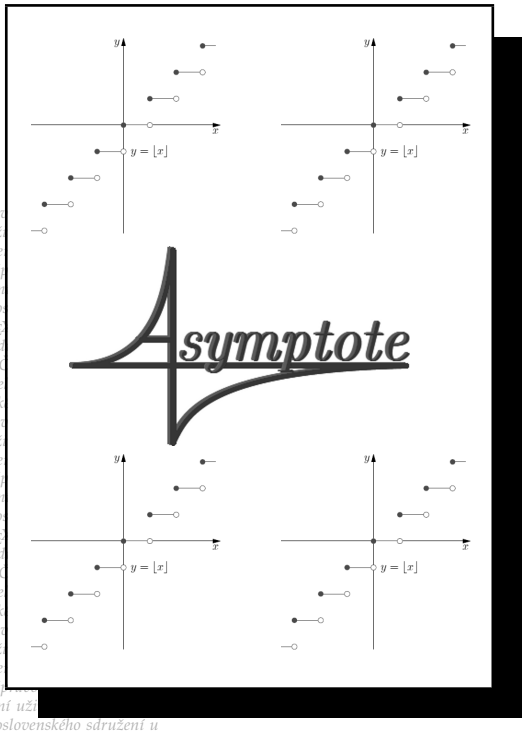
CSTUG



ZPRAVODAJ

Československého sdružení uživatelů TEXu

1–2
2015

ISSN 1211-6661 Tištěná verze

ISSN 1213-8185 Online verze

Ročník 25

OBSAH

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

Své příspěvky do Zpravodaje můžete zasílat v elektronické podobě, 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)

Úvodní slovo

Zdeněk Wagner

Otvíráte stránky poněkud opožděného, prvního a zároveň posledního dvojčísla ročníku 2015. Posledním je z toho důvodu, že se redakce snaží dohnat skluz ve vydávání, a to pokud možno trvale. Vzhledem k současnému nedostatku kvalitních článků jiná možnost není.

Redakční rada opět zvažovala, jak se Zpravodajem do budoucna naložit. Nelze slíbit, že bude vycházet pravidelně v konkrétním rozsahu, pokud neexistuje stabilní autorská základna. Je třeba si uvědomit, že členové dostávají časopis, ale členský příspěvek není předplatným časopisu. A práci redakce ztěžuje i skutečnost, že je domluva s některými autory obtížná. Přesto chce redakce časopis udržet jako periodický a především odborný jak po stránce obsahové, tak jazykové. Články tedy procházejí nejen recenzním řízením, ale až na velmi vzácné, okolnostmi vynucené výjimky též jazykovou korekturou.

Víme, jaký byl Zpravodaj v minulých letech. Redakce doufá, že i články v tomto dvojčísle čtenáře zaujmou. A jaký bude Zpravodaj v roce 2016 a letech následujících? To záleží především na autorech. Záleží na tom, jestli se mezi čtenáři, a nejen mezi nimi, najde dost odborníků, kteří by se chtěli o své zkušenosti podělit, kteří si najdou čas, aby článek napsali. Bude-li mít redakce dostatek příspěvků, bude se snažit, aby publikování bylo velice rychlé.

Zcela bez povšimnutí uniklo, že Zpravodaj má za sebou již čtvrtstoletí své existence. To čtvrtstoletí nebylo vždy jednoduché. Objevily se jisté dětské nemoci, které byly překonány. O současné situaci by se tedy mohlo psát obrazně jako o krizi středního věku. A jaké bude čtvrtstoletí další? Máme se obávat stařeckých neduhů, nebo raději očekávat moudrost zralého věku?

Nechci být přehnaně sentimentální a přát Zpravodaji další čtvrtstoletí úspěšné existence, kdy bude redakční rada s lehkostí překonávat všechny překážky, i když, pokud mi dovolíte parafrázi na jednu scénu z knihy *Hlava 22*, právě to bych mu velice přál.

„Písmo přežije svého tvůrce, i když je již pohřben pod zemí," říká arabské přísloví. Zpravodaj tedy přežije nás, ale jde o to, v jaké podobě přežívat bude, jaké se bude těšit aktivitě, a nebo snad bude také spát jako my. „Vše, čeho člověk dosáhne, plyne z jeho úsilí," říká další arabské přísloví. Budoucnost je tedy v našich rukou a nezbývá, než připomenout třetí arabské přísloví: „Začátek je polovina práce."

Zdeněk Wagner
zpravodaj@cstug.cz

Tvorba matematické grafiky pomocí programu Asymptote

Ondřej Kutal

Tento článek je vytvořen na základě diplomové práce autora (Kutal, 2012). Popisuje tvorbu matematické grafiky s programem Asymptote. Cílem textu je, aby čtenář z pestrých příkladů pochopil základní myšlenky práce s programem a zároveň měl dostatek informací o pozadí těchto postupů. V první části jsou vysvětlena základní syntaktická pravidla, v dalších částech je pak podrobně popsána metodika práce ve 2D a 3D. V závěrečné části jsou rozebrány algoritmy, které v Asymptote převádějí rovinné útvary do 3D reprezentace.

Úvod

Cílem tohoto článku je představit a popsat program Asymptote, účinný a pohodlný open source nástroj pro tvorbu 2D a 3D (i interaktivní) grafiky, vhodný především pro vědecké dokumenty. Asymptote je interpretovaný jazyk se syntaxí založenou na jazyce C++. Jeho potenciál je vidět především při psaní dokumentů v jazyce LaTeX, při kterém můžeme umisťovat bloky zdrojových kódů obrázků do speciálních prostředí a grafiku tak generovat společně se samotným dokumentem. Od roku 2004 na něm pracují John C. Bowman, Andy Hammerlindl a Tom Prince. Od té doby se Asymptote stále vyvíjí, každým rokem přibývají nové funkcionality a opravy chyb. Původně byl program koncipován jako alternativa k systému METAPOST, oproti kterému dovoluje například přesnější výpočty v plovoucí desetinné čárce či tvorbu interaktivní grafiky ve 3D.

V první kapitole je popsána instalace a nastavení programu Asymptote, včetně základních způsobů práce s programem. Ve druhé kapitole jsou popsány základy syntaxe programu. V další části jsou vysvětleny postupy pro generování 2D a 3D grafiky, včetně mnoha příkladů. Jelikož jsou však možnosti této práce omezené, doporučujeme se podívat na další příklady na oficiálních stránkách Asymptote (2004) či na Asymptote – Galeries d'exemples (2011). Pro všechny ukázky v práci byla použita verze Asymptote 2.14.

Instalace

Před instalací programu Asymptote je potřeba mít:

- implementaci TeXu
  např. TeXLive, lze stáhnout na `http://www.tug.org/texlive/`

 doi: 10.5300/2015-1-2/2

- Ghostscript GPL
  viz `http://sourceforge.net/projects/ghostscript`
- nějaký prohlížeč formátů `.ps` a `.pdf`
  např. GSview, Adobe Reader (pro správné zobrazení 3D scén v `pdf` je potřeba Adobe Reader 8.0 a vyšší)

Pro systémy Windows z oficiálních stránek programu Asymptote (2004) musíme stáhnout samotný Asymptote, například v podobě samorozbalovacího souboru `asymptote-x.xx-setup.exe`, kde `x.xx` označuje aktuální verzi programu. Pro uživatele linuxových systémů jsou k dispozici `tgz` i `rpm` balíky, pro podrobnější informace k distribuci viz oficiální stránky Asymptote (2004). Po instalaci je často potřeba provést nastavení souboru `config.asy`, viz následující odstavec.

Nastavení `config.asy`

Po nainstalování Asymptote je vhodné provést základní konfiguraci, a to především nastavit cesty k jednotlivým prohlížečům, TEXové implementaci, případně další parametry. Nastavení se provádí v souboru `config.asy`, který se nachází ve složce `.asy` v uživatelově domovské složce (na systémech Windows se nachází v `%userprofile%`), případně se dá nastavit alternativní cesta využitím systémové proměnné `ASYMPTOTE_HOME`.

Většinu systémových cest program Asymptote při instalaci nastaví sám, na linuxových systémech tato starost většinou úplně odpadá. Někdy je přesto potřeba něco nastavit ručně. Důležité jsou především cesty k PostSriptovému prohlížeči (`psviewer`), GhostScriptu (`gs`) a prohlížeči PDF (`pdfviewer`). Na linuxových systémech vše zpravidla funguje ihned po instalaci, na systémech Windows by nastavení `config.asy` mohlo vypadat například takto:

```
import settings;

gs="C:\Program Files\Ghostscript\gs9.00\bin\gswin32c.exe";
psviewer="C:\Program Files\Ghostgum\gsview\gsview32.exe";
pdfviewer="C:\Program Files\Adobe\Reader 9.0\Reader\AcroRd32.exe";
```

Další informace najdete na oficiálních stránkách Asymptote (2004).

Parametry příkazového řádku

Program Asymptote je možné spustit s různými parametry, které budou mít platnost pouze pro dané spuštění. Většina z těchto parametrů se dá nastavit i v souboru `config.asy`, a to použitím delšího názvu parametru (pokud má i zkrácenou verzi). Tabulka 1 shrnuje seznam některých parametrů (úplný seznam se zobrazí při spuštění Asymptote s parametrem `-h`).

`-a,-align C|B|T|Z`	zarovnání na stránce, Center, Bottom, Top, nebo Zero [`C`]
`-autoplay`	automatické spouštění 3D animací [`false`]
`-embed`	vloží renderovaný náhledový obrázek pro 3D scénu [`true`]
`-h,-help`	zobrazí seznam všech parametrů; pouze pro příkazovou řádku
`-f,-outformat format`	nastavení výstupního formátu
`-o,-outname name`	nastavení cesty pro výstupní soubor
`-prc`	vloží do `pdf` dokumentu 3D `PRC` model [`true`]
`-tex engine`	`latex`|`pdflatex`|`xelatex`|`tex`|`pdftex` |`context`|`none` [`latex`]
`-toolbar`	ukáže 3D toolbar v PDF výstupu [`true`]
`-version`	zobrazí verzi Asymptote
`-l,-listvariables`	vypíše seznam funkcí a proměnných [`false`]
`-V,-View`	zobrazí výstupní obrázek; pouze pro příkazovou řádku

Tabulka 1: Parametry příkazové řádky Asymptote

Pro vypnutí parametru stačí před název vložit `-no`, tj. například

```
asy -f pdf -notoolbar
```

nastaví výstupní formát `pdf` a ve výsledném dokumentu skryje pruh nástrojů pro 3D scénu.

Interaktivní mód

Do interaktivního módu programu Asymptote se dostaneme spuštěním `asy` v příkazové řádce systému. Pak stačí zadávat příkazy a potvrzovat klávesou Enter. Výstupní grafický soubor se ukládá do souboru `out.eps` v aktuálním adresáři (pokud nespecifikujeme jiný výstupní formát). Pokud máme správně nastavený PostScriptový prohlížeč (proměnná `psviewer` v souboru `config.asy`), uvidíme výstupní soubor automaticky při každé jeho změně (případně po kliknutí na obrázek nebo stisknutí klávesy R ve většině prohlížečů).

```
> size(0,50);
> fill(circle((-2,0),1),red);
> fill(circle((0,0),1),green);
> fill(circle((2,0),1),blue);
```

Obrázek 1: Ukázka výstupu

Zadávání dalších příkazů přidává objekty k dosud vykreslenému. Pro vyčištění kreslicí plochy slouží příkaz `erase`.

Užitečná je klávesa Tabulátor, při jejímž stisknutí program Asymptote doplní název funkce či proměnné (pokud je to jednoznačné), nebo nabídne možná doplnění. Zde je tabulka základních příkazů pro interaktivní mód:

`help`	zobrazí nápovědu
`erase`	vyprázdní plátno
`reset`	vrátí nastavení do původního stavu
`quit/exit`	ukončí program

Tabulka 2: Příkazy pro interaktivní mód

Dávkový (batch) mód

Dávkový mód se provede spuštěním `asy` s parametrem názvu souboru, ve kterém máme připravený kód. Program Asymptote za nás potom přečte sekvenci příkazů a vygeneruje grafiku do výstupního souboru. Implicitně je výstup ve formátu `eps`, ale vhodnou volbou parametrů (viz část Parametry příkazové řádky) lze nastavit výstup do jiných formátů[1]. Pokud například máme připravený kód v souboru `krivka.asy`, můžeme vygenerovat grafický výstup takto:

```
asy krivka.asy -V
```

kde parametr `-V` zajistí zobrazení výstupu v prohlížeči po ukončení.

Použití v LaTeXu

Pro použití Asymptote uvnitř LaTeXového dokumentu musíme mít správně umístěny příslušné LaTeXové balíky. K tomu, pokud to za nás neudělala instalace, musíme zkopírovat soubory `asymptote.sty`, `asycolors.sty` a `ocg.sty` do místa, kde LaTeX hledá balíky. V dokumentu pak načteme balík `asymptote` příkazem:

[1] Asymptote podporuje výstupní formáty `eps`, `pdf`, ale také `jpg`, `png` či `svg`, je však potřeba nainstalovat program ImageMagick pro konverzi, případně `dvisvgm`, více viz oficiální stránky Asymptote (2004).

```
\usepackage[inline]{asymptote}
```

Nyní můžeme použít prostředí `asy` a do něj umístit zdrojový kód obrázku.

```
\begin{asy}
...
\end{asy}
```

Další možností je mít kód v samostatném souboru a vkládat ho pomocí LaTeXového příkazu \asyinclude (například `\asyinclude{elipsa.asy}`).

Pro kompilaci grafiky pro dokument použijeme posloupnost příkazů

```
pdflatex -interaction=nonstopmode dokument.tex
asy -batchView dokument-*.asy
pdflatex -interaction=nonstopmode dokument.tex
```

Používáme-li nějaké prostředí pro psaní LaTeXových dokumentů, máme v nich někdy k dispozici tlačítko pro Asymptote (nebo dokonce pro posloupnost všech tří příkazů), jindy si jej musíme vytvořit (anebo kompilovat obrázky přes příkazovou řádku). Prostou obměnou předchozích příkazů můžeme generovat dokument do formátu `ps` (ovšem bez 3D grafiky), jen bychom místo předchozího napsali:

```
latex -interaction=nonstopmode dokument.tex
dvips -o dokument.ps dokument.dvi
asy -batchView dokument-*.asy
latex -interaction=nonstopmode dokument.tex
dvips -o dokument.ps dokument.dvi
```

Pro dokumenty s větším obsahem grafiky se může hodit utilita `latexmk`, pomocí níž se kompilují jen změněné obrázky (viz oficiální stránky Asymptote (2004). Případně je možné „ručně“ zkompilovat jen změněné obrázky běžným způsobem pro samostatné `asy` soubory. Pokud například víme, že se pozměněná grafika nachází pouze v souboru `dokument-2.asy`, použijeme

```
asy dokument-2.asy
```

a tím překompilujeme jen zvolený obrázek. Při příštím použití `pdflatex` (či podobného programu) se nová grafika vloží do dokumentu.

Při kompilaci dokumentu s vloženou grafikou je generováno velké množství dočasných souborů. Ty je někdy nutné promazat, a proto se hodí oddělit dočasné soubory od těch důležitých. To si můžeme usnadnit použitím LaTeXové proměnné `asydir`, která definuje podsložku pro dočasné soubory. Pro takové použití stačí vytvořit složku `temp`, na začátek dokumentu přidat

```
\renewcommand\asydir{temp}
```

a pak pouze mazat obsah složky `temp`. V takovém případě se však musí danému LaTeXovému prostředí dát vědět, aby spouštělo program Asymptote ve složce

`temp`, protože v původní složce nic nenajde. Toto nastavení záleží na konkrétním prostředí.

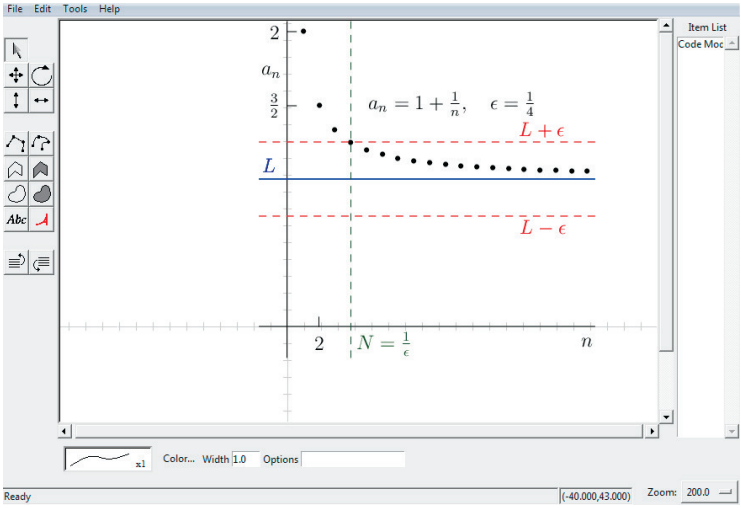
Jiný způsob, jak oddělit důležité soubory od těch dočasných, je umístit je do podsložky např. `files` a v původní složce mít pouze hlavní TEXový dokument a dočasné soubory. Pak stačí mazat všechny soubory kromě hlavního dokumentu a složky `files`.

LATEXový balík `asymptote.sty` také definuje prostředí `asydef`, do kterého můžeme vkládat globální nastavení pro celý dokument. Jedná se o obyčejné příkazy v Asymptote, je tedy možné mít nastavené například:

```
\begin{asydef}
settings.prc = false; //bez interaktivni 3D grafiky
\end{asydef}
```

Xasy

Asymptote v sobě obsahuje jednoduché uživatelské rozhraní Xasy, které je napsané ve skriptovacím jazyce Python, jehož interpret je nutný pro běh (`http://www.python.org`). Samotné Xasy zatím nabízí pouze omezené možnosti kreslení, hodí se spíše pro dolaďování již existujících obrázků.



Obrázek 2: Grafické rozhraní Xasy

Problémy

Práce v programu Asymptote vždy neprobíhá tak hladce, jak by si uživatel přál. Proto si zde uveďme nejčastější problémy a způsoby jejich řešení. Zaměřme se nejdříve na situaci, kdy nám ani **nevznikne výstupní grafický soubor**.

Při práci v LaTeXu doporučujeme v této situaci nejdřív promazat všechny dočasné soubory vytvořené Asymptote. Nejedná se však pouze o soubory `.asy`, ale také o některé `.tex`, `.ps`, `.pdf` a jiné soubory. To si můžeme usnadnit například použitím LaTeXové proměnné `asydir`.

Pokud nezabralo smazání dočasných souborů, jedná se zřejmě o syntaktickou chybu či zapomenutí nějakého balíku. V takovém případě je užitečné generovat grafiku z příkazové řádky a mít celý kód ve zvláštním `.asy` souboru. Pak jednoduše uvidíme chybová hlášení a měli bychom být schopni chybu odhalit. Případně můžeme zvýšit úroveň výpisů pomocí parametrů `-v`, `-vv` až `-vvvvv`. Asymptote dovoluje i ladění z příkazové řádky, včetně nastavení `breakpointů` apod. Více lze nalézt na oficiálních stránkách Asymptote (2004).

Dalším častým zdrojem problémů bývá chybné či nedostatečné nastavení souboru `config.asy`. Může se jednat o nesprávnou cestu k souborům TeXového kompilátoru, Ghostscriptu či samotných souborů Asymptote, více viz část Nastavení `config.asy`. Na závěr si uveďme řešení některých dalších problémů.

Kompilace se zastaví při zpracovávání programem Ghostscript, na výstupu se objeví seznam čísel a program čeká, až uživatel operaci potvrdí stiknutím klávesy Enter.

Tento problém nastává na systémech Windows při generování 3D grafiky s TeXovými popisy. Je způsoben použitím `gswin32.exe` místo `gswin32c.exe` v konfiguračním souboru `config.asy`. Po změně by měl problém zmizet.

Vše se zkompiluje, ale při prohlížení v Adobe Readeru je scéna po aktivaci příliš přiblížená.

Zde se jedná o chybu Javascriptu v linuxových verzích Adobe Readeru a projevuje se v systémech s neanglickým prostředím. Stačí spouštět Adobe Reader pomocí příkazu:

```
LC_NUMERIC=C acroread
```

Pro širší nastavení lze na vhodné místo do souboru `/usr/bin/acroread` vložit řádek:

```
export LC_NUMERIC="C"
```

V případě, že se ani uvedenými způsoby nepodaří problém vyřešit, je stále velká šance, že někdo měl již podobný problém. Doporučujeme proto diskusní fórum k programu Asymptote a často pokládané otázky (viz oficiální stránky Asymptote, 2004).

Syntaxe

Program Asymptote používá syntaxi velice podobnou jazykům C/C++ a Java, jsou zde však určité rozdíly. Tuto sekci probereme podrobněji, aby i čtenář neznalý jazyků C, C++ či Java mohl Asymptote využít. Přitom však předpokládáme znalosti alespoň základních principů programování.

Základy

Soubor se zdrojovým kódem Asymptote se skládá z posloupnosti příkazů oddělených středníkem. Příkazem

```
import balik;
```

lze načítat balíky pro nejrůznější potřeby. Balíky nejsou ve skutečnosti nic jiného než zdrojové soubory Asymptote, většinou dodávané tvůrci programu. Předchozí příkaz by tak vložil zdrojový kód ze souboru `balik.asy`. Takové soubory obsahují definice pokročilých typů a funkcí. Podrobnější informace o balících lze nalézt v části Balíky.

Vkládání komentářů se provádí stejně jako v C/C++. První možností je použít značku `//`, která zahajuje komentář, a následný text do konce řádku je tak při kompilaci ignorován. Pro rozsáhlejší komentáře slouží dvojice značek `/*` a `*/`, označujících postupně začátek a konec komentáře.

```
/*
Zde se se muze nachazet komentar
i na nekolik radku.
*/
```

```
prikaz1; //neco se provede zde
prikaz2; //a neco jineho zde
```

Asymptote ignoruje prázdná místa tvořená z více mezer či zalomením řádků. Pozor však na psaní mezer mezi dvěma čísly, to je chápáno jako násobení. Následující příklady zdrojových kódů tak generují stejný výsledek.

```
//priklad1.asy
prikaz1;
prikaz2;
prikaz3;
```

```
//priklad2.asy
prikaz1; prikaz2; prikaz3;
```

```
//priklad3.asy
prikaz1;    prikaz2;
prikaz3;
```

Datové typy

Většina proměnných, se kterými v programu Asymptote budeme pracovat, musí mít předem deklarovaný typ. Asymptote nabízí několik předdefinovaných datových typů, běžně známých z jiných programovacích jazyků. Jsou to `int` pro celá čísla, `real` pro čísla v plovoucí desetinné čárce a `bool` pro pravdivostní hodnotu výrazu `true` nebo `false`. Dále nabízí například `pair`, který slouží pro uchovávání dvojic čísel typu `real` a je také chápán jako komplexní číslo. Následuje podrobnější seznam základních datových typů.

`int` -- celé číslo, uloženo ve 64bitovém formátu, mohou se v něm uchovávat celá čísla v rozmezí -9223372036854775808 až 9223372036854775805 (tj. -2^{63} až $2^{63}-3$). Tyto limity jsou uloženy proměnných v `intMin` a `intMax`.

`real` -- číslo v plovoucí desetinné čárce, bývá implementováno jako nejpřesnější takový typ na dané architektuře. Používá `realDigits` význačných číslic a přesnost `realEpsilon`. Nejmenší a největší číslo tohoto typu je `realMin` a `realMax`.

`bool` -- booleovský typ, může nabývat hodnot `true` a `false`.

`bool3` -- rozšířený booleovský typ, kromě standardních `true` a `false` může nabývat hodnoty `default`.

`pair` -- uspořádaná dvojice čísel typu `real`. Často bývá chápáno jako komplexní číslo. K jednotlivým složkám lze přistupovat přes atributy `x` a `y` přidáním tečky, tj. například `z.x`.

`triple` -- uspořádaná trojice čísel typu `real`. Používáno pro body v prostoru či vektory. K jednotlivým složkám lze přistupovat přes atributy `x`, `y` a `z`.

`string` -- posloupnost znaků zadaná ve dvojitých uvozovkách (text).

Deklaraci proměnné provedeme napsáním názvu typu před název proměnné a oddělíme mezerou, tj. například

```
nazev_typu1 promenna1;
nazev_typu2 promenna2;
nazev_typu3 promenna3;
```

deklaruje proměnné `promenna1`, `promenna2` a `promenna3`, které mají postupně typy `nazev_typu1`, `nazev_typu2` a `nazev_typu3`. Součást deklarace může být přiřazení hodnoty (inicializace), které se provede operátorem `=`.

```
int n = 5;
real x = 0.01;
bool finished = false;
pair z = (1,2);
triple o = (0,1,0);
string label = "function f(x)";
```

Kromě uvedených typů je k dispozici spousta dalších, uživatelsky definovaných. Existuje typ `struct` podobný jako v C, pro více informací viz oficiální stránky

Asymptote (2004). Pokud nějaký typ definuje atributy nebo metody, přistupujeme k nim pomocí znaku tečky.

```
struct Kruh {
  real r; //polomer

  real obsah(){
    return pi*r^2;
  }
}

Kruh k;
k.r = 5;

write("Polomer: ",k.r); //vypise atribut r
write("Obsah: ",k.obsah()); //spocita a vypise obsah
```

Následuje přehled některých pokročilých typů.

frame -- plátno pro kreslení, pracuje v souřadnicích jazyka PostScript.

picture -- podobný typ jako **frame**, avšak využívá vlastní souřadný systém. Pokud není uvedeno jinak, uživatel kreslí do plátna **currentpicture** typu **picture**.

path -- představuje kubickou křivku.

guide -- představuje také kubickou křivku, avšak uloženou jako posloupnost ne nutně všech kontrolních bodů. Takto zadaná křivka se do typu **path** převádí až těsně před vykreslením.

pen -- pero pro kreslení, uchovává tloušťku, barvu, vzor apod. Pokud není uvedeno jinak, kreslí se perem **currentpen**.

