000%
\string\355\string\000\string\040\string\000j\string\000e%
\string\000\string\040\string\000z\string\000\string\341%
\string\000t\string\001\string\033\string\001\string\176}%
\pdfdest name{cviceni} xyz\relax % cíl odkazu
```

Sice je tímto způsobem na primitivní úrovni vytvořena záložka s textem „Cvičení je zátěž“, ale kdybychom to takto dělali pořád, byla by to opravdu zátěž. Proto jsem vytvořil makro PDFuni, které uživatelům $\mathcal{C}\mathcal{S}$plainu ulehčí práci. Uživatelé LaTeXu nic takového nepotřebují, protože mají balíček `hyperref`.

2. Použití balíčku PDFuni

Protože OPmac generuje do záložek texty bez akcentů, je třeba zavolat PDFuni až po `\input opmac`. Balíček PDFuni předefinuje implicitní konvertor z OPmac pro výstup do PDF stringů (konvertor do ASCII) na nový konvertor do UNICODE. Takže stačí na začátku dokumentu psát:

```
\input opmac
\input pdfuni
```

Od této chvíle příkaz `\outlines`⟨*num*⟩ generuje do záložek správně české a slovenské texty. Využívá přitom automaticky generovaný obsah. OPmac nabízí ještě příkaz `\insertoutline{`⟨*text*⟩`}`, který vloží do záložek jeden údaj. Tento příkaz vkládá ⟨*text*⟩ bez konverze, z čehož plyne, že tam české akcenty nebudou jednoduše fungovat. Proto je potřeba použít příkaz `\pdfunidef\makro{`⟨*text*⟩`}`, který je definován v balíčku PDFuni a který vloží do `\makra` zkonvertovaný ⟨*text*⟩ v kódování UNICODE s oktalovými přepisy. Kategorie backslashe tam je 12 (obyčejný znak). Takže je potřeba psát:

```
\pdfunidef\tmp{Tady je český text}
\insertoutline{\tmp}
```

PDF dokument může obsahovat PDF stringy různě kódované. Takže funguje například i toto:

```
    % tento PDF string je v UNICODE:
\pdfunidef\tmp{Čeština je dřina}\insertoutline{\tmp}
    % tento PDF string je v PDFDocEncoding:
\insertoutline{Jan Hus vnesl do jazyka velkou hloupost:
               zavedl akcenty.}
```

Příkaz pdfTEXu `\pdfinfo` zanáší do PDF dokumentu doplňující informace, například takto:

```
\pdfinfo {/Author (Autor)
          /Title (Titul)
          /Creator (csplain + OPmac)
          /Subject (clanek)
          /Keywords (klicova slova)}
```

Je nutné vědět, že PDF stringem je zde každý údaj uzavřený do závorky, přičemž některé tyto PDF stringy mohou být kódovány podle PDFDocEncoding a jiné podle UNICODE. Údaje je tedy možno zanést třeba takto:

```
\def\author{Petr Olšák}\pdfunidef\author{\author}
\def\title{PDFuni - akcenty v PDF záložkách}\pdfunidef\title{\title}
\def\subject{článek}\pdfunidef\subject{\subject}
\pdfinfo {/Author (\author)
          /Title (\title)
          /Creator (csplain + OPmac)
          /CreationDate (D:20130107000000)
          /Subject (\subject)
          /Keywords (TeX; pdfTeX; PDF; OPmac)}
```

Poznamenejme, že další informace `/Producer` a `/ModDate` vloží většinou nejlépe pdfTEX sám, nicméně uživatel je rovněž může předefinovat. Není-li ani jeden údaj `/CreationDate` a `/ModDate` vyplněn, oba obsahují čas vzniku PDF souboru. Můžete si všimnout, že chcete-li `/CreationDate` vyplnit podle svého, je nutné dodržet formát `D:YYYYMMDDhhmmss`.

3. Možnost rozšíření kódovací tabulky o další znaky

Příkaz `\pdfunidef` konvertuje správně všechny viditelné ASCII znaky s kategoriemi 11, 12 (písmena, znaky) a dále znaky s kategoriemi 7, 8 (^, _). Ostatní viditelné ASCII znaky jiných kategorií (např. `$`, `&`) jsou ignorovány. Chcete-li do PDF stringu dopravit znak jiné kategorie než 7, 8, 11, 12, musí mít před sebou backslash, tedy `\\`, `\{`, `\}`, `\$`, `\&`, `\#`, `\~`, `\%`.

Dále `\pdfunidef` konvertuje znaky kategorie 11 nebo 12, které nejsou ASCII, pokud jsou zahrnuty do seznamu `\pdfunichars`. Tento seznam implicitně obsahuje znaky Áá Ää Čč Ďď Éé Ěě Íí ĹĺĽľ Ňň Óó Öö Ôô Řř Řř Šš Ťť Úú Ůů Üü Ýý Žž. Oktalový zápis těchto znaků (tedy výstup konverze) ve stejném pořadí je uložen v seznamu `\pdfunicodes`. Kódy jsou tam zapisovány bez backslashů a bez první nuly a jsou od sebe odděleny příkazem `\or`. Podívejte se do souboru `pdfuni.tex` nebo do následující sekce, jak vypadají implicitní hodnoty těchto seznamů. Chcete-li rozšířit tuto konverzní tabulku, je možné použít příkaz `\addto` z OPmac například takto:

