Zpravodaj Československého sdružení uživatelů TeXu

Petr Olšák

Jednoduchá grafika PDF-primitivně

Zpravodaj Československého sdružení uživatelů TeXu, Vol. 23 (2013), No. 1, 13–30

Persistent URL: http://dml.cz/dmlcz/150076

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

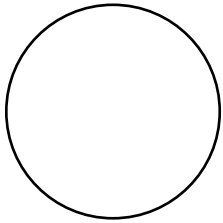
doprava. Druhý směr je $(0, -72)$, takže nová osa y' má stejnou jednotku, ale je orientovaná nikoli nahoru ale dolů. Počátek souřadného systému zůstává na stejném místě. Máme-li nyní nakreslit bod o souřadnicích $(2, 3)$, provedeme to přímočaře v novém souřadném systému, v nových jednotkách a směrech, 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
}
```

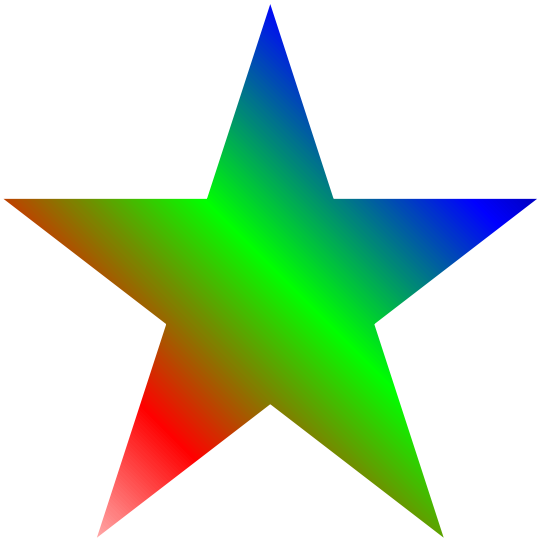
Pokud slovník naplníme jinak, dosáhneme duhy jiných vlastností. Význam jednotlivých údajů ve slovníku přesahuje rámec tohoto článku a zájemce odkazujeme na PDF referenci [8].

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