Vypsaný seznam není úplný, k dispozici je mnoho užitečných typů, specifických pro danou problematiku a balík. Více typů bude probráno v následujících kapitolách.

Pole

Deklaraci pole (konečné posloupnosti proměnných stejného typu) provedeme přidáním `[]` za název typu proměnné nebo za samotný název proměnné.

```
int[] pole;
real[] pole2 = {0.1, 0.2};
real pole3[];
```

Uvedený příklad vytváří tři pole, přitom `pole2` se přímo inicializuje s hodnotami 0,1 a 0,2. Pro přistupování k prvkům využijeme operátor `[]` s příslušným indexem uvnitř závorek. První prvek `pole2` bychom například získali příkazem `pole2[0]`. Pole má metody a atributy pro práci s ním, jako je mazání, vkládání apod.

`int length`	počet prvků pole
`void insert(int i ... T[] x)`	vloží prvek či pole prvků `x` na index `i`
`void delete(int i, int j=i)`	smaže prvky mezi indexy `i` a `j`
`T push(T x)`	vloží prvek na konec
`T pop()`	vyjme a vrátí poslední prvek

Tabulka 3: Základní atributy a metody polí

Můžeme vytvářet i tzv. anonymní pole, což jsou pole, která nemají žádný název, a to vynecháním názvu proměnné a znaku `=` a přidáním klíčového slova `new` před celý výraz. Následující kód by měl použití ozřejmit. Jedná se o definici čárkovaného pera, při které funkci předáváme pole dvou hodnot typu `real`.

```
pen dashed = linetype(new real [] {8 ,8});
```

Operátory

Pro práci s předdefinovanými typy nabízí program Asymptote prakticky stejné operátory, jako jsou dostupné v C/C++. Základní přehled poskytuje následující ukázka:

```
int i = 0;
++i; //to same co i=i+1
i += 2; //stejne jako i=i+2
i = 12 % 5; //zbytek po deleni 5
real a = 2pi; //stejne jako 2*pi
```

Zmíněné operátory jsou často použitelné i pro jiné typy operandů, než jsou základní typy. Jak ukazuje následující příklad, je možné objektu typu `pen` nastavit tloušťku pera (podrobněji o perech viz část Pera):

```
pen mypen;
mypen += 2.0;
mypen += red;
```

Funkce

Při práci s proměnnými využíváme funkce. Jejich základní seznam lze nalézt na oficiálních stránkách Asymptote (2004). Podrobnější seznam lze získat při spuštění programu Asymptote s parametrem `-l`. Ten však neobsahuje funkce z balíků. Proto bývá nejlepší prostudovat dokumentaci k jednotlivým balíkům, případně se podívat přímo do jejich kódu. V této části nebudeme popisovat jednotlivé funkce,

ale rozebereme si jejich syntaxi. S konkrétními funkcemi se setkáme v dalším textu.

Nyní se podívejme, jak hlavičky funkcí vypadají a jak je interpretovat. Zde se Asymptote poměrně liší od jazyka C++ i Javy. Dovoleny jsou implicitní argumenty kdekoliv v hlavičce, explicitní uvedení názvu argumentu při volání funkce či umístění argumentu ne nutně tam, kde je v hlavičce uveden. Nejlépe si to ilustrujeme na příkladech.

Funkce `time` z balíku `math` spočítá n-tý průsečík křivky a svislé přímky.

```
real time(path g, real x, int n=0);
```

Samotná deklarace začíná typem návratové hodnoty. Jako typ se zde může uvést `void`, což značí, že funkce nevrací žádnou hodnotu. V našem případě funkce vrací typ `real`, tj. reálné číslo. Zde označuje čas na křivce, ve kterém se realizuje průsečík. Dále je uveden název funkce, tj. `time`, následovaný seznamem argumentů v závorkách. Argumenty (s výjimkou funkcí) jsou uvedeny v pořadí název typu, název argumentu a případně implicitní hodnota za znakem `=`. Pokud není uvedena implicitní hodnota, musíme argument vždy zadat. V tomto případě máme celočíselný nepovinný argument `n` (index průsečíku, který chceme spočítat).

Při volání funkce také můžeme explicitně zmínit název argumentu, který zadáváme. To umožňuje například zadávat argumenty v jiném pořadí, než uvádí hlavička. Následující tři volání jsou proto všechna ekvivalentní.

```
real t;
path p = (0,0)..(2,1)..(0,2);

t = time(p, 1);
t = time(p, 1, 0);
t = time(n=0,x=1,g=p);
```

Zde se ještě zastavme u tzv. přetížených funkcí. Tento koncept je převzatý z jazyka C++ a zajišťuje větší flexibilitu a přehlednost. Nemusíme mít funkce s různými jmény k tomu, abychom udělali stejný úkon, jen v jiném kontextu. Například pro vytváření ploch existuje spousta funkcí s názvem `surface()`. Každá se však používá s jinými typy argumentů, takže nenastává konflikt. Výše zmíněná funkce `time()` má také další variantu. Její hlavička vypadá následovně:

```
real time(path g, pair z, int n=0);
```

Slouží k počítaní průsečíků, tentokrát křivky a vodorovné přímky. Od předchozí deklarace se liší argumentem `z` typu `pair`. Odpovídající vodorovná přímka je určená bodem `(0,z.y)`. První složka `z` je zde nepodstatná, avšak autoři zde nemohli použít typ `real`, poněvadž by nastal konflikt s předchozí verzí funkce `time()`.

```
real t;
```

```
path p = (0,0)..(2,1)..(0,2);

t = time(p, 1); //1. varianta
t = time(p, (0,1)); //2. varianta
```

Funkce se také mohou nacházet v argumentech nebo být vráceny jinou funkcí. Jejich zápis mezi argumenty vypadá podobně jako jejich deklarace, jen je vynechán název jednotlivých argumentů, tj. je uveden typ návratové hodnoty, název funkce a v závorce jednotlivé typy argumentů. Jako příklad uveďme následující variantu funkce `graph()` ze stejnojmenného balíku:

```
guide graph(picture pic=currentpicture, real f(real),
            real a, real b, int n=ngraph,
            real T(real)=identity,
            interpolate join=operator --);
```

Zde vidíme, že dva argumenty jsou funkce. První povinnou funkcí je zde `f`, která musí vracet typ `real` a přebírat jeden argument typu `real`. Zřejmě se jedná o reálnou funkci jedné reálné proměnné, jíž chceme vykreslit graf. Další funkcí je zde `T` a má stejný návratový typ a typy argumentů jako `f`. Význam se samozřejmě liší, jedná se totiž o transformaci nezávislé proměnné a mezí (implicitně identita).

Pokud má funkce jako návratový typ jinou funkci, musí se předem pojmenovat typ takové funkce pomocí klíčového slova `typedef`. Nejlépe bude vše prezentovat opět na příkladu.

```
typedef real func_type(real);

func_type f(int n){
  real temp(real x){
    return n*sin(x/n);
  }
  return temp;
}
```

Zde klíčové slovo `typedef` říká, že pod názvem `func_type` budeme nyní rozumět typ takové funkce, která přebírá argument typu `real` a rovněž vrací hodnotu stejného typu.

Podobně jako u polí, i funkce můžeme vytvářet anonymní. Takové funkce nemají název, protože je využíváme pouze krátkodobě, většinou jako argument pro jinou funkci. Deklarace anonymní funkce se od obyčejné liší pouze tím, že jí chybí název a před celý výraz je přidáno klíčové slovo `new`.

```
//"klasicky" zpusob
real f(real x){
  return sin(x);
}

draw(graph(f, -2pi, 2pi));
```

```
//s vyuzitim anonymni funkce
draw(graph(new real(real x){ return sin(x);}, -2pi, 2pi));
```

Cykly

Asymptote má syntaxi pro cykly `for`, `while` a `do` shodnou s C/C++ či Javou, od druhého jmenovaného jazyka si navíc propůjčuje syntaxi pro procházení prvků pole. Následující ukázka kódu prezentuje výpis čísel od 0 do 4 pomocí různých cyklů.

```
//ukazka for, while, do a foreach
for(int i = 0; i < 5; ++i){
  write(i);
}

int i = 0;
while(i < 5){
  write(i);
  ++i;
}

int i = 0;
do{
  write(i);
  ++i;
}while(i < 5);

int[] pole = {0,1,2,3,4};
for(int i : pole){
  write(i);
}
```

Balíky

Asymptote nabízí spoustu předpřipravených balíků pro různé účely, po jejichž načtení máme k dispozici celou řadu užitečných funkcí, například balík `graph` pro snadnou tvorbu grafů funkcí, `animation` pro vytváření animací nebo třeba `MetaPost`, obsahující funkce kompatibilní s tímto systémem. Asymptote však není prostředek pouze pro vykreslování grafiky, nabízí také různé matematické funkce, například pro řešení obyčejných diferenciálních rovnic (balík `ode`), řešení úlohy lineárního programování simplexovou metodou (balík `simplex`), či některé obecnější funkce (balík `math`). Dále si uživatel může balíky sám vytvářet, jsou to obyčejné soubory formátu `.asy` (příklad viz část Nespojitá funkce). Jejich vkládání se pak provádí příkazem `import`. Například

```
import three;
```

načte balík pro práci s vektory ve 3D (uložený v souboru `three.asy`). V této části si předvedeme základní balíky pro práci ve 2D i ve 3D.

Občas je nutné načíst nejen balíky programu Asymptote, ale také LaTeXové balíky. K tomu slouží funkce `usepackage()`:

```
void usepackage(string s, string options="");
```

```
usepackage("amssymb");
```

Hodí se nám tehdy, když v popiskách používáme konstrukce, které i v samotném LaTeXu vyžadují nějaký balík.

Matematické konstanty a funkce

Zde se již nebudeme zabývat syntaxí, ale uvedeme si základní matematické funkce. Jejich význam však není nijak vázán na to, zda pracujeme ve 2D nebo 3D, proto je tato část umístěna zde. Pro některé uvedené funkce a proměnné je potřeba načíst balík `math` příkazem:

```
import math;
```

Co se týče číselných konstant, používá Asymptote následující:

```
pi = 3.14159265358979;
I = (0,1); //komplexni jednotka
```

Z elementárních funkcí jsou k dispozici:

```
real sin(real x); //goniometricke funkce
real cos(real x);
real tan(real x);
real asin(real x); //cyklometricke funkce
real acos(real x);
real atan(real x);
real exp(real x); //exponencialni funkce
real log(real x); //prirozeny logaritmus
real log10(real x);
real sqrt(real x); //druha odmocnina
real cbrt(real x); //treti odmocnina
```

Dále jsou definovány také hyperbolické funkce s jejich inverzemi:

```
real sinh(real x); //hyperbolicke funkce
real cosh(real x);
real tanh(real x);
real asinh(real x);
real acosh(real x);
real atanh(real x);
```

K uvedeným funkcím `sin`, `cos`, `tan`, `exp`, `log` a `sqrt` existují také jejich komplexní varianty, jen místo argumentu `real` používají typ `pair`.

Výpočet absolutní hodnoty obstará funkce:

```
real abs(real x); //absolutni hodnota
```

Tato funkce má také varianty pro argumenty typu `pair` a `triple`, vracející velikost příslušných vektorů.

Pro převod mezi stupni a radiány lze využít:

```
real degrees(real radians);
real radians(real degrees);
```

Pro zaokrouhlování jsou k dispozici:

```
int floor(real x);
int ceil(real x);
int round(real x);
```

Faktoriál a kombinační čísla lze vytvořit funkcemi:

```
int factorial(int n);
int choose(int n, int k);
```

Náhodná čísla můžeme generovat použitím:

```
void srand(int seed); //(nepovinna) inicializace seminkem
int rand(); //nahodne cele cislo v intervalu [0,randMax]
real unitrand(); //nahodne cislo v intervalu [0,1]
```

Práce ve 2D

Plátna

Vše, co v programu Asymptote kreslíme, se odehrává na nějakém plátně. Pro reprezentaci plátna existují dva typy. Jsou jimi `frame` a `picture`. Liší se od sebe tím, že `frame` interpretuje všechny délky v jednotkách bp jazyka PostScript, zatímco `picture` umožňuje definovat vlastní jednotky. Ty se pak samy převádějí na jednotky jazyka PostScript. Většinou se používá právě typ `picture`.

Jednotky a typ `picture`

Pokud žádné jednotky nespecifikujeme, všechna čísla se budou chápat v jednotkách bigpoint jazyka PostScript (1 bp = 1/72 palce). Praktičtější je však nastavit vlastní jednotky v rámci plátna typu `picture`. Slouží k tomu funkce `unitsize`.

```
void unitsize(picture pic=currentpicture,
              real x, real y=x);
```

Pokud nespecifikujeme plátno `pic`, automaticky se použije `currentpicture`. Argumenty `x` a `y` určují, jaké jednotky se použijí ve směru osy x, případně y. Můžeme tak například napsat:

```
unitsize(1cm);
//stejne jako
//unitsize(1cm,1cm)
```

Vše, co potom vykreslíme na implicitním plátně `currentpicture`, bude v jednotkách centimetrů. Kromě `cm` lze použít také `mm`, `inches`, `bp` a `pt`. Další funkce se zdá být v praxi užitečnější, nastaví totiž výstupní rozměry pro daný obrázek.

```
void size(picture pic=currentpicture, real x, real y=x,
          bool keepAspect=Aspect);
```

Výsledný obrázek nebude mít šířku přesahující `x` a výšku přesahující `y`. Pokud se za proměnnou `x`, příp. `y` předá 0, nebude v daném směru kladeno žádné omezení. Pokud se za proměnnou `keepAspect` použije hodnota `Aspect` nebo `true`, bude zachován poměr stran a obrázek se vykreslí tak, aby šířka ani výška nepřesáhly zadané hranice. Pokud se použije hodnota `IgnoreAspect` nebo `false`, poměr stran se upraví tak, aby šířka i výška obrázku odpovídaly zadaným `x` a `y`.

```
void draw(picture pic=currentpicture, Label L="", path g,
          align align=NoAlign, pen p=currentpen,
          arrowbar arrow=None, arrowbar bar=None,
          margin margin=NoMargin, Label legend="",
          marker marker=nomarker);
```

Do obrázku `pic` vykreslí křivku předanou v parametru `g`. Přitom je možno nastavit pero `p`, případně popisek `L`. Jde o základní funkci pro vykreslování.

Jednotky jsou implementovány jako předdefinované konstanty typu `real` a jsou to násobky bigpointu. Například hodnota konstanty `mm` je přibližně 2,83, poněvadž 1 mm $\approx$ 2,83 bp. Z tohoto důvodu je při použití `unitsize` nežádoucí, uvádět kdekoliv jinde v kódu jednotky (s výjimkou míst, která jsou na jednotkách plátna nezávislé). Z povahy implementace by totiž takto vynásobená čísla s největší pravděpodobností neodpovídala zamýšlené délce. Ze stejného důvodu je nevhodné pro názvy proměnných používat jednotky.

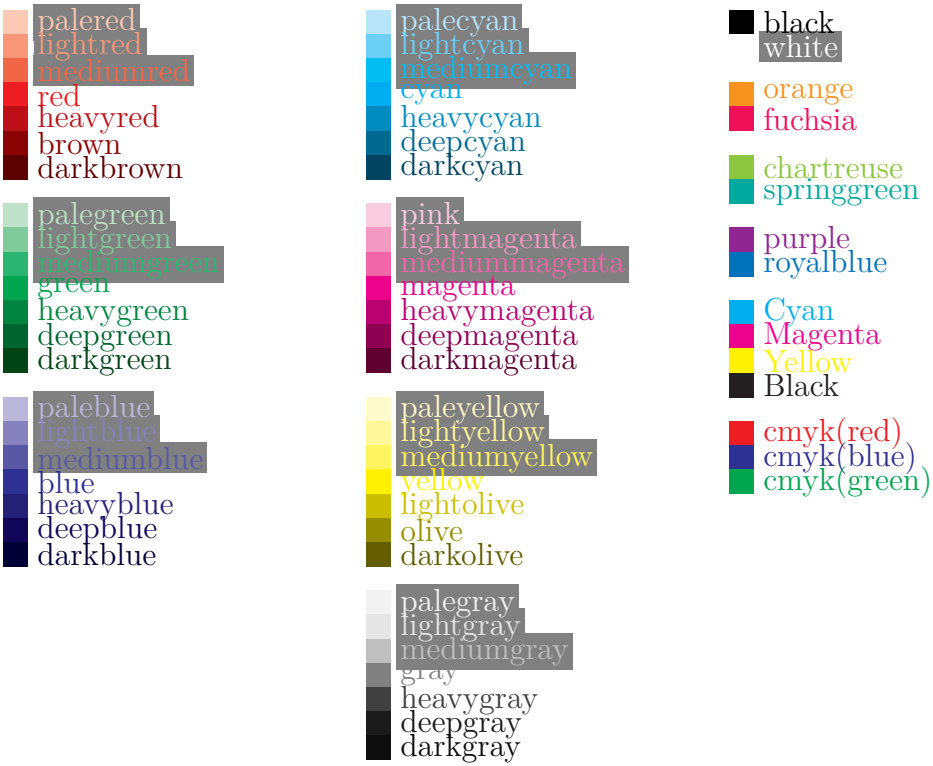
Pera

Pera hrají při kreslení velmi důležitou roli.. S jejich pomocí uživatel nastaví tloušťku, barvu, velikost písma a mnoho dalších. Pera mají typ `pen` a nejčastěji se používají jako argumenty čtyř základních funkcí pro vykreslování `draw()`,

`fill()`, `clip()` a `label()`. Pokud při kreslení neuvedeme žádné pero, je použito `currentpen`. Můžeme také globálně nastavit, aby `currentpen` mělo požadované vlastnosti.

V dalších částech si popíšeme, jak se vytvářejí pera určitých vlastností. Takto vytvořená pera se dají kombinovat binárním operátorem `+`, případně operátorem `*` pro násobení hodnot barevných složek číslem.

Barvy



Obrázek 3: Barevná paleta

Pera určité barvy můžeme vytvářet na základě RGB i CMYK příslušnými funkcemi.

```
pen rgb(real r, real g, real b);
```

```
pen cmyk(real c, real m, real y, real k);
```

Argumenty `r`, `g`, `b`, případně `c`, `m`, `y`, `k` leží v intervalu $[0; 1]$. Pokud chceme definovat například pero s červenou barvou, můžeme napsat:

```
pen cervena = rgb(1,0,0);
```

Většinou však využijeme předpřipravené palety, případně namícháme barvy pomocí operátoru `+`. Například

```
pen p = red+green;
```

vytvoří žlutou barvu. Následuje přehled předdefinovaných barev.

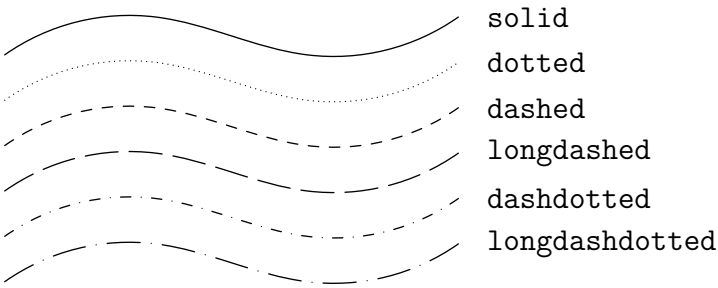
Typy čar

Typy čar jsou v programu Asymptote určeny posloupností čísel typu `real` (polem) spolu s dodatečnými parametry. Uveďme si nejdříve hlavičku funkce pro sestrojení pera s určitým typem čáry. Potom popíšeme význam jednotlivých argumentů.

```
pen linetype(real[] a, real offset=0,
             bool scale=true, bool adjust=true);
```

Pole čísel `a` odpovídá délkám po sobě jdoucích úseků, kde první číslo udává délku viditelného úseku, druhé číslo pak délku následujícího neviditelného úseku, třetí číslo opět délku následujícího viditelného úseku atd. Zde 0 odpovídá tečce. Argument `offset` určuje počáteční posunutí. Pomocí `scale` nastavíme, jestli se zadané délky úseků mají chápat jako násobky tloušťky čáry (pro `scale=true`). Argument `adjust` zajistí, aby se délky mezer mezi úseky přizpůsobovaly délce dané křivky, takže se nestane, aby křivka skončila mezerou nebo jen částečnou čárkou. Důležité také je, že zadané délky nejsou nijak svázané s jednotkami plátna a jsou tak v jednotkách `bp` jazyka PostScript.

Podobně jako u barev i zde Asymptote nabízí předdefinované konstanty pro nejpoužívanější typy čar. Implicitně se využívá typ `solid`.



Obrázek 4: Typy čar

Pro lepší pochopení původní definice se ještě podívejme, jak je definován čárkovaný (`dashed`) a čerchovaný (`dashdotted`) typ čar.

```
pen dashed=linetype(new real[] {8,8});
pen dashdotted=linetype(new real[] {8,8,0,8});
```

U obou typů je argument `scale` implicitně roven `true`, a proto se zadané délky budou chovat jako násobky aktuální tloušťky pera `linewidth(p)`. Čárkovaná čára je tedy definovaná tak, že úsek délky $8 \cdot$ `linewidth(p)` je vždy viditelný a následuje ho prázdný úsek délky $8 \cdot$ `linewidth(p)`. Podobně je to s čerchovanou čárou, kde 0 znamená tečku.

Křivky

Základním útvarem, který budeme v programu Asymptote konstruovat, přímo či nepřímo, jsou křivky. Všechny úsečky, grafy funkcí nebo třeba i popisky jsou uloženy jako segmenty kubických křivek. Pro křivky v rovině má Asymptote typy `guide` a `path`. Typ `guide` pouze uchovává posloupnost bodů, ale nemá ještě dopočítány všechny údaje (je vhodnější ve fázi sestrojování křivky). Typ `guide` se těsně před vykreslením převádí na typ `path`, ve kterém už jsou dopočítány všechny kontrolní body.

Křivku tvoříme navazováním bodů pomocí vhodných operátorů, které mezi nimi určují způsob interpolace. Pokud má být křivka uzavřená, jako poslední bod uvedeme `cycle`. Pro lineární interpolaci (spojení úsečkou) používáme `operator -`. Například jednotkový čtverec můžeme vytvořit takto:

```
path ctverec = (0,0)--(1,0)--(1,1)--(0,1)--cycle;
```

Jednotkový čtverec je mimochodem v Asymptote předdefinovaný právě tímto způsobem, a to v proměnné `unitsquare`.

Pro interpolaci Bézierovou kubikou používáme `operator ..`; ten se dá použít s různými parametry, popišme si ho proto podrobněji.

Bézierova kubika je zadaná celkem čtyřmi kontrolními body. Vždy budeme muset zadat krajní body, kterými prochází. Zbytek může Asymptote dopočítat. Pokud chceme zadat i kontrolní body, použijeme zápis:

```
a .. controls c0 and c1 .. b
```

V případě, že nezadáváme kontrolní body přímo, vypadá zápis s parametry takto:

```
a{dir1} .. tension t1 and t2 .. {dir2}b
```

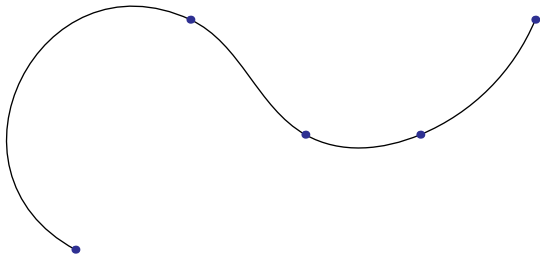
Parametry `dir1` a `dir2` udávají směry tečných vektorů segmentu v prvním a druhém krajním bodě (jsou typu `pair`). Parametry `t1` a `t2` určují míru zakřivení v podobě reálného čísla většího nebo rovného 0,75. Čím větší tato hodnota je, tím víc se tvar blíží přímce. Pokud nejsou zadány, volí se hodnota 1. Žádný z těchto parametrů však zadávat nemusíme, nezadané směry a výsledné kontrolní body se určí pomocí tzv. Hobbyho algoritmu.

Pro samotné vykreslování křivek slouží funkce `draw()`.

```
void draw(picture pic=currentpicture, Label L="",
          path g, align align=NoAlign,
          pen p=currentpen, arrowbar arrow=None,
          arrowbar bar=None, margin margin=NoMargin,
          Label legend="", marker marker=nomarker);
```

Tato varianta vykreslí křivku `g` do plátna `pic`. Před popisem dalších argumentů si uveďme jednoduchý příklad.