```
\addto\pdfunichars{Ëë}  \addto\pdfinicodes{\or00313\or00353}
```

Uvedený příklad rozšiřuje seznam `\pdfunichars` o znaky Ë, ë a dále rozšiřuje seznam `\pdfunicodes` o kódy `\000\313` a `\000\353`. Na tyto kódy se budou znaky Ë, ë (v tomto pořadí) konvertovat.[1]

Příkaz `\pdfunidef` ještě před začátkem konverze provede úplnou expanzi svého parametru ⟨*text*⟩. Před touto expanzí je vhodné předefinovat některá makra, například `\def\TeX{TeX}`. Taková činnost je zanesena do seznamu

[1] Aby uvedený příklad fungoval, musejí mít znaky Ëë svou reprezentaci ve vnitřním kódování TEXu. To je u 16bitového TEXu zařízeno automaticky. Pro $\mathcal{CS}$plainu a encTEXu je možné použít třeba `\input t1code`.

`\pdfunipre`, který je (uvnitř skupiny) předřazen před expanzí parametru ⟨*text*⟩. Kromě speciálních expanzí maker tam jsou povelem `\let\makro\relax` zabezpečena makra, která expandovat v dané chvíli nechceme. Jsou to makra `\ae`, `\P` atd., která posléze vstoupí do konverze a promění se na svůj oktalový kód. Přidělení tohoto kódu se děje v seznamu `\pdfunipost` pomocí příkazu `\odef\makro`⟨*6ciferný kód*⟩⟨*mezera*⟩. Například příkazem `\odef\ae000346` říkáme, že makro `\ae` se zkonvertuje na `\000\346`. Čtenáři doporučuji podívat se na obsahy seznamů `\pdfunipre` a `\pdfunipost` do souboru `pdfuni.tex` nebo do následující sekce.

Chcete-li rozšířit konverzní tabulku týkající se maker, lze to udělat pomocí `\addto` například takto:

```
\addto\pdfunipre{\let\endash\relax}
\addto\pdfunipost{\odef\endash040023 }
```

Tento příklad přidává do tabulky údaj o konverzi makra `\endash`. Ve fázi expandování parametru je jeho expanze potlačena a dále se zkonvertuje na kód `\040\023`, což je dle UNICODE pomlka na půlčtverčík.

Kontrolní sekvence, které nejsou během expanze parametru ⟨*text*⟩ expandovány a nemají deklarován svůj kód pomocí `\odef`, jsou při konverzi ignorovány.

Nyní už čtenář patrně ví, proč při použití $\mathcal{CS}$plainu s encTEXem funguje správně i právnický znak §, ačkoli není uveden v seznamu `\pdfunichars`. EncTEX jej totiž mapuje na kontrolní sekvenci `\S`, takže kdykoli napíšeme §, promění se to automaticky v `\S`. Znak `\S` je ošetřen v seznamu `\pdfunipre`, kde je napsáno `\let\S\relax`. A v seznamu `\pdfunipost` je definice `\odef\S000247`.

4. Popis interních maker a s nimi spojených triků

Makra z `pdfuni.tex` obsahují několik zajímavých triků. Jejich popis je určen pro pokročilé TEXisty.

Soubor `pdfuni.tex` je kódován v UTF-8 kódování. Je nutné, aby byl přečten encTEXem nebo 16bitovým TEXem, protože chceme s jednotlivými akcentovanými znaky v TEXu pracovat jako s jedním tokenem. Proto je na začátku souboru test, zda je použit správný TEXový překladač:

```
\def\tmp#1#2\end{\if$#2$\else
\errmessage {This file is UTF-8 encoded. Use TeX+encTeX
             or 16bit TeX engine.}\fi}%
\tmp č\end
```

Dále na začátku souboru definujeme makro `\Bslash`, které expanduje na backslash kategorie 12. Totéž dělá OPmac s makrem `\bslash`, ale OPmac hodnotu tohoto makra v různých místech mění. Zde potřebujeme mít jistotu, že to bude pořád backslash. Kromě toho potřebujeme pracovat s pomocným čítačem

`\tmpnum`. Protože nevíme, zda je zavolán OPmac (makro PDFuni funguje i bez něj), je za předpokladu nedefinovanosti `\newcount` tento čítač deklarován.

```
\ifx\tmpnum\undefined \csname newcount\endcsname\tmpnum \fi
{\lccode`\?=`\\  \lowercase{\gdef\Bslash{?}}}
```

Následuje výpis implicitní hodnoty seznamů `\pdfunichars` a `\pdfunicodes`:

```
\edef\pdfunichars{\Bslash()
   ÁáÄäČčĎď ÉéĚěÍíĹĺĽľ ŇňÓóÖö ŐőŘřŔŕŠš ŤťÚúŮůÜü ÝýŽž
}
\def\pdfunicodes {\or00134\or00050\or00051\or             % \ ()
   00301\or00341\or00304\or00344\or01014\or01015\or       % ÁáÄäČč
   01016\or01017\or00311\or00351\or01032\or01033\or       % ĎďĚéĚě
   00315\or00355\or01071\or01072\or01075\or01076\or       % ÍíĹĺĽľ
   01107\or01110\or00323\or00363\or00326\or00366\or       % ŇňÓóÖö
   00324\or00364\or01124\or01125\or01130\or01131\or       % ŐőŘřŔŕ
   01140\or01141\or01144\or01145\or00332\or00372\or       % ŠšŤťÚú
   01156\or01157\or00334\or00374\or00335\or00375\or       % ŮůÜüÝý
   01175\or01176%                                         % Žž
}
```

Je vidět, že do seznamu jsou zařazeny i hodnoty `\,` `(`, `)`. Je to tím, že tyto znaky nechceme konvertovat do podoby `\000`⟨*znak*⟩, protože by v PDF stringu způsobovaly problémy (backslash je escape prefix a kulaté závorky ohraničují string). Je tedy bezpečnější je do PDF stringu zapsat v úplné oktalové notaci.

Pokračuje výpis implicitní hodnoty `\pdfunipre` a `\pdfunipost` a některých speciálních kódů.

```
\def\pdfunipre {\def\TeX{TeX}\def\LaTeX{LaTeX}\def~{ }\def\ { }%
   \let\ss\relax \let\l\relax \let\L\relax
   \let\ae\relax \let\oe\relax \let\AE\relax \let\OE\relax
   \let\o\relax \let\O\relax \let\i\relax \let\j\relax \let\aa\relax
   \let\AA\relax \let\S\relax \let\P\relax \let\copyright\relax
   \let\dots\relax \let\dag\relax \let\ddag\relax
   \let\clqq\relax \let\crqq\relax \let\flqq\relax \let\frqq\relax
   \let\promile\relax \let\euro\relax
   \def\\{\Bslash}\let\{\relax \let\}\relax
   \def\${\string$}\def\&{\string&}\def\_{\string_}\def\~{\string~}%
}
\def\pdfunipost {\odef\ss000337 \odef\l001102 \odef\L001101
   \odef\AE000306 \odef\ae000346 \odef\OE001122 \odef\oe001123
   \odef\o000370 \odef\O000330 \odef\aa000345 \odef\AA000305
   \odef\i001061 \odef\j002067  \odef\S000247 \odef\P000266
   \odef\copyright000251 \odef\dots040046 \odef\dag040040
   \odef\ddag040041 \odef\clqq040033 \odef\crqq040034
   \odef\flqq000253 \odef\frqq000273 \odef\promile040060
```

```
   \odef\euro040254 \odef\{000173 \odef\}000175 \odef\#000043
}
\def\pdfinitcode{\Bslash376\Bslash377}
\def\pdfspacecode{\Bslash000\Bslash040}
\def\pdfcircumflexcode{\Bslash000\Bslash136}
\def\pdflowlinecode{\Bslash000\Bslash137}
```

Nyní přistupme k tomu hlavnímu, k definici makra `\pdfunidef`⟨*makro*⟩`{`⟨*text*⟩`}`.

```
\def\pdfunidef#1#2{\bgroup \def\tmpa{\noexpand#1}%
   \pdfunipre \edef\tmp{#2}\edef\out{\pdfinitcode}%
   \def\odef##1##2##3##4##5##6##7
      {\def##1{&\edef\out{\out\Bslash##2##3##4\Bslash##5##6##7}}}%
            \pdfunipost
   \def\insertbslashes ##1##2##3##4##5{%
            \Bslash0##1##2\Bslash##3##4##5}%
   \def\gobbletwowords ##1 ##2 {}%
   \tmpnum=0
   \loop
      \sfcode\tmpnum=0 \advance\tmpnum by1
      \ifnum \tmpnum<128 \repeat
   \tmpnum=0
   \expandafter \pdfunidefA \pdfunichars {}%
}
```

Makro `\pdfunidef` nejprve zahájí skupinu a zapamatuje si jméno ⟨*makra*⟩, které má definovat, do `\tmpa`. Dále spustí `\pdfunipre` a expanduje parametr ⟨*text*⟩. Do pracovního makra `\out` bude postupně střádat výsledek konverze. Výchozí hodnota je `\pdfinitcode`, což je `\376\377`, neboli prefix UNICODE stringu. Dále je definováno pomocné makro `\odef`⟨*makro*⟩⟨*6 oktalových cifer*⟩⟨*mezera*⟩ poněkud trikoidním způsobem. Makro provede `\def`⟨*makro*⟩`{&`⟨*činnost*⟩`}`. Proč je tam znak `&`, vysvětlíme později. Makro přidá odpovídající oktalové cifry do výstupního makra `\out`, přitom k těmto cifrám přidá backslashe. Seznam `\pdfunipost` obsahuje jednotlivé příkazy `\odef`, tedy připraví makra ke konverzi.

Dále je připraveno pomocné makro `\insertbslashes`⟨*5 oktalových cifer*⟩, které přidá šestou nulu na začátek a vloží na správné místo backslashe. Makro `\gobbletwowords` odstraní dvě následující slova ukončená mezerou. Budeme to potřebovat za chvíli.

V závěru této části cyklus uloží každému znaku s kódem menším než 128 sfkód rovný nule. Podle této hodnoty sfkódu poznáme, že znak je z dolní ASCII tabulky a bude kódován na `\000`⟨*znak*⟩. Ostatní znaky vyjmenované v seznamu `\pdfunichars` budou mít každý svůj sfkód postupně se zvětšující od jedničky. To zajistí makro `\pdfunidefA`, které postupně ze seznamu `\pdfunichars` odlupuje jednotlivé znaky a dává jim odpovídající sfkódy.

