

# Zpravodaj Československého sdružení uživatelů TeXu

---

Přemysl Šedivý; Miroslav Brož; Jana Gřondilová; Michal Píše; Karel Houfek

Kreslíme METAFONTem

*Zpravodaj Československého sdružení uživatelů TeXu*, Vol. 8 (1998), No. 1, 1–63

Persistent URL: <http://dml.cz/dmlcz/149808>

## Terms of use:

© Československé sdružení uživatelů TeXu, 1998

Institute of Mathematics of the Czech Academy of Sciences provides access to digitized documents strictly for personal use. Each copy of any part of this document must contain these *Terms of use*.

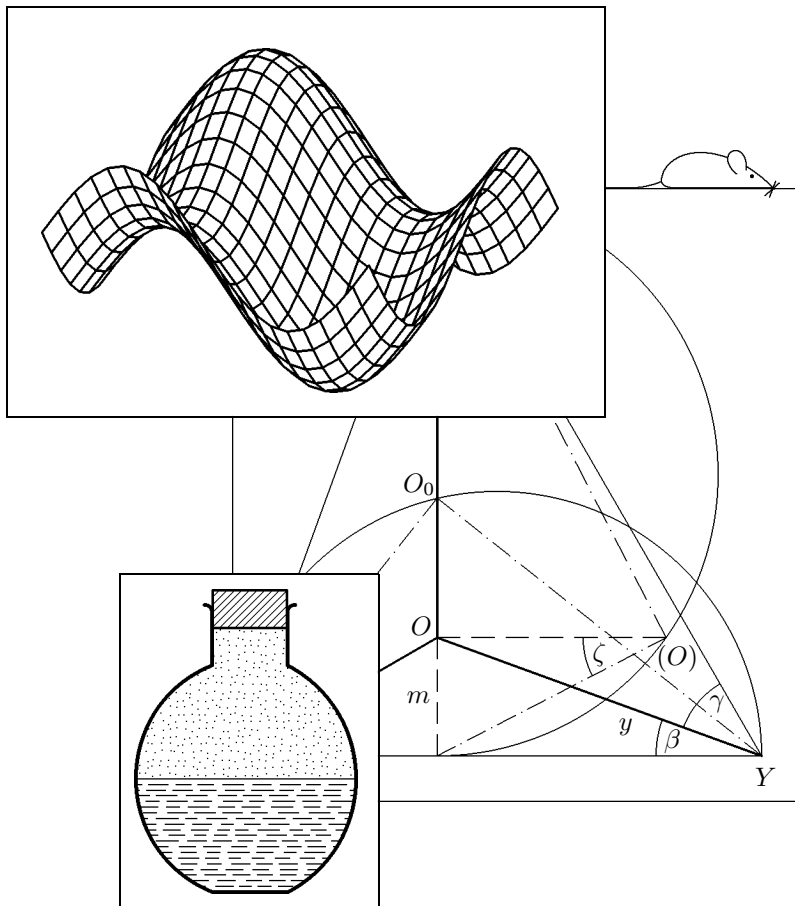


This document has been digitized, optimized for electronic delivery and stamped with digital signature within the project *DML-CZ: The Czech Digital Mathematics Library* <http://dml.cz>

---

# KRESLÍME METAFONTEM

---



---

PŘEMYSL ŠEDIVÝ, MIROSLAV BROŽ,  
JANA GŘONDILOVÁ, MICHAL PÍŠE, KAREL HOUFEK

---

# OBSAH

Úvod . . . . .	3
1 První kroky . . . . .	3
2 Body . . . . .	7
3 Problémy s výpočty . . . . .	9
4 Cesty . . . . .	11
5 Afinní transformace . . . . .	13
6 Operace s cestami . . . . .	16
7 Skládání obrázků . . . . .	18
8 Pera, štětce, gummy . . . . .	21
9 Podmínky, cykly . . . . .	25
10 Makra . . . . .	27
11 Grafy funkcí . . . . .	29
12 Náhodná čísla . . . . .	32
13 Kosoúhlé zobrazení . . . . .	34
14 Axonometrické zobrazení . . . . .	35
15 Graf funkce dvou proměnných . . . . .	37
A Práce s celými obrázky . . . . .	39
A.1 Příkaz <code>\special</code> . . . . .	40
A.2 Program <code>bm2font</code> . . . . .	40
A.3 Převod obrázku do formátu <code>pcx</code> . Dávka <code>dvi2pcx</code> . . . . .	42
B Makra . . . . .	43
C Elektrotechnické značky . . . . .	51
Literatura a internetové odkazy . . . . .	58
Rejstřík . . . . .	59

# Úvod

Tento text je určen uživatelům  $\TeX$ u, kteří zvládli sazbu textu a potřebují své dokumenty doplňovat různými schématy, grafy, planimetrickými nebo stereometrickými obrázky, nákresy fyzikálních aparatur a dalšími jednoduchými ilustracemi, které bývají označovány jako „pérovky“, neboť donedávna byly vytvářeny převážně pomocí rýsovacího pera a tuše. Program METAFONT je pro tyto účely jedinečným prostředkem.

Autoři používají instalaci  $\zeta\TeX$ u 95 pro DOS s úpravami z 6. 10. 1997 a dokumenty sázejí ve formátu  $\LaTeX$  2.09. Kapitola 1 a Dodatek A předpokládají stejné podmínky. Velká většina textu však může být užitečná všem uživatelům, ať pracují pod DOSem, či Unixem, píší dokumenty v plainu, či v  $\LaTeX$ u.

V této příručce jsme se snažili shrnout své zkušenosti, které jsme získali během několika let při tvorbě letáků fyzikální olympiády, fyzikálního korespondenčního semináře MFF UK a obrázků do učebnic fyziky.

WWW stránka věnovaná této publikaci má URL

<http://www.cstug.cz/kreslime/>

Zde je možné stáhnout archiv `kreslime.zip`, který obsahuje soubory nutné pro tisk a soubory maker `gjkt.mf`, `znacky.mf`.

V Hradci Králové 1. 11. 1997

Za kolektiv autorů

Přemysl Šedivý  
Gymnázium J. K. Tyla  
502 23 Hradec Králové

## 1 První kroky

METAFONT byl vytvořen jako program pro tvorbu písma. Obrázky, které pomocí něj nakreslíme, budou proto také jen jakási „písmena“ námi vytvořeného „fontu“ a jako s písmeny s nimi můžeme v dokumentu zacházet. Ukažme si to na jednoduchém příkladu.

Do textu, nazvaného `ukazka_1`, který popisuje vlastnosti pravoúhlého trojúhelníka, potřebujeme jeho obrázek. Přejdeme proto do hlavního menu, do sloupce METAFONT, položky `Parameters`. Otevře se okénko, kde jsou vypsány parametry ovlivňující činnost METAFONTu a konvertoru GfToPK:

```
'%MF% file (without extension):'ukazka_1
'METAFONT' &plain \mode=laserjet; mag=1.0; input %MF%
'Converter' gftopk %MF%.300 %MF%.pk
```

Na prvním řádku se nabízí stejný název pro METAFONTový soubor, jaký má právě editovaný textový soubor. Můžeme zvolit jiný název, ale v našem příkladu to neuděláme.

Na druhém řádku je uvedeno, že METAFONT bude využívat makropříkazů obsažených v bázi `plain`. Dále je parametrem `mode=laserjet` zvoleno rozlišení 300 dpi pro laserové a tryskové tiskárny. Pokud používáme tiskárny s rozlišením 180 dpi (24 jehličkové), přepíšeme parametr na `mode=lqlores`, pro rozlišení 240 dpi (9 jehličkové tiskárny) napíšeme `mode=epsonfx`. Parametr `mag=1.0` znamená, že obrázek vygenerujeme v základní velikosti. Změnou číselné hodnoty můžeme dosáhnout zvětšení nebo zmenšení obrázku. Např. `mag=2.0` vede ke zvětšení na dvojnásobek.<sup>1)</sup>

Na třetím řádku musí být uvedena hustota bodů v dpi u vygenerovaného fontu. Vypočítá se jako součin rozlišení a zvětšení (v našem případě 300).<sup>2)</sup>

Když jsme zkontrolovali, případně i upravili parametry, přejdeme do položky **Edit(mf)**, a napíšeme program `ukazka_1.mf` pro nakreslení obrázku:

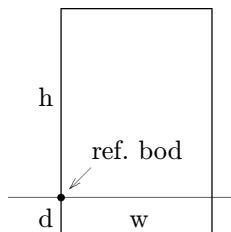
```
mode_setup;
beginchar(1,20mm#,15mm#,0);
pickup pencircle scaled .3mm;
draw(0,0)--(20mm,0)--(0,15mm)--(0,0);
endchar;
end
```

Všimněte si, že v METAFONTu neoznačujeme klíčová slova příkazů zpětným lomítkem `\` jako v  $\TeX$ u. Jiná než klíčová slova se tu totiž nevyskytují. Dále vidíme, že jednotlivé příkazy je nutno oddělovat středníkem.

Příkaz `mode_setup` na prvním řádku nastaví hodnoty délkových jednotek podle parametrů `mode` a `mag`. V METAFONTu používáme dva druhy délkových jednotek. První druh, který slouží k určení rozměru obrázku, odvozujeme od jednotky  $\text{pt} \doteq 0,3515 \text{ mm}$  (tiskový bod). Tyto jednotky nejsou závislé na rozlišení výstupního zařízení. Patří sem

`pt#=1, mm#=2.84528, cm#=28.45276, in#=72.27.`

Příkazem `beginchar(1,20mm#,15mm#,0);`, kterým začíná definice znaku — obrázku, jsme zvolili číslo znaku v kódu ASCII, šířku obrázku, výšku nad účařím a hloubku pod účařím. Rozměry se uloží do proměnných `w`, `h` a `d`. Stejného výsledku bychom dosáhli příkazem `beginchar(1,56.90552,42.67914,0);`, kde bychom rozměry obrázku udali přímo v tiskových bodech.



<sup>1</sup>Parametry druhého řádku jsou předvoleny v dávce `\EMTEX\MNU\mfbat.bat`, kde je můžeme případně upravit podle potřeby.

<sup>2</sup>Parametr třetího řádku je předvolen v souboru `\EMTEX\MNU\texset.bat` na řádku začínajícím `set GFOPT=...`

Druhý druh jednotek, který používáme k určení souřadnic bodů uvnitř obrázku, je závislý na rozlišení výstupního zařízení. Základní jednotkou je pixel (mezibodová vzdálenost rastru bitmapy, kterou uvidíme na obrazovce a kterou vytiskne tiskárna). Při rozlišení 300 dpi platí

`pt=4.1511, mm=11.81102, cm=118.11024, in=300.`

Počátek vztažné soustavy obrázku — *referenční bod* — leží v našem případě, kdy jsme zvolili nulovou hloubku `d`, v levém dolním rohu obrázku. Příkaz `draw(0,0)--(20mm,0)--(0,15mm)--(0,0)`; způsobí nakreslení trojúhelníku jednou čarou, která půjde z levého dolního rohu do pravého dolního rohu, levého horního rohu a zpět. Stejněho výsledku bychom dosáhli příkazem

`draw(0,0)--(236.22,0)--(0,177.17)--(0,0)`;

anebo velmi jednoduše příkazem `draw(0,0)--(w,0)--(0,h)--(0,0)`;

Poměrně dlouhý příkaz `pickup pencircle scaled .3mm`; provádí volbu jakéhosi pomyslného pera, které bude kreslit po naší bitmapě. Tentokrát se jedná o pero s kruhovým hrotem o průměru 0,3 mm. Program pro nakreslení obrázku zakončíme příkazem `endchar`; a celý zdrojový text příkazem `end`. Po dokončení zdrojového textu přejdeme do menu a spustíme program **Metafont**, který zkontroluje, zda jsme neudělali nějakou syntaktickou chybu. Svá zjištění zapíše do souboru `ukazka_1.log`. Je-li vše v pořádku, vytvoří soubory `ukazka_1.300` a `ukazka_1.tfm`.

Nakonec spustíme program **Convertor**, který soubor `ukazka_1.300` převede do souboru `ukazka_1.pk`, a příprava obrázku je skončena.

Po zařazení obrázku do textu je třeba, aby prohlížeč hledal font nejen v adresáři `\EMTEX\FONTS`, ale také v aktuálním adresáři. To lze předvolit v souboru `\EMTEX\DATA\scr300.cnf`. Příslušný řádek by měl vypadat takto:

`+font-files=${DVIDRVFONTS:pixel.1j}\@Rrdpi\,}of{.pk,.px1}`

Podobnou změnu je třeba udělat i v `cnf` souborech pro tiskárnu a prohlížení při jiném rozlišení.

Nyní se můžeme vrátit k editaci v  $\text{\LaTeX}$ u a náš obrázek umístit do textu. Do preambule dokumentu napíšeme příkaz `\font\obr=ukazka_1` zavádějící název námi vytvořeného fontu. (Název fontu nesmí obsahovat číslice, proto jsme museli provést přejmenování.) Obrázek umístíme na zvolené místo v textu vytvořením samostatné skupiny `{\obr\char1}`. Náš první pokus v  $\text{\LaTeX}$ u 2.09 může vypadat třeba takto:

`\documentstyle[czech]{article}`

`\font\obr=ukazka_1`

`\begin{document}`

`\noindent V~pravoúhlém`

`trojúhelníku jsou dvě`

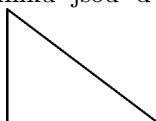
`strany navzájem kolmé:`

`{\obr\char1}`

`\end{document}`

V pravoúhlém trojúhelníku jsou dvě

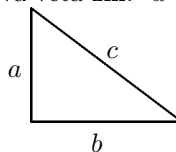
strany navzájem kolmé:



Vidíme, že náš obrázek se v textu zatím chová jako větší písmeno, které je od předcházejícího textu odděleno normální mezerou a které zvětšilo výšku řádku, do kterého je zařazeno. My ovšem obvykle potřebujeme obrázek umístit samostatně a opatřit ho vhodným popisem. Jak toho můžeme dosáhnout, vidíme v další ukázce napsané v  $\text{\LaTeX} 2_{\epsilon}$ :

```
\documentclass{article}
\usepackage{czech}
\font\obr=ukazka_1
\unitlength=1mm
\begin{document}
\noindent Pythagorova věta zní:
\ $a^2+b^2=c^2\$,.$
\begin{center}
\mbox{} \put(-3,6){\a$}
\put(8,-4){\b$}
\put(10,8.5){\c$}
{\obr\char1}
\end{center}
\end{document}
```

Pythagorova věta zní:  $a^2 + b^2 = c^2$ .