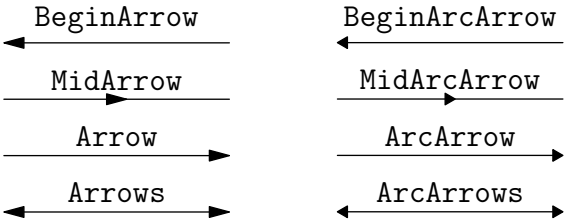
```
size(200,0);

pair[] z = {(0,0),(1,2),(2,1),(3,1),(4,2)};

path p =z[0]..z[1]..z[2]..z[3]..z[4];

draw(p);
dot(z,blue);
```



Obrázek 5: Vykreslení typu `path` spolu s kontrolními body

Mezi další argumenty patří popisek `L` a jeho směr zarovnání `align`. Dále `arrow` umožňuje přidat na křivku šipku. Možné hodnoty jsou vidět na obrázku 6. Každá z těchto šipek jde navíc volat jako stejnojmenná funkce s argumenty určující velikost, tvar apod. Pro více informací doporučujeme balík `plain_arrows`.

```
picture left, right;

size(left,3cm);
size(right,3cm);

path p = (0,0)--(2,0);

pair align = N;

draw(left,"\tt BeginArrow",p,align,BeginArrow);
draw(left,"\tt MidArrow",shift(0,-0.5)*p,align,MidArrow);
draw(left,"\tt Arrow",shift(0,-1)*p,align,Arrow);
draw(left,"\tt Arrows",shift(0,-1.5)*p,align,Arrows);

draw(right,"\tt BeginArcArrow",p,align,BeginArcArrow);
draw(right,"\tt MidArcArrow",shift(0,-0.5)*p,align,MidArcArrow);
draw(right,"\tt ArcArrow",shift(0,-1)*p,align,ArcArrow);
draw(right,"\tt ArcArrows",shift(0,-1.5)*p,align,ArcArrows);

add(left.fit(),(0,0),20W);
add(right.fit(),(0,0),20E);
```

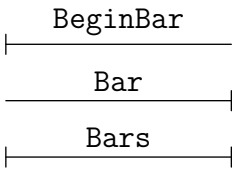


Obrázek 6: Šipky

Použitím `bar` můžeme přidat příčku na konci křivky:

```
size(3cm);

path p = (0,0)--(2,0);

pair align = N;

draw("\tt BeginBar",p,align,bar=BeginBar);
draw("\tt Bar",shift(0,-0.5)*p,align,bar=Bar);
```

```
draw("\tt Bars",shift(0,-1)*p,align,bar=Bars);
```



Obrázek 7: Příčky

Jelikož jsou křivky orientované a často určují nějakou oblast (například pro vyplnění), hodí se nám funkce `reverse` pro změnu parametrizace na opačný směr.

```
path reverse(path p);
```

Další funkce (pro počítání průsečíků apod.) lze nalézt na oficiálních stránkách Asymptote (2004).

Vyplňování

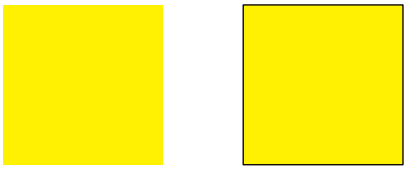
Uzavřené křivky můžeme vyplnit funkcí `fill()`, případně `filldraw()`.

```
void fill(picture pic=currentpicture, path g,
          pen p=currentpen);
```

```
void filldraw(picture pic=currentpicture, path g,
              pen fillpen=currentpen,
              pen drawpen=currentpen);
```

Liší se od sebe pouze tím, že druhá funkce vykreslí i obrys, viz obr. 8.

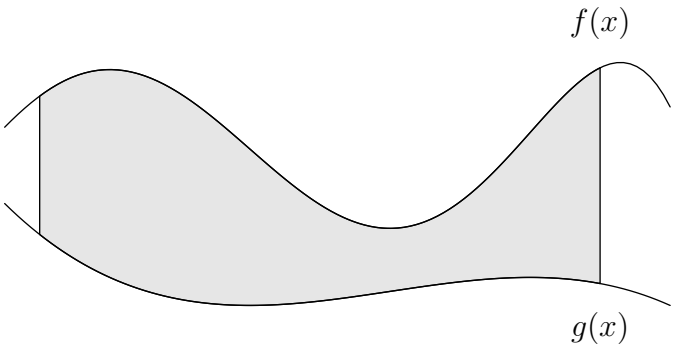
```
size(150);

fill(unitsquare,yellow);
filldraw(shift(1.5,0)*unitsquare,yellow,black);
```



Obrázek 8: Vyplnění funkcemi `fill()` a `filldraw()`

Funkci můžeme předat i křivky, které jsou vytvořené funkcemi pro tvorbu grafů. Následující příklad ilustruje, jak vyplnit oblast mezi grafy funkcí $f(x)$ a $g(x)$ (podrobněji o grafech viz část Grafy funkcí).

```
import graph;

size(250,0);

real f(real x){
  return -1/40*x^5+1/4*x^4-2/3*x^3+9/8*x+2;
}

real g(real x){
  return -1/20*x^3+9/20*x^2-6/5*x+2;
}

filldraw(graph(f,0.5,4.5)--graph(g,4.5,0.5)--cycle,
         lightgray);

draw(graph(f,0.25,5),black);
draw(graph(g,0.25,5),black);

label("$f(x)$",(4.5,f(4.5)),3N);
label("$g(x)$",(4.5,g(4.5)),3S);
```

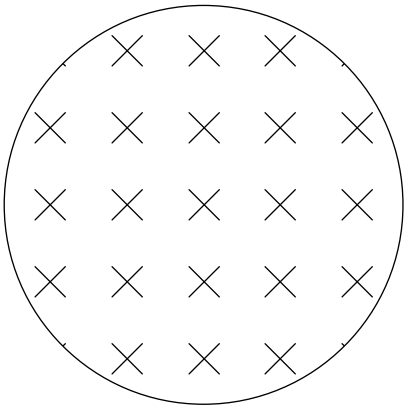


Obrázek 9: Vyplnění funkcí `fill`

Pero nemusí obsahovat jen základní informace jako barvu, ale může definovat i vzor pro vyplnění, kterým může být prakticky libovolný obrázek.

```
import patterns;

size(0,150);
```

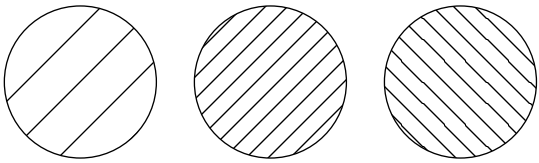
```
//krizek
picture custom;
real h = 2mm;
path[] cross = (-1,-1)--(1,1)^^(1,-1)--(-1,1);
draw(custom,scale(h)*cross);

//pridani do vzoru
add("custompattern",custom,(3h,3h));

filldraw(unitcircle,pattern("custompattern"));
```



Obrázek 10: Vyplnění vlastním vzorem

V uvedeném příkladu jsme do vlastního plátna nakreslili křížek. Použitím funkce `add` jsme ho pak přidali do seznamu vzorů. Funkce `pattern` potom vytvořila příslušné pero, se kterým byla vyplněna kružnice. Pro praktické použití máme v balíku `patterns` k dispozici funkce pro vytváření obrázků, které se nám hodí pro různé typy šrafování.

```
import patterns;

size(200,0);

add("hatch",hatch());
add("hatch2",hatch(2mm));
add("hatchback2",hatch(2mm,NW));

filldraw(unitcircle,pattern("hatch"));
filldraw(shift(2.5,0)*unitcircle,pattern("hatch2"));
filldraw(shift(5,0)*unitcircle,pattern("hatchback2"));
```



Obrázek 11: Šrafování

Ořezávání

```
void clip(picture pic=currentpicture, path g, stroke=false,
          pen fillrule=currentpen);
```

Funkcí `clip` můžeme oříznout obsah plátna `pic` křivkou `g`. Pro matematickou grafiku jsou však zajímavější následující dvě funkce.

```
void xlimits(picture pic=currentpicture, real min=-infinity,
             real max=infinity, bool crop=NoCrop);
```

```
void ylimits(picture pic=currentpicture, real min=-infinity,
             real max=infinity, bool crop=NoCrop);
```

```
void crop(picture pic=currentpicture);
```

Tyto funkce nastaví meze obrázku ve směrech os x a y a následné volání funkce `crop` ořeže existující křivky na plátně podle zadaných mezí. To je praktické zejména při kreslení grafů, kdy chceme, aby graf nepřesahoval zadané meze. Použití lze vidět např. na obrázku 14 v části Nespojitá funkce.

Transformace

Pro účely afinních transformací existuje typ `transform`. Proměnnou tohoto typu můžeme vynásobit zleva libovolnou proměnnou typu `pair`, `guide`, `path`, `pen`, `string`, `frame` nebo `picture`. Po provedení dané transformace obdržíme proměnnou stejného typu. Popišme si, co transformace pro konkrétní typy znamená.

`pair` -- transformace se aplikuje na bod v rovině.

`guide, path` -- transformace se aplikuje na všechny body daných křivek včetně kontrolních bodů (Bézierovy křivky jsou však invariantní vůči afinním transformacím).

`string` -- převede se na typ `Label`, což není nic jiného než křivka, na kterou se následně aplikuje transformace.

`frame, picture` -- transformace se aplikuje na obsah plátna (těsně před vykreslením).

Uživatel si samozřejmě může definovat vlastní transformace. Obecná afinní transformace

$$t(x) = \begin{pmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{pmatrix} x + \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} \tag{1}$$

se vytvoří příkazem:

```
transform t = (b1,b2,a11,a12,a21,a22);
```

Pro jednodušší práci jsou nejpoužívanější transformace předdefinovány. Následuje seznam a popis funkcí, které vytvářejí a vracejí příslušné transformace.

```
transform identity();
```

Identická transformace. Slouží víceméně jen proto, aby se dala použít jako implicitní transformace.
Translace (posunutí) určené buď vektorem `z`, nebo jednotlivými složkami `x`, `y` zvlášť:

```
transform shift(pair z);
transform shift(real x, real y);
```

Otočení o úhel `angle` (ve stupních) kolem bodu `z`:

```
transform rotate(real angle, pair z=(0,0));
```

Změna měřítka, opět je možno zadat více způsoby. Funkce `scale()` mění měřítko ve směru obou os. Zbylé funkce umožňují měnit měřítka pro obě osy zvlášť:

```
transform scale(real s);
transform scale(real x, real y);
transform xscale(real x);
transform yscale(real y);
```

Převrácení kolem přímky určené body `a` a `b`:

```
transform reflect(pair a, pair b);
```

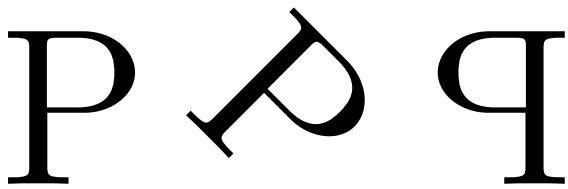
K dané transformaci `t` můžeme získat inverzní transformaci pomocí funkce `inverse`:

```
transform inverse(transform t);
```

Transformace můžeme skládat, a to pomocí operátoru násobení (`*`). Na závěr uveďme ilustrační příklad.

```
size(0,200);

Label l = scale(7)*"P";
```

```
//pismeno "P"
label(l);

//otocene "P" o -45 stupnu
label(shift(80,0)*rotate(-45)*l);

//prevracene "P"
label(shift(160,0)*reflect((0,0),(0,1))*l);
```



Obrázek 12: Transformace

Grafy funkcí

Funkce dané explicitně

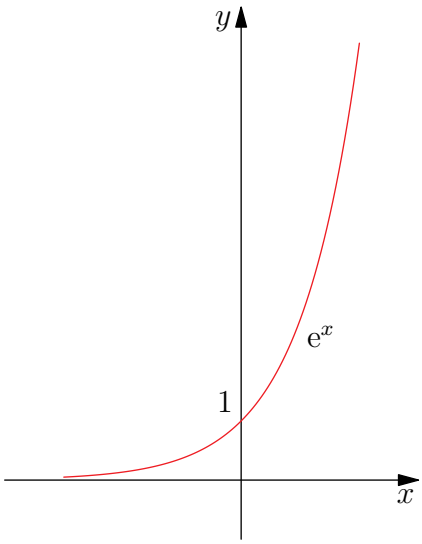
Při vykreslování grafů funkcí budeme využívat balík `graph`. Pro vykreslení reálné funkce v explicitním tvaru $y = f(x)$ využijeme následující variantu funkce `graph()`:

```
guide graph(picture pic=currentpicture, real f(real), real a,
            real b, int n=ngraph, real T(real)=identity,
            interpolate join=operator --);
```

Funkce spočítá funkční hodnoty podle zadaných parametrů a vrátí vytvořenou křivku. Argument `f` musí být funkce s jedním argumentem typu `real` a vracející hodnotu stejného typu (tj. odpovídající reálná funkce jedné reálné proměnné). Argumenty `a` a `b` určují meze na ose x. Mezi nepovinnými argumenty můžeme zadat plátno `pic`, případně dělení `n`. Předveďme si použití na vykreslení grafu funkce $f(x) = \mathrm{e}^x$.

```
import graph;

size(200);

real f(real x){
```

```
  return exp(x);
}

draw(graph(f,-3,2),red);

xaxis("$x$",-4,3,Arrow);
yaxis("$y$",-1,8,Arrow);

labely("$1$",1,NW);
label("$\mathrm{e}^x$",(1,f(1)),SE);
```



Obrázek 13: Funkce $f(x) = \mathrm{e}^x$

Nespojité funkce

V případě nespojitých funkcí (příp. funkcí nedefinovaných na celém zadaném intervalu) je situace komplikovanější. Program Asymptote totiž při počítání funkčních hodnot v bodech, kde není definována funkční hodnota, vyvolá chybu a graf se nevykreslí. U nespojitých funkcí mu naopak chybí informace o tom, že v daném místě nemá části grafu spojovat. To se dá vyřešit několika způsoby.

Řešením prvního problému by mohlo být přidání definic funkčních hodnot pro ty body, kde funkční hodnota není definována. Jedná se o však o zcela nevhodné řešení, protože pak dostáváme jiný graf. Trochu lepším řešením je vykreslovat části grafu zvlášť, a to na intervalech, kde je funkce definována. Řešením, které

umožňuje vypořádat se i s druhým zmíněným problémem, je argument `cond` ve funkci `graph()`. Jeho hlavička je očekávána ve tvaru:

```
bool3 cond(real x);
```

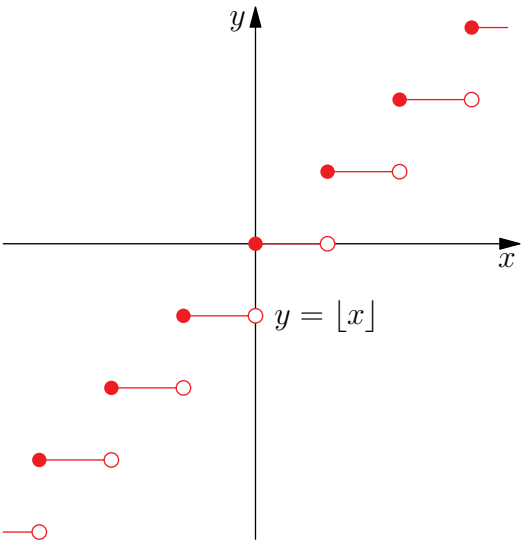
Tato, námi zadaná funkce umožňuje sdělovat funkci `graph()`, které hodnoty na ose x smí či nesmí vyhodnocovat. To vše navíc tak, že bude vědět, kdy začíná další větev grafu, takže nedojde k jejich propojení. Pro lepší pochopení následujícího příkladu (a pro opětovnou využitelnost) si definujeme vlastní jednoduchý balík `discont`, který za nás provede výše popsaný postup. My pouze specifikujeme seznam intervalů, na nichž není funkce spojitá nebo není definována (tj. intervaly, kde její graf nechceme vykreslovat).

```
//discont.asy
//verejne promenne pro nastaveni
pair[] discontIntervals;
int discontLastIndex = 0;

//funkce pouzitelna jako argument cond v graph()
bool3 discontCond(real x){
  if (discontLastIndex >= discontIntervals.length){
    //za poslednim intervalem nespojitosti
    return true;
  }

  if (x <= discontIntervals[discontLastIndex].x){
    //stale vlevo od pristiho intervalu nespojitosti
    return true;
  } else if (x >= discontIntervals[discontLastIndex].y) {
    //interval nespojitosti prekrocen
    ++discontLastIndex;
    return default;
  }

  //v okoli bodu nespojitosti
  return false;
}
```

Balík umístíme do stejné složky s hlavním souborem, příp. do složky s ostatními balíky programu Asymptote. V hlavním souboru balík načteme, nastavíme příslušný seznam intervalů `discontIntervals` a nakonec funkci `graph()` předáme jako argument funkci `discontCond` z našeho balíku. Seznam je implementován jako pole hodnot typu `pair`, kde jednotlivé složky značí krajní body daného intervalu (bod můžeme zadat jako interval s totožnými krajními body). V případě více grafů navíc před každým kreslením vynulujeme pozici aktuálního intervalu, a to příkazem `discontLastIndex = 0`. Ukažme si použití na funkci $y = \lfloor x \rfloor$.

```
import graph;
import discont;
```

```
size(200);

real f(real x){
  return floor(x);
}

//intervaly kolem hodnot -4,-3,...,4
real eps = 0.01;
for(int i=-4;i<=4;++i){
  discontIntervals.push((i-eps,i+eps));
}

draw(graph(f,-3.5,3.5,n=2000,discontCond),red);

path circle = scale(0.1)*unitcircle;
for(int i=-3;i<=3;++i){
  fill(shift((i,f(i)))*circle,red);
  filldraw(shift((i,f(i)-1))*circle,white,red);
}

xaxis("$x$",Arrow);
yaxis("$y$",Arrow);

label("$y=\lfloor x\rfloor$",(0,-1),2E);
```



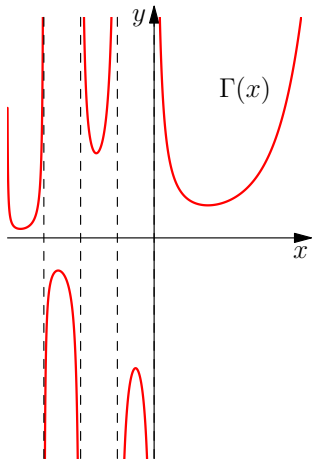
Obrázek 14: Graf funkce $y = \lfloor x \rfloor$

V seznamu jsme tedy jako intervaly nespojitosti zadali $(n-\epsilon, n+\epsilon)$ pro $n = -4, -3, \ldots, 4$ a $\epsilon = 0{,}01$. V těch se při sestrojování grafu nevyhodnocovaly funkční hodnoty a zároveň byly okolní větve správně odděleny. Pro použití u jiné funkce stačí pouze nahradit pole `discontIntervals` vlastními intervaly či body. Uživatel musí zajistit, aby intervaly byly disjunktní a seřazené dle krajních hodnot od nejmenší po největší. Následující příklad ukazuje použití na gamma funkci

$$y = \Gamma(x) = \int_0^\infty t^{x-1} \mathrm{e}^{-t} \mathrm{d}t, \tag{2}$$

která v bodech $-4, -3, \ldots, 0$ není definována.

```
import graph;
import discont; //vlastni balik

size(0,200,IgnoreAspect);

real f(real x){
  return 1/x;
}

//intervaly kolem hodnot -4,-3,-2,-1 a 0
real eps = 0.01;
for(int i=-4;i<=0;++i){
  discontIntervals.push((i-eps,i+eps));
}

draw(graph(gamma,-4,4,n=2000,discontCond),red+1.0);

ylimits(-6,6);
crop();
xaxis("$x$",Arrow);
yaxis("$y$",Arrow);

for(int i=-3;i<=0;++i){
  xequals(i,dashed);
}

label("$\Gamma(x)$",(3.5,gamma(3.5)),2NW);
```



Obrázek 15: Graf funkce $y = \Gamma(x)$

Křivka zadaná parametricky
Graf křivky $f(t) = \big(x(t), y(t)\big)$ vytvoříme následující verzí funkce `graph()`:

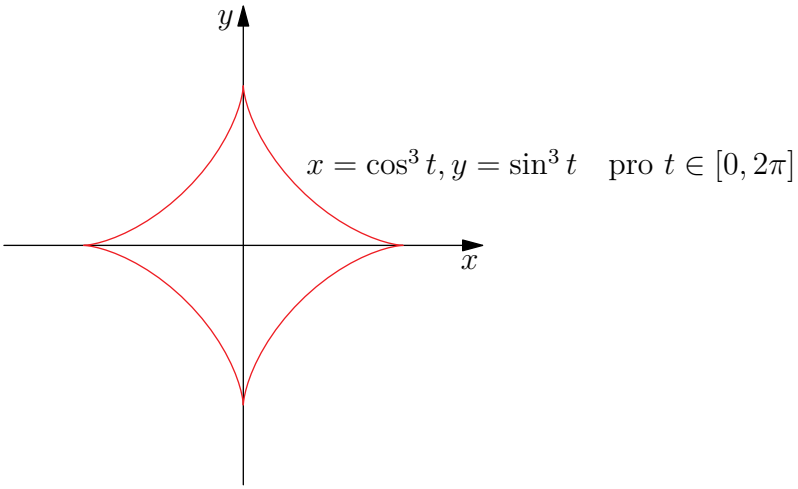
```
guide graph(picture pic=currentpicture, real x(real),
            real y(real), real a, real b,
            int n=ngraph, real T(real)=identity,
            interpolate join=operator --);
```

Od předešlé verze `graph()` se liší jen možností zadat dvě reálné funkce. Například graf asteroidy, zadané parametricky

$$x = \cos^3 t, \quad y = \sin^3 t \qquad \text{pro } t \in [0, 2\pi], \tag{3}$$

můžeme vykreslit takto:

```
import graph;

size(0,180);

real x(real t) {return cos(t)^3;}
real y(real t) {return sin(t)^3;}

label("$x=\cos ^3 t,\ y=\sin ^3 t
       \quad\mbox{pro $t\in [0,2\pi]$}$",
       (x(pi/4),y(pi/4)),NE);

draw(graph(x,y,0,2pi),red);
xaxis("$x$",-1.5,1.5,Arrow);
yaxis("$y$",-1.5,1.5,Arrow);
```



Obrázek 16: Parametrická křivka

Implicitní funkce

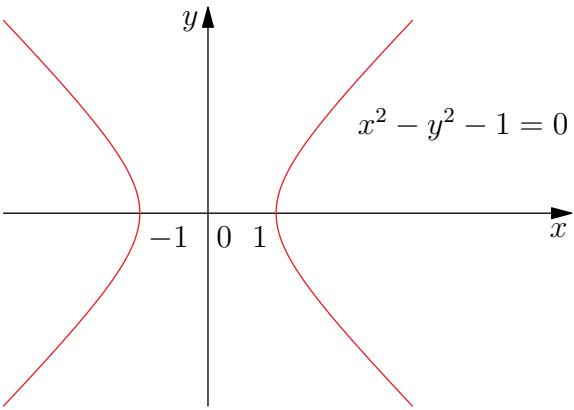
Graf funkce v implicitním tvaru $F(x, y) = 0$ můžeme vykreslit pomocí funkce `contour` ze stejnojmenného balíku. Ta slouží pro vytváření vrstevnic ve tvaru $F(x, y) = \mathrm{C}$.

```
guide[][] contour(real F(real, real), pair a, pair b,
                  real[] c, int nx=ngraph, int ny=nx,
                  interpolate join=operator --,
                  int subsample=1);
```

V poli `c` předáváme množinu hodnot pro C, pro které chceme vytvořit vrstevnice $F(x, y) = \mathrm{C}$, v rozsazích hodnot od `a` do `b`.

```
import contour;
import graph;

size(0,150);

real f(real x, real y){
  return x^2-y^2-1;
}

draw(contour(f,(-3,-3),(3,3),new real[]{0},join=operator..),red);

xaxis("$x$",Arrow);
yaxis("$y$",Arrow);
label("$x^2-y^2-1=0$",(2,sqrt(3)),2SE);
```

```
labelx(-1,SE);
labelx(0,SE);
labelx(1,SW);
```



Obrázek 17: Implicitní funkce

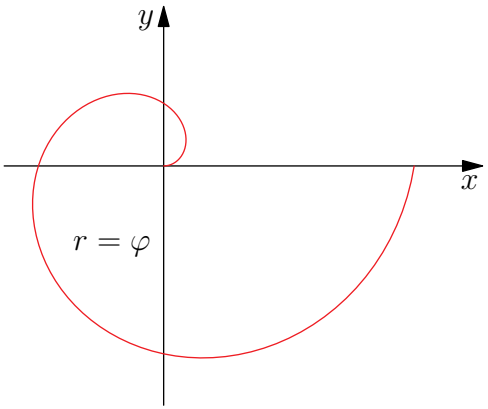
Křivka zadaná v polárních souřadnicích
Křivku v polárních souřadnicích, zadanou ve tvaru $r = r(\varphi)$ pro $\varphi \in [a, b]$, můžeme vytvořit pomocí funkce `polargraph` z balíku `graph`.

```
guide polargraph(picture pic=currentpicture,
                 real r(real), real a,
                 real b, int n=ngraph,
                 interpolate join=operator --);
```