```
\def\pdfunidefA #1{\if\relax#1\relax%
      \def\next{\let\next= }\afterassignment\pdfunidefB
         \expandafter\next\tmp\end
   \else \advance\tmpnum by1 \sfcode‘#1=\tmpnum \expandafter
        \pdfunidefA
   \fi
}
```

Postupné odlupování končí, když je načten prázdný parametr (poznáme to podle `\if\relax#1\relax`, to se totiž sejde první `\relax` s druhým). V takovém případě zahájíme postupné odlupování expandovaného ⟨*text*⟩, který máme v `\tmp`. Konec druhého řádku ukázky po provedení `\expandafter` a následné expanzi `\next` vypadá takto:

`\let\next=` ⟨*expandovaný text*⟩`\end`

Mezera za rovnítkem je důležitá, protože další případná mezera se skutečně přiřadí. Příkaz `\let\next` odloupne z ⟨*expandovaného textu*⟩ první token a po přiřazení spustí `\pdfunidefB`.

```
\def\pdfunidefB {%
   \ifcat x\noexpand\next \pdfunidefC
   \else \ifcat.\noexpand\next \pdfunidefC
   \else \ifcat\noexpand\next\space \edef\out{\out\pdfspacecode}%
   \else \ifcat^\noexpand\next \edef\out{\out\pdfcircumflexcode}%
   \else \ifcat_\noexpand\next \edef\out{\out\pdflowlinecode}%
   \else \expandafter\ifx\expandafter&\next
   \fi\fi\fi\fi\fi\fi
   \ifx\next\end \edef\next{\def\tmpa{\out}}\expandafter\egroup\next
      \else
      \def\next{\let\next= }\afterassignment\pdfunidefB
          \expandafter\next
   \fi
}
```

Makro `\pdfunidefB` prošetří obsah `\next` a podle jeho typu provede jeden krok konverze. V závěru své činnosti zavolá rekurzivně samo sebe a odloupne z ⟨*expandovaného textu*⟩ další token. To dělá tak dlouho, dokud nenarazí na `\end`. Až se tak stane, vloží do `\next`:

`\def`⟨*makro*⟩`{`⟨*zkonvertovaný výstup*⟩`}`

a nejprve expanduje `\next` pomocí `\expandafter` (tehdy si ještě pamatuje obsah `\next`) a vzápětí poté ukončí skupinu, vše tím zapomene a provede výsledek připravené expanze. Dílo je dokonáno.

Vraťme se k makru `\pdfunidefB`. Toto makro vyhodnotí především kategorii odloupnutého tokenu. Je-li to mezera, znak `^` nebo `_`, přidá do `\out` jejich odpovídající kód. Je-li to písmeno nebo jakýkoliv jiný znak, provede `\pdfunidefC`. Podívejme se tedy, co dělá `\pdfunidefC`:

```
\def\pdfunidefC {\edef\tmp{\expandafter\gobbletwowords\meaning\next}%
   \edef\out{\out\pdfunidefD}%
}
```

Makro `\pdfunidefC` potřebuje zpětně zjistit, jaký znak je v `\next` uložen. K tomu použije příkaz `\meaning\next`, který vypíše třeba `the character B`. Pomocí `\gobbletwowords` odstraníme dvě slova `the character` a v makru `\tmp` zbude jen `B`. Do `\out` pak přidáme výsledek expanze `\pdfunidefD`.

```
\def\pdfunidefD{%
   \ifnum\sfcode\expandafter`\tmp=0 \Bslash 000\tmp
   \else
      \expandafter \insertbslashes \ifcase
             \sfcode\expandafter`\tmp\pdfunicodes
         \else 000077\fi
   \fi
}
```

Toto makro musí pracovat jen s expandovatelnými primitivy. Nejprve zkontroluje, zda konvertovaný znak (skrytý v `\tmp`) má sfkód roven nule. V takovém případě expanduje na `\000`⟨*znak*⟩. Jinak pomocí `\expandafter` rozvine `\ifcase\sfcode`⟨*znak*⟩. Protože tato podmínka není terminovaná, expanduje se i tělo tohoto `\ifcase` skryté v makru `\pdfunicodes`. Víme, že tam je psáno

`\or`⟨*5ciferný kód*⟩`\or`⟨*5ciferný kód*⟩`\or`⟨*5ciferný kód*⟩...

Celý `\ifcase` je uzavřen pomocí `\else 000077\fi`, což je kód otazníku. Expandafter před `\ifcase` způsobí, že toto `\ifcase` expanduje na svou odpovídající větev podle sfkódu, kde je ⟨*5ciferný kód*⟩. Před tímto výsledkem se ovšem rozprostírá makro `\insertbslashes`, které do tohoto ⟨*5ciferného kódu*⟩ doplní první nulu a backslashe.

Zajímavý trik je na řádku těsně nad řadou `\fi\fi\fi\fi\fi\fi` v těle makra `\pdfunidefB`. Je tam řečeno `\expandafter\ifx\expandafter&\next`. Představme si, že `\next` je makro definované pomocí `\odef`. V takovém případě po vykonání těch dvou `\expandafter` dostáváme:

`\ifx&&`⟨*činnost*⟩

a provede se požadovaná činnost. Je-li `\next` cokoli jiného, není to shodné se znakem `&` a neprovede se nic. Je-li `\next` přímo znak `&`, neprovede se také nic.

Soubor `pdfuni.tex` je ukončen definicí makra z OPmac, která mění konvertor spuštěný při činnosti `\outlines`⟨*num*⟩ z původní hodnoty (konverze do ASCII) na novou hodnotu (konverze pomocí `\pdfunidef`).