Obrázek je od předchozího textu oddělen vložení do prostředí `center`. Příkaz `\mbox{}` slouží k zavedení pomocného referenčního bodu, od kterého příkazem `\put(xr, yr){popis}` vynášíme souřadnice levých dolních bodů jednotlivých detailů popisu a do kterého nakonec umístíme levý dolní bod obrázku. Souřadnice  $x_r$ ,  $y_r$  udáváme pouze číselnými hodnotami. Délková jednotka musí být už dříve, nejlépe v preambuli dokumentu, definována příkazem `\unitlength=míra`.

Souřadnice popisu určíme nejrychleji tak, že nejprve vytvoříme dokument s obrázkem bez popisu, zobrazíme jej v režimu **View**, do referenčního bodu obrázku umístíme střed osového kříže a pomocí něj polohu jednotlivých částí popisu odhadneme a zapíšeme na papír. Pak se vrátíme do editoru, napíšeme potřebné příkazy `\put(xr, yr){popis}` a dokument, tentokrát už s popisem, znovu zobrazíme. Není-li popis umístěn přesně podle našich představ, vrátíme se do editoru a provedeme opravy souřadnic. Takto postupujeme, až jsme s výsledkem spokojeni.

Před vypnutím počítače smažeme už nepotřebné soubory `ukazka_1.300`, `ukazka_1.log` a ponecháme jen zdrojový text obrázku `ukazka_1.mf`, soubor `ukazka_1.tfm`, kde jsou uloženy rozměry obrázku, a soubor `ukazka_1.pk`, což je komprimovaná bitmapa. Úklid nepotřebných souborů lze provést automaticky před opuštěním  $\text{\LaTeX}$ u použitím položky menu `clear(dvi)` v okénku **File**.

## 2 Body

Polohu bodu na obrázku určujeme nejčastěji pomocí uspořádané dvojice souřadnic polohového vektoru  $z_{index}=(x_{index},y_{index})$ . Tento vektor můžeme také chápat jako obraz komplexního čísla v *Gaussově rovině*. Například vektoru  $z_1=(2,1.5)$  přísluší bod o souřadnicích  $x_1 = 2$  pixely,  $y_1 = 1,5$  pixelu. Tyto souřadnice můžeme volat také jako  $xpart\ z_1, ypart\ z_1$ .

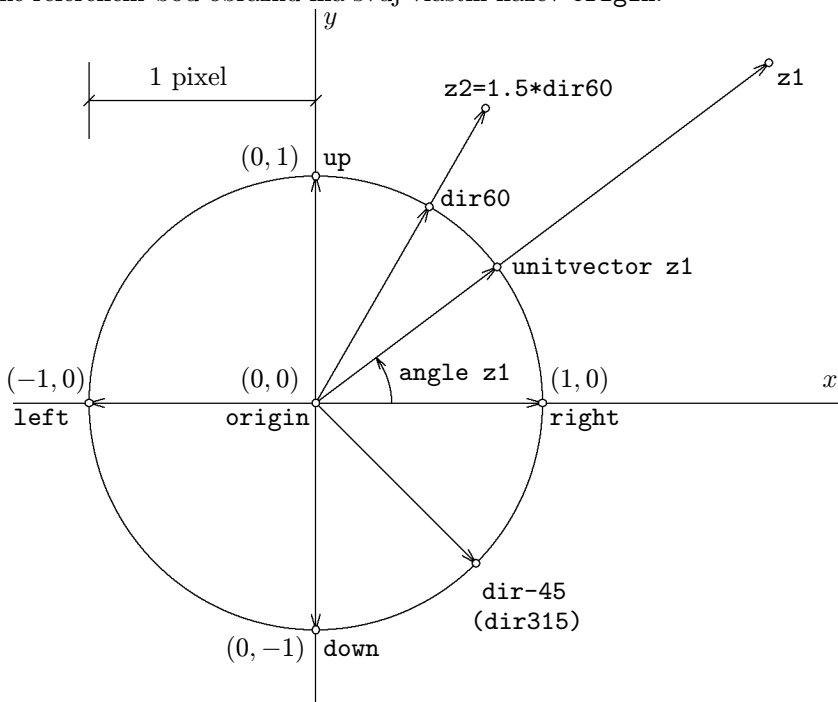
METAFONT umí určit argument komplexního čísla  $angle\ z$  ve stupních, absolutní hodnotu komplexního čísla  $abs\ z$  v pixelech a komplexní jednotku se stejným argumentem  $unitvector\ z$ . Úhly se zásadně vyjadřují ve stupních. Příkazem  $angle$  dostaneme hodnotu argumentu v základním intervalu  $(-180^\circ, 180^\circ)$ , jinak můžeme pracovat s úhly v intervalu  $(-4095^\circ, 4095^\circ)$ . Komplexní jednotku s argumentem  $30^\circ$  zavedeme jako

$$dir30=(cosd30,sind30),$$

kde  $cosd$  a  $sind$  znamenají hodnoty funkcí  $\cos$  a  $\sin$  ve stupňové míře. Často používané komplexní jednotky mají vlastní symboly:

$$right=dir0, left=dir180, up=dir90, down=dir270$$

Také referenční bod obrázku má svůj vlastní název  $origin$ .



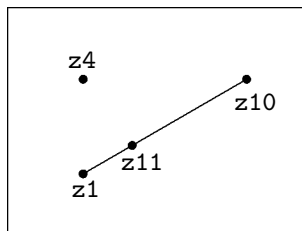
Komplexní číslo můžeme zapsat v goniometrickém tvaru. Například vektor s argumentem  $60^\circ$  a velikostí 1,5 pixelu zadáme jako  $z_2=1.5*dir60$ .



Při kreslení obrázku ovšem obvykle neudáváme souřadnice a velikosti vektorů v pixelech, ale v milimetrech nebo centimetrech. Velmi výhodné je zavedení vlastní jednotky, např. `u`, jejíž velikost mohu podle potřeby změnit a tím případně upravit všechny rozměry obrázku najednou.

Číselné hodnoty souřadnic vektorů můžeme vkládat přiřazovacím příkazem `:=`, který dovede přepsat dříve zavedené hodnoty. Častěji však používáme příkazy, které obsahují samotné rovnítko `=` a mají charakter lineárních rovnic vyjadřujících vektorové vztahy. Takovéto rovnice a jejich soustavy METAFONT automaticky vyřeší a potřebné souřadnice doplní. Použití obou způsobů ilustruje následující příklad:

```
mode_setup;
u#=1mm#;
define_pixels(u); %% provede přepočet dekl. jednotky na pixely
                  %% podle rozlišení výstupního zařízení
beginchar(4,40u#,30u#,0)
z1=(w,0); z2=(w,h); z3=(0,h);
pickup pencircle scaled (.2u);
draw origin--z1--z2--z3--origin;
x1:=10u; y1:=8u; % změna zadání
z10=z1+25u*dir30;
(z11-z1)=.3*(z10-z1);
z4=(x1,y10);
draw z1--z10;
pickup pencircle scaled 1.2u;
for i=1,10,11,4:drawdot z[i];endfor;
endchar;
end
```



Podobně jako v  $\text{T}_{\text{E}}\text{X}$ u můžeme ke zdrojovému textu připisovat poznámky a komentáře začínající znakem `%`. Dále jste si mohli všimnout jednoho ze způsobů, jak METAFONT používá příkaz cyklu `for`, tentokrát s výčtem hodnot proměnné `i`. Z ukázky je také zřejmé, že výrazy `z4` a `z[4]` jsou rovnocenné.

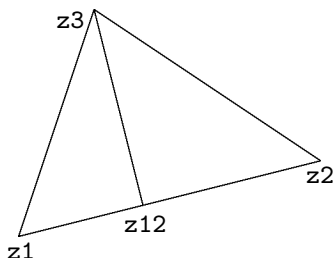
Příkaz  $(z_{11}-z_1)=.3*(z_{10}-z_1)$  určil na spojnici bodů `z1` a `z10` bod s dělicím poměrem 0,3. Stejného výsledku lze dosáhnout příkazem `z11=.3[z1,z10]`. Pomocí dělicího poměru můžeme určit polohu kteréhokoliv bodu přímky. Například `z12=.5[z1,z2]` je střed úsečky `z1 z2`. Bod, který leží za bodem `z1` a je od něj vzdálen desetkrát méně než bod `z2`, určíme jako `z13=(-.1)[z1,z2]` nebo `z13=1.1[z2,z1]`.

Při zadávání polohy bodu pomocí soustavy lineárních rovnic často používáme pomocné neznámé označené jednotně symbolem `whatever`. Například průsečík `z5` přímek `z1 z2` a `z3 z4` určíme příkazem:

```
z5=whatever [z1, z2]=whatever [z3, z4]
```

V dalším příkladu použijeme neznámou `whatever` při určení výšky trojúhelníka:

```
z1=origin;  
z2=(40mm,10mm);  
z3=(10mm,30mm);  
z12=whatever [z1, z2];  
z12=z3+whatever*dir (angle(z2-z1)-90);  
draw z12--z3--z1--z2--z3;  
show z12/mm, abs(z12-z3)/mm;
```



Příkaz `show z12/mm, abs(z12-z3)/mm` způsobí, že během překladu zdrojového textu programem METAFONT se na obrazovce objeví souřadnice bodu `z12` a velikost výšky v milimetrech:

```
>> (16.47069,4.11768)
```

```
>> 26.67892
```

Kdybychom zapomněli napsat `/mm`, dostali bychom hodnoty v pixelech.

### 3 Problémy s výpočty

Numerické proměnné, stejně jako body  $z$  (*index*), není nutno deklarovat. Stačí napsat např. rovnici `a=5`, která určuje hodnotu proměnné `a`<sup>3</sup>). Oproti proměnným typu bod je tu jeden rozdíl. Zavedeme-li souřadnice bodu, např. `z1=(20,0)`, pak tyto souřadnice platí pouze v rámci skupiny vymezené příkazy `beginchar` a `endchar`. Chceme-li v tomtéž souboru vytvořit nový obrázek a v něm opět použít bod `z1` o stejných souřadnicích, musíme znovu vložit rovnici `z1=(20,0)` na rozdíl od proměnné `a`, která bude mít stále hodnotu 5 a můžeme ji dále používat. Chceme-li ale ve druhém obrázku používat proměnnou `a` s jinou hodnotou (např. 7), musíme jí tuto hodnotu přiřadit přiřazovacím příkazem `a:=7`.

Pro operace s čísly METAFONT používá symboly uvedené v tabulce:

---

<sup>3</sup>Při zavádění nových proměnných nezapomeňte, že v proměnných `w`, `d`, `h` jsou uloženy rozměry obrázku.

symbol	příklad	význam
+	a+b	$a + b$
-	a-b	$a - b$
*	a*b	$ab$
/	a/b	$a/b$
**	a**b	$a^b$
++	a++b	$\sqrt{a^2 + b^2}$
+-+	a+-+b	$\sqrt{a^2 - b^2}$
sind	sind a	$\sin a$
cosd	cosd a	$\cos a$
sqrt	sqrt a	$\sqrt{a}$
mexp	mexp a	$e^{a/256}$
mlog	mlog a	256a
max	max(a,b,c)	maximum z množiny $\{a, b, c\}$
min	min(a,b,c)	minimum z množiny $\{a, b, c\}$
round	round 14.5=15	zaokrouhlování
floor	floor 14.7=14	zaokrouhlování směrem dolů
ceiling	ceiling 14.3=15	zaokrouhlování směrem nahoru
mod	a mod b	zbytek čísla $a$ po dělení číslem $b$

Nevýhodou METAFONTu je jeho číselné omezení. Zlomek  $\frac{1}{65536}$  je nejmenší nenulové kladné číslo a značíme jej symbolem *epsilon*. Pro největší číslo, s nímž můžeme v METAFONTu provádět výpočty, používáme symbol *infinity* a jeho hodnota je 4096-*epsilon*, což dává 4095.99998. Při počítání hodnoty výrazu se může stát, že mezivýsledek výpočtu překročí hodnotu *infinity*. Pokud tato hodnota nepřesáhne 32768, je vše v pořádku. Jinak se výpočet přeruší a objeví se chybové hlášení ! *Aritmetic owerflow*. Takovéto nepříjemnosti by vznikaly zvláště při použití Pythagorovy věty, kde by k překročení dovolené hodnoty mezivýsledku došlo například už při výpočtu hodnoty výrazu `sqrt(400**2+300**2)`. Proto je v METAFONTu zavedeno pythagorejské sčítání a odčítání: `400++300` dává výsledek 500, `500+-+300` dává výsledek 400.