Křivku zadanou v polárních souřadnicích jako $r = \varphi$ můžeme vytvořit například takto:

```
import graph;

size(0,150);

real r(real phi){
  return phi;
}

draw(polargraph(r,0,2pi),red);

xaxis("$x$",-4,8,Arrow);
yaxis("$y$",-6,4,Arrow);
```

```
pair polarpoint(real r, real phi){
  return (r*cos(phi),r*sin(phi));
}
label("$r=\varphi$",polarpoint(5pi/4,r(5pi/4)),3NE);
```



Obrázek 18: Křivka v polárních souřadnicích

Osy

Pro vykreslení os nabízí program Asymptote několik funkcí. Především jsou to `xaxis()` a `yaxis()`.

```
void xaxis(picture pic=currentpicture, Label L="",
           axis axis=YZero, real xmin=-infinity,
           real xmax=infinity, pen p=currentpen,
           ticks ticks=NoTicks, arrowbar arrow=None,
           bool above=false);
```

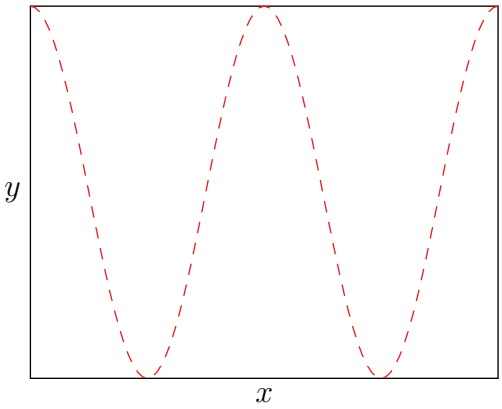
```
void yaxis(picture pic=currentpicture, Label L="",
           axis axis=XZero, real ymin=-infinity,
           real ymax=infinity, pen p=currentpen,
           ticks ticks=NoTicks, arrowbar arrow=None,
           bool above=false, bool autorotate=true);
```

Jak je vidět z deklarací, nemají tyto funkce žádné povinné argumenty. Pro základní osy bez popisků a šipek je tak možné zavolat pouze:

```
xaxis();
yaxis();
```

Popišme zde zbylé argumenty pro osu x (osa y analogicky). Specifikovat můžeme popisek `L`, rozsah vykreslení `xmin` a `xmax`. Dále můžeme nastavit styl šipky `arrow`, pero `p`, argument `above` určující, zda se má osa vykreslit nad již existující objekty na plátně. Argument `axis` určuje umístění osy. Implicitně se osa x vykreslí na přímce $y = 0$. Pomocí tohoto argumentu je možné nechat vykreslit osu nahoře anebo dole na plátně. Slouží k tomu hodnoty `Bottom`, `Top` a `BottomTop` (pro osu y jsou to `Left`, `Right` a `LeftRight`). Běžné použití os je vidět na obrázku 16. Zde si ukažme umístění os do krajů.

```
import graph;

size(0,150);

draw(graph(new real(real x){return 5*cos(x);},-2pi,2pi),dashed+red)
    ;

xaxis("$x$",axis=BottomTop);
yaxis("$y$",axis=LeftRight);
```



Obrázek 19: Umístění os

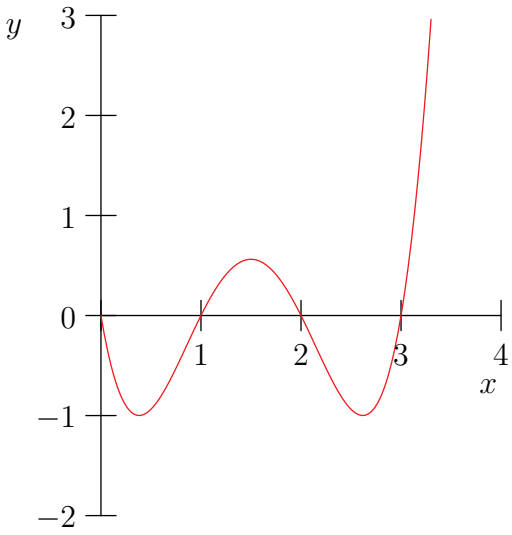
Posledním argumentem je `ticks`, tj. čárkování. Pro základní použití máme několik předdefinovaných hodnot: `LeftTicks`, `RightTicks` a `Ticks`. V takovém případě čárky budou nalevo, napravo anebo obojí. Širší možnosti nabízejí stejnojmenné funkce `LeftTicks()`, `RightTicks()` a `Ticks()`. Význam argumentů je u všech analogický, popišme si proto jen první z nich.

```
ticks LeftTicks(Label format="", ticklabel ticklabel=null,
                bool beginlabel=true, bool endlabel=true,
                int N=0, int n=0, real Step=0, real step=0,
```

```
                bool begin=true, bool end=true,
                tickmodifier modify=None, real Size=0,
                real size=0, bool extend=false,
                pen pTick=nullpen, pen ptick=nullpen);
```

Použitím `N` uvádíme, na kolik intervalů se osa rozdělí velkými čárkami. S `n` je to podobné, udává počet intervalů mezi dvěma velkými čárkami, které budou odděleny malými čárkami. Jiný způsob je specifikovat vzdálenosti pomocí `Step` a `step` mezi velkými, resp. malými čárkami. Parametry `Size` a `size` určují velikost čárek.

```
import graph;

size(0,200);

real f(real x){
  return x*(x-1)*(x-2)*(x-3);
}

draw(graph(f,0,3.3),red);

xaxis("$x$",0,4,Ticks(beginlabel=false,n=1));
yaxis("$y$",-2,3,Ticks(n=1));
```



Obrázek 20: Čárkování na osách

Popisky

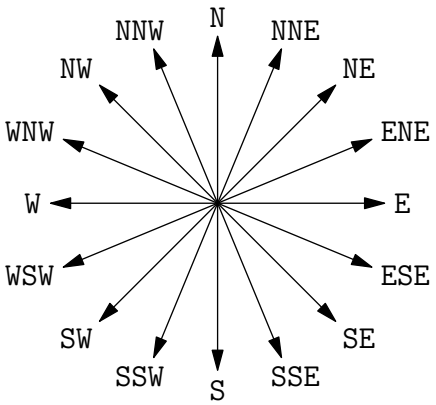
Pro vložení popisku na zadanou souřadnici můžeme použít funkci `label()`, pro vykreslení podél os funkce `labelx()` a `labely()`.

```
void label(picture pic=currentpicture, Label L,
           pair position, align align=NoAlign,
           pen p=nullpen, filltype filltype=NoFill);
```

```
void labelx(picture pic=currentpicture, Label L="",
            real x, align align=S, string format="",
            pen p=nullpen);
```

```
void labely(picture pic=currentpicture, Label L="",
            real y, align align=W, string format="",
            pen p=nullpen);
```

Rozeberme si nejprve funkci `label()`. Ta má jako povinné argumenty pouze popisek `L` (který můžeme zadat jako typ `string`) a souřadnice pro vykreslení `position`. Důležitý je také argument `align`, který určuje, v jakém směru a vzdálenosti od zadané pozice má být popisek umístěn. Většinou se používají konstanty odpovídající světovým stranám v angličtině (viz obr. 21), příp. jejich násobky.



Obrázek 21: Přehled předdefinovaných směrů

Pro vykreslování popisků podél os využitím `labelx()`, příp. `labely()` stačí zadat směr zarovnání a pozici na dané ose. Pokud například na ose y budeme chtít v bodě $y = 1$ vykreslit popisek 1, můžeme použít jednu z variant:

```
labely(1,E);
label("$1$",(0,1),E);
```

Pro další příklady odkazujeme na ukázky v předchozích částech.

Práce ve 3D

V této části uvedeme postupy pro generování 3D grafiky. Většinou se bude jednat o obdobu toho, co již bylo napsáno v předchozí části.

Popišme stručně, jak program Asymptote pracuje při generování 3D grafiky pro PDF dokumenty. Nejprve se vygeneruje 3D model ve formátu PRC[2]. Pokud pracujeme v LaTeXu, vygeneruje se také náhledový obrázek. Ten se při další kompilaci LaTeXu automaticky vloží do dokumentu. Scéna se pak aktivuje až po kliknutí na tento obrázek. Někdy však můžeme chtít dokument bez interaktivního 3D modelu (kvůli velikosti, nebo protože tam není potřebný či je přímo nežádoucí). V takovém případě stačí přidat na začátek `asy` bloku pro daný obrázek příkaz:

```
settings.prc = false;
```

Jindy naopak můžeme chtít, aby se v dokumentu nacházel jen samotný model, tj. bez náhledového obrázku. V takovém případě stačí do kódu obrázku na začátek přidat:

```
settings.embed = false;
```

Ve 3D se také neobejdeme bez balíku `three`, který definuje zobecněné typy známé z 2D. Proto se téměř v celé této kapitole předpokládá jeho načtení:

```
import three;
```

Kamera

Nastavení kamery se provádí pomocí promítání typu `projection`. Promítání se mění pomocí proměnné `currentprojection`, kterou můžeme nastavit na rovnoběžné nebo perspektivní promítání.

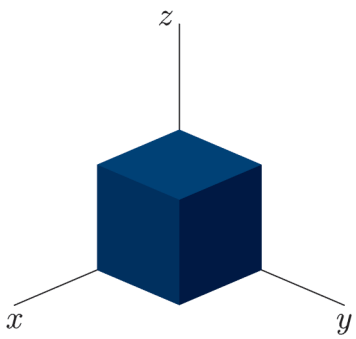
```
orthographic(triple camera, triple up=Z,
             triple target=0, real zoom=1,
             pair viewportshift=0, bool showtarget=true,
             bool center=false);
```

```
perspective(triple camera, triple up=Z,
            triple target=0, real zoom=1,
            real angle=0, pair viewportshift=0,
            bool showtarget=true, bool autoadjust=true,
            bool center=autoadjust);
```

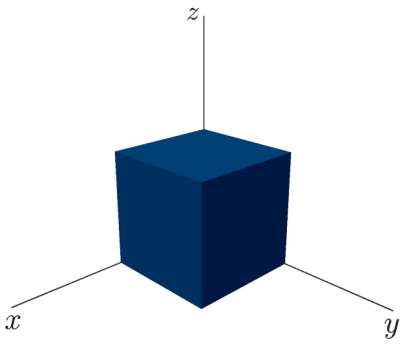
Kamera bude umístěna do bodu (příp. ve směru bodu pro rovnoběžné promítání) `target+camera`, natočena směrem k `target`. Vektor `up` na obrazovce určuje směr

[2] Viz Specifikace formátu PRC (2008).

nahoru (implicitně `(0,0,1)`). Oběma těmto funkcím můžeme zadat jednotlivé složky vektoru `camera` zvlášť, výsledek je ale stejný. Pokud neuvedeme žádné promítání, použije se `perspective(5,4,2)`.



Obrázek 22: `orthographic(3,3,2)`



Obrázek 23: `perspective(3,3,2)`

Tím možnosti promítání nekončí. K dispozici je dále kosoúhlé promítání `obliqueX`, `obliqueY`, `obliqueZ`, případně si uživatel může definovat vlastní. Pro vyčerpávající popis viz oficiální stránky Asymptote (2004).

Světla

Nastavení světel je reprezentováno typem `light`, který je zadáván ve vykreslovacích příkazech. Pokud jej nezadáme, využije se `currentlight`, které je implicitně nastaveno na `Headlamp`. Další předdefinovaná světla jsou `Viewport`, `White` a pro scénu bez světel `nolight` (přesněji pro scénu bez stínů). Pro vlastní osvětlení můžeme využít funkci `light`:

```
light light(pen diffuse=white, pen ambient=black,
            pen specular=diffuse, pen background=nullpen,
            bool viewport=false, real x, real y, real z);
```

To umožňuje nastavit pera (především barvy) `diffuse`, `ambient`, `specular` a pero pro pozadí – `background`. Argument `viewport` uvádí, zda se má zdroj světla pohybovat společně s pohybem kamery. Zbylé argumenty určují souřadnice cíle světla. Pokud chceme ve scéně používat jedno nastavení světel, například `nolight`, stačí na začátku kódu uvést příkaz:

```
currentlight = nolight;
```

Obrázek 24: `nolight`

Obrázek 25: `White`

Obrázek 26: `Headlamp`

Obrázek 27: `Viewport`

Pera

Pro většinu kreslení ve 3D se využívá typ `pen`, známý ze 2D (viz část Pera). Pro některé případy však balík `three_light` (importován balíkem `three`) zavádí rozšířený typ `material`. Ten uchovává především 4 pera/barvy `diffuse`, `ambient`, `emissive` a `specular` (uloženo v podobě per, umožňuje definovat i vzor). Dále uchovává (ne)průhlednost `opacity` a lesklost `shininess`. Materiál těchto vlastností vytvoříme pomocí:

```
material material(pen diffusepen=black, pen ambientpen=black,
                  pen emissivepen=black,
                  pen specularpen=mediumgray,
                  real opacity=opacity(diffusepen),
                  real shininess=defaultshininess);
```

Většinou si vystačíme s materiálem indukovaným z pera typu `pen`. Takový materiál pak převezme zadané pero pro popis `diffuse` barvy a zbylé hodnoty doplní implicitními. Provedeme to například takto:

```
material m = black; //black je typu pen
```

Případně předáme typ `pen` přímo v místě, kde se zadává materiál, a převod se provede automaticky.

Zastavme se zde ještě u balíku `palette`. Ten nám umožňuje vykreslit oblasti či plochy se zadanou (nejen) barevnou paletou. Můžeme tak například zvýraznit výškové rozdíly v grafu, jak ilustruje následující ukázka. Více informací o tomto balíku viz oficiální stránky Asymptote (2004).

```
import graph3;
import palette;

size(250,IgnoreAspect);

currentprojection=orthographic(2,4,1);
currentlight=Viewport;

real f(pair z) {
  real x=z.x,y=z.y;
  return sin(x)*sin(y);
}

surface s=surface(f,(0,0),(2pi,2pi),40,Spline);
s.colors(palette(s.map(zpart),Rainbow()));

draw(s);
```

Křivky

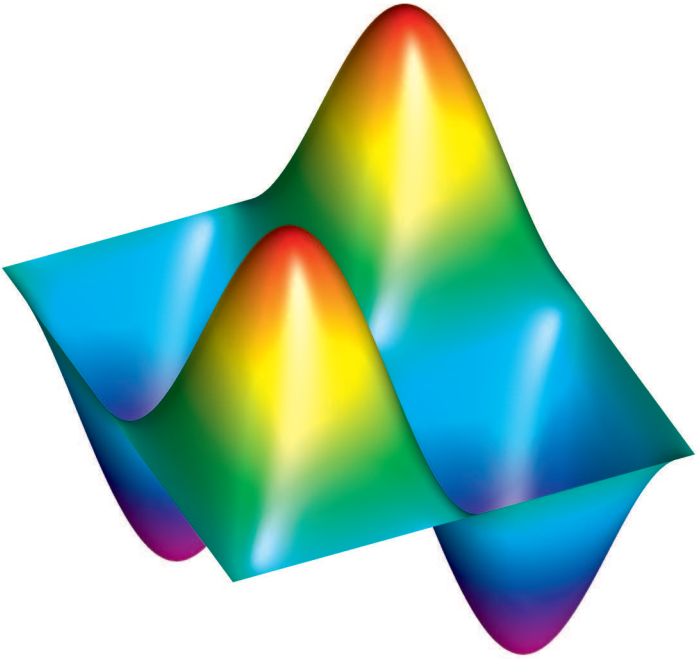
Stejně jako ve 2D je i ve 3D základním kamenem všeho křivka. Balík `three` definuje typy `guide3` a `path3`, se kterými se pracuje téměř identicky jako s jejich 2D verzemi `guide` a `path`.

Stejně jako tomu bylo ve 2D, i zde můžeme při vykreslování dané křivky specifikovat typ šipky, případně příčky. Ty zadáváme při vykreslování pomocí `draw()` nebo při používání nějaké vyšší funkce jako například `graph()` (viz Grafy funkcí). Použitelné hodnoty jsou `BeginArrow3`, `MidArrow3`, `EndArrow3`, `Arrow3` a `Arrows3` pro obyčejné šipky, dále `BeginArcArrow3`, `EndArcArrow3`, `ArcArrow3`, `MidArcArrow3` a `ArcArrows3` pro šipky s kratším hrotem. Pro příčky slouží `None`, `Blank`, `BeginBar3`, `EndBar3` a `Bar3`.

Více o křivkách vytvořených ze zadaného předpisu viz část Grafy funkcí.

Transformace

Transformace ve 3D se velice podobají těm ve 2D. Jejich typ je `transform3`. Uveďme si zde jen v krátkosti seznam nejdůležitějších z nich.



Obrázek 28: Využití balíku `palette`

```
transform3 shift(triple v);
```

```
transform3 xscale3(real x);
transform3 yscale3(real y);
transform3 zscale3(real z);
transform3 scale3(real s);
transform3 scale(real x, real y, real z);
```

```
transform3 rotate(real angle, triple v);
transform3 rotate(real angle, triple u, triple v);
```

```
transform3 reflect(triple u, triple v, triple w);
```

Použití lze vidět, mimo jiné, na obr. 39 a 40.

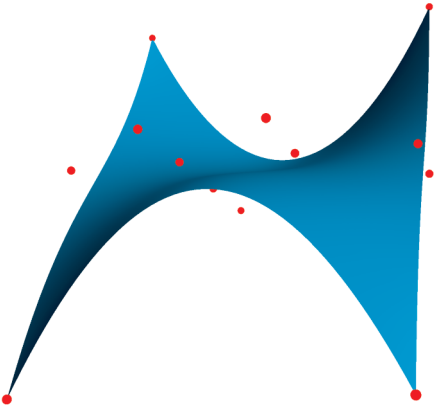
Plochy

Plochy jsou v balíku `three` reprezentovány typem `surface`. Zde jsou uloženy jako pole Bézierových plátů (typ `patch`). Funkce v balících pro práci ve 3D často produkují typ `surface`, případně takový typ, který se na typ `surface` převádí, když je to potřeba (viz část Rotační tělesa). Následující funkce slouží pro vytvoření plochy výčtem kontrolních bodů.

```
surface surface(triple[][][] P,
                triple[][] normals=new triple[][],
                pen[][] colors=new pen[][],
                bool3 planar=default);
```

Tato varianta je vhodná pro případ, kdy chceme zadat plochu výčtem všech kontrolních bodů všech Bézierových plátů. Trojrozměrné pole `P` se očekává jako pole plátů, kde je každý z plátů očekáván jako pole čtyř čtveřic bodů. Dále lze pro rohové body nastavit normálové vektory `normals` a také barvy `colors`. Následuje ukázka vytvoření jednoho Bézierova plátu zadáním kontrolních bodů.

```
import three;

size(0,150);

currentprojection = perspective(5,2,-1);

triple[][][] P={
  {
    {(0.0, 0.0, 1.0),(0.5, 0.0, 0.0),
     (1.0, 0.0, 0.0),(1.5, 0.0, -1.0),},
    {(0.0, 0.5, 0.0),(0.5, 0.5, 0.0),
     (1.0, 0.5, 0.0),(1.5, 0.5,  0.0),},
    {(0.0, 1.0, 0.0),(0.5, 1.0, 0.0),
     (1.0, 1.0, 0.0),(1.5, 1.0,  0.0),},
    {(0.0, 1.5, 1.0),(0.5, 1.5, 0.0),
     (1.0, 1.5, 0.0),(1.5, 1.5, -1.0),}
  }
};

draw(surface(P),cyan);

for(int i=0;i<4;++i)
  for(int j=0;j<4;++j)
    dot(P[0][i][j],red);
```



Obrázek 29: Bézierův plát zadán šestnácti kontrolními body

Plochy lze dále vytvářet generováním grafu funkce, vytažením ze zadaných křivek či rotací křivek, jak ukážeme v následujících částech.

Grafy funkcí

Při kreslení grafů funkcí v prostoru využijeme balík `graph3`[3], jenž je obdobou balíku `graph` pro kreslení grafů funkcí v rovině.

Funkce $f(x, y)$

Graf funkce dvou proměnných vykreslíme tak, že pomocí funkce z balíku `graph3` vygenerujeme Bézierovu plochu typu `surface`, kterou posléze vykreslíme příkazem `draw`. Použijeme k tomu následující funkci:

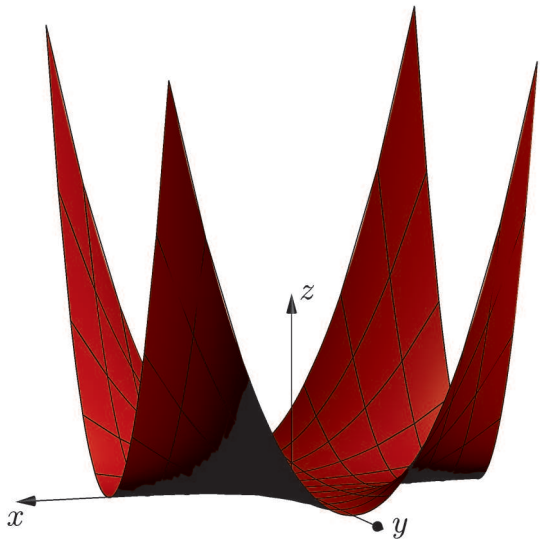
```
surface surface(real f(pair z), pair a,
                pair b, int nx=nmesh, int ny=nx,
                splinetype xsplinetype,
                splinetype ysplinetype=xsplinetype,
                bool cond(pair z)=null);
```

Povinné argumenty jsou zde funkce dvou proměnných `f` a meze `a` (dolní meze) a `b` (horní meze). Celočíselné argumenty `nx` a `ny` určují dělení ve směru os x a y. Ukažme si použití na příkladu funkce $f(x, y) = |xy|^2$:

```
import graph3;

size(200,200,keepAspect=false);
```

[3]Balík `graph3` v sobě již zahrnuje načítání balíku `three`, a ten tak není nutno načítat zvlášť.

```
currentprojection=orthographic(3,9,5);

real f(pair z){
    real x=z.x,y=z.y;
  return abs(x*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);
```



Obrázek 30: Funkce $f(x, y) = |xy|^2$

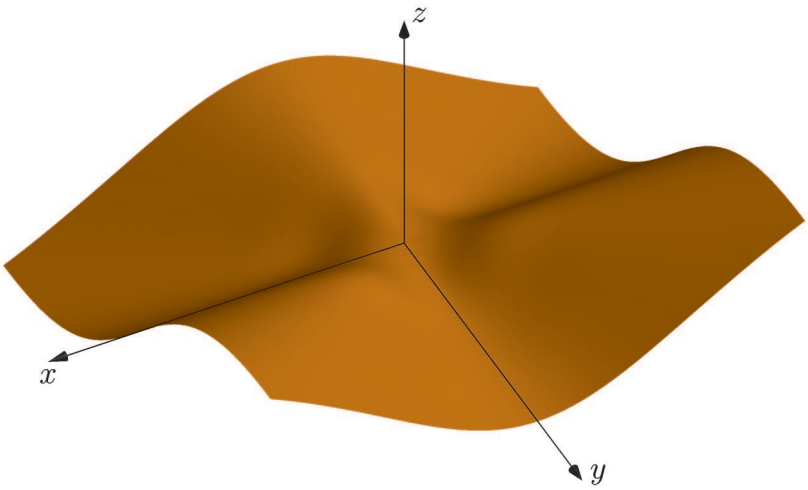
Nespojité funkce

Pro nespojité funkce je v současné verzi programu Asymptote 2.14 pouze mizivá podpora a výsledky nevypadají příliš povedeně. Existuje sice stejný argument `cond`, jako byl popsán v části Nespojitá funkce na straně 30, avšak graf je zpravidla velmi nekvalitní („zubatý“).