```
\def\cnvhook #1#2{#2\pdfunidef\tmp\tmp} % the default convertor
                                        % in OPmac is redefined here
```

5. Literatura

[1] `http://www.adobe.com/devnet/acrobat/pdfs/pdf_reference_1-7.pdf`

[2] `http://petr.olsak.net/opmac.html`

Summary: PDFuni – accents in PDF bookmarks

Bookmarks in the PDF document (pdfoutlines) are texts which are displayed by PDF viewer under certain circumstances. They are clickable and connected to their destination in the document. They are organized in a tree structure. The bookmarks are never printed. Many TEX users may have noticed that Czech and Slovak texts are sometimes destroyed in the bookmarks: the accented letters are displayed badly. This article explains the technical background of this problem and describes the PDFuni macro package for plain TEX which solves this problem for Czech and Slovak texts in bookmarks and in the others PDF strings.

Key words

PDF, outlines, encoding, TEX, macro package, PDFuni

LaTeX, nebo ConTeXt? První zkušenosti se sazbou ConTeXtem

Tomáš Hála

Po mnoha letech strávených ve světě LaTeXu se autor rozhodl opustit vyjeté koleje a vyzkoušet jinou nadstavbu – ConTeXt. V tomto článku budou představeny první zkušenosti získané při přípravě knižní publikace týkající se podstatných rysů zpracování, jako jsou nastavení sazebního obrazce, sazba na řádkový rejstřík, strukturní značkování apod. Vybrané rysy obou nadstaveb jsou vzájemně porovnány a zhodnoceny.

Klíčová slova

TeX, ConTeXt, LaTeX, srovnání nadstaveb

Úvod

V roce 1992 jsem měl možnost se seznámit s TeXem, přesněji řečeno s jeho nadstavbou LaTeX. Pro počítačovou přípravu dokumentů se v té době jednalo o převratnou novinku. „Nepohodlné“ rozdělení procesu přípravy a zpracování na tři kroky, editaci, překlad, prohlížení (tzv. non-WYSIWYG), bylo bohatě vyváženo matematicky přesnou sazbou na základě Knuthova TeXu, díky čemuž bylo možné dosáhnout lepších výsledků než s tehdejšími systémy pracujícími v režimu WYSIWYG. Ukázalo se, že TeX, resp. LaTeX, lze využít pro všechny typy dokumentů, které jsem potřeboval připravovat.

Čas od času jsem se však setkával s poněkud nesrozumitelnými chybovými hlášeními, jejichž původ se dal jen obtížně vystopovat studiem maker LaTeXu. Jak jsem později zjistil, bylo to proto, že začátečník není schopen rozpoznat rozdíl mezi „TeXovými“ a „LaTeXovými“ problémy.

Olšák (1997a) vyjádřil podobné pocity ve své eseji o důvodech, proč nemá rád LaTeX. Nejprve formuloval čtyři základní potřeby, které podle něj vedly ke vzniku LaTeXu:

– odstínit poměrně složitou problematiku TeXu od koncového uživatele;
– vytvořit vlastní jazyk vstupních textů;
– umožnit formátovat jednoduché dokumenty podle předzpracovaných stylů;
– umožnit snadnou výměnu dokumentů a jejich nové přeformátování.

doi: 10.5300/2013-1/57

Uvedené čtyři body lze přijmout, ovšem jak Olšák (1997a) uvádí dále, nenaplní očekávání uživatele, a proto analyzoval detailně jedenáct aspektů chování LaTeXu (nedostatečnost jazyka TeXu; snaha LaTeXu o odstínění složitosti TeXu; utajení skutečností; nerozlišování mezi LaTeXem a TeXem; vlastní jazyk vstupních textů; strukturované značkování; ideální dělba práce; předzpracované styly; složitost maker LaTeXu; přenositelnost dokumentů; nestálost verzí LaTeXu).

Některé z nich poslouží v tomto textu pro srovnání s chováním ConTeXtu. V tomto článku je popisována verze z distribuce TeXCollection (2010), kterou jsem použil pro sazbu publikace spřáteleného pracoviště.

Odstínění relativní složitosti TeXu

LaTeX jako nadstavba TeXu má za cíl odstínit běžného uživatele od relativní složitosti základního TeXu. Rozsáhlý soubor specifických maker LaTeXu se snaží překrýt původní struktury pro zjednodušení uživatelské práce, nezdařilo se to však úplně.

Olšák (1997a) demonstruje tuto situaci na příkladu začátečníka, který v řádku použil o sloupec více, než si určil v záhlaví tabulky, a který se po obdržení chybového hlášení ocitne poněkud na rozpacích:

```
! Extra alignment tab has been changed to \cr
```

Není si vědom, že by makropříkaz `\cr` definoval či použil, kromě toho jej ani nenalezne v běžných publikacích o LaTeXu, zmiňují jej pouze Mittelbach a Goossens (2004, s. 898) v seznamu chybových hlášení.

Podobně i ConTeXt umí učinit uživatele nešťastným. V jiné situaci uživatel obdrží podobně nesrozumitelné hlášení týkající se příkazu `\omit`:

```
! Misplaced \omit
\!ttuse #1->\ifnum #1>\plusone \omit
                            \global \TABLEdivisionfalse ...
l.274 ... value \VL ... values\use{3}
                                                  \MR\HL
```

Tento příkaz `\omit` (stejně jako `\cr` v předchozí ukázce) nepatří mezi makropříkazy ConTeXtu. V tomto bodě tedy neshledáváme rozdíl mezi oběma nadstavbami.

ConTEXt jako typografický systém

Řízení vzhledu dokumentu

ConTEXt obsahuje velmi účinný a současně pohodlný nástroj pro přípravu návrhu sazby – makropříkazy `\setuplayout` a `\setuppapersize` zahrnující všechny sazební parametry.

Kromě toho existuje makropříkaz `\showlayout`, který zobrazí rozvržení jednotlivých částí strany, což lze považovat snad za nejužitečnější pomocný nástroj sazeče.

Naproti tomu v LaTeXu se nastavuje všechno samostatně, jednak příkazy LaTeXu (např. `\textheight`, `\textwidth`), jednak pomocí příkazů pracujících přímo s vlastnostmi formátu PDF (např. `\paperwidth`, `\paperheight`), což je poněkud komplikované a nepřehledné pro začínající uživatele.

Sazba na řádkový rejstřík

TeX, a proto ani LaTeX nebyly navrženy prvoplánovitě pro sazbu na řádkový rejstřík. Neexistuje ani žádná jednoduchá cesta jak rejstříkové uspořádání obecně v LaTeXu vynutit (Kroonenberg, 2004). To dokazuje i Kohm (2008), který svůj balíček `gridset` hodnotí jen jako krok na cestě k sazbě do řádkového rejstříku, neboť neimplementuje automatickou sazbu, ale předkládá pouze nástroje, jimiž si lze ručně pomoci v určitých situacích.

Jiný, dosti komplikovaný způsob navrhl Mittelbach (2000), jehož algoritmus se snaží řešit umístění plovoucích objektů ve vícesloupcových dokumentech, což je v základním LaTeXu velice obtížné.

V kratších dokumentech, jako jsou technické zprávy, články apod., se velmi často užívá meziodstavcových mezer, obvykle o velikosti poloviny vzdálenosti dvou účaří. V LaTeXu je navíc tato vertikální mezera implicitně nastavena jako pružná, vyrovnávající meziodstavcové mezery v rámci jedné strany. To činí sazbu na řádkový rejstřík téměř nemožnou, a proto je potřeba vždy toto chování předefinovat.

Řešení pro plainTeX připravil Olšák (1996). Je založeno na postupném, „ručním odlamování“ řádků z předpřipraveného vertikálního boxu. I přesto že tento způsob může být použit i v LaTeXu (což autor tohoto článku před lety vyzkoušel), příprava sazebních definic u rozsáhlejších, složitějších textů se ukáže jako dosti pracná. Jedná se tedy o metodu vhodnou pro kratší texty, například pro sazbu do sloupců, s předem nařízenými, pevně umístěnými obrázky. Skutečnost, že pro řešení sazby na řádkový rejstřík je nutno užít kompletně řešení v plainTeXu, připomíná Olšákovu (1997a) úvahu, zda je tedy LaTeX vůbec nutný pro sazbu.

V kontrastu k popsaným problémům stojí nativní, plně funkční podpora sazby na řádkový rejstřík v ConTEXtu, navíc uživatelsky překvapivě jednoduše přístupná:

```
\setuplayout[grid=yes]
```

Tento přístup se blíží systémům typu WYSIWYG, kde se sazba na řádkový rejstřík aktivuje jedním kliknutím někde v nabídce. Je vhodné na tomto místě poznamenat, že tato jednoduché aktivace spojená s plnou funkčností řešení byla hlavním důvodem, proč se autor tohoto článku rozhodl začít experimentovat s ConTeXtem.

ConTeXt jako překladač

Časová náročnost překladu

Hoekwater (1998) měřil čas potřebný pro překlad téhož dokumentu pomocí plainTeXu, LaTeXu a ConTeXtu. Zjistil, že ConTeXt spotřebuje nejvíce času ze všech tří možností. Je to způsobeno jednak složitějším konceptem příkazů a zpracováním jejich parametrů, jednak složitější výstupní rutinou.

Na linuxových systémech (CentOS 5.3, 5.6) s instalací TeXlive (2010) zůstává poměr spotřebovaného času mezi LaTeXem a ConTeXtem (měřeno v roce 2011) obdobný. Na rozdíl od ostatních implementací TeXu, kde uživatel sám rozhoduje o počtu překladů v souvislosti s různými odkazy, tvorbou obsahu apod., ConTeXt si potřebný počet překladů určí sám. Tyto automaticky volané opakované překlady mají rovněž vliv na spotřebovaný čas. Při běžné sazečské praxi však není vždy potřeba pracovat s dokumentem, který prošel všemi nezbytnými překlady. Značnou část času ušetříme použitím volby `-once`, potlačující opakované překlady.

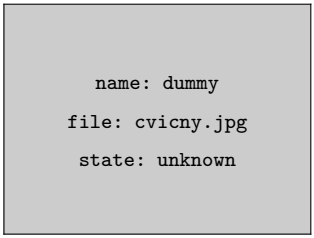
Varovná hlášení

Má-li tabulka ve skutečnosti větší počet sloupců, než bylo deklarováno, například při opomenutém konci řádku, překladač vytvoří pouze varovnou zprávou (`[missing row]`), kterou zapíše přímo do tabulky mezi první a druhý nadbytečný sloupec:

	CZK	česká koruna
	USD	americký dolar NOR *sloupce* `[missing row]`
norská koruna *navíc*		
	GBP	britská libra

Poněkud nebezpečné je také opomenutí externích souborů, například obrázků, do dokumentu. Použijeme-li `\includegraphics` v LaTeXu, překlad se přeruší,

neboť se jedná o závažnou chybu, a záleží na rozhodnutí uživatele, jak bude pokračovat. Naproti tomu ConTeXt vypisuje podobně jako u tabulek pouze varovné hlášení, které vloží namísto požadovaného obrázku:



Toto poněkud nezvyklé řešení vyžaduje výrazně pečlivější kontrolu hotového dokumentu před jeho odevzdáním či odesláním do tisku.

Nevýhoda tohoto přístupu je zřejmá. Této vlastnosti však lze i využít, například tehdy, když nemáme k dispozici příslušné externí soubory. Překlad není zbytečně přerušován množstvím chybových hlášení. Proto tuto vlastnost nelze hodnotit jednoznačně a je otázkou, zda by uživatelskému komfortu neprospěla nějaká volba ovlivňující chování překladu.

Protokol o překladu
Ve srovnání s LaTeXem se protokol o překladu (log file) vytvořený ConTeXtem zdá lépe uspořádán a obsahuje podrobnější informace o všech vstupních i výstupních událostech.

Na druhé straně standardní výstup ConTeXtu je téměř totožný s obsahem zapisovaným do souboru. Ačkoliv je podrobný výpis jinak velmi užitečný, pro optimální práci v terminálovém okně by se určitě uplatnil nějaký nástroj, kterým by bylo možné vybrat jen důležité řádky výpisu.

ConTeXt jako nadstavba

Základní koncept strukturních příkazů používá dvojice `\start`*něco* a `\stop`*něco*, které ohraničují příslušnou část dokumentu. Ačkoliv se to zdá méně důležité, psaní příkazů bez přehnaného množství složených závorek, jako je tomu v LaTeXu, se jeví pohodlnější.

Kromě dvojice `\starttext` a `\stoptext` ostatní příkazy typu start/stop podle očekávání otevírají a uzavírají skupiny.[1]

[1] Tato nedůslednost byla reportována a následně v dalších verzích ConTeXtu odstraněna.

Objekty sazby je možné velmi snadno konfigurovat. Slouží k tomu jednak příkazy `\setup`*objekty* pro všechny objekty a `\setup`*objekt* pro jednu jeho konkrétní instanci, jednak rozsáhlý repertoár parametrů v režimu *klíč-hodnota* pro všechny objekty.

S nastavováním pomocí klíče a hodnoty se setkáváme často i v LaTeXu, obvykle je využit balíček `keyval.sty` (Carlisle, 1993), ale tento způsob není implementován systematicky. Proto uživatelé LaTeXu musejí studovat příslušný způsob v závislosti na použitém balíčku či příkazech.

Velkou výhodou je také definiční příkaz pro vlastní plovoucí objekty. Otten a Hagen (2006) jej ukazují na příkladu vsuvky: `\definefloat[vsuvka][vsuvky]` vytvoří další příkazy – `\placevsuvka[][]{}{}` pro použití plovoucí vsuvky v dokumentu, dále `\startvsuvkatext` a `\stopvsuvkatext`, mezi něž se vkládá text, a nakonec dvojici `\placelistofvsuvky` a `\completelsitofvsuvky`, oba pro umístění seznamu vsuvek. Ve srovnání se základním vybavením LaTeXu se jedná o velmi jednoduchý a pohodlný způsob vytvoření všech potřebných nástrojů pro práce s vlastními plovoucími objekty.

Z tohoto pohledu se ConTeXt chová jako skutečná nadstavba, přináší nové vlastnosti, jednotný konfigurační postup pro všechny objekty dokumentu a kromě toho neusiluje o překrytí základů plainTeXu.

Důkazem může být i využití plainTeXu například pro vytvoření nového způsobu zarovnání odstavce (do bloku, s posledním řádkem na středu):