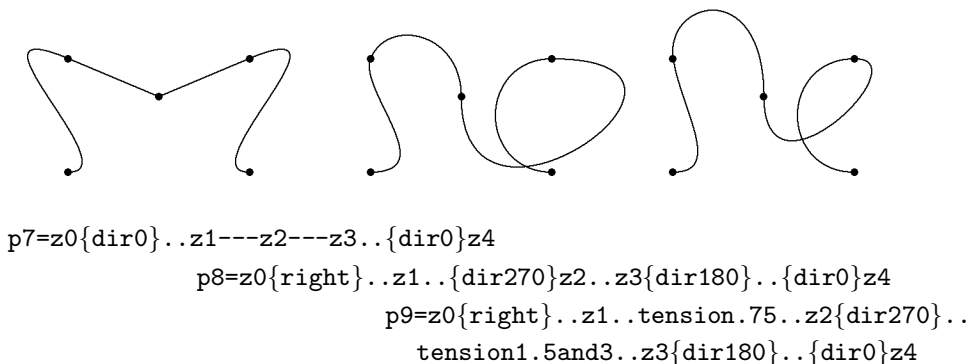
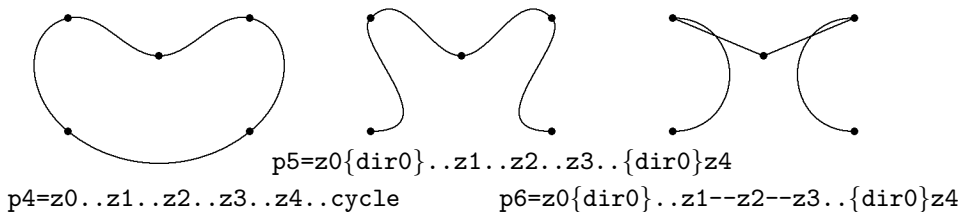
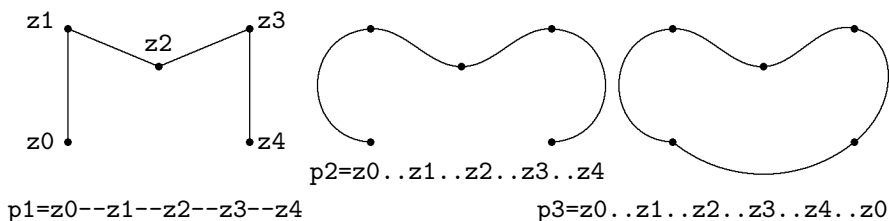
Hodnotu numerických výrazů METAFONT vždy zakrouhluje na nejbližší násobek čísla  $\frac{1}{65536}$ . Například skutečná hodnota čísla .1 je  $\frac{6554}{65536}$  stejně jako hodnota čísla .099999 nebo .10001. Toto zaokrouhlování má za následek, že při složitějších výpočtech můžeme dostat značně nepřesný výsledek. Například hodnota výrazu `2000*(.015**2)` je v METAFONTu .45776, podobně `2.3/.002=1150.63359` nebo `.000001/.004=0`, protože číslo 0,000001 je menší než  $\frac{1}{65536}$  a METAFONT je zaokrouhlí na nulu. Těmto problémům lze většinou předejít vhodným rozšířením zlomku. Například `.2*(1.5**2)=.45` a `2300/2=1150`. Podobně zlomek `.000001/.004` upravíme na `1/4000=.00024`.

Omezením čísel, s nimiž METAFONT počítá, jsou v závislosti na rozlišení omezeny i rozměry obrázku, který chceme vytvořit. Například při rozlišení 300 dpi musí být rozměry obrázku v cm menší než  $\frac{4096}{300} \cdot 2,54 \doteq 35$  cm.

## 4 Cesty

Spojité čáry vytvořené jedním tahem nazýváme cesty. METAFONT chápe cesty jako proměnné typu `path`. Pro některé operace s cestami je vhodné provést po `beginchar` jejich deklaraci, která umožní cesty označit symboly a příkazy napsat přehledným způsobem. Například deklarace `path p[]` umožní pracovat s cestami `p0`, `p1`, `p2`, atd.

METAFONT disponuje celou řadou příkazů, kterými můžeme ovlivnit tvar cesty, což ilustrují následující ukázky:

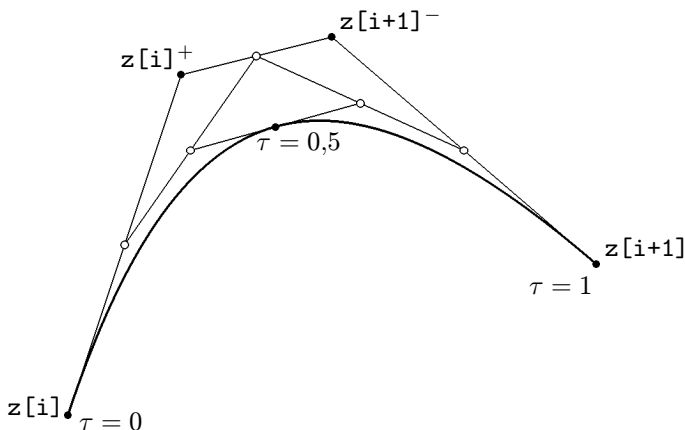


METAFONT vytváří cesty z na sebe navazujících Bézierových křivek. Uvažujme o cestě, která vychází z bodu  $z[0]$ , prochází body  $z[1]$ ,  $z[2]$ , ... a končí v bodě  $z[n]$ . Pro výpočet Bézierovy křivky mezi body  $z[i]$  a  $z[i+1]$  se nejprve zvláštním algoritmem určí dva *kontrolní body*  $z[i]^+$  a  $z[i+1]^-$ . Průběh Bézierovy křivky je pak vypočítán pomocí parametrických vztahů:

$$x(\tau + i) = (1 - i)^3 x_i + 3\tau(1 - \tau)^2 x_i^+ + 3\tau^2(1 - \tau) x_{i+1}^- + \tau^3 x_{i+1},$$

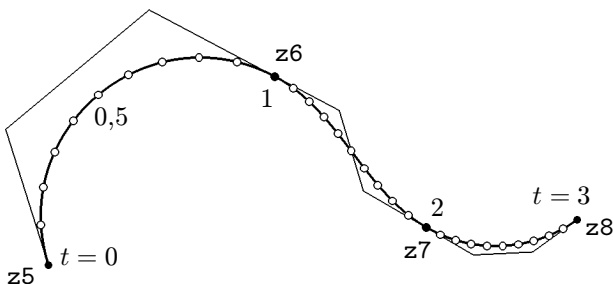
$$y(\tau + i) = (1 - i)^3 y_i + 3\tau(1 - \tau)^2 y_i^+ + 3\tau^2(1 - \tau) y_{i+1}^- + \tau^3 y_{i+1},$$

kde  $0 \leq \tau \leq 1$ . Z těchto vztahů je možno odvodit jednoduchou grafickou konstrukci bodu s parametrem  $\tau = 0,5$ , která je založena na půlení úseček:

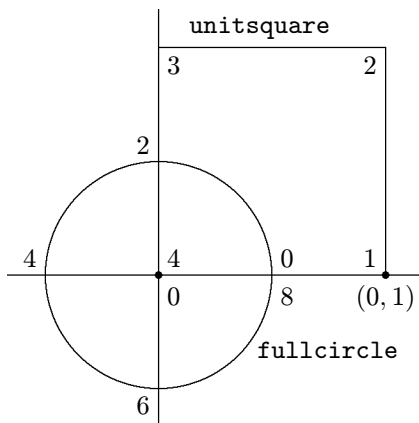
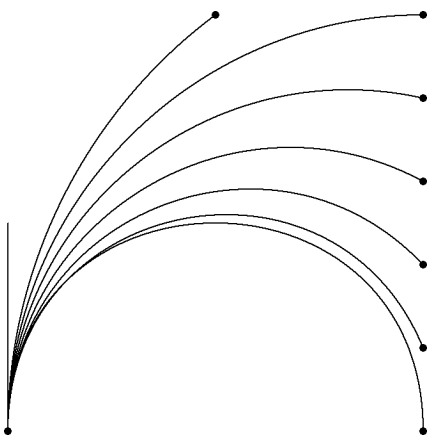


Opakováním konstrukce s využitím kontrolních bodů označených prázdným kroužkem bychom dostali body s parametry  $\tau = 0,25$  a  $\tau = 0,75$  atd. Součet  $t = \tau + i$  se nazývá čas - *time* - a kreslení cesty si představujeme jako děj, který začíná v bodě  $z_0$ , nakreslení jednoho úseku trvá jednotkovou dobu a na konci cesty je celkový čas `length(p)` číselně roven počtu nakreslených úseků. Na následující obrázku je nakreslena cesta definovaná příkazem `p=z5..z6..z7..z8` složená ze tří Bézierových křivek. Na křivce je vyznačen čas po 0,1 jednotky. Lomená čára spojuje kontrolní body, které si program sám vypočítal.<sup>4</sup> Je zřejmé, že každá trojice bodů  $z[i]^-, z[i]$  a  $z[i]^+$  leží v jedné přímce.

<sup>4</sup>Podrobnou informaci o kontrolních bodech získáme příkazem `show p` v souboru s příponou `log`.



Jednoduché cesty definované příkazem typu `p=z1{diruhel}..z2` jsou počítány METAFONTEM tak, aby se co nejméně lišily od kruhového oblouku. To platí zejména pro oblouky se středovým úhlem menším než  $45^\circ$ , kde jsou odchylky hluboko pod rozlišovací schopností obrazovek a tiskáren. Na tom jsou založeny definice cest nazvaných `quartercircle`, `halfcircle` a `fullcircle`, které slouží k aproximaci čtvrtkružnice, půlkružnice a celé kružnice o poloměru 0,5 pixelu. Jsou tvořeny oblouky se středovým úhlem  $45^\circ$ . Cesta `fullcircle` je sestavena z osmi oblouků a čas na ni tedy probíhá v kladném smyslu od 0 do 8.



Jinou důležitou cestou je `unitsquare`, což je čtverec o straně 1 pixel s levým dolním rohem v referenčním bodě. Počátek této cesty je v referenčním bodě a při obíhání v kladném smyslu se čas mění od 0 do 4.

## 5 Afinní transformace

METAFONT umí provést jakoukoliv afinní transformaci v rovině, tj. zobrazení v rovině, ve kterém obrazem přímky je opět přímka, a dvě různé rovnoběžky se opět zobrazují jako různé rovnoběžky. Dělicí poměr na přímce se zachovává.

Transformovat můžeme bod, cestu nebo tvar pera a některé transformace můžeme provádět i s celými obrázky.

Pro sedm nejjednodušších transformací existují zvláštní příkazy, jejichž argumentem je jedno reálné nebo komplexní číslo. Jsou to:

posunutí o vektor  $(a, b)$

$$(x, y) \text{ shifted } (a, b) = (x + a, y + b),$$

stejnolehlost se středem v počátku a koeficientem  $a$

$$(x, y) \text{ scaled } a = (ax, ay),$$

$a$  násobné zvětšení ve směru osy  $x$

$$(x, y) \text{ xscaled } a = (ax, y),$$

$a$  násobné zvětšení ve směru osy  $y$

$$(x, y) \text{ yscaled } a = (x, ay),$$

zkosení ve směru osy  $x$

$$(x, y) \text{ slanted } a = (x + ay, y),$$

otočení okolo počátku o úhel  $\vartheta$

$$(x, y) \text{ rotated } \vartheta = (x \cos \vartheta - y \sin \vartheta, x \sin \vartheta + y \cos \vartheta),$$

stejnolehlost se středem v počátku a koeficientem  $\text{abs}(a, b)$  spojené s otočením o úhel  $\text{angle}(a, b)$

$$(x, y) \text{ zscaled } (a, b) = (ax - by, bx + ay).$$

Jen o málo složitější jsou transformační příkazy se dvěma parametry:

osová souměrnost podle osy určené body  $(a, b)$ ,  $(c, d)$

$$(x, y) \text{ reflectedabout } ((a, b), (c, d)),$$

otočení okolo bodu  $(a, b)$  o úhel  $\vartheta$

$$(x, y) \text{ rotatedaround } ((a, b), \vartheta).$$

Následující obrázky znázorňují jednoduché transformace cesty

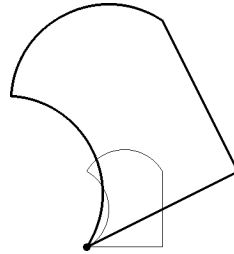
```
p0=origin--(10mm,0)--(10mm,10mm){dir120}..
{dir210}(0,10mm){dir330}..origin;
```



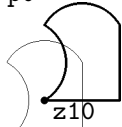
p0



p0 slanted 1/3;



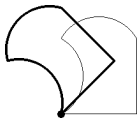
p0 zscaled (2,1);



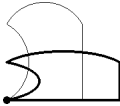
p0 shifted z10;



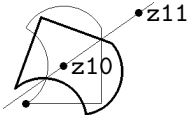
p0 scaled 0.5;



p0 rotated 45;



p0 xscaled 1.5  
yscaled 0.5;



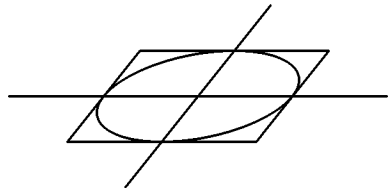
p0 reflected about (z10,z11);



p0 rotated around (z10,45);

Základní transformace můžeme skládat v libovolném pořadí za sebou a vytvořit tak kterékoli afinní zobrazení. Používáme-li takovou složenou transformaci vícekrát, vyplatí se deklarovat pro ni zvláštní označení. Deklaraci složené transformace je nutno zahájit předdefinovanou transformací identity, která sama neudělá nic.

```
transform T;
T=identity xscaled 25mm yscaled 12mm
  slanted .8 shifted (20mm,20mm);
draw fullcircle transformed T;
draw (left--right) transformed T;
draw (up--down) transformed T;
draw unitsquare shifted (-.5,-.5)
  transformed T;
```



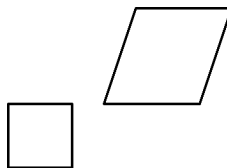
Transformaci můžeme určit tak, že ke třem bodům, které neleží v jedné přímce, zvolíme jejich obrazy. Například:



```

transform T;
(0,0) transformed T =(300,200);
(100,0) transformed T =(500,200);
(0,100) transformed T =(600,500);
show T;
draw unitsquare scaled 100;
draw unitsquare scaled 100 transformed T;

```



Zobrazení bodu příkazem `z1=z transformed T` je popsáno soustavou lineárních rovnic:

$$x_1 = t_x + t_{xx}x + t_{xy}y, \quad y_1 = t_y + t_{yx}x + t_{yy}y.$$

Na příkaz `show T` v předcházející ukázce vypíše METAFONT koeficienty transformačních rovnic jako vektor

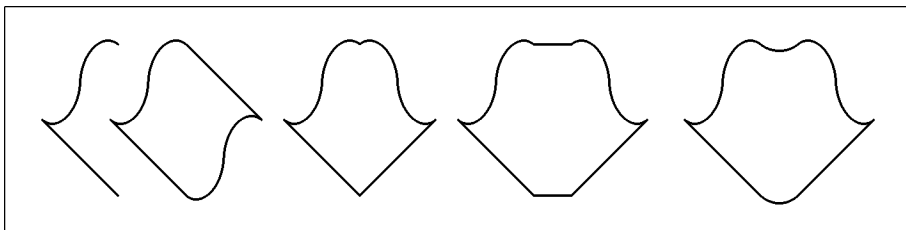
$$(t_x, t_y, t_{xx}, t_{xy}, t_{yx}, t_{yy}) = (150; 100; 1,5; 0,5; 0; 1,5).$$

Souřadnice tohoto vektoru můžeme použít samostatně jako

`xpart T`, `ypart T`, `xxpart T`, `xy part T`, `yxpart T`, `yypart T`.