Proto doporučujeme použít vhodné parametrické vyjádření plochy či rozdělení plochy na části. Jedná-li se o odstranitelnou nespojitost, je možné také rozšíření definice funkce, jak ukazuje druhý řádek těla funkce v následujícím příkladu grafu

funkce $f(x,y) = \frac{x^2y}{x^2+y^2}$.

```
import graph3;

size(300);
currentprojection=orthographic(3,6,6);

real f(pair z){
  real x=z.x,y=z.y;
  if (x==0.0 && y==0.0) return 0;
  return (x^2*y)/(x^2+y^2);
}

draw(surface(f,(-3,-3),(3,3),xsplinetype=Spline),orange);

xaxis3("$x$",0,4,Arrow3);
yaxis3("$y$",0,4,Arrow3);
zaxis3("$z$",0,3,Arrow3);
```



Obrázek 31: Funkce $f(x,y) = \frac{x^2y}{x^2+y^2}$

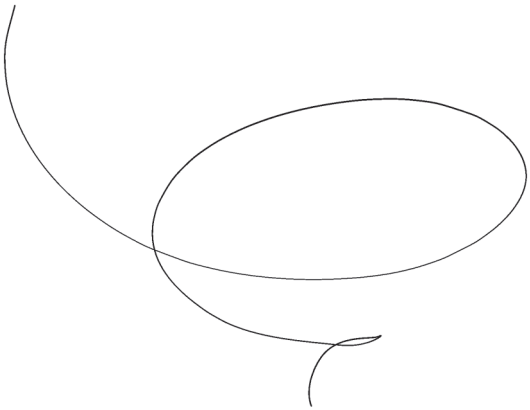
Parametrická křivka

Pro parametrickou křivku $f(t) = \big(x(t), y(t), z(t)\big)$ máme následující variantu funkce `graph()`.

```
guide3 graph(picture pic=currentpicture, real x(real),
             real y(real), real z(real), real a, real b,
             int n=ngraph, interpolate3 join=operator --);
```

Předáváme tři funkce `x()`, `y()` a `z()`. Ty musí mít jeden argument typu `real` a vracet hodnotu typu `real` (tj. odpovídající reálné funkce jedné reálné proměnné). Dále zadáme meze intervalu `a` a `b`, případně dělení `n` a také typ interpolace `join`.

```
import graph3;

size(0,150);

triple f(real t){
  return (t*cos(t),t*sin(t),t);
}

draw(graph(f,0,4pi,join=operator .. ,n=10));
```



Obrázek 32: Prostorová křivka

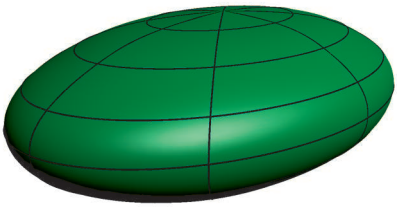
Parametrická plocha
Plochu zadanou ve tvaru $F(u, v)$ můžeme vytvořit následující variantou funkce `surface()` z balíku `graph3`:

```
surface surface(triple f(pair z), pair a, pair b,
                int nu=nmesh, int nv=nu,
                splinetype[] usplinetype,
                splinetype[] vsplinetype=Spline,
                bool cond(pair z)=null);
```

Zde dvojice hodnot `a`, příp. `b` určuje dolní, příp. horní meze pro oba parametry. Jemnost dělení lze nastavit pomocí `nu` a `nv`. Typ interpolace nastavíme pomocí `usplinetype`, příp. `vsplinetype`. U těchto argumentů je dobré si všimnout, že jsou to pole typu `splinetype`, přesněji je očekáváno tříprvkové pole. Hodnota

`Spline` je však přetížena a lze ji zde použít (hodnoty `Straight`, `operator..` ani `operator-` však nelze). Pokud budeme chtít striktně lineární interpolaci, můžeme použít podobnou verzi `surface`, která se liší jen v tom, že nevyžaduje `usplinetype` ani `vsplinetype`.

```
import graph3;

size(150,0);

real a=3,b=2,c=1;
triple F(pair z){
  real phi=z.x, theta=z.y;
  return (a*cos(phi)*cos(theta),
          b*cos(phi)*sin(theta),
          c*sin(phi));
}

draw(surface(F,(-pi/2,0),(pi/2,2pi),nu=8,Spline),green,meshpen=
    black+0.5);
```



Obrázek 33: Plocha zadaná parametricky

Implicitní funkce

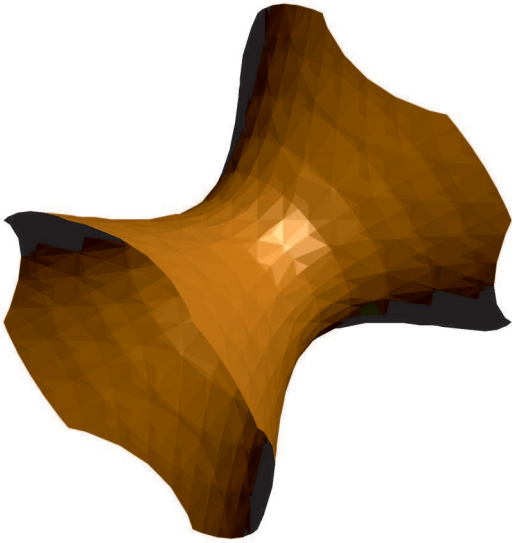
Plochu zadanou v implicitním tvaru $F(x, y, z) = 0$ můžeme vykreslit funkcí `countour3` ze stejnojmenného balíku.

```
vertex[][] contour3(real F(real, real, real),
                    triple a, triple b,
                    int nx=nmesh, int ny=nx, int nz=nx,
                    projection P=currentprojection);
```

Body `a` a `b` jsou rohy příslušného kvádru, ze kterého se při sestrojování grafu volí hodnoty x, y a z.

```
import graph3;
import contour3;
```

```
size(0,200);

currentprojection = orthographic(1,1,2);

real F(real x, real y, real z){
  return -x^2+y^2+z^2-1;
}

draw(surface(contour3(F,(-2,-2,-2),(2,2,2),nx=10)),orange);
```



Obrázek 34: Plocha zadaná implicitně

Pro lepší výsledky se však doporučuje použít parametrické vyjádření plochy.

Sférické a cylindrické souřadnice
Pro kreslení křivek zadaných ve sférických souřadnicích máme k dispozici funkci `polargraph` z balíku `graph3`. Očekávaný tvar křivky je

$$\theta = f(t),\ \varphi = g(t),\ r = h(\theta, \varphi), \quad t \in [0, 1]$$

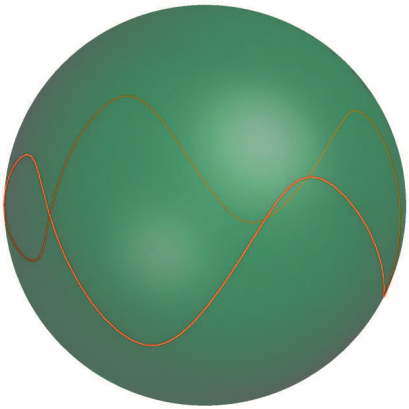
a příslušná funkce pro vytvoření

```
guide3 polargraph(real r(real,real), real theta(real),
                  real phi(real), int n=ngraph,
                  interpolate3 join=operator --);
```

Následuje příklad použití na křivce dané předpisem

$$\theta = \frac{1}{8}\sin(8\pi t)\pi + \frac{\pi}{2}, \quad \varphi = 2\pi t, \quad r = 1, \quad t \in [0,1].$$

```
import solids;
import graph3;

size(0,150);

real phi(real t){return t*2pi;}
real theta(real t){return 1/8*sin(4*t*2pi)*pi+pi/2;}
real r(real theta, real phi){return 1;}

draw(surface(sphere(1)),green+opacity(0.5));
draw(polargraph(r,theta,phi),red);
```

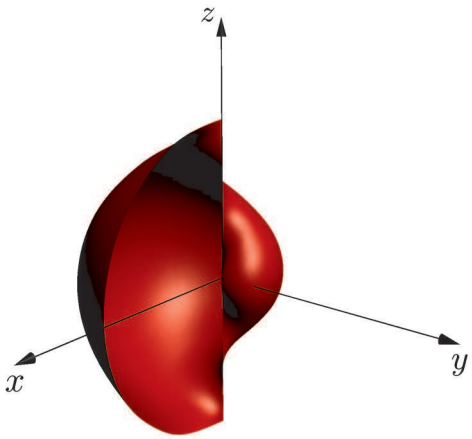


Obrázek 35: Křivka ve sférických souřadnicích

Balíky Asymptote v současné verzi nenabízejí funkce pro vytváření křivek v cylindrických souřadnicích ani pro plochy ve sférických či cylindrických souřadnicích. Není však těžké využít parametrického tvaru v kartézských souřadnicích, a pomocí nich vytvořit vlastní funkce. Následující dva příklady ilustrují takové dvě funkce, `sphericalgraph` a `cylindricalgraph`. První příklad představuje plochu danou předpisem $r(\theta, \varphi) = \frac{1}{5}\varphi$ ve sférických souřadnicích.

```
import graph3;

size(0,160);
```

```
surface sphericalgraph(real f(real, real), pair a, pair b,
 int nu=nmesh, int nv=nu, splinetype[] usplinetype=Spline,
 splinetype[] vsplinetype=Spline, bool cond(pair z)=null){
  triple F(pair z){
    real phi=z.x,theta=z.y;
    real r = f(phi,theta);
    return (r*cos(phi)*sin(theta),
            r*sin(phi)*sin(theta),
            r*cos(theta));
  }
  return surface(F,a,b,nu,nv,usplinetype,vsplinetype,cond);
}

real r(real phi,real theta){ return phi/5;}
draw(sphericalgraph(r,(0,0),(2pi,pi),nu=10),red);

xaxis3("$x$",0,2,Arrow3);
yaxis3("$y$",0,2,Arrow3);
zaxis3("$z$",0,2,Arrow3);
```

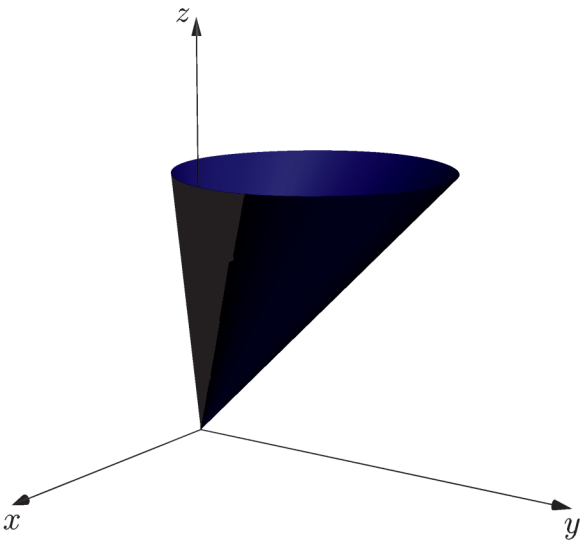


Obrázek 36: Plocha ve sférických souřadnicích

Druhá ukázka představuje plochu danou předpisem $r(\varphi, z) = z\sin(\varphi)$ v cylindrických souřadnicích.

```
import graph3;

size(0,200);

surface cylindricalgraph(real f(real, real), pair a, pair b,
   int nu=nmesh, int nv=nu, splinetype[] usplinetype=Spline,
```

```
   splinetype[] vsplinetype=Spline, bool cond(pair z)=null){
  triple F(pair t){
    real phi=t.x,z=t.y;
    real r = f(phi,z);
    return (r*cos(phi),r*sin(phi),z);
  }
  return surface(F,a,b,nu,nv,usplinetype,vsplinetype,cond);
}

real r(real phi,real z){ return sin(phi)*z;}
draw(cylindricalgraph(r,(0,0),(2pi,1),nu=8,nv=4),blue);

xaxis3("$x$",0,1,Arrow3);
yaxis3("$y$",0,1.5,Arrow3);
zaxis3("$z$",0,1.5,Arrow3);
```



Obrázek 37: Plocha v cylindrických souřadnicích

Osy

Pro vykreslení os ve 3D použijeme funkce `xaxis3()`, `yaxis3()` a `zaxis3()` z balíku `graph3()`. Formát jejich hlavičky je analogický, uveďme si jej pro `xaxis3()`.

```
void xaxis3(picture pic=currentpicture, Label L="",
            axis axis=YZZero, real xmin=-infinity,
            real xmax=infinity, pen p=currentpen,
```

```
            ticks3 ticks=NoTicks3, arrowbar3 arrow=None,
            bool above=false);
```

Funkce se podobá funkci `xaxis()` pro 2D případ, jen `axis`, `ticks` a `arrow` používají své trojrozměrné varianty. Šipka `arrow` může nabývat jedné z hodnot uvedených v části Křivky. Popišme si zbylé argumenty pro určení polohy os a čárkování.

Argument `axis` může nabývat hodnoty `YZZero` a potom osa x leží na přímce $y = 0$, $z = 0$ (jde o implicitní hodnotu). Analogicky, pro osy y a z máme hodnoty `XZZero` a `XYZero`. Pro specifikaci vlastního umístění můžeme použít funkce `YZEquals()`, `XZEquals()` nebo `XYEquals()`.

```
axis YZEquals(real y, real z);
axis XZEquals(real x, real z);
axis XYEquals(real x, real y);
```

Případné použití by pak mohlo vypadat třeba takto:

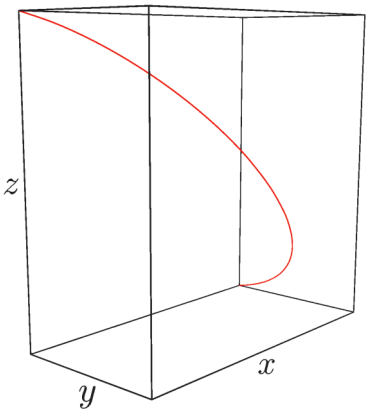
```
//osa x bude lezet na primce y=2, z=3
xaxis3("$x$",YZEquals(2,3));
```

Zbývajícím typem umístění je `Bounds`, které umístí osu do všech čtyř pozic podle rozměrů aktuálního modelu. Jedná se o trojrozměrnou analogii k `BottomTop` či `TopRight`.

```
import graph3;

size(0,150);

draw((0,0,0)..(1,1,1)..(2,0,2),red);

xaxis3("$x$",Bounds);
yaxis3("$y$",Bounds);
zaxis3("$z$",Bounds);
```

Zbylý argument `ticks` určuje styl čárkování na osách. Přednastavené hodnoty jsou `NoTicks3`, `InTicks`, `OutTicks` a `InOutTicks`. Pro pokročilejší čárkování můžeme volat stejnojmenné funkce a specifikovat například počet čárek, krokování apod. Uveďme si hlavičku alespoň jedné z nich.

```
ticks3 InOutTicks(Label format="", ticklabel ticklabel=null,
                  bool beginlabel=true, bool endlabel=true,
                  int N=0, int n=0, real Step=0, real step=0,
                  bool begin=true, bool end=true,
                  tickmodifier modify=None, real Size=0,
                  real size=0, bool extend=false,
                  pen pTick=nullpen, pen ptick=nullpen);
```



Obrázek 38: Osy s využitím `Bounds`

Význam argumentů je zde totožný jako ve 2D. Důležité jsou zde především `N`, `n` (počet dílků oddělených velkými, příp. malými čárkami), `Step`, `step` (vzdálenost mezi velkými, příp. malými čárkami), `Size` a `size` (velikost velkých, příp. malých čárek).

Popisky

Obdobně, jako tomu bylo ve 2D, také ve 3D můžeme sázet LaTeXové popisky. Využijeme k tomu:

```
void label(picture pic=currentpicture, Label L,
           triple position, align align=NoAlign,
           pen p=currentpen, light light=nolight,
           string name="", render render=defaultrender,
           interaction interaction =
           settings.autobillboard ? Billboard : Embedded);
```

Význam je analogický jako v odpovídající 2D variantě, popisek `L` se vysází na pozici `position`. Za povšimnutí stojí argument `interaction` určující, zda se má popisek natáčet s pohybem kamery (hodnota `Billboard`, implicitní chování), nebo zda se má natočit staticky ve směru zadané projekce (hodnota `Embedded`).

Po načtení balíku `graph3` máme k dispozici i varianty pro snazší sázení popisků kolem os.

```
void labelx(picture pic=currentpicture, Label L="", triple v,
            align align=-Y, string format="", pen p=nullpen);
```

Analogicky `labely()` a `labelz()`.

Rotační tělesa

S využitím typu `revolution` z balíku `solids` můžeme vytvářet plochy vzniklé rotací křivky kolem zadaného vektoru. Typ `revolution` v sobě obsahuje právě tyto údaje, tj. křivku, která se otáčí, a vektor, kolem kterého se otáčí. Abychom získali plochu, musíme ho převést na typ `surface`. Podívejme se, jak vytváříme proměnné typu `revolution`.

```
revolution cylinder(triple c=0, real r,
                    real h, triple axis=Z);
```

```
revolution cone(triple c=0, real r,
                real h, triple axis=Z,
                int n=nslice);
```

```
revolution sphere(triple c=0, real r, int n=nslice);
```

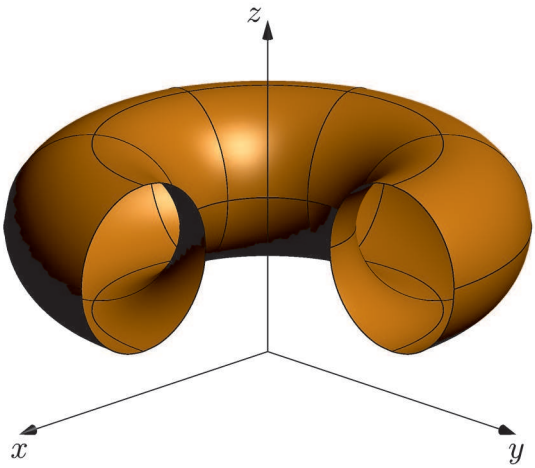
V případě válce (`cylinder`) zadáváme střed podstavy `c`, poloměr podstavy `r`, výšku `h` a směr osy válce `axis`. Podobně je to s kuželem (`cone`) a koulí (`sphere`), kde můžeme navíc zadat jemnost dělení stran `n`.

S těmito základními tvary bychom si však těžko vystačili. Ukažme si proto, jak vytvořit plochu vzniklou rotací obecné křivky kolem zadané osy.

```
revolution revolution(triple c=0, path3 g, triple axis=Z,
                      real angle1=0, real angle2=360);
```

Funkci předáme střed otáčení `c` (bod, z nějž povede osa otáčení), křivku `g`, osu otáčení `axis` (implicitně ve směru osy z, vektor však může být libovolný) a rozsah úhlů otáčení `angle1` a `angle2` ve stupních.

```
import solids;

size(0,175);
currentprojection=orthographic(2,2,1);

path3 p = rotate(90,(1,0,0))*shift(2,1.5,0)*unitcircle3;

revolution r=revolution(p,90,360);

draw(surface(r,n=5),orange,meshpen=black+0.5);

xaxis3("$x$",0,4,Arrow3);
yaxis3("$y$",0,4,Arrow3);
zaxis3("$z$",0,4,Arrow3);
```



Obrázek 39: Plocha vzniklá rotací kružnice

Ukažme si ještě, jak vytvořit plochu vzniklou rotací funkce $z = f(x)$.

```
revolution revolution(triple c=0, real f(real x),
                      real a, real b, int n=ngraph,
                      interpolate3 join=operator --,
                      triple axis=Z, real angle1=0,
                      real angle2=360);
```

Zde `c` je střed otáčení, `f` je příslušná funkce v mezích `a` až `b` s jemností dělení `n`. Dále je možné specifikovat typ interpolace argumentem `join`, osu otáčení `axis` a meze úhlu rotace `angle1` a `angle2`.

Vytažené plochy

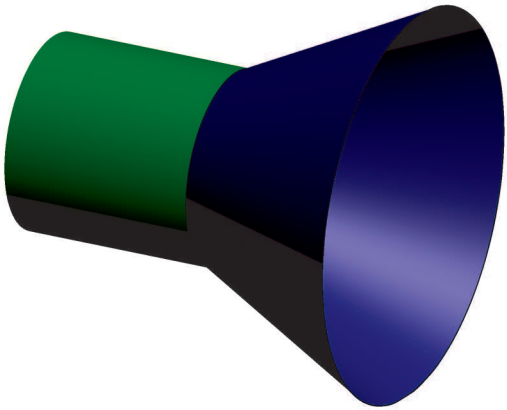
Dalším užitečným způsobem, jak vytvářet plochy, je vytažení vhodných křivek v určitém směru, případně mezi dvěmi křivkami. Balík `three` pro tento účel definuje několik verzí funkce `extrude()`. Uveďme si dvě základní varianty.

```
surface extrude(path3 p, triple axis=Z);
surface extrude(path3 p, path3 q);
```

V první verzi je křivka `p` vytažena ve směru `axis` (o délku vektoru `axis`). Druhá verze vytváří přímkovou plochu mezi dvěma zadanými křivkami. Pro ilustraci opět uvádíme příklad.

```
import three;
```

```
size(0,150);
currentprojection = orthographic(1,-2,1);

//dve kruznice
path3 p = rotate(90,(0,1,0))*unitcircle3;
path3 q = shift((2,0,0))*scale3(2)*p;

draw(p);
draw(q);

draw(extrude(p,(-2,0,0)),green);
draw(extrude(p,q),blue);
```

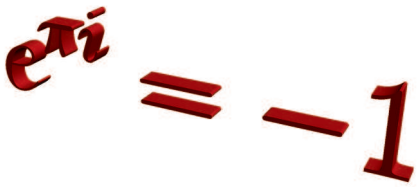


Obrázek 40: Vytažené plochy

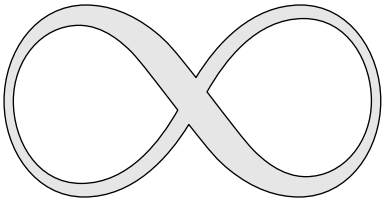
Algoritmy pro 3D reprezentaci rovinných oblastí

Jednou z vlastností programu Asymptote odlišujících ho od podobných programů je schopnost převádět rovinné systémy křivek (jakými je například font) do reprezentace plochami. S takovými objekty je možno pracovat jako s jinými 3D objekty, tj. aplikovat transformace, vytažení apod.

K tomu Asymptote používá algoritmus `BEZULATE`. `BEZULATE` přebírá jednoduše souvislou oblast a postupem podobným triangulaci ji převádí na systém menších, jednodušších oblastí (Bézierových plátů). Prvotní oblast je určena systémem jednoduše uzavřených křivek. Těm však obecně neodpovídá jednoduše souvislá oblast (obr. 42). Z tohoto důvodu se před `BEZULATE` provádí algoritmus `PARTITION`, který zadané oblasti rozdělí tak, že už budou jednoduše souvislé.



Obrázek 41: 3D text vytažený z $e^{\pi i} = -1$



Obrázek 42: Oblast určená znakem ∞

Rozdělení (`PARTITION`)
Předpokládejme konečnou množinu jednoduše uzavřených křivek. Těm odpovídají nějaké oblasti. Tento algoritmus je rozdělí na menší, jednoduše souvislé oblasti (ve tvaru množiny křivek). Provede se to tak, že se křivky nejdříve seřadí podle toho, jak jsou v sobě vzájemně obsaženy. Takové rekurzivní seřazení provádí procedura `SORT`. Pak se mezi vnějšími a vnitřními křivkami vytváří vhodné úsečky, které dané oblasti propojují (procedura `MERGE`).

Algoritmus 1 `PARTITION(D)`

Vstup: množina jednoduše uzavřených křivek `D`
Výstup: množina uzavřených křivek `A`

```
 1: A ← ∅
 2: for G ∈ SORT(D) do
 3:    innerGroups ← SORT(G.inners)
 4:    for H ∈ innerGroups do
 5:       A ← A ∪ PARTITION(H.inners)
 6:    innerTops ← ∅
 7:    for K ∈ innerGroups do
 8:       innerTops ← innerTops ∪{K.top}
 9:    A ← A ∪ MERGE(G.top,innerTops)
10: return A
```

Popišme si použité procedury `SORT` a `MERGE`. Jak již bylo zmíněno, procedura `SORT` seřadí křivky v závislosti na tom, jak jsou v sobě obsaženy. To se dá provést efektivně, protože křivky jsou jednoduše uzavřené. Na test pak stačí použít libovolný bod křivky.

Algoritmus 2 `SORT(D)`

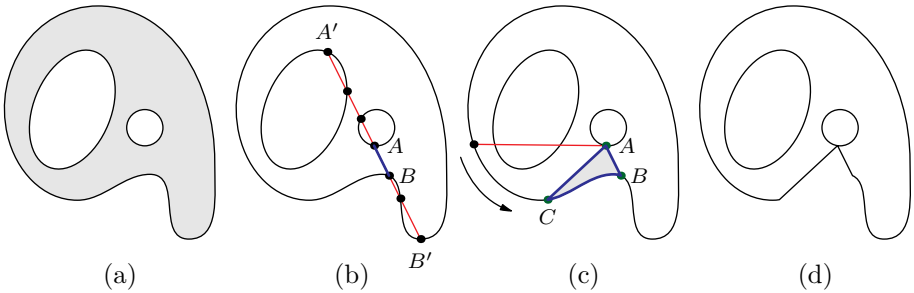
Vstup: množina jednoduše uzavřených křivek `D`
Výstup: množina `A` dvojic „vrchních“ a příslušných, v nich obsažených křivek

```
 1: G ← ∅
 2: for C ∈ D do
 3:   found ← false
 4:   for H ∈ G do
 5:     if C leží uvnitř H.top then
 6:       H.inners ← H.inners ∪ {C}
 7:       found ← true
 8:       break
 9:     else if H.top leží uvnitř C then
10:       H.inners ← H.inners ∪ {H.top}
11:       H.top ← C
12:       for H' ∈ G \ {H} do
13:         if H'.top leží uvnitř C then
14:           H.inners ← H.inners ∪ H'.inners
15:           H.inners ← H.inners ∪ {H'.top}
16:           G ← G \ {H'}
17:       found ← true
18:       break
19:   if not found then
20:     G ← G ∪{(C, ∅)}
21: return G
```

Procedura `MERGE` prochází křivky `inners`, které jsou obsaženy v křivce `top`, a snaží se je vhodnými úsečkami propojit (obr. 43).