```
  \installalign{blocklastcenter}{%
    \leftskip=0pt plus1fil \rightskip=0pt plus-1fil
    \parfillskip=0pt plus2fil\parindent0pt
  }
% Inspirace: Olšák (1997b)
```

Koncept *setup/start/stop* vede uživatele ke strukturovanému značkování. Uživatel sice může používat některé značky tzv. vizuálně, práce s nimi však není efektivní, pokud se nejedná o řešení zvláštních případů. To přirozeně podporuje dělbu práce mezi písaře (autora) a sazeče.

Závěr

Ačkoliv jsem se zkušenosti s ConTeXtem rozhodl získat netradičně sazbou rovnou celé knihy namísto postupného zkoušení jednoduchých příkladů, přímočarý a jednotný koncept konfigurace jednotlivých objektů mi velmi usnadnil práci. Kromě kladného subjektivního pocitu přinesl i zjištění, že finalizace sazby je časově méně náročná než v LaTeXu, a to přes množství tabulek a obrázků, které se dařilo umísťovat bez větších problémů na patřičná místa.

Je sice pravda, že například sazbu některých prvků mohou v LaTeXu zjednodušit některé volně dostupné balíčky, na druhé straně většina potřebných vlastností je v ConTeXtu již přítomna jako součást základní funkcionality, a to činí systém konzistentním.

Zhodnotíme-li jednotlivé zkoumané aspekty, nelze říci, že se ConTeXt se vším vypořádal výrazně lépe. Podstatné rysy – sazba na řádkový rejstřík i snadná úprava nastavení jednotlivých objektů – jsou však pádnými argumenty, proč tato nadstavba v některých rysech překonává LaTeX.

I přes některé nesnáze a překvapení mohu svoji první *exkurzi* do ConTeXtu uzavřít vyjádřením celkově kladného dojmu i doporučením ostatním uživatelům ConTeXt si vyzkoušet.

Literatura

Carlisle, David. *keyval.sty*. Package for LaTeX. 1993–1999.

Hoekwater, Taco. *Comparing* ConTeXt *and LaTeX*. MAPS 20/1998, s. 280–285.

Kohm, Markus. *Semi-Manual Grid Setting Using* gridset. `http://ctan.mackichan.com/macros/latex/contrib/gridset/gridset.dtx` 2008/11/12 v0.1

Kroonenberg, Siep. *Exact layout with LaTeX*. MAPS 31/2004, s. 67–70.

Mittelbach, Frank. Formatting documents with floats. A new algorithm for LaTeX2ε. *TUGBoat*, Vol. 21 (2000), No. 3 – Proceedings of the 2000 Annual Meeting.

Mittelbach, Frank; Goossens, Michael. *The LaTeXCompanion*. Addison-Wesley, 2004 xxvii+1090 s. ISBN 0-201-36299-6.

Olšák, Petr. *Makro na sazbu článku „Kam se poděla dobrá typografie“ pro časopis MENSA*. 28. 3. 1996 [cit. 2011-09-08] Dostupné na: `http://math.feld.cvut.cz/ftp/olsak/vyuka/mensa.kam`

Olšák, Petr. Proč nerad používám LaTeX. *Zpravodaj CSTUG*, č. 1–2/1997a, s. 89–99. (doi: 10.5300/1997-1-2/89)

Olšák, Petr. *TeXbook naruby*. Brno : Konvoj, 1997b. 486 s.

Otten, Ton; Hagen, Hans. Exkurze do ConTeXtu. Přeložil Vít Zýka a kol. *Zpravodaj CSTUG*, č. 2–4/2006, s. 57–224. (doi: 10.5300/2006-2-4/57)

TeXCollection [DVD]. TeXlive. 2010.

Summary: LaTeX, or ConTeXt? First experience with ConTeXt typesetting

After a number of years spent in LaTeX civilisation, the author decided to fly the track and tried ConTeXt for book publishing. This paper covers the author's experience with some necessary elements for typesetting books, e.g., layout setup, grid, structure markuping, etc. Selected features of both superstructures have been compared and discussed.

Key words

TeX, ConTeXt, LaTeX, comparison of superstructures

Poděkování

Je mou milou povinností na tomto místě poděkovat Československému sdružení uživatelů TeXu za finanční podporu mé účasti na 5th ConTeXt meeting v Bessange-Boirs v Belgii.

thala@mendelu.cz,
Mendelova univerzita v Brně, Provozně ekonomická fakulta,
ústav informatiky, Zemědělská 1, CZ 613 00 Brno,
KONVOJ, spol. s r. o., Bystřinova 4, 612 00 Brno, konvoj@konvoj.cz

Zpravodaj Československého sdružení uživatelů TEXu
ISSN 1211-6661 (tištěná verze), ISSN 1213-8185 (online verze)

Vydalo:	Československé sdružení uživatelů TEXu vlastním nákladem jako interní publikaci
Obálka:	Antonín Strejc
Počet výtisků:	400
Uzávěrka:	15. 1. 2013
Odpovědný redaktor:	Zdeněk Wagner
Redakční rada:	Zdeněk Wagner (šéfredaktor), Ján Buša, Jiří Demel, Tomáš Hála, Jaromír Kuben, Michal Růžička, Jiří Rybička, Petr Sojka, Pavel Stříž, Jan Šustek
Technický redaktor:	Tomáš Hála
Tisk a distribuce:	KONVOJ, spol. s r. o., Berkova 22, 612 00 Brno, tel. +420 541 245 548
Adresa:	CSTUG, c/o FEL ČVUT, Technická 2, 166 27 Praha 6
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

`gacstug@cstug.cz`
grantová agentura CSTUGu

`secretary@cstug.cz`, `orders@cstug.cz`
korespondence administrativní síle sdružení, objednávky CD a DVD

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

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

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

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

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

CONTENTS