## 6 Operace s cestami

Složitější cesty můžeme získat složením z několika jednodušších. Pokud na sebe jednotlivé úseky bezprostředně navazují, použijeme operátor `&`, jinak použijeme rovné spojky `--` nebo plynulé napojení pomocí `..`:



```

path p;
p=(origin--(-2,2){dir-40}..(-1,3)..{dir-40}(0,4)) scaled 5mm;
pickup pencircle scaled .3mm;
draw p shifted(15mm,5mm);
draw p shifted(24mm,5mm)& p
  rotated 180 shifted (24mm,25mm);
draw p shifted(47mm,5mm)& reverse p
  reflectedabout(up,down) shifted (47mm,5mm);
draw p shifted(70mm,5mm)--reverse p

```

```

reflectedabout(up,down) shifted(75mm,5mm)--cycle;
draw p shifted(100mm,5mm)..reverse p
reflectedabout(up,down) shifted(105mm,5mm)..cycle;

```

Další možné operace s cestami jsou použity v následujících dvou příkladech. První znázorňuje válcovou misku v kosoúhlém promítání:

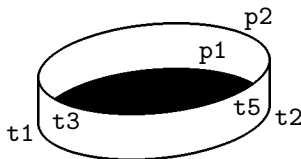
```

beginchar(1,40mm#,25mm#,0);
path p[];
p1=fullcircle scaled 30mm yscaled .4 slanted .5
  shifted (w/2,h/2-3mm);
p2=p1 shifted (0,6mm);
t1=directiontime down of p1;
t2=directiontime up of p1;
(t3,t4)=p1 intersectiontimes p2;
(t5,t6)=reverse p1 intersectiontimes reverse p2;
pickup pencircle scaled .3mm;
filldraw subpath(t3,length(p1)-t5) of p1--
  (reverse subpath(t4,length(p2)-t6) of p2)--cycle;
draw point t1 of p2--
  subpath(t1,8) of p1 & subpath(0,t2) of p1--point t2 of p2;
draw p2;
endchar;

```

Operací  $(t3,t4)=p1 \text{ intersectiontimes } p2$  získáme vektor, jehož souřadnice  $t3, t4$  jsou časy na cestě  $p1$  resp.  $p2$  určující jejich průsečík.

Pomocí  $directiontime \ z \ of \ p$  určíme čas, kdy má cesta  $p$  směr daný vektorem  $z$ .



Cesta  $subpath(t1,t2) \ of \ p$  je částí cesty  $p$  mezi časy  $t1$  a  $t2$ . Na konci příkladu jsme byli nuceni  $subpath$  rozepsat na dvě části spojené operátorem  $\&$ . Stejněho výsledku bychom dosáhli použitím  $subpath(t1,8+t2) \ of \ p1$ .

Proměnná  $length(p)$  představuje celkový čas cesty. Jednou jsme v příkladu napsali místo  $length(p)$  přímo číslici 8, což je celkový čas cesty  $fullcircle$ .

$reverse \ p$  nám cestu  $p$  otočí, tj. změni smysl počítání času. Hodí se při spojování cest nebo při hledání průsečíků. Pokud se cesty protínají vícekrát, nalezneme pomocí  $reverse$  místo prvního průsečíku průsečík poslední.

Na druhém obrázku je znázorněna dráha komety obíhající okolo Slunce, okamžitá poloha komety a vektor okamžité rychlosti po průchodu afelem v okamžiku, kdy průvodič svírá s hlavní osou elipsy úhel  $10^\circ$ .

```

beginchar(2,40mm#,25mm#,0);
path p[]; numeric t[];
a:=17.5mm; b:=7.5mm; e:=a++b; show a,b,e;

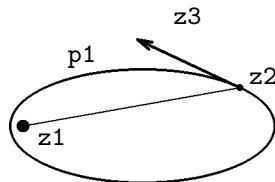
```

```

p1=fullcircle xscaled (2*a) yscaled (2*b) shifted (w/2,h/2);
z1=(w/2-e,h/2);
p2=(origin--dir10) scaled 100mm shifted z1;
z2=p1 intersectionpoint p2;
(t1,t2)=p1 intersectiontimes p2;
pickup pencircle scaled .3mm;
draw p1;
pickup pencircle scaled .4mm;
c:=angle(direction t1 of p1);
z3=z2+15mm*dir(c);
draw z2--z3;
draw (dir165--(0,0)--dir195) scaled 2mm rotated c shifted z3;
pickup pencircle scaled .2mm;
draw p1;
draw z1--z2;
filldraw fullcircle scaled 1.5mm shifted z1;
filldraw fullcircle scaled .75mm shifted z2;
endchar;

```

V obrázku je použita pomocná směrová úsečka p2, jejíž délku volíme tak, aby určitě prořela elipsu p1. Výsledkem p1 intersectionpoint p2 je bod, kde se cesty protínají.



Oproti minulému příkladu jsme navíc použili makra `direction t of p`, které zjistí směr cesty p v čase t. Nejedná se však o jednotkový vektor, jeho velikost se mění s polohou na cestě. Proto můžete často vidět konstrukci `dir(angle(direction t of p))`.

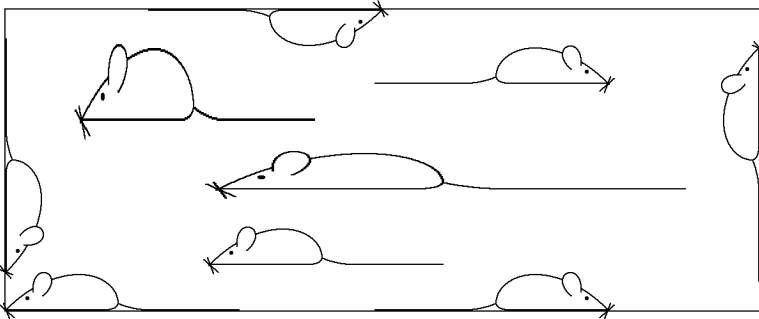
Ještě si všimněte, že jsme museli deklarovat číselné pole `t[]`. Z předchozího znaku se totiž zachovaly hodnoty `t1` a `t2` a při provádění řádku `(t1,t2)=...` by se objevilo chybové hlášení ! `Inconsistent equation`. Jediné proměnné, které se znovu inicializují při `beginchar`, jsou `x`, `y` a `z`.

## 7 Skládání obrázků

Opakuje-li se v obrázku některá část vícekrát, stačí ji nakreslit jen jednou a uložit jako hodnotu proměnné typu `picture`, kterou ovšem musíme nejprve deklarovat. Takto připravený detail můžeme posouvat, otáčet o násobek  $90^\circ$ , zrcadlově převracet okolo os, jejichž směrové úhly jsou násobkem  $45^\circ$  a zvětšovat jeho rozměry na celočíselný násobek. Transformovaný detail přidáme k původnímu obrázku příkazem

`addto currentpicture also transformovaný detail.`

V následující ukázce byla nejprve nakreslena první myš v levém dolním rohu obrázku a uložena do proměnné s názvem `mouse`. Její několikerou transformací a přikopírováním byl pak „zamyšován“ celý obrázek.



```

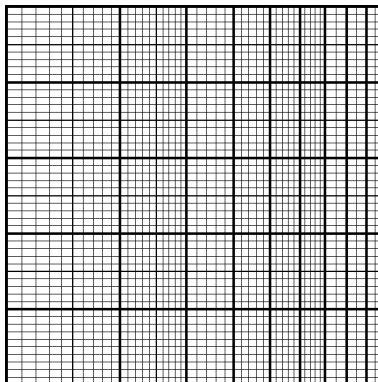
picture mouse; % deklarace
pickup pencircle scaled .2u; % vykreslení první myši
draw ((0,.4)---(25,.4)..{dir70}(30,2){up}..(26,8)..(15,9)
..{dir226}origin)scaled .5u;
draw ((30,2)..(38,.4)---(62,.4))scaled .5u;
erase fill ((7.5,5){dir100}..(11,10)..{dir215}(10,4)--cycle)
scaled .5u;
draw ((7.5,5){dir100}..(11,10)..{dir215}(10,4))scaled .5u;
draw (3dir-35--2dir130)scaled .5u;
draw (2.5dir-60--2dir100)scaled .5u;
fill fullcircle scaled .5u shifted (3u,1.7u);
mouse=currentpicture;
addto currentpicture also mouse shifted(27mm,6mm);
addto currentpicture also mouse xscaled 2 shifted (28mm,16mm);
addto currentpicture also mouse yscaled 2 shifted (10mm,25mm);
addto currentpicture also mouse rotated 180 shifted (50mm,h);
addto currentpicture also mouse
  reflectedabout((40mm,0),(40mm,h));
addto currentpicture also mouse
  reflectedabout((40mm,0),(40mm,h))shifted(30mm*up);
addto currentpicture also mouse
  reflectedabout(origin,dir45)shifted(0,5mm);
addto currentpicture also mouse
  reflectedabout(origin,dir-45)shifted(w,35mm);

```

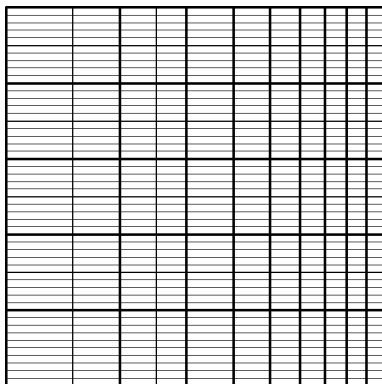
METAFONT ukládá bitmapy po řádcích do formátu `gf` a konvertor `GFtoPK` je převádí do úsporného formátu `pk`. Na řádcících registruje jen změny z černé

na bílou a naopak. Obsahuje-li obrázek velký počet svislých čar nebo v něm chceme vytvořit jemný rastr, může se stát že překročíme paměťové možnosti METAFONTu a po spuštění DOSovského programu **Convertor** se objeví hlášení: **Ran out of internal memory for row counts !** V takovém případě pomůže rozkreslení obrázku na několik samostatných částí, které složíme až v T<sub>E</sub>Xovském dokumentu. Semilogaritmickou síť v následující ukázce bylo nutno složit ze tří samostatných obrázků pomocí příkazů

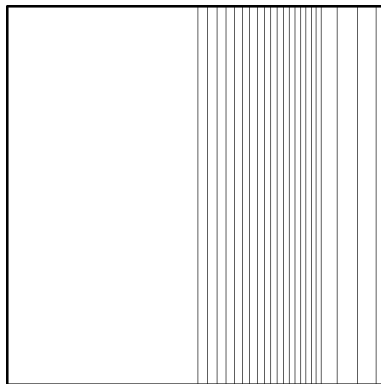
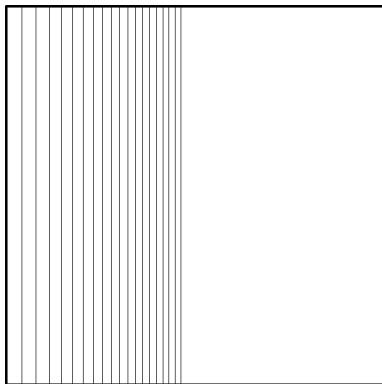
```
\begin{center}
\mbox{
\put(0,0){\obrazky\char1}
\put(0,0){\obrazky\char2}
{\obrazky\char3}
\end{center}
```



První dílčí obrázek má následující zdrojový program, další dva byly vytvořeny podobně:



```
beginchar(1,50mm#,50mm#,0);
pickup pencircle scaled 4;
draw unitsquare xscaled w yscaled h;
for i=1 upto 10:
  draw(mlog(i)/mlog(10)*50u,0)
  --(mlog(i)/mlog(10)*50u,h);endfor;
for i=1 upto 4:
  draw(0,10u*i)--(w,10u*i);endfor;
pickup pencircle scaled 2;
for i=1.5,2.5:
  draw(mlog(i)/mlog(10)*50u,0)
  --(mlog(i)/mlog(10)*50u,h);
endfor;
for i=1 upto 4:
  draw(0,10u*i+5u)--(w,10u*i+5u);
endfor;
pickup pencircle scaled 1;
for i=1 upto 49:
  draw(0,u*i)--(w,u*i);endfor;
endchar;
```



## 8 Pera, štětce, gumy

Chceme-li nakreslit bod nebo cestu, musíme nejprve zvolit pero příkazem  
`pickup druh pera scaled zvětšení;`

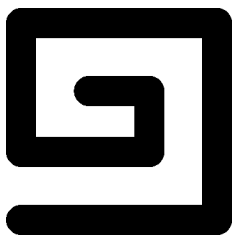
METAFONT nabízí tři základní druhy pera:

`pencircle` ... kruhové pero o průměru 1 pixel,

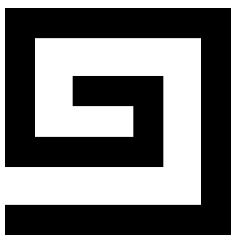
`pensquare` ... čtvercové pero o straně 1 pixel, jedna strana je vodorovná,

`penrazor` ... vodorovná úsečka nulové tloušťky a délky 1 pixel.