Libovolný bod na vnější křivce spojí s libovolným bodem na vnitřní křivce. Výsledná přímka obsahuje dva průsečíky, A s vnitřní, B s vnější křivkou takové, že mezi A a B se již jiné průsečíky nenacházejí (b). Dále se hledá bod C na vnější křivce takový, že přímka AC kromě krajních bodů neprotíná žádnou z křivek, a zároveň oblast určená úsečkami $\overline{AB}$, $\overline{AC}$ a křivkou $\overline{BC}$ (úsek na vnější křivce) neobsahuje žádnou další z křivek v `inners` (c). To se nám po případném opakovaném dělení segmentů sousedících s B vždy podaří. Nakonec se nalezená oblast oddělí a pokračuje se se zbylými křivkami (d).

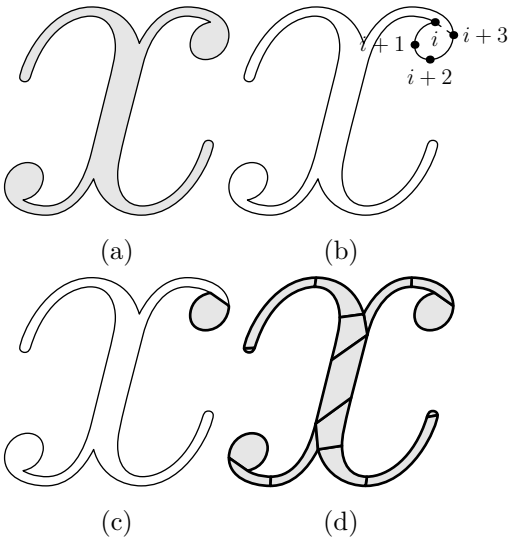
Ve výsledku se tedy `PARTITION` redukuje na zjištění, zda bod leží uvnitř



Obrázek 43: procedura `MERGE`

křivky, a na počítání průsečíků přímky a křivky. Počítání průsečíků Bézierovy kubiky s přímkou vede k hledání kořenů kubického polynomu, které dovedeme spočítat. Pro zjištění, zda bod leží uvnitř oblasti zadané uzavřenou křivkou, se může použít např. Winding number (způsob použitý v programu Asymptote).

Algoritmus `BEZULATE`



Obrázek 44: Algoritmus `BEZULATE`

Dostáváme se k popisu algoritmu `BEZULATE`, hlavního algoritmu této části. Na

Algoritmus 3 `MERGE`(`top`,`inners`)

Vstup: křivka `top` a množina křivek `inners` obsažených v `top`
Výstup: množina `A` uzavřených křivek

```
 1: A ← ∅
 2: while inners ≠ ∅ do
 3:   D ← libovolná křivka z inners
 4:   A'B' ← úsečka mezi libovolnými body B' na top a A' na D
 5:   B ← ten z průsečíků A'B' a top nejblíž bodu A'
 6:   t ← čas bodu B na křivce top
 7:   A ← ten z průsečíků BA' s křivkami z inners nejblíže bodu B
 8:   D' ← křivka z inners, na které se realizuje průsečík A
 9:   Δ ← 2
10:   found ← false
11:   repeat
12:     Δ ← Δ/2
13:     for sgn ∈ {−1, 1} do
14:       C ← bod na křivce top v čase t+sgnΔ
15:       l ← subpath(top,B,C) + CA + AB
16:       if l neobsahuje ani neprotíná žádnou křivku
          z inners s výjimkou bodu A then
17:         found ← true
18:   until found
19:   A ← A ∪ {l}
20:   top ← subpath(top,C,B) + BA+ subpath(D',A,A) +AC
21:   inners ← inners \ {D'}
22: A ← A ∪ {top}
23: return A
```

vstupu již předpokládáme jednoduše uzavřenou křivku C, jíž odpovídá jednoduše souvislá oblast. Účelem tohoto algoritmu je převést oblast na sjednocení menších oblastí ohraničených nejvýše čtyřmi Bézierovými kubikami. Popišme, jakým způsobem `BEZULATE` sestavuje tyto ohraničující křivky.

Nechť křivka C prochází n kontrolními body $P_0, P_1, \ldots, P_{n-1}$ (body mezi jednotlivými segmenty). Algoritmus postupně prochází body P_i a testuje, zda oblast určená segmenty P_iP_{i+1}, $P_{i+1}P_{i+2}$, $P_{i+2}P_{i+3}$ a úsečkou $\overline{P_{i+3}P_i}$, leží uvnitř křivky C. Na to stačí zkontrolovat jen zmíněnou úsečku. V případě, že tam oblast leží, vhodnou rekonstrukcí křivky se tato oblast „vyjme“, a algoritmus se opakuje pro zbylou část. Pokud se vhodnou oblast, tj. trojici, případně čtveřici bodů nepodaří nalézt, křivka C se zjemní přidáním dodatečného bodu doprostřed každého segmentu. Celý proces se pak opakuje, dokud délka původní křivky neklesne na

4 a méně. Při dostatečně jemném dělení se původní křivka stává nerozlišitelnou od mnohoúhelníku a celý proces se redukuje na triangulaci. Algoritmus se proto vždy dokončí.

Algoritmus 4 `BEZULATE(D)`

Vstup: množina uzavřených křivek `D`
Výstup: množina uzavřených křivek `A` délek 3 nebo 4

```
1: A ← ∅
2: for C ∈ D do
3:   while C.length > 4 do
4:     found ← false
5:     for n = 3 to 2 do
6:       for i = 0 to C.length-1 do
7:         L ← úsečka mezi uzly i a i+n na C
8:         if L protíná C právě ve dvou bodech and
           prostřední bod L leží uvnitř C then
9:           p ← subpath(C,i,i + n)
10:          q ← subpath(C,i + n,i + C.length)
11:          A ← A ∪ {p + L}
12:          C ← L + q
13:          found ← true
14:          break
15:      if found then
16:        break
17:    if not found then
18:      zjemni C přidáním bodu doprostřed každého segmentu
19: return A
```

Coonsova bilineární plocha

Posledním krokem je získání Bézierova plátu ze čtyř okrajových Bézierových křivek. K tomu využijeme obecný postup, tzv. Coonsovu bilineární plochu, která je určená vztahem:

$$Q(u,v) = (1-u,u)\begin{pmatrix} Q(0,v) \\ Q(1,v) \end{pmatrix} + (Q(u,0),Q(u,1))\begin{pmatrix} 1-v \\ v \end{pmatrix} - \\ - (1-u,u)\begin{pmatrix} Q_{0,0} & Q_{0,1} \\ Q_{1,0} & Q_{1,1} \end{pmatrix}\begin{pmatrix} 1-v \\ v \end{pmatrix} \tag{4}$$

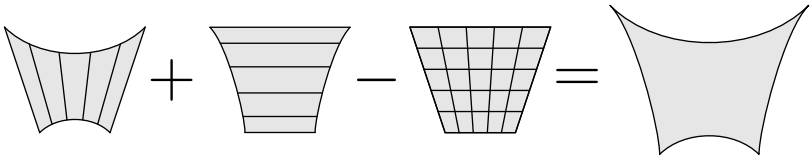
kde $u, v \in [0,1] \times [0,1]$. Body $Q(0,v)$, $Q(1,v)$, $Q(u,0)$ a $Q(u,1)$ známe, poněvadž leží na okrajových křivkách.

V našem případě však chceme plochu v podobě Bézierova plátu. Proto potřebujeme z předchozího obecného zápisu vyjádřit 4 vnitřní body pro speciální případ s Bézierovými okrajovými křivkami. Bézierův plát je zadán šestnácti body $P_{i,j}$ a předpisem:

$$f(u,v) = \sum_{i,j=0}^{3} B_i(u)B_j(v)P_{i,j} \quad \text{pro} \quad u,v \in [0,1] \tag{5}$$

kde $B_i(t) = \binom{3}{i}t^i(1-t)^{3-i}$ je Bernsteinův polynom třetího stupně. Známe všechny body $P_{i,j}$ kromě vnitřních $P_{1,2}$, $P_{2,1}$, $P_{1,1}$ a $P_{2,2}$. Položením rovnosti s výrazem 4 a porovnáním koeficientů dostaneme příslušné vztahy.

$$\begin{aligned}
P_{1,2} &= \frac{1}{9}(6P_{0,2} + 6P_{1,3} - 4P_{0,3} + 3P_{1,0} + 3P_{3,2} - 2P_{0,0} - 2P_{3,3} - P_{3,0}) \\
P_{2,1} &= \frac{1}{9}(6P_{2,0} + 6P_{3,1} - 4P_{3,0} + 3P_{0,1} + 3P_{2,3} - 2P_{0,0} - 2P_{3,3} - P_{0,3}) \\
P_{1,1} &= \frac{1}{9}(6P_{0,1} + 6P_{1,0} - 4P_{0,0} + 3P_{1,3} + 3P_{3,1} - 2P_{3,0} - 2P_{0,3} - P_{3,3}) \\
P_{2,2} &= \frac{1}{9}(6P_{3,2} + 6P_{2,3} - 4P_{3,3} + 3P_{2,0} + 3P_{0,2} - 2P_{3,0} - 2P_{0,3} - P_{0,0})
\end{aligned}$$

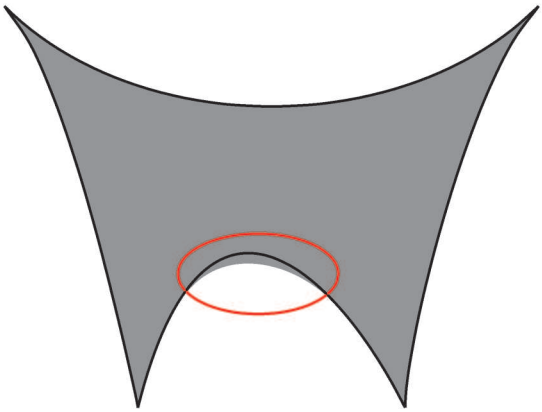


Obrázek 45: Coonsova bilineární plocha

Tím celý algoritmus ještě nekončí. V případě, kdy by byl výsledný Bézierův plát degenerovaný, tj. nebyl by difeomorfizmem ze čtverce $[0,1] \times [0,1]$, vznikaly by při renderování grafické artefakty (obr. 46). Z tohoto důvodu se v programu Asymptote kontroluje postačující podmínka pro degenerovanost. Podmínka je založena na faktu, že Jakobián pro zmíněnou plochu musí být všude nenulový (více viz Surface Parametrization . . . , 2011).

Výrazy

$$\sum_{i+k=p}\sum_{j+l=q}\left[(P_{i+1,j} - P_{i,j}) \times (P_{k,l+1} - P_{k,l})\right]_3 \binom{2}{i}\binom{3}{k}\binom{3}{j}\binom{2}{l} \tag{6}$$



Obrázek 46: Grafický artefakt degenerovaných ploch

pro $p, q = 0, 1, \ldots, 5$, musí mít stejné zmaménko, pokud je plocha nedegenerovaná. Zde operace $[\cdot \times \cdot]_3$ na vektorech z $\mathbb{R}^2$ odpovídá třetí složce vektorového součinu příslušných vektorů v $\mathbb{R}^3$, tj. $[u \times v]_3 = u_x v_y - u_y v_x$. V případě, že se podaří degenerovanost zjistit, zkouší se dále degenerovanost na okraji plochy. K tomu slouží následující věta.

Věta 1. *Nechť p je uzavřená kladně orientovaná rovinná křivka délky 4. Dále nechť vnitřní úhly svírané vcházející a vycházející tečnou jsou ve všech jejích uzlech menší nebo rovny 180°. Dále nechť $J(u, v)$ je Jakobián odpovídajícího Bézierova plátu s kontrolními body $P_{i,j}$. Položme*

$$f(u) = \frac{1}{3} J(u, 0) = \sum_{i,j=0}^{3} B_i'(u) B_j(u) P_{i,0} \times (P_{j,1} - P_{j,0}). \tag{7}$$

Jestliže pro všechna $u \in (0, 1)$ taková, že $f'(u) = 0$, platí $f(u) \geq 0$, pak $J(u, 0) \geq 0$ na $[0, 1]$. V opačném případě se minimální hodnota $J(u, 0)$ nachází v některém u takovém, že $f'(u) = 0$.

Důkaz. Předpokládejme, že pro všechna $u \in (0, 1)$, $f'(u) = 0$ platí $f(u) \geq 0$. Dále předpokládejme, že $J(u, 0) < 0$ pro nějaké $u \in [0, 1]$. Jelikož $J(u, 0)$ je spojitá funkce, nabývá svého minima v nějakém $\overline{u} \in [0, 1]$. Jistě je pak $J(\overline{u}, 0) < 0$. Dále platí

$$J(0, 0) = 3f(0) = 9(P_{1,0} - P_{0,0}) \times (P_{0,1} - P_{0,0}) \geq 0$$
$$J(1, 0) = 3f(1) = 9(P_{3,0} - P_{2,0}) \times (P_{3,1} - P_{3,0}) \geq 0$$

poněvadž se jedná o vektorové součiny tečných vektorů na křivce p, která je kladně orientovaná, a úhly svírané těmito tečnami nepřesahují 180°. Proto $\overline{u} \in (0,1)$. Protože však $J(u,0) = 3f(u)$, nabývá svého minima v $\overline{u}$ i funkce f. Potom zřejmě $f'(\overline{u}) = 0$ a $f(\overline{u}) < 0$, což je spor. □

Nejdřív tedy rozdělíme okraj v uzlech, kde úhel mezi vcházející a vycházející tečnou přesahuje 180°. Potom se numerickými metodami naleznou kořeny polynomu 4. stupně $f'(u)$ na $(0,1)$. V nalezených bodech u se potom zkontrolují znaménka $f(u)$. V případě zápornosti již stačí spočítat $J(u,0)$ ve všech nalezených kořenech, a zjistit tak, kde je Jakobián nejmenší. Takto spočítané body určují optimální rozdělení okrajových křivek. Daným způsobem se počet výsledných plátů snižuje oproti jiným, rychlejším metodám. To je v našem konkrétním případě výhodné, protože chceme vytvořit pouze reprezentaci původní oblasti a časová složitost je tu méně podstatná. Uvedenou Větu použijeme analogicky pro nalezení degenerace i na zbylých třech okrajových křivkách.

V případě, že se degenerace na okraji nezjistí, musí být degenerace uvnitř plochy. V takovém případě se rozdělení provede náhodně ve zvolených místech na okraji. Celý proces se potom rekurzivně opakuje.

Summary: Mathematical graphics with the Asymptote program

The aim of this paper is to describe creating of mathematical graphics with the program Asymptote with emphasis on 3D graphics. The task of this text is to make learning of the program easier with various examples used, while at the same time providing a solid description of the background of techniques used. In the first part, the basic syntax rules are explained, then in following chapters the routines for 2D and 3D are described. In the last chapter, algorithms converting planar regions into 3D surfaces are explained.

Literatura

ASYMPTOTE – Galeries d'exemples [on-line]. 2011. [cit. 2011-12-15]. Dostupné na: `http://marris.org/asymptote/`.

Asymptote [Oficiální stránky] [on-line]. 2004. [cit. 2011-12-15]. Dostupné na: `http://asymptote.sourceforge.net/`.

Kutal, O. *Tvorba matematické grafiky pomocí programu Asymptote.* (Diplomová práce.) Brno : Masarykova univerzita, 2012. 105 s.

Specifikace formátu PRC [on-line]. 2008. [cit. 2012-02-16]. Dostupné na: `http://livedocs.adobe.com/acrobat_sdk/9/Acrobat9_HTMLHelp/API_References/PRCReference/PRC_Format_Specification/`.

Surface Parametrization of Nonsimply Connected Planar Bézier Regions [on-line]. 2011. [cit. 2011-12-15]. Dostupné na: `http://www.math.ualberta.ca/~bowman/publications/cad10.pdf`.

Ondřek Kutal, kutal@math.muni.cz

Matematika na čtečkách e-knih

Roman Plch

Mobilní zařízení se stala nedílnou součástí našeho každodenního života, a je proto přirozené, že ovlivňují i výukový proces. Tištěné výukové materiály jsou postupně nahrazovány digitálními a jejich využívání se stále častěji přesouvá z počítačů na tato mobilní zařízení (mobily, tablety, čtečky e-knih). Tento příspěvek popisuje možnosti zobrazování matematických textů na mobilních zařízeních a dále se věnuje konverzi matematického textu do podoby vhodné k zobrazení na čtečkách e-knih.

Formáty e-knih

Formát EPUB

EPUB (Electronic Publication, 2015) je standardizovaný otevřený souborový formát určený pro digitální knihy, vyvinutý organizací International Digital Publishing Forum (IDPF). Je založený na XML. Zjednodušeně se dá říci, že se jedná o sadu HTML stránek zabalenou s dalšími metadaty do ZIP archivu. Formát EPUB je pro čtenáře výhodný v tom, že umožňuje automaticky přizpůsobit text různým velikostem obrazovek zařízení (včetně chytrých telefonů, netbooků a čteček elektronických knih). EPUB také podporuje obsah s pevným rozvržením.

Poslední verze tohoto formátu (EPUB 3.01) už podporuje i jazyk MathML a umožňuje tak zobrazování matematických formulí přirozeným způsobem, tj. matematický text se zvětšuje/zmenšuje společně s ostatním textem, vzorečky se správně zarovnávají vzhledem k okolnímu textu a je možné v nich vyhledávat a kopírovat jejich obsah.

Bohužel podpora tohoto formátu je zatím omezená a mnohem častěji se používá starší verze EPUB 2, která pro zobrazování matematiky využívá obrázků, stejně jako všechny dále uváděné formáty.

Formát EPUB 2 podporují téměř všechny čtečky e-knih (významnou výjimkou je pouze Kindle od Amazonu).

Formáty MOBI, AZW, KF8

Čtečky Kindle od Amazonu nepodporují formát EPUB a používají vlastní formáty pro e-knihy. Na rozdíl od formátu EPUB se jedná o proprietární formáty. První čtečky Kindle používaly formát AZW, což je vlastně formát MOBI doplněný o podporu DRM restrikcí. Spolu se čtečkou Kindle Fire se objevuje formát KF8

 doi: 10.5300/2015-1-2/70

(Kindle format 8), známý též jako AZW3. Ten už podporuje i některé prvky HTML 5 a CSS3. Matematika může být ale uvedena pouze ve formě obrázků.

Pro převody z formátu EPUB do MOBI (AZW3) můžeme použít aplikaci KindleGen nebo níže uvedený program Calibre pro oboustranné převody.

Formát PDF

I tento základní souborový formát pro elektronické publikování má na čtečkách e-knih stále své místo, jeho „nevýhodou“ je ale fixní rozložení stránky, které neumožňuje přizpůsobovat velikost textu rozměrům čtečky.

Některé čtečky sice dnes už mají funkci „PDF Reflow“, která se pokusí původní strukturu dokumentu převést tak, aby se text dobře vešel na displej čtečky. Jak však uvidíme později, je funkce PDF Reflow prakticky nepoužitelná na texty s matematikou.

Kromě těchto základních formátů se můžeme setkat i s méně častými formáty, jako je eReader (PDB), FictionBook (FB2) a Microsoft Lit (LIT). V tomto příspěvku se ale budeme věnovat výhradně nejrozšířenějšímu formátu EPUB.

Typy čteček

Čtečky e-knih můžeme dělit do dvou základních kategorií – na softwarové a hardwarové čtečky.

Hardwarové čtečky

Jedná se o specializovaná elektronická zařízení na bázi elektronického inkoustu. Oproti zařízením s LCD displejem spotřebovává tato zobrazovací technologie mnohem méně energie a text je dobře čitelný i na intezivním slunečním světle.

Mezi populární čtečky e-knih patří např. Kindle (Amazon), Nook (Barnes & Noble), Kobo (Rakuten), Cybook (Bookeen), PocketBook, SonyReader, JetBook (Ectaco) a Boox (Onyx). Podporu formátu EPUB 3 ale uvádí pouze Kobo u své čtečky Kobo Glo HD (Kobo, 2015).

Softwarové čtečky

Druhou kategorii tvoří programy a aplikace umožňující čtení e-knih na počítači, tabletu a mobilním telefonu. Do výběru jsme zařadili některé programy, které jsou šířeny bezplatně, jsou multiplatformní a podporují více formátů e-knih (a především formát EPUB 3). Kompletní přehled čteček s podporou EPUB 3 najdeme např. na `http://docs.mathjax.org/en/latest/misc/epub.html`.

Calibre
(`http://calibre-ebook.com/`)
Volně šiřitelná multiplatformní aplikace Calibre je univerzální správce a konvertor elektronických knih. K dispozici jsou nástroje pro organizaci i prohlížení knih, editaci metadat i konverzi. Do osobní knihovny je možné přidávat knihy jednotlivě nebo v dávkách, je podporován široký rozsah nejrůznějších formátů.

Readium
(`http://idpf.org/forum/topic-2429`)
Rozšíření pro Google Chrome, umožňující čtení e-knih ve formátu EPUB 3.

Gitden Reader
(`http://gitden.com/`)
Jedna z nejlépe hodnocených čteček e-knih pro Android a IOS. Na základě našich zkušeností nejlépe zvládá zobrazování e-knih s matematickým textem ve formátu EPUB 3 na mobilních platformách.

Hardwarové čtečky s Androidem
Jedná se o zařízení s elektronickým inkoustem běžící pod operačním systémem Android. Umožňuje tedy instalaci Android aplikací a spojuje tak výhody hardwarových a softwarových čteček v jediném zařízení. Jako příklad můžeme uvést např. inkBOOK Onyx nebo eReading 4 Touch Light.

Konverze do EPUB

Po tomto stručném úvodu se budeme věnovat přípravě matematického textu pro čtečky e-knih. Vycházíme z předpokladu, že máme k dispozici zdrojový kód v LaTeXu a chceme ho převést do formátu EPUB.

TeX4ebook
Jedná se o konvertor z LaTeXu do formátu EPUB a MOBI. Ke konverzi využívá program tex4ht a skripty LuaTeXu. Instalační balíček a dokumentaci najdeme na `https://github.com/michal-h21/tex4ebook`, návod na instalaci najdeme např. v bakalářské práci (Antol, 2015).

Konverzi spustíme příkazem

```
tex4ebook [volby] jméno_souboru
```

Implicitní výstupní formát je EPUB, změnu na EPUB 3 (MOBI) provedeme parametrem `-f epub3` (`mobi`). Pro matematiku v MathMl přidáme na konec příkazu ještě parametr `mathml`:

```
tex4ebook -f epub3 soubor.tex mathml
```

Při praktických testech konverze delšího matematického textu (Slovák, Panák a Bulant, 2013) jsme na instalaci TEXLive 2014 narazili na chybové hlášení

```
Undefined control sequence.
\pgfsys@svg@newline ->\Hnewline
```

Jedná se o známou chybu, kterou podle (TEX – LATEX Stack Exchange, 2014) vyřešíme úpravou v souboru `pgfsys-tex4ht.def`, kde řádek s textem `\def\pgfsys@svg@newline{\Hnewline}` nahradíme za `\def\pgfsys@svg@newline{^^J}`.

Při výstupu do EPUB 2 se matematika ukládá do obrázků formátu PNG, pomocí poměrně komplikovaného nastavení je možné docílit změny formátu na SVG. Potřebné nastavení je popsáno v (TEX – LATEX Stack Exchange, 2015).

Shrnutí, výhody a nevýhody
Konverze i složitějšího matematického textu s tabulkami sázenými pomocí balíčku `tabularx` proběhla v pořádku, zobrazení matematiky pomocí Calibre na počítači bylo taktéž bez chyb. Poznámky pod čarou se umisťují ihned za odstavcem textu, ve kterém je příkaz uveden. Do textu je možné vkládat obrázky ve formátech EPS, JPG a PNG.

Nevýhodou je poměrně komplikovaná instalace[1] a pro méně zkušené uživatele nutnost spouštět konverzi z příkazového řádku.

Pandoc
Pandoc je univerzální multiplatformní dokumentový konvertor podporující desítky různých formátů včetně EPUB 3.

Instalace je v tomto případě jednoduchá, stačí ze stránek `http://pandoc.org/` stáhnout a spustit instalační soubor. K dispozici je i online verze (`http://pandoc.org/try/`), která má však omezené možnosti a nepodporuje výstup do EPUB.

Program se ovládá pouze z příkazového řádku (nemá grafické uživatelské rozhraní), jeho syntaxi si ukážeme na modelovém příkladu

```
pandoc soubor.tex -f latex -t epub -mathml -o soubor.epub
```

kde přepínač `f` zadává vstupní formát, přepínač `t` výstupní formát a přepínač `o` jméno výstupního souboru. Pro výstup ve formátu EPUB 3 zadáme parametr `mathml`. Seznam všech možných parametrů získáme přepínačem `h`.