Na pero můžeme aplikovat transformační příkazy, kterými měníme jeho velikost, případně i tvar. Nejčastěji používáme pero kruhového tvaru. Čtvercové pero volíme k ostrému vykreslení rohů čar zalomených v pravém úhlu. Pero ve tvaru úsečky umožňuje kaligrafické efekty:



`pencircle scaled 4mm`



`pensquare scaled 4mm`



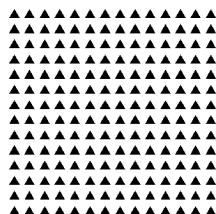
`penrazor scaled 4mm  
rotated 30`

Pomocí příkazu `makepen název=cesta` můžeme vytvořit vlastní pero libovolného konvexního tvaru. Název pera musíme předem deklarovat. V následující ukázce je použito pero trojúhelníkového tvaru pro vyrastování plochy:

```

pen triangle;
triangle=makepen(dir90--dir210--dir330--cycle);
pickup triangle scaled .8u;
for i=1 upto 14: for j=1 upto 14:
drawdot (2i*u,2j*u);
endfor; endfor;

```



Příkaz `draw`, kterým kreslíme cesty, můžeme použít i pro kreslení jednotlivých bodů, lepšího výsledku však dosáhneme při použití příkazu `drawdot`. Rozdíl vidíme v následující ukázce, kde byly kruhovým perem o průměru 1 mm vedle sebe nakresleny dva body a obrázek byl pak desetkrát zvětšen.

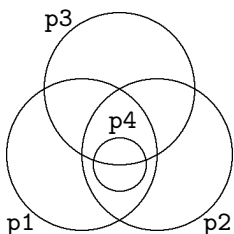
```

pickup pencircle scaled 1mm;
draw(.5mm,.8mm);
drawdot(2mm,.8mm);
addto currentpicture
    also currentpicture scaled 10;

```



METAFONT dovede vyčernit oblast ohraničenou uzavřenou cestou. Používá se k tomu příkaz `fill cesta`. Dovede také setřít jednu vrstvu černé barvy z již vyčerněné oblasti příkazem `unfill cesta`, nebo oblast úplně vyčistit příkazem `erase fill cesta`. Kombinací těchto příkazů můžeme dosáhnout zajímavých efektů, jak vidíme z následující ukázky:



```

fill p1;fill p2;
unfill p3;
unfill p4;

```



```

fill p1;fill p2;
erase fill p3;
erase fill p4;

```

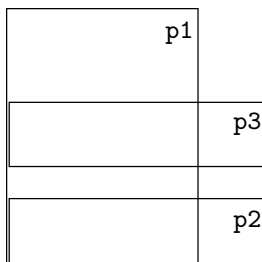


METAFONT pracuje s konečným počtem bodů připravované bitmapy, jakýchsi čtverečků ve čtvercové síti, a každému z nich přiřadí číslo, které udává, kolikrát byl vybarven nebo gumován. Ukažme si to na zvětšeném detailu takové bitmapy, který byl připraven příkazy:

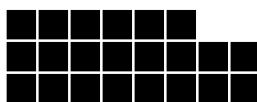
```

path p[];
p1=unitsquare xscaled 6 yscaled 8;
p2=unitsquare xscaled 8 yscaled 2;
p3=unitsquare xscaled 8 yscaled 2 shifted (0,3);
fill p1; fill p2; unfill p3;

```



1	1	1	1	1	1	0	0
1	1	1	1	1	1	0	0
1	1	1	1	1	1	0	0
0	0	0	0	0	0	-1	-1
0	0	0	0	0	0	-1	-1
1	1	1	1	1	1	0	0
2	2	2	2	2	2	1	1
2	2	2	2	2	2	1	1



Na hotové bitmapě jsou vyčerněny body, jejichž hodnota je kladná. Body, jejichž hodnota je nulová nebo záporná, zůstanou bílé.

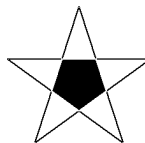
Příkazem `cull currentpicture keeping (min, max)` vymažeme všechny body, jejichž hodnota neleží v intervalu  $\langle min, max \rangle$ . Ostatním přiřadí hodnotu 1. V následujícím obrázku, kde p1 až p4 jsou stejné cesty jako v ukázce na předcházející stránce, zůstanou jen body vybarvené dvakrát a třikrát:

```
fill p1; fill p2; fill p3; fill p4;
cull currentpicture keeping (2,3);
```

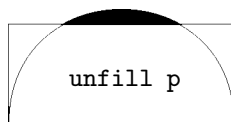
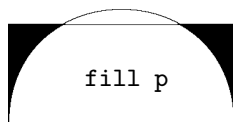


Cesta, která omezuje vyplňovanou oblast musí být uzavřena pomocí příkazu `cycle`. Nestačí tedy, aby první bod cesty byl současně jejím koncem, tedy: nikoli `p=z1--z2--z3--z1`, ale `p=z1--z2--z3--cycle`. Je-li cesta složitá, mohou být některé části plochy vybarveny vícekrát. Vnitřní pětiúhelník hvězdy na následujícím obrázku je vybarven dvakrát.

```
path hvezda;
hvezda=
  (dir90--dir-126--dir18--dir162--dir-54--cycle)
  scaled 10u shifted(w/2,h/2);
pickup pencircle scaled .2u;
filldraw hvezda; draw hvezda;
cull currentpicture keeping (2,2);
```



Cesta musí být určena tak, aby jednoznačně rozdělovala body roviny na vnitřní a vnější. Jinak dochází k situacím jako na následujícím obrázku:



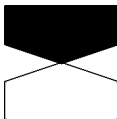
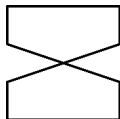


Častěji však dojde k přerušení běhu METAFONTu a k ohlášení chyby

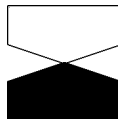
**! Strange path (turning number is zero).**

Stiskneme-li **Enter**, překlad zdrojového textu pokračuje i s pokusem o vybarvení plochy. Takový případ nastane například u cesty ve tvaru osmičky:

```
p1=origin--(0,5u)--(15u,10u)--(15u,15u)--
(0,15u)--(0,10u)--(15u,5u)--(15u,0)--cycle;
```

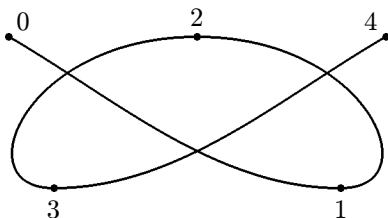


fill p



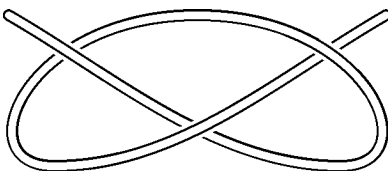
fill reverse p

Na vyčerněné ploše můžeme kreslit bílé body a čáry pomocí příkazu **undraw**, který odstraní v místě čáry jednu vrstvu barvy, a pomocí příkazu **erase draw**, který odstraní barvu až na podklad. Tyto příkazy používáme také pro kreslení dvojitých čar stále šířky a pro vyznačení viditelnosti vymazáním části čáry, která se má jevit jako spodní:



```
path p[];
p1=((0,20){dir-30}..(44,0){dir 0}
..(25,20)..{dir0}(6,0)
..{dir30}(50,20))scaled u;
```

```
pero(1.6u);
draw p1;
pero(u);
erase draw p1;
pero(2.5u);
erase draw subpath(.15,.25)of p1;
erase draw subpath(1.6,1.7)of p1;
erase draw subpath(3.2,3.5)of p1;
pero(1.6u);
draw subpath(.1,.3)of p1;
draw subpath(1.5,1.8)of p1;
draw subpath(3.1,3.6)of p1;
pero(u);
erase draw subpath(.05,.35)of p1;
erase draw subpath(1.45,1.85)of p1;
erase draw subpath(3.05,3.65)of p1;
```



## 9 Podmínky, cykly

Syntaxe METAFONTovské podmínky je následující:

```
if podm1:text1;
elseif podm2:text2
elseif podm3:text3;
:
else:text999;
fi;
```

Povinná je pouze část na prvním řádku začínající `if` a ukončení příkazu `fi`, ostatní části se uvádět nemusí. METAFONT postupuje tak, že nejprve vyhodnotí podmínku za `if` — když je splněna, provede text uvedený za ní. Pokud splněna není, pokračuje částmi `elseif`. Ty se vyhodnocují v pořadí, v jakém byly uvedeny. V případě, že není splněna ani jedna podmínka, provede METAFONT text za `else`. Chceme-li například spojit bod `z5` s nejbližším z bodů `z12` a `z13`, a pokud jsou stejně daleko, nakreslit obě spojnice, napíšeme:

```
if length(z12-z5)<length(z13-z5): draw z5--z12;
elseif length(z13-z5)<length(z12-z5): draw z5--z13;
else: draw z5--z12;
      draw z5--z13;
fi;
```

Relační operátory METAFONTu jsou: `>`, `>=`, `<`, `<=`, `=` a `<>`. Jako logické operátory se kromě klasických `and`, `or` a `not` používají ještě `numeric`, `path`, `boolean`, `string`, `pen`, `picture`, `transform` a `pair`. Tyto operátory mají hodnotu `true` jen tehdy, pokud je jejich operand daného typu — tedy `path p=true` jen když `p` je typu `path`. Operátory `known`, `unknown` a `cycle` jsou `true` jen když je proměnná za nimi po řadě známá, neznámá a uzavřená cesta. Operátor `odd` testuje lichost. Podmínky však nemusí figurovat jen jako samostatné příkazy, ale mohou se i vkládat na různá místa. Text

```
for i=0 upto 720:
  drawdot (.05*i*mm,if sind(i)>0: sind(i)*10mm else: 0 fi);
endfor;
```

způsobí velmi husté vytečkování „ořezané“ sinusoidy:

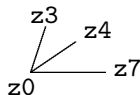


V předchozím příkladě byl použit cyklus. Cyklus je část textu, která se pro různé hodnoty tzv. řídicí proměnné vyhodnotí několikrát. METAFONT užívá několik druhů cyklu. První z nich má syntaxi:

```
for prom = hodn1, hodn2, ...:text endfor
```

Tento zápis METAFONT vyhodnotí tak, že do proměnné *prom* dosadí hodnotu *hodn1* a proběhne tělem cyklu *text*, poté dosadí hodnotu *hodn2* a proběhne podruhé atd. Pokud např. potřebujeme spojit bod *z0* s body *z3*, *z4* a *z7*, můžeme napsat cyklus:

```
for i=z3,z4,z7:
  draw z0--i;
endfor;
```



Hodnoty řídicí proměnné často tvoří aritmetickou posloupnost. V takovém případě použijeme cyklus:

```
for prom = dolni step krok until horni : text endfor
```

Pokud je *krok* 1 resp.  $-1$  můžeme místo *step*  $\pm 1$  until napsat pouhé *upto* resp. *downto*. Tento druh cyklů se většinou využije v různém rastrování a generování pravidelných vzorů. Milimetrový papír o rozměrech  $1 \times 1$  cm vygenerujeme cyklem:

```
for i=0 step 1mm until 1cm:
  draw (i,0)--(i,1cm);
  draw (0,i)--(1cm,i);
endfor;
```



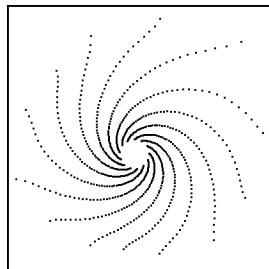
Tělo cyklu, *text*, nemusí být vždy kompletním příkazem zakončeným středníkem. V případě, že je *text* pouze částí příkazu, středník za ním nepíšeme (viz kapitola Grafy funkcí).

Poslední typ cyklu, který zde uvedeme, je nekonečný cyklus:

```
forever: text endfor
```

Takovýto cyklus nemá konec, vyskočit z něj lze pouze příkazem *exitif*. Tento příkaz může být uveden v těle jakéhokoli cyklu; pokud je podmínka za ním splněna, METAFONT okamžitě přeruší zpracování tohoto cyklu a pokračuje textem za *endfor*. Následující příklad zobrazuje průchod bodu Mandelbrotovým fraktálem:

```
beginchar(0,35mm#,35mm#,0);
pair a;
a=(0,0);
z0=(-.4,-.579);
pickup pensquare scaled 3;
forever:
  a:=(a zscaled a) shifted z0;
  exitif length(a)>4;
  fill unitsquare scaled 3
    shifted (a scaled 600+(420,380));
  exitif (totalweight currentpicture)>.1;
endfor;
draw unitsquare xscaled w yscaled h;
endchar;
```



## 10 Makra

Makra umožňují zkrácení a zjednodušení zdrojového textu zavedením vlastních příkazů. Makro musíme definovat před jeho prvním použitím, bývá však dobrým zvykem shromáždit všechny definice na začátku zdrojového textu.

Definice makra bez parametrů má syntaxi:

```
def jméno = text enddef;
```

Takto definované makro se pak volá příkazem *jméno*. Makro, které nakreslí rámeček okolo obrázku, může mít definici:

```
def frame=draw unitsquare xscaled w yscaled (h+d) shifted (0,-d)
enddef;
```

V dalším textu pak stačí napsat příkaz `frame`; . Makrem bez parametrů můžeme nahradit také jen část delšího příkazu, např.:

```
def penc=pickup pencircle scaled enddef;
```

a v textu pak volíme kruhové pero o průměru 0,3 mm příkazem `penc.3mm`.