Shrnutí, výhody a nevýhody
Pandoc nezvládl konverzi celé řady matematických symbolů (balíček `amssymb`), dále nefunguje konverze matematických prostředí definovaných pomocí balíčku

[1] TEX4ebook se stal součástí aktuální instalace TEXlive 2015, což situaci výrazně zjednodušuje.

amsthm. Po konverzi dále nefungují odkazy vytvářené pomocí příkazů \label a \ref. Ani konverze tabulek neproběhla bez chyb. Poznámky pod čarou jsou umístěny na konec knihy a v textu jsou umístěny odkazy. Do textu je možné vkládat obrázky ve formátech JPG a PNG.

Mezi výhody patří rychlá a snadná instalace a velké množství konverzních formátů. Pro složitější matematické texty není ale možné Pandoc doporučit.

TEX4ht + Calibre

V tomto případě rozdělíme konverzi do dvou částí. Nejdříve pomocí programu TEX4ht (je součástí instalace TEXLive) zkonvertujeme LATEXový zdroj do jazyka HTML. Získaný soubor poté pomocí Calibre převedeme do EPUB.

Syntaxe:

```
htlatex soubor.tex "xhtml, charset=utf-8,mathml" " -cunihtf -utf8"
```

Výstupem bude soubor soubor.html v kódování UTF8 s matematikou zapsanou v jazyce MathML.

Následně pomocí příkazu

```
ebook-convert soubor.html soubor.epub
```

získáme výsledný soubor ve formátu EPUB 3. Pokud při konverzi vynecháme parametr mathml, bude matematika ve formě obrázků a výstup bude ve formátu EPUB 2. Stejně tak při volbě jiného výstupního formátu (např. MOBI) bude matematika uložena ve formě obrázků. Tyto konverze je možné provádět i pomocí grafického uživatelského rozhraní Calibre.

Shrnutí, výhody a nevýhody

Výstup je srovnatelný s výsledky získanými pomocí TEX4ebook. Poznámky pod čarou jsou umístěny na konci dokumentu, v textu jsou umístěny odkazy. Mnohem jednodušší je ale instalace, TEX4ht je součástí TEXLive, stačí tedy nainstalovat pouze Calibre. Komplikací zůstává, že TEX4ht je při generování MathML velmi citlivý na čistotu zápisu matematiky ve zdrojovém textu dokumentu. Například zápis `$M=\{x|x$ je liché $\}$` je korektní TEXový zápis. TEX4ht ale v tomto případě nemá informaci o párování složených závorek. Je tedy nutné použít správnější zápis: `$M=\{x|x \text{je liché}\}$`, viz Sojka a Růžička (2008).

LATEXML

LATEXML (http://dlmf.nist.gov/LaTeXML/) konvertuje zdrojový kód LATEXu do formátů XML, HTML a EPUB. Primárním výstup je do XML, dalším zpracováním získáme formáty HTML a EPUB, matematické formule mohou být ve formě obrázků nebo MathML.

Vlastní použití je jednoduché, pomocí příkazu
`latexmlc soubor.tex --destination=soubor.epub`
dostaneme výsledný soubor ve formátu EPUB3.

Shrnutí, výhody a nevýhody
Konverze i složitějšího matematického textu proběhla v pořádku, text je na čtečkách na PC dobře čitelný (obr. 1), také konverze tabulek proběhla korektně. Poznámky pod čarou se zobrazují v samostatném okně (po najetí kurzorem myši na text poznámky). Do textu je možné vkládat obrázky ve formátu PDF, EPS, JPG a PNG.

Komplikací pro autory může být to, že pokud mají v dokumentu vlastní definice maker, musí tuto definici napsat i pro LaTeXML, aby věděl, jak tato makra expandovat. Dle našeho názoru se ale jedná o perspektivní a dobře použitelné řešení (pokud ovšem dojde k vylepšení softwaru pro čtení formátu EPUB 3 na čtečkách e-knih a mobilních platformách).

$$(1+ap)^{p^{\ell-1}} \equiv (1+ap^{\ell-1})^p \pmod{p^{\ell+1}}.$$

Z binomické věty přitom plyne

$$(1+ap^{\ell-1})^p = 1 + p\cdot a\cdot p^{\ell-1} + \sum_{k=2}^{p}\binom{p}{k}a^k p^{(\ell-1)k}$$

a vzhledem k tomu, že pro $1<k<p$ platí $p \mid \binom{p}{k}$, stačí ukázat $p^{\ell+1} \mid p^{1+(\ell-1)k}$, což je ekvivalentní s $1 \le (k-1)(\ell-1)$. Rovněž pro $k=p$ dostáváme díky předpokladu $\ell \ge 2$ vztah $p^{\ell+1} \mid p^{(\ell-1)p}$. ∎

Lemma. *Buď p liché prvočíslo, $\ell \ge 2$ libovolné. Pak pro libovolné $a \in \mathbb{Z}$ splňující $p \nmid a$ platí, že řád čísla $1+ap$ modulo p^ℓ je roven $p^{\ell-1}$.*

Důkaz. Podle předchozího lemmatu je

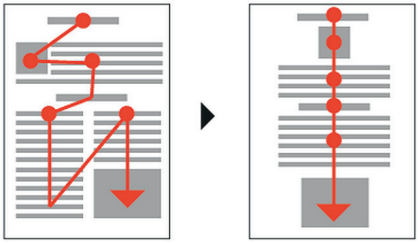
Obrázek 1: Export do EPUB 3 zobrazený pomocí Calibre na PC

PDF formát na čtečkách e-knih

Formát PDF zachovává fixní rozložení stránky, proto se obecně nepovažuje za formát vhodný pro čtečky. Stále je však nejběžnější pro publikování elektronických dokumentů a při jeho použití máme jistotu, že i složitý matematický text bude vypadat přesně tak, jak jsme zamýšleli. V této části ukážeme, jak PDF soubory přizpůsobit k zobrazování na čtečkách e-knih.

PDF Reflow

Tato funkce se pokusí původní strukturu dokumentu převést tak, aby se text dobře vešel na displej čtečky, aby nikde nevyčníval (nepřetékal mimo displej) – aby se tedy dokument choval jako nativní elektronická kniha nebo webová stránka s průběžným zobrazením obsahu tak, jak jej nastavil uživatel, včetně velikosti písma a rozměrů stránky.



Obrázek 2: Schéma práce funkce PDF reflow, převzato (Bohdal, 2014)

Výhodou této funkce je, že umožňuje také určité zmenšení nebo zvětšení písma. Touto funkcí ale nejsou vybaveny všechny čtečky. Funguje jen na PDF se sazbou, nikoliv skenované soubory, které jsou vlastně jen obrázkem zabaleným do formátu PDF.

Čím jednodušší je dokument PDF, tím lépe funkce PDF Reflow pracuje. Pokud máte jen souvislý text s odstavci a kapitolami, např. běžný román, budete pravděpodobně s převodem vždy spokojeni. Pokud však máte odborný text s obrázky, tabulkami a vzorečky, výsledek může být katastrofální.

Změna rozměrů PDF souboru

K2pdfopt (`http://www.willus.com/k2pdfopt/`) je multiplatformní program, který optimalizuje soubory ve formátu PDF pro čtečky. Vstupním i výstupním souborem je PDF, přičemž volíme rozměry výsledného souboru. Nevýhodou je

nesprávné dělení slov a matematických výrazů na konci řádků a také velikost výsledného souboru, která několikanásobně převyšuje velikost původního souboru. Hodí se tedy pouze v případě, že máme k dispozici jen výsledný soubor a nikoliv zdrojový kód.

Podobně funguje celá řada dalších nástrojů, najít je můžeme např. na `http://www.willus.com/k2pdfopt/pdf_conversion.shtml`. Za vyzkoušení stojí i online nástroj `http://convert-kindle.com/`.

Balíček `geometry`, šablony

Mnohem výhodnější je ale upravit rozměry a vzhled výsledného PDF dokumentu ještě před překladem. K tomu můžeme využít např. balíček `geometry`, hodí se i balíček `extsizes`, který umožňuje měnit velikost základního fontu pro dokument.

Nejjednodušší možností je ale použít některou z šablon, dobré zkušenosti máme se šablonou eBook (`http://www.latextemplates.com/template/ebook`). Použitím šablony získáme výsledný soubor ve formátu PDF, který svými rozměry, nastavením okrajů, velikostí použitého fontu, číslováním stránek a celkovým vzhledem co nejlépe odpovídá zobrazení na dnešních čtečkách e-knih (obr. 3).

Závěr

Základním problémem zobrazování matematických textů na čtečkách e-knih je to, že hardwarové čtečky nepodporují formát EPUB 3. To vede k situaci, kdy se matematické knihy v tomto formátu prakticky nevydávají. Nedostatek poptávky pak způsobuje, že se výrobcům čteček nevyplatí investovat do podpory formátu EPUB 3. Pokud by se podařilo tento „začarovaný kruh“ nějakým způsobem vyřešit, tak je formát EPUB 3 určitě do budoucna perspektivním řešením.

Protože ani softwarové čtečky nejsou v současné době schopny korektně zobrazovat složitější matematický text, zůstává pro autory matematických textů jediné řešení – formátovat výsledný PDF dokument kromě verze pro tisk i ve verzi pro zobrazení na čtečkách e-knih, např. pomocí šablony eBook uvedené výše v článku.

Summary: Mathematics for readers of e-books

Mobile devices have become an integral part of our daily lives and that is why they have been affecting a learning process. Printed educational materials are being gradually replaced by a digital version. Displaying is increasingly shifting from desktop computers to the mobile devices (i.e. phones, tablets and e-book readers). This paper describes possibilities of displaying mathematical texts on the mobile devices and a possible conversion of mathematical texts for e-book readers is also analysed.

Důkaz. Plyne snadno z binomické věty s využitím matematické indukce vzhledem ℓ.
I. Pro $\ell = 2$ tvrzení zřejmě platí.
II. Nechť tvrzení platí pro ℓ, dokážeme jej i pro $\ell + 1$.
Tvrzení pro ℓ umocníme na p-tou, dostaneme

$$(1 + ap)^{p^{\ell-1}} \equiv (1 + ap^{\ell-1})^p \pmod{p^{\ell+1}}.$$

Z binomické věty přitom plyne

$$(1 + ap^{\ell-1})^p = 1 + p \cdot a \cdot p^{\ell-1} + \sum_{k=2}^{p} \binom{p}{k} a^k p^{(\ell-1)k}$$

a vzhledem k tomu, že pro $1 < k < p$ platí $p \mid \binom{p}{k}$, stačí ukázat $p^{\ell+1} \mid p^{1+(\ell-1)k}$, což je ekvivalentní s $1 \leq (k-1)(\ell-1)$. Rovněž pro $k = p$ dostáváme díky předpokladu $\ell \geq 2$ vztah $p^{\ell+1} \mid p^{(\ell-1)p}$. □

Lemma. *Buď p liché prvočíslo, $\ell \geq 2$ libovolné. Pak pro libovolné $a \in \mathbb{Z}$ splňující $p \nmid a$ platí, že řád čísla $1 + ap$ modulo p^ℓ je roven $p^{\ell-1}$.*

Obrázek 3: Ukázka použití šablony eBook

Literatura

Antol, F. *Matematika na čtečkách e-knih* [on-line]. (Bakalářská práce.) Brno: Masarykova univerzita, 2015. [cit. 2015-07-17]. xii+29 s. Dostupné na: `http://is.muni.cz/th/408834/prif_b/`.

Bohdal, T. Jak číst PDF na ebook čtečkách [on-line]. In *Ebook Expert.cz.* 2014 [cit. 2015-07-29]. Dostupné na: `http://ebookexpert.cz/jak-cist-pdf-na-ebook-cteckach/`.

EPUB [on-line]. In *International Digital Publishing Forum : The Trade and Standards Organization for the Digital Publishing Industry.* 2015 [cit. 2015-05-07]. Dostupné na: `http://idpf.org/epub`.

Error using pgfsysdriver with TEX4ht, only shows up with TEXlive 2014, ok with

TEXlive 2013 [on-line]. In *TEX – LATEX Stack Exchange.* 2014 [cit. 2015-07-29]. Dostupné na: http://tex.stackexchange.com/questions/185349/.

Kobo Device Comparison [on-line]. 2015. [cit. 2015-07-30]. Dostupné na: https://www.kobo.com/devices/compare#ereaders.

Sojka, P., Růžička, M. Parallel Electronic Publications. *Zpravodaj Československého sdružení uživatelů TEXu*, 2008, vol. 18, issue 3, s. 116–129.

Slovák, J., Panák, M., Bulant, M. *Matematika drsně a svižně.* 1. vyd. Brno : Masarykova univerzita, 2013. 773 s. ISBN 978-80-210-6307-5.

The best way for math in epub – font or SVG? (tex4ebook) [on-line]. In *TEX – LATEX Stack Exchange.* 2015 [cit. 2015-07-29]. Dostupné na: http://tex.stackexchange.com/questions/235423/the-best-way-for-math-in-epub-font-or-svg-tex4ebook.

Roman Plch, plch@math.muni.cz

The Trends in the Usage of TEX for the Preparation of Theses and Dissertations at the Masaryk University in Brno

Vít Novotný

This article is a statistical analysis of theses and dissertations written and defended during the period 2010–2015 at the Masaryk University in Brno (MU). The author assesses the trends in the usage of TEX and tests the hypothesis that theses and dissertations written using TEX received significantly better rating than those not written using TEX.

Key words:

thesis, dissertation, statistics

Introduction

A statistical analysis of theses defended at the Masaryk University in Brno (MU) between years 2010 and 2015 was carried out by the author of this article. The sample data for the analysis were kindly provided by doc. Ing. Michal Brandejs, CSc., the head of the Computer Systems Unit at the Faculty of Informatics at MU.

Table 1 shows the distribution of theses written and defended during the years 2010–2015 across the faculties of MU and Table 2 illustrates how many of these theses were written using TEX. Table 3 then presents the trends in the usage of TEX by the students of bachelor's, master's and doctoral degree programmes at the Faculty of Informatics (FI) and the Faculty of Science (Sci). Other faculties of MU were not considered since the number of theses written at these faculties using TEX was statistically insignificant (see Table 2). Theses written by students of lifelong education programmes were likewise ignored since none of them were written using TEX.

Analysis

A thesis was considered to be written using TEX if one or more files submitted with it satisfied one or more of the following conditions:

- The suffix was `tex`.
- The magic number was that of a DVI file.

 doi: 10.5300/2015-1-2/80

Faculty	#	%
Arts	10 000	21.98
Education	8 219	18.07
Social Studies	5 599	12.31
Science	5 275	11.60
Law	4 824	10.60
Economics & Administration	4 591	10.09
Informatics	2 904	6.38
Sports Studies	2 062	4.53
Medicine	2 014	4.43
Total	**45 488**	**100.00**

Table 1: The distribution of theses defended during 2010–2015 across the faculties of MU

Faculty	**With TEX**	**Total**	%
Informatics	1 716	2 904	59.09
Science	786	5 275	14.90
Economics & Administration	64	4 591	1.39
Arts	69	10 000	0.69
Medicine	8	2 014	0.40
Law	15	4 824	0.31
Education	19	8 219	0.23
Social Studies	12	5 599	0.21
Sports Studies	3	2 062	0.15
Total	**2 692**	**45 488**	**5.92**

Table 2: The distribution of theses written using TEX, which were defended during 2010–2015 across the faculties of MU

Degree	Fac.	2010	2011	2012	2013	2014	R
Bachelor's	FI	58.92	59.44	49.54	53.77	59.06	−0.195
	Sci	11.55	13.00	15.90	19.79	15.16	+0.703
	All	5.08	6.19	6.00	6.08	6.24	+0.731
Master's	FI	60.61	59.91	60.08	64.50	57.96	−0.046
	Sci	19.38	13.54	13.75	13.78	17.71	−0.180
	All	6.02	4.88	5.22	6.59	6.29	+0.490
Doctoral	FI	100.00	76.67	71.88	83.87	90.91	−0.155
	Sci	18.09	10.71	12.75	10.19	8.85	−0.830
	All	8.83	8.23	8.41	9.38	7.43	−0.361
All	**FI**	**60.83**	**60.53**	**54.92**	**60.57**	**59.34**	**−0.188**
	Sci	**14.86**	**12.96**	**14.74**	**16.55**	**15.45**	**+0.577**
	All	**5.67**	**5.70**	**5.73**	**6.41**	**6.28**	**+0.855**

Table 3: The percentage of theses written using TEX which were defended in each year during the years 2010–2014 and the sample correlation coefficient R between the percentage and the years with remarkably strong correlations emphasized

- The MIME type was `application/postscript` and the file contained the `TeXDict` substring suggesting that the file was a PostScript document which had been created using the dvips utility.
- The MIME type was `application/pdf` and either the `Creator` or the `Producer` PDF header contained the `TeX` substring suggesting that the file had been created using either the dvipdfm utility or a TEX engine which supports PDF output.

Provided the heuristic is sound, there was a marked and steady increase in the use of TEX for the typesetting of theses at MU during the period 2010–2014 (see Table 3). This, however, does not necessarily hold true for individual faculties and degree study programmes with some of them showing barely any correlation between the years and the use of TEX others showing a strong negative correlation. A particularly striking example of the latter case is the pronounced downward trend in the use of TEX for the typesetting of doctoral theses at the Faculty of Science.

At first the null hypothesis h_1 was supposed that the grades awarded to theses written using and not using TEX, respectively, have the same distribution on the significance level $\alpha = 0.05$. The one-tailed Pearson's χ^2 test (Pearson, 1900) of the goodness of fit was applied to the observations of awarded grades (see Table

	Without TEX	E(**With TEX**)	O(**With TEX**)	$(E-O)^2/E$
A	15 476	987.635	1 181	37.858
B	9999	638.108	587	4.093
C	7 926	505.815	381	30.799
D	4 020	256.545	194	15.248
E	2 783	177.603	128	13.853
F	1 979	126.294	145	2.771
Total	**42 183**	**2 692**	**2 692**	**104.623**

Table 4: The contingency table of the numbers of marks awarded to theses written and defended during 2010–2015 with Pearson's goodness-of-fit measure $(E-O)^2/E$ between the expected (E) and the observed (O) numbers of marks awarded to theses written using TEX

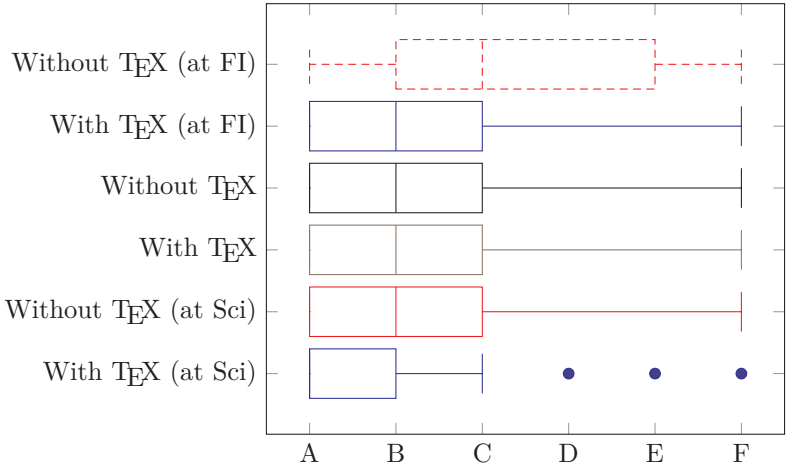


Figure 1: A box plot of the grades of theses written and defended during the period 2010–2015 at FI, Sci and all the faculties of MU with and without TEX

4). Since

$$\sum_{\mathrm{A,B,\ldots,F}} (\mathrm{E}-\mathrm{O})^2/\mathrm{E} = 104.623 \gg 11.07 = \chi^2_{1-\alpha}(5) \tag{1}$$

the null hypothesis h_1 was refused and it was concluded that the grades are indeed differently distributed on the significance level α.

Having shown that the distribution of grades awarded to theses written using and not using TEX is different, the author proceeded to test if this holds for individual grades. The null hypothesis h_A was supposed that the distribution of grade A being awarded to theses written using and not using TEX is equivalent. The two-tailed Mann-Whitney U test (Mann and Whitney, 1947) was applied to the observations of grade A being and not being awarded to theses written using and not using TEX:

$$\begin{aligned}
m_1 &= 15\,476 && \text{(Without TEX (grade A))}\\
m_2 &= 1\,181 && \text{(With TEX (grade A))}\\
n_1 &= 42\,183 && \text{(Without TEX (total))}\\
n_2 &= 2\,692 && \text{(With TEX (total))}\\
U_1 &= m_1(m_2 \cdot 0.5 + (n_2 - m_2)) + (n_1 - m_1)((n_2 - m_2) \cdot 0.5) && (2)\\
&= 52\,699\,952.5\\
U_2 &= m_2(m_1 \cdot 0.5 + (n_1 - m_1)) + (n_2 - m_2)((n_1 - m_1) \cdot 0.5) && (3)\\
&= 60\,856\,683.5\\
U &= \min(U_1, U_2) = U_1 = 52\,699\,952.5 && (4)
\end{aligned}$$

Since $n_1 n_2 \gg 20$, $U \sim \mathrm{N}\left(\frac{n_1 n_2}{2}, \frac{n_1 n_2 (n_1+n_2+1)}{12}\right)$. After normalization to

$$\mathrm{N}(0,1) \sim z = \frac{U - \frac{n_1 n_2}{2}}{\sqrt{\frac{n_1 n_2 (n_1+n_2+1)}{12}}} \approx -\frac{4\,078\,365.5}{651\,662.46} \approx -6.258 \tag{5}$$

the two-tailed p-value β was computed as follows:

$$\begin{aligned}
&\arg\min_\beta P(\Phi^{-1}_{\beta/2} \le z \le \Phi^{-1}_{1-\beta/2}) = \beta && (6)\\
&\iff \Phi^{-1}_{\beta/2} = -6.258 \iff \beta/2 = 1 - \Phi(6.258) \iff \beta \approx 0
\end{aligned}$$

Since $\beta < \alpha$ the null hypothesis h_A on the significance level α was refused. Following a similar procedure for grades from B to F, the following conclusions on the significance level α could be made:

- Theses written using TEX had been awarded grade A significantly more often than those not written using TEX.

- Theses written using TEX had been awarded grades C and D significantly less often than those not written using TEX.
- No significant difference was observed in the distributions of grades B, E and F being awarded to theses written using and not using TEX.

A box plot of the grades is shown in Figure 1.

Conclusion

The author has shown that there was a marked and steady increase in the usage of TEX at MU the period during 2010–2014 and that the usage of TEX correlated with statistically significantly better grades during the period 2010–2015.

Literatura

Mann, H. B., Whitney, D. R. On a Test of Whether one of Two Random Variables is Stochastically Larger than the Other. *The Annals of Mathematical Statistics*, 1947, Vol. 18, No. 1, s. 50–60.

Pearson, C. On the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling. *Philosophical Magazine Series 5*, 1900, Vol. 50, Iss. 302, s. 157–175.

Vít Novotný
witiko@mail.muni.cz

Načítání souboru s argumenty v TEXu

Jan Šustek

Abstrakt: Cílem článku je simulovat v TEXu načítání souboru s argumenty z terminálu. Bude představeno několik řešení tohoto problému, od jednodušších po složitější. Přitom si čtenář připomene některá pravidla TEXu, s nimiž se tak často nesetká. Jako ukázku vytvoříme soubor brozura.tex, který po zavolání z terminálu `pdftex brozura kniha` načte dokument kniha.pdf a přerovná jeho strany do souboru brozura.pdf. Ten pak lze vytisknout jako brožuru.

Klíčová slova: TEX, argumenty, brožura, scanargs.

1 Popis problému

Běžně program TEX voláme s argumentem, kterým je název vstupního souboru. Když zavoláme

```
1 tex vstup
```

bude TEX zpracovávat vstupní soubor vstup.tex. Cílem článku je nasimulovat načítání vstupního souboru s „jeho“ argumenty, například

```
2 tex vstup arg1 arg2
```

Ukažme si nejprve, co se stane, když TEX zavoláme jako na řádku 2. Znaky

```
3 vstup arg1 arg2
```

jsou TEXu poslány jako nultý vstupní řádek. Input procesor tyto znaky načte jako[1]

```
4 \input vstup arg1 arg2§
```

kde § je znak s hodnotou `\endlinechar`.

Vstupní soubor obvykle obsahuje makro `\end{document}` nebo `\bye`, případně jinou ukončovací značku. Když hlavní procesor TEXu dostane příkaz k ukončení, přestane číst další vstupní text a po nezbytných formalitách ukončí svoji činnost. Ke znakům

```
5 arg1 arg2§
```

se tak TEX nedostane.

[1] Příkaz `\input` se provede pouze interně a token `input` ve čtecí frontě ve skutečnosti nebude.

 DOI: 10.5300/2015-1-2/86

2 Návrh řešení

Aby se TEX k uvedeným znakům dostal, nesmí se hlavní procesor při čtení vstupního souboru dostat k žádnému ukončovacímu příkazu. V takovém případě se vstupní soubor celý přečte a čtecí fronta bude pokračovat znaky na řádku 5. TEX se ale nějak ukončit musí, proto do vstupního souboru musíme ukončovací příkaz vložit trikově a musíme zajistit, aby se k němu hlavní procesor dostal až po přečtení znaků na řádku 5.

Základní myšlenka je jednoduchá – napsat na konec vstupního souboru řádky

```
6 \catcode\endlinechar=2
7 \afterassignment\bye%
8 \def\ARGS{%
```

Nastavení `\catcode` způsobí, že se znak `§` na řádku 5 promění na token $\boxed{\S}_2$ a bude se chovat jako `}`. Nastavení `\afterassignment` způsobí, že se po přiřazení `\def\ARGS` vloží do čtecí fronty token $\boxed{\texttt{bye}}$. Pokud na konec vstupního souboru napíšeme řádky 6–8, bude čtecí fronta po zavolání řádku 2 a provedení řádku 8 vypadat následovně:

```
9 \def\ARGS{arg1 arg2}\bye
```

A máme, co jsme chtěli. Dostali jsme se k argumentům a trikově jsme ukončili běh TEXu.[2]

Problém je, že toto jednoduché řešení nemůže fungovat. Důvodem je konec vstupního souboru uvnitř definice makra `\ARGS`. TEX totiž při ukončení načítání souboru testuje mimo jiné, zda právě nenačítá definici makra. Pokud k tomu dojde, TEX předpokládá, že je to chyba uživatele, který zapomněl napsat uzavírací závorku za definicí. V tom případě TEX nahlásí chybu a ukončí načítání definice.

Řešením je ukončit načítání souboru před zahájením definice. K tomu se použije příkaz `\noexpand` na úplném konci souboru. Příkaz `\noexpand` načte ze čtecí fronty jeden token. Jelikož okamžitě za příkazem končí soubor, je čtení souboru ukončeno a čtecí fronta pokračuje znaky na řádku 5. Příkaz `\noexpand` tak přečte token $\boxed{\texttt{a}}_{11}$. Je víceméně vedlejší, že se tím tokenu přiřadí příznak *no_expand_flag*. Důležité je, že k tomu dojde před zahájením definice. TEX pak čte tokeny $\boxed{\texttt{def}}$ $\boxed{\texttt{ARGS}}$ $\boxed{\{}_1$ $\boxed{\texttt{a}}_{11}$ z paměti a během čtení definice nenarazí na konec souboru. Zbývá zajistit, aby se TEX k příkazu `\noexpand` skutečně dostal dříve než k příkazu `\def`. K tomu poslouží příkaz `\expandafter`. Funkční řešení na posledních řádcích vstupního souboru vypadá následovně.

```
10 \catcode\endlinechar=2
11 \afterassignment\bye%
12 \expandafter\def\expandafter\ARGS\expandafter{\noexpand%
```

[2] Řádky 7 a 8 je nutné zakončit procenty, aby token procesor nevytvořil token $\boxed{\S}_2$ i na koncích těchto řádků.

Jako příklad vytvoříme soubor vypis.tex, který po zavolání

```
13 tex vypis nejake argumenty
```

vypíše na terminál

```
14 Argumenty jsou nejake argumenty.
```

Rozdíl oproti řádkům 10–12 bude v tom, že se po přiřazení neukončí běh TEXu, ale že se zavolá makro `\RUN`, které vypíše řádek 14 a až poté ukončí běh TEXu. Připomínám, že `\bye` je `\outer` makro, a proto se token `bye` nesmí vyskytnout uvnitř definice jiného makra.

```
15 \def\RUN{\message{Argumenty jsou \ARGS.}
16   \csname bye\endcsname}
17 \catcode\endlinechar=2
18 \afterassignment\RUN%
19 \expandafter\def\expandafter\ARGS\expandafter{\noexpand%
```

Všimněme si, že pokud budou argumenty obsahovat nějaká makra, vypíšou se tato po úplné expanzi. Čtenář si lehce domyslí, jak tuto expanzi při výpisu potlačit.

3 Využití registrů `\toks`

Řádky 17–19 nevypadají z uživatelského pohledu příliš přívětivě. Lépe by vypadalo, kdybychom si nadefinovali makro `\SCAN` a uživatel by pak na konci souboru psal pouze

```
20 \SCAN
```

Zde ale narazíme na další problém, a sice na nepárovou závorku na řádku 19, která se uvnitř definice makra `\SCAN` nesmí vyskytnout. Problém lze vyřešit pomocí registru `\toks`, který místo otevírací závorky umožňuje použít token `bgroup`, jenž nepodléhá kontrole na párování závorek. V následujícím řešení je navíc nastavení `\catcode` umístěno do skupiny, aby nemohlo způsobit další komplikace v případě uživatelovy neopatrnosti.[3]

```
21 \def\SCAN{\begingroup \catcode\endlinechar=2
22   \afterassignment\RUN
23   \toks0=\expandafter\bgroup\noexpand}
24 \def\RUN{\xdef\ARGS{\the\toks0}\endgroup
25   \message{Argumenty jsou \ARGS.}
26   \csname bye\endcsname}
27 \SCAN%
```

[3] Na konci řádku 22 už procento být nemusí, protože ke změně `\catcode` dojde až na řádku 27, kdy už je konec řádku 22 dávno zpracován.

Použití `\xdef` na řádku 24 neprovede úplnou expanzi argumentů, ale úplnou expanzi `\the\toks0`. Jejím výsledkem jsou neexpandované tokeny přesně tak, jak je zadal uživatel na terminálu.

Vadou na kráse ovšem stále zůstává znak `%` na řádku 27, který je v tuto chvíli nutný, aby se potlačil znak konce řádku 27, ze kterého by vznikl token §$_2$. Tím by se ale ukončilo načítání `\toks0` a k argumentům z terminálu bychom se nedostali.

V následujícím řešení tohoto faktu využijeme a odstraněním procenta ukončíme přiřazení do `\toks0`. Na toto místo totiž můžeme vložit další token pomocí `\afterassignment`. A expanzí tohoto tokenu analogicky načteme argumenty z terminálu. Takto dostáváme nový soubor vypis.tex, který už uvedenou vadu na kráse nemá.

```
\def\SCAN{\begingroup \catcode\endlinechar=2
  \afterassignment\SCANc
  \toks0=\bgroup}
\def\SCANc{\afterassignment\RUN
  \toks2=\expandafter\bgroup\noexpand}
\def\RUN{\xdef\ARGS{\the\toks2}\endgroup
  \message{Argumenty jsou \ARGS.}
  \csname bye\endcsname}
\SCAN
```

Pro pochopení se podrobně podívejme, jak se bude postupně měnit čtecí fronta na řádku 36, pokud zavoláme řádek 13. Text na následujících řádcích, který není v rámečcích, je text již přečtený input procesorem, ale zatím nedotčený token procesorem.

```
\SCAN§nejake argumenty§
SCAN §nejake argumenty§
begingroup catcode endlinechar =12 212 ␣10
    afterassignment SCANc toks 012 =12 bgroup §nejake argumenty§
toks 012 =12 bgroup §2 nejake argumenty§
SCANc nejake argumenty§
afterassignment RUN toks 212 =12
                  expandafter bgroup noexpand nejake argumenty§
toks 212 =12 bgroup n11 e11 j11 a11 k11 e11 ␣10
                                a11 r11 g11 u11 m11 e11 n11 t11 y11 §2
RUN
```

4 Příprava brožury

V dalším příkladu, inspirovaném makry Petra Olšáka (2016), vytvoříme užitečný soubor brozura.tex, který po zavolání

```
pdftex brozura kniha
```

načte dokument kniha.pdf a přeuspořádá jeho strany do souboru brozura.pdf, jehož oboustranným vytištěním a přeložením dostaneme brožuru.

Základ souboru je stejný jako v předcházejících příkladech. Rozdíl je jen v makru `\RUN`. V něm se nejprve příkazem `\pdflastximagepages` zjistí počet stránek souboru kniha.pdf a pak se v cyklu po jednoduchém výpočtu vloží příslušné stránky do souboru brozura.pdf. Je použit příkaz `\shipout`, kterým se obejde algoritmus stránkového zlomu i output rutina a bez nutnosti nastavování záhlaví a zápatí se příslušný box přímo vysází do souboru brozura.pdf.

Soubor brozura.tex vypadá následovně.

```
\def\SCAN{\begingroup \catcode\endlinechar=2
  \afterassignment\SCANc
  \toks0=\bgroup}
\def\SCANc{\afterassignment\RUN
  \toks2=\expandafter\bgroup\noexpand}
\newcount\aL \newcount\aR \newcount\bL \newcount\bR
\def\RUN{\xdef\nazev{\the\toks2}\endgroup
  \pdfpagewidth=297mm \pdfpageheight=210mm
  \pdfhorigin=0pt \pdfvorigin=0pt
  \pdfximage{\nazev.pdf}
  \aL=\pdflastximagepages
  \advance\aL by3 \divide\aL by4 \multiply\aL by4
  \aR=1 \bL=2 \bR=\aL \advance\bR by-1
  \loop
    \shipout\hbox{\strana\aL \strana\aR}
    \shipout\hbox{\strana\bL \strana\bR}
    \advance\aR by2 \advance\aL by-2
    \advance\bR by-2 \advance\bL by2
  \ifnum\aL>\aR
  \repeat
  \csname end\endcsname}
\def\strana#1{%
  \ifnum#1>\pdflastximagepages
    \hbox to.5\pdfpagewidth{\hss}%
  \else
    \pdfximage width.5\pdfpagewidth page#1 {\nazev.pdf}%
    \pdfrefximage\pdflastximage
  \fi}
\SCAN
```

Pokud se soubor brozura.tex umístí do adresářové struktury TEXu, může uživatel pohodlně psát řádek 45 přímo v adresáři, v němž se nachází soubor kniha.pdf a v němž se pak vytvoří soubor brozura.pdf.

Soubor brozura.tex by se samozřejmě mohl vylepšit. Mohl by například testovat, zda náhodou uživatel nezadal název souboru včetně přípony .pdf, zda zadaný soubor existuje, a podobně. To ale není náplní tohoto příkladu.

5 Řešení v balíčku

Cílem této části je uložit dříve definovaná makra do balíčku scanargs.tex, aby mohl uživatel zprovoznit načítání argumentů z terminálu pouhým načtením balíčku.

```
... \input scanargs ...
\bye
```

Naše řešení spočívá v tom, že se zevnitř balíčku scanargs.tex načte zbytek hlavního souboru až po závěrečné `\bye`, pak se uloží argumenty z terminálu a vrátí se k načtenému textu. Problém ale je, že pokud text jednou načteme, není pak možné v něm měnit kategorie znaků, což občas uživatelé potřebují. Dále není možné načíst text, který obsahuje nějaké `\outer` makro, například `\newcount`.

Toto lze obejít dvouprůchodovým načítáním hlavního souboru. Balíček scanargs.tex změní kategorie speciálních znaků na 12 a zbytek hlavního souboru přeskočí. Přeskakované znaky bude načítat jako verbatim a jejich separátorem bude `\bye`, konkrétně tokeny $\boxed{\backslash}_{12}$ $\boxed{\texttt{b}}_{11}$ $\boxed{\texttt{y}}_{11}$ $\boxed{\texttt{e}}_{11}$. Dále se načtou a uloží argumenty z terminálu. Pak se znovu načte hlavní soubor a opět jako verbatim se přeskočí začátek souboru až po text `\input scanargs`, konkrétně po tokeny $\boxed{\backslash}_{12}$ $\boxed{\texttt{i}}_{11}$ $\boxed{\texttt{n}}_{11}$ $\boxed{\texttt{p}}_{11}$ $\boxed{\texttt{u}}_{11}$ $\boxed{\texttt{t}}_{11}$ $\boxed{\sqcup}_{12}$ $\boxed{\texttt{s}}_{11}$ $\boxed{\texttt{c}}_{11}$ $\boxed{\texttt{a}}_{11}$ $\boxed{\texttt{n}}_{11}$ $\boxed{\texttt{a}}_{11}$ $\boxed{\texttt{r}}_{11}$ $\boxed{\texttt{g}}_{11}$ $\boxed{\texttt{s}}_{11}$. Tím činnost balíčku scanargs.tex končí a zbytek hlavního souboru se zpracuje obvyklým způsobem, včetně koncového `\bye`.

V následujícím výpisu souboru scanargs.tex se navíc na řádcích 87–88 a 91–105 zjišťuje počet argumentů oddělených mezerou a po jednom je ukládá do paměti. Počet argumentů je pak uložen v registru `\ARGSCOUNT` a i-tý argument lze získat zavoláním `\ARG[`i`]`. Při počítání argumentů je jako pomocný separátor použit token $\boxed{\texttt{\^{}\^{}A}}_{8}$, jehož použití v argumentech je velmi nepravděpodobné.

```
\def\SCAN{\begingroup
  \def\do##1{\catcode`##1=12}\dospecials\SCANa}
{\catcode`\|=0 \catcode`\\=12 |long|gdef|SCANa#1\bye{|SCANb}}
\def\SCANb{\endgroup
  \begingroup\catcode\endlinechar=2
  \afterassignment\SCANc
  \toks0=\bgroup}
```

```
\def\SCANc{\afterassignment\RUN
  \toks2=\expandafter\bgroup\noexpand}
\def\RUN{\xdef\ARGS{\the\toks2}\endgroup
  \expandafter\expandafter\expandafter\SAVE
    \expandafter\ARGS\space^^A%
  \begingroup\def\do##1{\catcode`##1=12}\dospecials
  \expandafter\SKIP\input\jobname\relax}
\newcount\ARGSCOUNT
\def\SAVE#1 #2^^A{%
  \def\next{#1}%
  \ifx\next\empty
  \else
    \advance\ARGSCOUNT1
    \expandafter\def\csname ARG[\the\ARGSCOUNT]\endcsname{#1}%
  \fi
  \def\next{#2}%
  \ifx\next\empty
    \let\next\relax
  \else
    \def\next{\SAVE#2^^A}%
  \fi
  \next}
\def\ARG[#1]{\csname ARG[#1]\endcsname}
{\catcode`\|=0 \catcode`\ =12 \catcode`\\=12
|long|gdef|SKIP#1\input scanargs{|endgroup}}
\expandafter\SCAN\noexpand
```

Balíček scanargs.tex lze načíst víceméně na libovolném místě před prvním použitím makra `\ARG` nebo `\ARGSCOUNT`. V případě, že by uživatel definoval makro se stejným jménem, jaké má nějaké pomocné makro v balíčku scanargs.tex, nevadí to, pokud makro definuje až po načtení balíčku.

Balíček scanargs.tex bude fungovat, jestliže bude hlavní soubor končit řádkem 76 s nepotlačeným koncem řádku. Bude fungovat i v případě, že za řádkem 76 budou prázdné řádky s komentářem. Pokud ale za řádkem 76 bude následovat prázdný řádek bez komentáře, vznikne z něj token $\boxed{\S}_2$, a argumenty se z terminálu nenačtou.

6 Ukázka použití balíčku

V následujícím jednoduchém příkladu si ukážeme použití balíčku scanargs.tex. Vytvoříme dokument, ve kterém budou vypsány jednotlivé argumenty, a to jak v expandovaném, tak neexpandovaném tvaru.

```
\input opmac
\input scanargs
\rightskip0ptplus1fil
\def\bezzacatku#1>{}
Počet argumentů: \the\ARGSCOUNT\par
\tmpnum=0
\loop
  \ifnum\tmpnum<\ARGSCOUNT \advance\tmpnum1
  Argument~\the\tmpnum: \ARG[\the\tmpnum]\csname par\endcsname
  Argument~\the\tmpnum\ bez expanze: {\tt
   \expandafter\expandafter\expandafter\expandafter
   \expandafter\expandafter\expandafter\bezzacatku
   \expandafter\expandafter\expandafter\meaning
     \ARG[\the\tmpnum]}
   \csname par\endcsname
\repeat
\bye
```

7 Problém nula argumentů

Nyní se zaměříme na související problém, který se autorovi tohoto článku nepodařilo vyřešit. Týká se všech dříve uvedených způsobů načítání argumentů.

Předpokládejme, že uživatel na terminálu načte soubor vypis.tex a nezadá argumenty, tj. zavolá TEX pouze

```
tex vypis
```

Rozeberme si podrobně tuto situaci. Čtecí fronta se bude vyvíjet následovně.

```
\input vypis§
```

Příkaz `\input` si vyžádá tokeny, ze kterých poté složí název souboru.

v$_{11}$ ypis§
v$_{11}$ y$_{11}$ pis§
v$_{11}$ y$_{11}$ p$_{11}$ is§
v$_{11}$ y$_{11}$ p$_{11}$ i$_{11}$ s§
v$_{11}$ y$_{11}$ p$_{11}$ i$_{11}$ s$_{11}$ §

Znak `\endlinechar` má v tuto chvíli (na začátku běhu TEXu) kategorii 5, proto se znak § promění na token ␣$_{10}$.

v$_{11}$ y$_{11}$ p$_{11}$ i$_{11}$ s$_{11}$ ␣$_{10}$

Token s ASCII-hodnotou 32 slouží jako oddělovač názvu souboru a je hlavním procesorem pohlcen. Čtecí fronta tak zůstane prázdná a důležitý token §$_2$ se

nevytvoří. Pak se načte soubor vypis.tex. Makra na konci tohoto souboru očekávají, že ve čtecí frontě ještě nějaké znaky jsou. To však není pravda, a proto TEX vypíše na terminál hvězdičku a tím uživatele vyzve ke vložení znaků.

Uživatel pak píše znaky úplně stejně, jako by je psal na konci řádku 127. To znamená, že se za tyto znaky automaticky vloží token $\boxed{\S}_2$. Tím se ukončí načítání `\toks2`, za nímž se již pak makro `\RUN` vykoná obvyklým způsobem.

Pokud na výzvu TEXu uživatel žádné znaky nenapíše a pouze stiskne enter, pak bude ve čtecí frontě pouze znak `§` vložený input procesorem. Z tohoto znaku se stane token $\boxed{\S}_2$ a vše bude opět v pořádku. Registr `\toks2` a makro `\ARGS` budou prázdné.

Aby uživatel nebyl zmaten hvězdičkou vypsanou na terminálu, bylo by dobré, kdyby existoval způsob, jak za běhu TEXu zjistit, zda je čtecí fronta prázdná, nebo zda ještě něco obsahuje. Autorovi článku se však takový způsob najít nepodařilo.

Literatura

Olšák, P. *TEX pro pragmatiky.* Brno: CSTUG, v tisku. 150 s.
ISBN 978-80-901950-1-1.

Summary: Reading Files with Arguments in TEX

The paper shows a possibility in TEX how to input a file with arguments from the terminal. There are simple solutions and also more complicated solutions with several advanced rules of TEX explained. As an application there is file brochure.tex. The user inputs this file by `pdftex brochure book`. Then TEX reads the document book.pdf and reorders its pages into file brochure.pdf so that it can be printed as a brochure.

Key words: TEX, arguments, brochure, scanargs.

Jan Šustek, jan.sustek@osu.cz
Ostravská univerzita, Přírodovědecká fakulta, Katedra matematiky
30. dubna 22, CZ-701 03 Ostrava, Czech Republic

Informace o Zpravodaji CSTUG a pokyny autorům

Zpravodaj CSTUG je recenzovaný vědecký a odborný časopis s původními pracemi vycházející od r. 1991. Od roku 1997 je tištěná verze opatřena číslem ISSN 1211-6661. Od roku 2002 vychází časopis i elektronicky, on-line verzi bylo přiděleno ISSN 1213-8185. Periodikum je vydáváno v Brně.

Profil periodika: Zpravodaj CSTUG se zaměřuje na problematiku typografického systému TEX, jeho implementaci a vývoj, na jeho následníky (LATEX, pdfTEX, XETEX, LuaTEX aj.) a na příbuzné a podpůrné programové vybavení. Dále se též zabývá problematikou typografie obecně, zpracováním textu, sazbou a souvisejícími tématy, s cílem podporovat využití systému, zejména mezi českými a slovenskými uživateli. Přijímají se též texty z příbuzných oblastí, jako jsou například knižní produkce, nakladatelská a vydavatelská činnost, polygrafická produkce, technologie apod.

Zpravodaj CSTUG slouží ke sdílení nových poznatků, postupů a metod. Uveřejňuje původní práce, projektová řešení a práce přehledové i osvětové.

Časopis nabízí publikační příležitost jak renomovaným odborníkům, tak i začínajícím autorům. Uveřejněny mohou být rovněž i zkrácené verze závěrečných prací stejně jako výstupy projektů grantových agentur.

Pokyny pro autory: Redakce vítá široké spektrum *článků* spadajících do kategorií: teoretické stati, empirické studie a projektová řešení či přehledové práce. Přijímá též *krátké texty* k aktuálním tématům, jako jsou zprávy, informace o dění ve sdružení, o nových knihách, recenze apod., zejména recenze typografických knih a softwaru.

Články se přijímají v jazyce českém či slovenském, případně anglickém. Články jsou strukturovány do podkapitol a sestávají z částí odpovídajících obvyklé struktuře článku (úvod, stav řešené problematiky, použité postupy a metody, výsledky a interpretace, diskuse a závěry); autor se však může od této struktury odchýlit s ohledem na charakter svého díla.

Nezbytnými součástmi článků jsou: *název článku, abstrakt* (v délce max. 1 200 znaků včetně mezer) *a klíčová slova*, vždy v českém, nebo slovenském jazyce a v jazyce anglickém; *jméno, příjmení, e-mail a poštovní adresa* autora nebo jeho pracoviště; dále *seznam použité literatury a odkazy* na tuto literaturu v textu článku. V seznamu použité literatury se mohou uvést i překlady názvů děl do angličtiny. Redakce je oprávněna chybějící anglické části doplnit bez účasti autora.

Původní články většího rozsahu se doporučuje zakončit podrobnějším, zpravidla dvoustránkovým, souhrnem v anglickém jazyce a dle uvážení i obsahem.

Články či projekty, financované z grantových zdrojů, jako jsou Grantová agentura České republiky (GAČR), Fond rozvoje vysokých škol (FRVŠ), Evropský sociální fond (ESF), Interní grantová agentura Československého sdružení uživatelů TEXu, univerzitní interní grantové agentury aj., musí obsahovat údaj o financování z příslušného projektu.

Jednotlivé články procházejí recenzním řízením typu „peer-review“. Články jsou recenzovány vždy dvěma odborníky na dané téma. Recenzenti pocházejí z jiné instituce, než je pracoviště autora článku. Recenzent, až na odůvodněné výjimečné případy, není členem redakční rady. Redakční rada zachovává recenzenty v anonymitě a vyhrazuje si právo článek odmítnout.

Ostatní příspěvky (úvodník, diskuse, zprávy, pozvánky, recenze atd.) posuzuje redakční rada. Podrobnější informace se nacházejí na `http://www.cstug.cz/pokyny/`.

Texty k publikaci ve Zpravodaji se přijímají ve spisovném jazyce, dle členění podle stylové příznačnosti, ve vrstvě knižní či neutrální, vždy však s dodržením správné odborné terminologie.

Veškeré texty procházejí jazykovou korekturou a je-li potřeba, i jazykovou úpravou. Redakční rada může dodaný rukopis jazykově či formálně upravit. Zásahy dle nejlepšího vědomí a svědomí však redakční rada diskutuje s autory.

Odevzdáním článku redakci autor prohlašuje, že se jedná dosud nepublikované dílo, a souhlasí se zveřejněnými pokyny pro autory a zavazuje se jimi řídit, stejně jako případnými dalšími pokyny redakční rady Zpravodaje CSTUG. Odevzdáním článku autor dále prohlašuje že dílo nebylo nabídnuto jinému časopisu a že bez souhlasu vydavatele Zpravodaje CSTUG tak ani neučiní, a – v případě přijatého odborného článku – se zavazuje po dobu jednoho roku od vydání článku ve Zpravodaji CSTUG dílo nepublikovat v žádné formě včetně vystavení na internetu, leda by k tomu vydavatel Zpravodaje CSTUG předem udělil souhlas.

Šablonu v TEXu pro autory pro snazší přípravu zdrojového textu článku naleznete na `http://bulletin.cstug.cz/`.

Složení redakční rady je uváděno v tiráži každého čísla a dále na webu Zpravodaje CSTUG, viz `https://www.cstug.cz/redakcni-rada/`.

Kontaktovat redakční radu můžete prostřednictvím e-mailu `zpravodaj@cstug.cz`, pro kontakt s **administrací sdružení** slouží `secretary@cstug.cz`.

Zpravodaje CSTUG vydává od jeho založení Československé sdružení uživatelů TEXu, z. s., IČ 005 36 580. Evidenční číslo registrace vedené Ministerstvem kultury dle zákona č. 46/2000 Sb. je E 7629. Bližší informace o vydavateli naleznete na webu sdružení `https://www.cstug.cz/kontakty`.

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
Ilustrace na obálce:	Ondřej Kutal
Počet výtisků:	400
Uzávěrka:	15. 7. 2015
Odpovědný redaktor:	Zdeněk Wagner
Redakční rada:	Zdeněk Wagner (šéfredaktor), Ján Buša, Jiří Demel, Tomáš Hála, Jaromír Kuben, Michal Růžička, Jiří Rybička, Petr Sojka, Jan Šustek
Technický redaktor:	Jan Šustek
Tisk a distribuci zajišťuje:	KONVOJ, spol. s r. o., Berkova 22, 612 00 Brno, konvoj@konvoj.cz
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