Definice makra s jedním nebo více parametry má syntaxi:

```
def jméno (expr param1 , param2 ,...)= text enddef;
```

Makro s parametry se pak volá příkazem *jméno* (*param1* , *param2* ,...). Příkladem makra s parametrem může být příkaz `overdraw`:

```
def overdraw(expr p)=erase fill p; draw p enddef;
```

Toto makro nakreslí danou uzavřenou cestu a vymaže její vnitřek. Při použití se cesta musí napsat do závorek, např.

```
overdraw(fullcircle scaled 5mm shifted(10mm,15mm));
```

Přerušovanou spojnicí dvou bodů můžeme nakreslit pomocí makra, ve kterém jako parametry uvedeme polohy počátečního a koncového bodu a počet přerušování:

```
def dashline(expr zac,kon,opak)=
  for t=0 upto opak:
    draw (3t/(3opak+2)) [zac,kon] --((3t+2)/(3opak+2)) [zac,kon];
  endfor
enddef;
```

Parametry `expr` mohou být různého typu (numeric, pair, transformation, path, ...), ale musí samy o sobě dávat smysl, tj. nelze např. jako hodnotu parametru předávat `scaled .3mm`. METAFONT totiž parametr před dosazením do makra vyhodnotí (vypočítá číselnou hodnotu, souřadnice bodu, ...).

Není bez zajímavosti, že METAFONT je více jazykem maker než „holých“ příkazů, makra se v něm dají najít prakticky všude. Příkazy `beginchar` a `endchar` jsou makra, `draw` je také makro, dokonce i značka rovné čáry `--` je makro. Definice těchto maker jsou uvedeny v souboru `plain.mf`.

Uživatel METAFONTu si může vytvořit vlastní soubor oblíbených maker, nazvat jej např. `moje_mak.mf` a umístit jej do aktuálního adresáře. Na začátku práce na zdrojovém textu jej načteme příkazem `input moje_mak`.

Někdy je účelné zavést uvnitř makra *lokální proměnné*. V takovém případě vymežíme text makra jako samostatnou *skupinu* pomocí příkazů `begingroup` a `endgroup` a v takto definované skupině pak použijeme příkaz

`save prom1 , prom2 , ... ,`

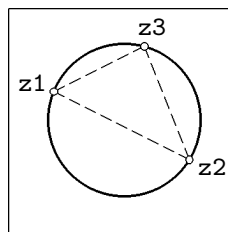
který způsobí, že všechny hodnoty a typy uvedených proměnných jsou překryty a mohou se libovolně měnit. Po příkazu `endgroup` METAFONT všechny proměnné opět obnoví.

Následující příklad makra s lokálními proměnnými slouží k sestrojení kružnice určené třemi různými body, které neleží v přímce. Příkaz `save x,y,R`; uloží souřadnice  $x$ ,  $y$  všech bodů a hodnotu proměnné  $R$ , pokud byla už zavedena.

```
def circ(expr boda,bodb,bodc)=
  begingroup
    save x,y,R;
    z1=boda; z2=bodb; z3=bodc;
    z0=0.5[z1,z2]+whatever*((z1-z2) rotated 90);
    z0=0.5[z2,z3]+whatever*((z2-z3) rotated 90);
    R=length(z1-z0);
    draw fullcircle scaled 2R shifted z0;
  endgroup
enddef;
```

Zdrojový text dalšího obrázku je tvořen prakticky jen makry z tohoto článku. Jak to přispělo k jeho stručnosti a přehlednosti je zřejmé na první pohled:

```
beginchar(2,30u#,30u#,0);
z1=(6u,19u);z2=(24u,10u);z3=(15u,25u);
penc.3u;
circ(z1,z2,z3);
penc.15u;
frame;
dashline(z1,z2,8);
dashline(z2,z3,6);
dashline(z3,z1,5);
for i=1 upto 3:
  overdraw(fullcircle scaled u shifted z[i]);
endfor;
endchar;
```

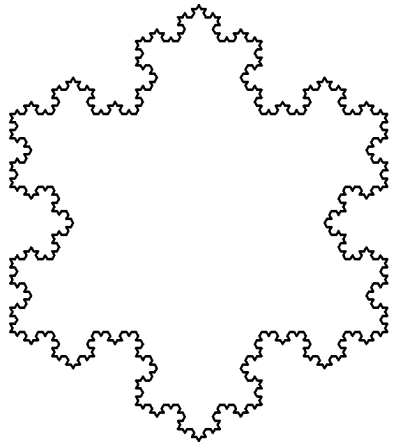


METAFONT umožňuje i rekurzivní makrodefinice jako třeba následující pro nakreslení křivky Kochové:

```

def spoj (expr n,a,f)=
if n>0:begingroup
    save b,c,e;
    pair b,c,e;
    b=1/3[a,f];
    e=2/3[a,f];
    c=a rotatedaround (b,-120);
    spoj(n-1,a,b);
    spoj(n-1,b,c);
    spoj(n-1,c,e);
    spoj(n-1,e,f);
endgroup;
else: draw a--f;
fi;
enddef;

```



```

beginchar(0,30u#,35u#,0);
pickup pencircle scaled .3u;
spoj(4,right*30u,origin);
spoj(4,origin,dir60*30u);
spoj(4,dir60*30u,right*30u);
endchar;

```

## 11 Grafy funkcí

Ve fyzice nebo v matematice často potřebujeme nakreslit graf funkce, u které známe její analytický předpis.

Při kreslení grafu využijeme toho, že Bézierova křivka, kterou proložíme dostatečným počtem bodů, jejichž polohu vypočteme podle funkčního předpisu, velmi dobře odpovídá skutečnému tvaru křivky. V první ukázce nakreslíme graf funkce  $y = \sin 2x + 2 \sin x$  v intervalu  $\langle 0, 2\pi \rangle$ :

Nejprve si zvolíme počátek souřadnicové soustavy  $\mathbf{z0}$  a nakreslíme souřadné osy a měřítko (viz komentář u zdrojového textu). Chceme, aby interval  $\langle 0, 2\pi \rangle$  byl na ose  $x$  zobrazen úsečkou délky 90 mm a aby jednotka na ose  $y$  měla velikost 6 mm.

Tvar křivky grafu popíšeme pomocí cesty  $\mathbf{p0}$ . Musíme zvolit dostatečný počet bodů cesty, aby tvar křivky odpovídal skutečnosti. Cesta však nesmí obsahovat víc než 300 bodů. V našem případě bohatě postačí, když zvolíme krok po deseti stupních. To znamená, že náš interval  $\langle 0, 2\pi \rangle$ , tj.  $\langle 0^\circ, 360^\circ \rangle$ , rozdělíme na 36 úseků.

Začneme v krajním bodě grafu (v našem případě je to bod o souřadnicích  $(0,0)$ ). Pak pomocí  $\mathbf{for}$  cyklu s parametrem  $i$  vypočítáme polohy dalších bodů

```
(i,2*sind (10i)+sind(20i))
```

a současně jimi proložíme Bézierovu křivku. Příkazem

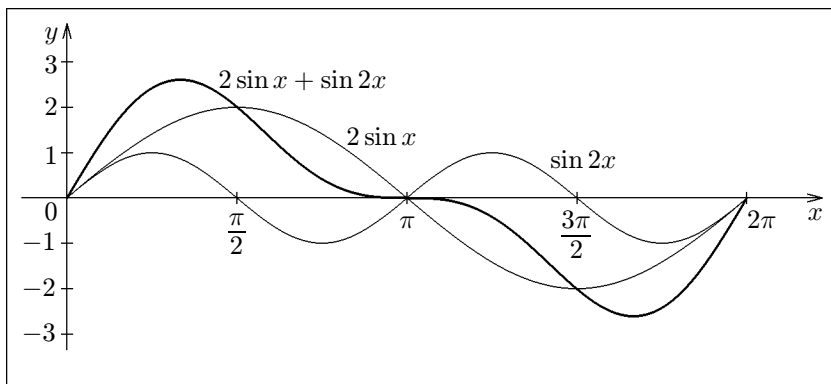
```
draw p0 xscaled 2.5u yscaled 6u shifted z0;
```

pak vykreslíme tuto cestu zvětšenou tak, aby graf odpovídal požadovanému měřítku (osa  $x$ :  $36 * 2,5 u = 90 u = 90 \text{ mm}$ , osa  $y$ :  $1 * 6 u = 6 \text{ mm}$ ) a posunutou do počátku souřadné soustavy, tj. do bodu  $z0$ .

Pro názornost jsou na obrázku ještě grafy funkcí  $2 \sin x$  a  $\sin 2x$ . Všimněte si, že obě křivky jsou nakresleny pomocí téže cesty  $p10$ . Kreslíme v podstatě graf funkce  $\sin x$  v intervalu  $\langle 0, 2\pi \rangle$ , který nejprve dvakrát zvětšíme ve směru osy  $y$  (funkce  $2 \sin x$ ) a podruhé zmenšíme na polovinu ve směru osy  $x$  (funkce  $\sin 2x$ ). V případě funkce  $\sin 2x$  nakreslíme nejprve první periodu a pak tentýž graf posuneme o délku periody vpravo.

Také si všimněte, že při kreslení funkce  $\sin x$  jsme dosáhli uspokojivého výsledku i při volbě většího kroku  $30^\circ$ .

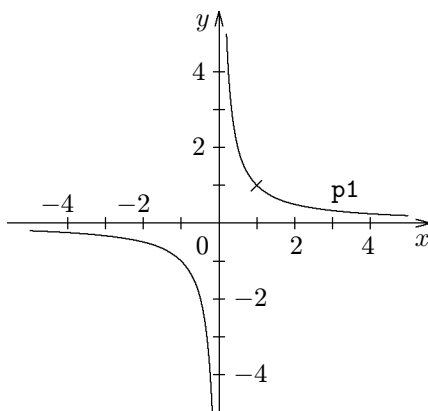
```
beginchar(1,110u#,50u#,0);
path p[]; %% deklarace cest
z0=(8u,h/2); %% počátek soustavy souřadnic
pickup pencircle scaled .2u;
draw unitsquare xscaled w yscaled h;
draw (2u,y0)--(w-2u,y0); %% osa x
draw (x0,5u)--(x0,h-2u); %% osa y
p1=-2u*dir 15--origin--2u*dir 165; %% šipka
draw p1 shifted (w-2u,y0); %% šipka na ose x
draw p1 rotated 90 shifted (x0,h-2u); %% šipka na ose y
for i=1 upto 4: %% měřítko na ose x
draw (up--down) scaled .75u shifted (z0+90u/4*i*right);
endfor;
for i=-3 upto 3: %% měřítko na ose y
draw (left--right) scaled .75u shifted (z0+6u*i*up);
endfor;
p0=(0,0) for i:=1 upto 36:
..(i,2*sind(10i)+sind(20i)) endfor;
pickup pencircle scaled .3u;
draw p0 xscaled 2.5u yscaled 6u shifted z0;
p10=(0,0) for i:=1 upto 12:
..(i,sind (30i)) endfor;
pickup pencircle scaled .15u;
draw p10 xscaled 7.5u yscaled 12u shifted z0; %% 2sin x
draw p10 xscaled (7.5u/2) yscaled 6u shifted z0; %% sin 2x
draw p10 xscaled (7.5u/2) yscaled 6u shifted (z0+45u*right);
endchar;
```



Funkční hodnoty nemusíme vždy počítat v celém zobrazovaném intervalu. V mnoha případech můžeme dobře využít symetrie zobrazované křivky. Například při kreslení grafu funkce  $y = \frac{1}{x}$  využijeme symetrie podle os kvadrantů a podle počátku soustavy souřadnic.

Nejprve výše uvedeným způsobem vytvoříme graf funkce v intervalu  $\langle 1, 5 \rangle$  (cesta **p1**). Zrcadlením cesty **p1** podle osy prvního a třetího kvadrantu a podle osy druhého a čtvrtého kvadrantu a jejím otočením okolo počátku soustavy souřadnic o  $180^\circ$  dostaneme zbývající části grafu.

```
p0=(1,1) for i:=2 upto 10:
  ..(i/2,2/i) endfor;
p1=p0 scaled 5u shifted z0;
draw p1;
draw p1 reflectedabout
      (z0,z0+dir 45);
draw p1 reflectedabout
      (z0,z0+dir-45);
draw p1 rotatedaround (z0,180);
```



Podobným způsobem postupujeme i při kreslení křivek zadaných parametricky. Epicykloidu popsanou rovnicemi

$$x = R \cos(\Omega t) + r \cos(\omega t), \quad y = R \sin(\Omega t) + r \sin(\omega t),$$

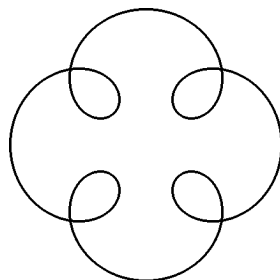
kde  $R = 12 \text{ mm}$ ,  $r = 6 \text{ mm}$ ,  $\omega = 5\Omega$ , nakreslíme jediným příkazem:



```

beginchar(3,36u#,36u#,0);
pickup pencircle scaled .3u;
draw
((3,0)
for i=1 upto 180:
  ..(2cosd(2i)+cosd(10i),2sind(2i)+sind(10i))
endfor)
scaled 6u shifted(w/2,h/2);
endchar;

```



## 12 Náhodná čísla

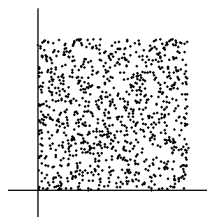
METAFONT je vybaven dvěma generátory náhodných čísel.

Příkaz `x=uniformdeviate t` generuje náhodná čísla rovnoměrně rozdělená v intervalu  $(0, t)$  pro  $t > 0$ , nebo v intervalu  $(t, 0)$ , jestliže  $t < 0$ .

```

for i=1 upto 1000:
draw z0+(uniformdeviate 20mm,uniformdeviate 20mm);
endfor;

```

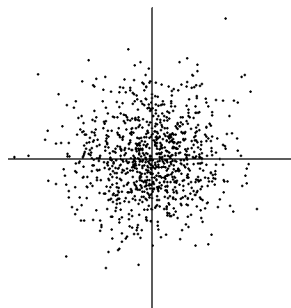


Příkaz `x=normaldeviate` generuje náhodná čísla v normálním rozdělení popsaném Gaussovou křivkou četnosti se středem v bodě 0 a střední kvadratickou hodnotou rovnou 1. Pravděpodobnost výskytu je přímo úměrná výrazu  $e^{-x^2/2}$ . Z vygenerovaných čísel splňuje 68 % nerovnost  $|x| < 1$ , pro 95 % platí  $|x| < 2$  a pro 99,7 % platí  $|x| < 3$ .

```

for i=1 upto 1000:
draw z0+(normaldeviate,normaldeviate)*5mm;
endfor;

```

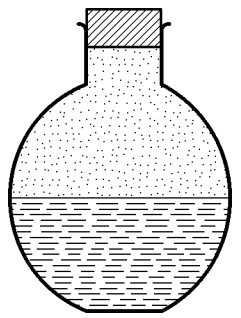
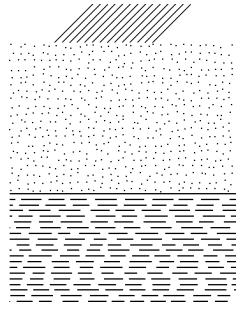


Počáteční stav generátorů `uniformdeviate` a `normaldeviate` můžeme nastavit příkazem `randomseed:=číselný výraz`. Pokud to neprovedeme, volí se implicitně `randomseed:=day + time * epsilon` a po každém spuštění generátoru vznikne jiná číselná množina.

Náhodná čísla můžeme dobře využít pro vytvoření polopravidelných rastrů znázorňujících kapalinu nebo plyn:

```
beginchar(4,30mm#,40mm#,0);
randomseed:=1;           % kapalina
krok:=mm;
pickup pencircle scaled .15mm;
a=w; b=w;
for i=1 upto 15mm/krok:
  y:=i*krok;
  b:=b-w;
  if a>w: a:=a-w; draw (a,y)--(b,y);
  else: draw (0,y)--(b,y);
  fi;
  forever:
    a:=b+krok*(1+uniformdeviate 1);
    b:=a+krok*(2+uniformdeviate 2);
    if b<w:draw (a,y)--(b,y);
    elseif a<w:draw (a,y)--(w,y);
    fi;
    exitif b>w;
  endfor;
endfor;
draw(0,15mm)--(w,15mm);
for i=6 upto 19:         % zátka
  draw((i,35)--(i+5,40))scaled mm;
endfor;

pickup pencircle scaled .2mm;
for j=15 upto 34:       % plyn
  for i=0 upto 29:
    draw(uniformdeviate .6+i,uniformdeviate .6+j+.2)*mm;
  endfor;
endfor;
filldraw((10,40)--(10,30){dir202}..(10,0)--(20,0){dir28}
  ..(20,30)--(20,40)--cycle) scaled mm;           % silueta
pickup pencircle scaled .3mm;
cull currentpicture keeping (2,2);
draw unitsquare xscaled 9.8mm yscaled 5mm shifted (10.1mm,35mm);
```



```

pickup pencircle scaled .5mm;
draw((9,38)..{down}(10,37)--(10,30){dir202}..(10,0)--
(20,0){dir28}..(20,30)--(20,37){up}..(21,38)) scaled mm; % obrys
endchar;

```

### 13 Kosoúhlé zobrazení

Kosoúhlé zobrazení je určeno směrem a zkrácením průmětu souřadnicové osy kolmé k nákresně. Je účelné zavést nejprve pomocné transformace pro vytvoření průmětů půdorysu a bokorysu. V následující ukázce je zobrazena koule o poloměru  $r$  a středu  $[2,2r; 2r; 2,5r]$  a její průměty do půdorysny nárysny a bokorysny.

```

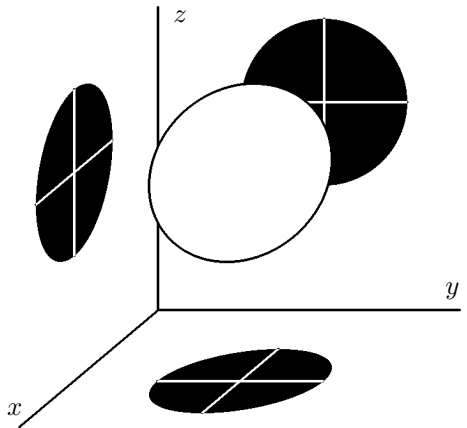
mode_setup; u#:=1mm#;
define_pixels(u);

```

```

beginchar(1,60u#,60u#,0.15u#);
  transform t[]; path p;
% parametry zobrazení:
z0=(20u,18u); f:=40; k:=.6;
% transformace půdorysu:
(0,0)transformed t1=z0;
(1,0)transformed t1=z0+(1,0);
(0,1)transformed t1=
  z0+(-k*cosd(f),-k*sind(f));
% transformace bokorysu:
(0,0)transformed t2=z0;
(0,1)transformed t2=z0+(0,1);
(1,0)transformed t2=
  z0+(-k*cosd(f),-k*sind(f));

```



```

% zadání koule:
r:=11u; sx:=2.2r; sy:=2r; sz:=2.5r;
% zobrazení průmětů koule:
p:=fullcircle scaled 2r;
filldraw p shifted (z0+(sy,sz));
filldraw p shifted (sy,sx) transformed t1;
filldraw p shifted (sx,sz) transformed t2;
pickup pencircle scaled .3u;
draw (up--down) scaled r shifted (z0+(sy,sz));
draw (left--right) scaled r shifted (z0+(sy,sz));
draw (up--down) scaled r shifted (sy,sx) transformed t1;
draw (left--right) scaled r shifted (sy,sx) transformed t1;
draw (up--down) scaled r shifted (sx,sz) transformed t2;
draw (left--right) scaled r shifted (sx,sz) transformed t2;
cull currentpicture keeping (1,1);
% zobrazení souřadnicových os:
draw ((40u,0)--origin--(0,40u)) shifted z0;

```

```

draw(origin--(40u,0))transformed t2;
% zobrazení koule:
z10=(sy,sx)transformed t1+sz*up;
erase fill (p xscaled(sqrt(1+k*k)) rotated(f) shifted z10);
draw (p xscaled(sqrt(1+k*k)) rotated(f) shifted z10);
endchar;

```

## 14 Axonometrické zobrazení

Axonometrii můžeme určit odchylkami  $\beta$ ,  $\alpha$  průmětů os  $x$  a  $y$  od základny axonometrického trojúhelníka. Vzdálenosti měřené ve směrech souřadných os se v axonometrickém průmětu zkracují. Určíme nejprve odchylku  $\zeta$  osy  $z$  od axonometrické průmětny. Zkrácení axonometrického průmětu osy  $z$  je rovno  $\cos \zeta$  a zkrácení průmětu spádové přímky roviny  $xy$  je rovno  $\sin \zeta$ . Užitím Euklidovy věty o výšce dostaneme:

$$\sin \zeta = \frac{m}{\sqrt{m \cotg \beta \cdot m \cotg \alpha}} = \sqrt{\tg \beta \tg \alpha}, \quad \gamma = 90^\circ - \beta - \alpha,$$

$$\cos \zeta = \sqrt{1 - \sin^2 \zeta} = \sqrt{\frac{\cos(\beta + \alpha)}{\cos \beta \cos \alpha}} = \sqrt{\frac{\sin \gamma}{\cos \beta \cos \alpha}}.$$

Obdobně určíme odchylku  $\xi$  osy  $x$  a odchylku  $\eta$  osy  $y$ :

$$\cos \xi = \sqrt{\frac{\sin \alpha}{\cos \beta \cos \gamma}} \quad \cos \eta = \sqrt{\frac{\sin \beta}{\cos \alpha \cos \gamma}}.$$

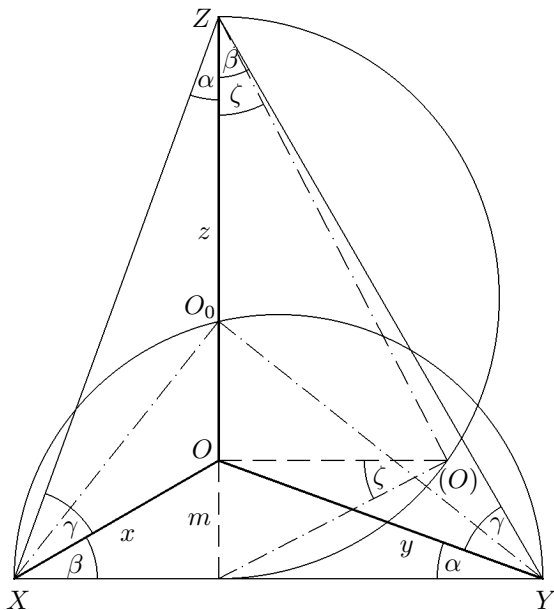
Na další ukázce je zobrazena koule o poloměru  $r$  se středem  $[2,2r; 2r; 2,1r]$  a její průměty do rovin souřadných os. Průmět koule do půdorysny se zobrazuje jako elipsa, jejíž hlavní osa je kolmá k obrazu osy  $z$  a má velikost  $2r$  a vedlejší osa má velikost  $2r \sin \gamma$ . Obdobné vlastnosti mají obrazy průmětů koule do nárýsny a bokorysny.

```

mode_setup; u#:=1mm#; define_pixels(u);

beginchar(2,70u#,65u#,0.15u#);
% parametry zobrazení:
z0=(28u,20u); b:=25; a:=10; c:=90-b-a;
% koeficienty zkrácení:
lx=sqrt(sind(a)/(cosd(b)*cosd(c))); kx=sqrt(1-lx**2);
ly=sqrt(sind(b)/(cosd(a)*cosd(c))); ky=sqrt(1-ly**2);
lz=sqrt(sind(c)/(cosd(b)*cosd(a))); kz=sqrt(1-lz**2);
r:=15u;

```

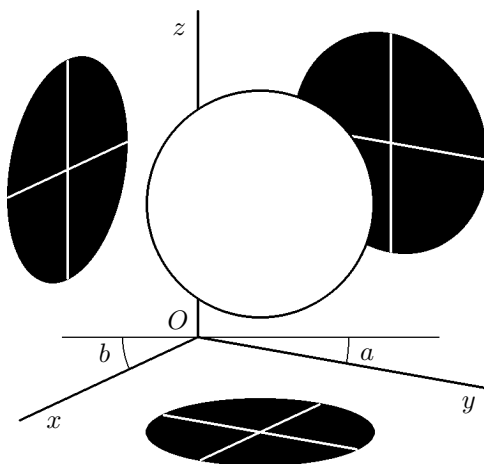


Obrázek 1: Axonometrický trojúhelník

```

% průměty poloměrů rovnoběžných s osami:
z1=r*lx*dir(180+b); z2=r*ly*dir-a; z3=r*lz*up;
% obrazy průmětů koule do nakreslen:
z11=z0+2z2+2.1z3; z12=z0+2.2z1+2.1z3; z13=z0+2.2z1+2z2;
pickup pencircle scaled .3u;
filldraw fullcircle scaled 2r xscaled kx rotated b shifted z11;
filldraw fullcircle scaled 2r xscaled ky rotated -a shifted z12;
filldraw fullcircle scaled 2r yscaled kz shifted z13;
draw(z3--(-z3))shifted z11; draw(z2--(-z2))shifted z11;
draw(z3--(-z3))shifted z12; draw(z1--(-z1))shifted z12;
draw(z1--(-z1))shifted z13; draw(z2--(-z2))shifted z13;
cull currentpicture keeping (1,1);
% obrazy os:
draw(z0+3z1)--z0--(z0+3z3); draw z0--(z0+3z2);
% obraz koule:
erase fill fullcircle scaled 2r shifted (z0+2.2z1+2z2+2.1z3);
draw fullcircle scaled 2r shifted (z0+2.2z1+2z2+2.1z3);
pickup pencircle scaled .15u;
draw(10u,y0)--(60u,y0);
draw (z0+10u*left){down}..(z0-10u*dir b);

```



Obrázek 2: Axonometrické zobrazení koule a jejích průmětů

```
draw (z0+20u*right){down}..(z0+20u*dir-a);
endchar;
```

## 15 Graf funkce dvou proměnných

Následující příklad ukazuje použití METAFONTu pro vykreslení grafu funkce dvou proměnných v axonometrickém promítání. V makru `setview` nastavíme azimut a výšku pohledu na graf a transformaci. Pomocí `setdiv` určíme dělení sítě a skutečné rozměry boxu, do kterého budeme kreslit graf. Abychom mohli předat tyto parametry do makra `fce`, musíme využít globálních proměnných. V makru `fce` nejprve vypočteme axonometrické průměty všech bodů sítě a uložíme je do pomocných polí `xp [] []` a `yp [] []`. Nakonec „odzadu vybělíme a obtáhneme“ všechny čtyřúhelníky.

```
mode_setup;
```

```
def overdraw(expr p)=erase fill p; draw p; enddef;
```

```
def setview(expr A,H,T)= %% nastavení pohledu a transformace
  transform t; t:=T;
  a:=cosd(A);b:=-sind(A);c:=b*sind(H);d:=-a*sind(H);e:=cosd(H);
enddef;
```

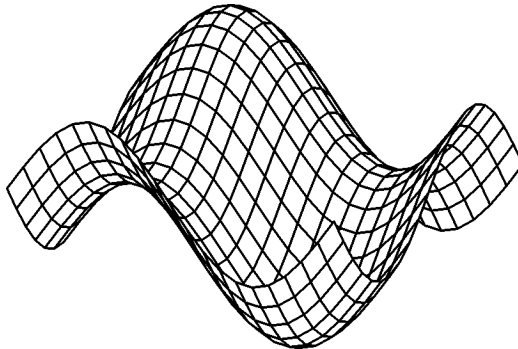
```

def setdiv(expr NX,NY,XM,YM,ZM)= %% nastavení dělení a měřítek
    nx:=NX; ny:=NY; xm:=XM/nx; ym:=YM/ny; zm:=ZM;
enddef;

def fce(text f)(expr minx,maxx,miny,maxy,minz,maxz)= %% 3D graf
    numeric xp[] [],yp[] [];
    a:=a*xm; b:=b*ym; c:=c*xm; d:=d*ym; e:=e*zm/(maxz-minz);
    for i=0 upto nx:
        for j=0 upto ny:
            x:=minx+i*(maxx-minx)/nx; y:=miny+j*(maxy-miny)/ny;
            xp[i][j]:=a*i+b*j; yp[i][j]:=c*i+d*j+e*(f-minz);
        endfor;
    endfor;
    for i=0 upto nx-1:
        for j=0 upto ny-1:
            overdraw((xp[i][j],yp[i][j])--(xp[i][j+1],yp[i][j+1])
                --(xp[i+1][j+1],yp[i+1][j+1])--(xp[i+1][j],yp[i+1][j])
                --cycle) transformed t);
        endfor;
    endfor;
enddef;

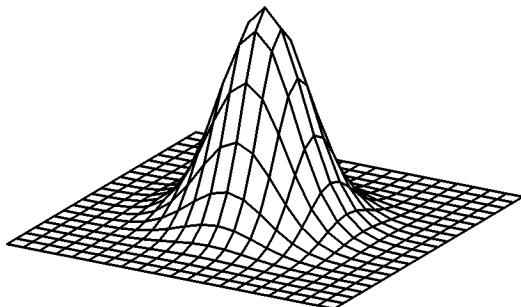
beginchar(1,75mm#,50mm#,0); %% příklad použití
    pickup pencircle scaled .3mm;
    setview(30,10,identity shifted (27.5mm,20mm));
    setdiv(20,20,50mm,50mm,20mm);
    fce((sind(x)+sind(y)))(0,360,0,360,-1,1);
endchar;
end.

```



$$f(x, y) = \sin(x) + \sin(y)$$

Protože čtveřice bodů, určující elementy prostorového grafu, neleží obecně v jedné rovině, může jejich průmět do axonometrické roviny vytvořit někdy „mašličku“, kterou METAFONT neumí vyplnit. Například v následující ukázce se přeruší běh METAFONTu s chybovým hlášením ! **Strange path**. Tomu můžeme zabránit příkazem `turningcheck:=0`; zařazeným před volání makra `fce`.



$$f(x, y) = e^{-(x^2+y^2)}$$

Další vylepšení těchto maker, např. vykreslení souřadnicových os, jejich dělení, pohled z azimutu ( $90^\circ$ ;  $360^\circ$ ), jiná definice referenčního bodu, stínování podle dopadajícího světla, přenecháme laskavému čtenáři.

## A Práce s celými obrázky

V praxi se často setkáme s úkolem začlenit do textu vysázeného v  $\text{T}_\text{E}_\text{X}$ u obrázky, které pocházejí z jiných programů a jsou uloženy v nějakém běžném grafickém formátu (PCX, BMP, GIF, TIFF aj). Opačnou úlohou je využití obrázků vytvořených METAFONTEM v jiných programech (např. Microsoft Word).

Pro práci s celými obrázky potřebujeme některé pomocné programy, které nejsou součástí standardní instalace  $\text{T}_\text{E}_\text{X}$ u. Přístup k nim je uveden v kapitole Literatura a internetové odkazy.

Velice užitečný je sharewarový program Graphic Workshop 7.0 (GWS), který umožňuje konvertovat barevné obrázky na černobílé (funkce `dither`), ořezávat (`crop`), zmenšovat či zvětšovat (`scale`).

Pro úpravy rastrových obrázků existuje velké množství programů. Známý je PC Paintbrush, pracující pod DOSem, nebo skvělý shareware Paint Shop Pro, který vyžaduje Windows.

Programy `scr2pcx` Rudolfa Čejky nebo Screen Thief slouží k převedení obsahu obrazovky počítače do grafického souboru `pcx` nebo `gif`. Protože obrázky jsou při rozlišení 300 dpi příliš malé, zvětšíme je pomocí GWS na dvojnásobek. Rychlé prohlížení obrázků ve formátu `pcx` umožňuje Čejkův program `pcxview`.

Programem `bm2font`, který umožňuje vložit rastrové obrázky do  $\text{T}_\text{E}_\text{X}$ ovského dokumentu, se budeme podrobněji zabývat na následujících stránkách.



## A.1 Příkaz `\special`

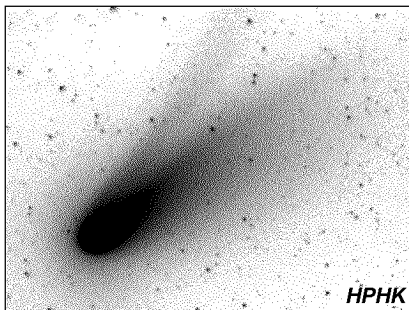
Pomocí příkazu `\special{em: graph jmeno}` ve zdrojovém souboru můžeme ovladači `dvipsr`, `dvihplj` nebo `dvidot` říci, aby do dokumentu na dané místo vložil černobílý obrázek ve formátu `pcx`, `bmp` aj.

Protože  $\TeX$  o obrázku „neví“, nevynechá pro něj žádné místo. Musíme si tedy vypočítat, jaké místo obrázek zabírá při rozlišení naší tiskárny a volné místo nebo obtékání textu zajistit sami. Někdy nám bude stačit jednoduché `\vspace{50mm}`, jindy můžeme zkusit složitější konstrukce pomocí prostředí `minipage`, jako v následujícím příkladu. Všimněte si, že *referenční bod je levý horní roh obrázku*.

```
\noindent\begin{minipage}[t]{56mm}
Tento snímek komety Hale--Bopp byl pořízen ...
\end{minipage}
\hfill
\begin{minipage}[t]{55mm}
\unitlength=1mm
\mbox{}
\vspace{-8mm}

{\special{em: graph halebopp.pcx}}
\end{minipage}
```

Tento snímek komety Hale–Bopp byl pořízen na hvězdárně v Hradci Králové 1. 4. 1997, 20:30 UT fotografickým objektivem 2,8/80. Jako detektor byla použita CCD kamera SBIG ST5, která má matici tvořenou  $320 \times 240$  pixely velikosti  $10 \mu\text{m}$ . Snímek byl pro naše účely dvakrát zvětšen.



Popsaným způsobem můžeme vložit obrázek jen v případě, že při tisku nedochází k otočení textu (jako např. při umístění dvou stránek A5 na list A4). Obrázek vsazený příkazem `\special` totiž ignoruje potřebné transformace.

## A.2 Program `bm2font`

Volně šiřitelný program `bm2font` slouží ke konverzi bitmapových obrázků do fontů  $\TeX$ u. Obrázek je rozdělen na malé pravoúhlé části, z nichž se vytvoří „písmena“ jednoho nebo několika fontů. Jejich složením vznikne na zvoleném místě dokumentu opět celý obrázek. Konvertovat je možno formáty `bmp`, `pcx`, `gif`, `tiff`. Je-li obrázek barevný, program provede dithering (tj. převod na černobílý). Rozměry obrázku lze podle potřeby upravit. Program spustíme příkazem

## bm2font jmeno [parametry]

Výstupem je font (soubor `jmena.pk`), jeho metrika (soubor `jmena.tfm`) a textový soubor `jmeno.tex`, obsahující příkazy k opětnému spojení „písmen“ do původního obrázku. Ten zavedeme do dokumentu příkazem `\input{jmeno}` a pak na zvoleném místě příkazem `\setjmeno` obrázek vysadíme na stránce. Je-li obrázek velký nebo složitý, vzniknou kromě fontu `jmena` ještě fonty `jmenob`, `jmenoc`, atd.

V následující tabulce je uveden přehled parametrů programu:

-f	jméno obrázku pro T <sub>E</sub> X (nesmí obsahovat číslice), standardně jméno souboru
-h	vodorovné rozlišení obrázku v bodech na palec (dpi), standardně 300
-v	svislé rozlišení obrázku v dpi, standardně 300
-a	y nebo n, ukazuj obrázky, standardně n
-l	délka řádku bitové mapy (pokud není uvedeno ve formátu)
-i	inverze obrázku, standardně n
-w	y nebo n, bílá bude světle šedá, standardně y
-b	počet polotónů
-u	počet bodů vodorovně v „šedém“ obdélníčku, maximálně 8
-c	počet bodů svisle v „šedém“ obdélníčku, maximálně 8
-t	kontrast v %, standardně 70
-z	oblast kontrastu v %, standardně 70
-m	skutečná vodorovná velikost obrázku v mm, není-li uvedeno, je skutečná velikost obrázku dána velikostí v bodech a rozlišením
-n	skutečná svislá velikost obrázku v mm
-k	barva, podle které se provede dithering (c – světle modrá, m – fialová, y – žlutá, k – černá)

Tab. 1 – Přehled parametrů programu `bm2font`.

Ukážeme si ještě několik příkladů volání programu `bm2font`.

`bm2font obrazek.bmp`

vygeneruje font pro 300 dpi, na začátek napíšeme `\input{obrazek}` a znak vysázíme příkazem `\setobrazek`

`bm2font nnn.gif -fobr -h240 -iy -ay`

font bude mít tentokrát rozlišení 240 dpi, obrázek bude inverzní, program nám jej ukáže na obrazovce a v dokumentu píšeme `\input{obr}`, `\setobr`

`bm2font nnn.pcx -h180 -n70 -wn`

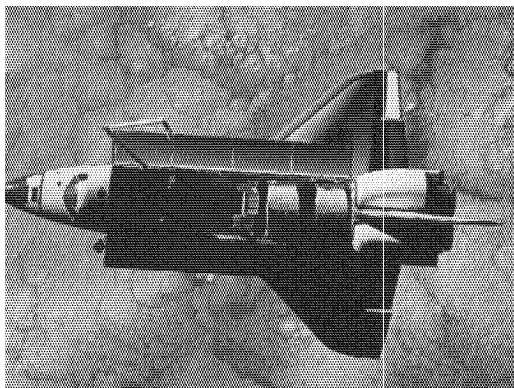
rozlišení fontu je 180 dpi, obrázek se upraví tak, aby jeho výška byla 70 mm (není-li uveden parametr `-m`, zachová se poměr stran obrázku), původně bílá místa zůstanou bez pokrytí

V následující ukázce jsme původně barevný obrázek nejprve zpracovali programem `bm2font`

```
bm2font -m68 shuttle.bmp
```

a vznikly soubory `shuttle.tex`, `shuttle.tfm`, `shuttlea.pk`. Vložení obrázku do dokumentu jsme provedli na jediném řádku:

```
\begin{center}\input{shuttle}\setshuttle\end{center}
```



### A.3 Převod obrázku do formátu `pcx`. Dávka `dvi2pcx`

Ovladač `dvidot.exe` s parametrickým souborem `pcx.dot` je schopen ze souboru `dvi` zhotovit obrázky ve formátu `pcx`, vhodné ke zpracování v ne $\TeX$ ovských programech. Pokud chceme vyrobit ze zdrojového textu METAFONTu obrázky `pcx`, musíme je vysázet samostatně na jednotlivé stránky. Nezapoměňte zrušit číslování stránek příkazem `\pagestyle{empty}`.

Dávkový soubor `dvi2pcx.bat`, který si pro tento účel vytvoříme, může vypadat třeba takto:

```
c:\emtex\bin\dvidotc:\emtex\data\pcx.dot @dvi2pcx.cnf %1 str??
```

V konfiguračním souboru `dvi2pcx.cnf` zapíšeme rozlišení, cesty k fontům a ke knihovnám; přepínač `+minimize=on` zajistí, aby se ořezaly přebytečné bílé okraje:

```
+resolution=300
+minimize=on
+font-libraries=c:\emtex\fonts\lj_{cbas,more}
+font-files={c:\emtex\fonts\pixel.lj@Rrdpi\,}@f{.pk,.pxl}
```

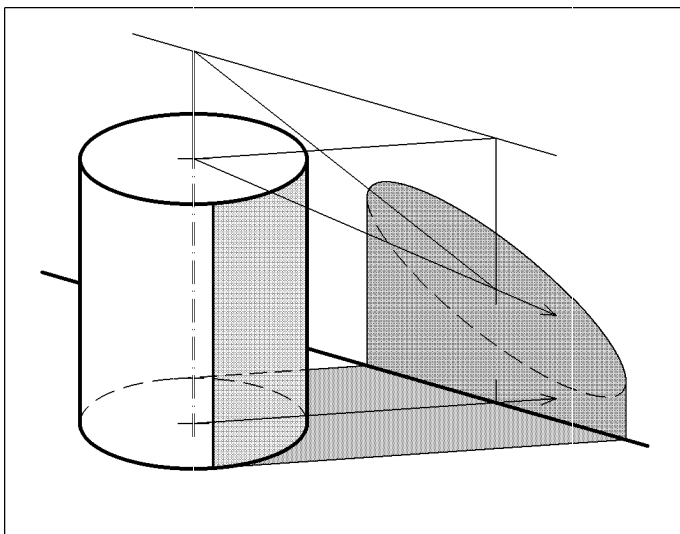
Všechny cesty v těchto souborech je samozřejmě nutné upravit podle místní instalace. Jednotlivé stránky budou uloženy jako obrázky nazvané `str01.pcx`, `str02.pcx`, ...

Předpokládejme, že již máte vytvořen zdrojový text obrázku `obr.mf`. Přeložte jej METAFONTEM, převedte do formátu `pk` a napište soubor `obr.tex`, ve kterém obrázek vysázíte samostatně na stránku:

```
\documentstyle{article}
\pagestyle{empty}
\font\obr=obr
\begin{document}
{\obr\char1}
\end{document}
```

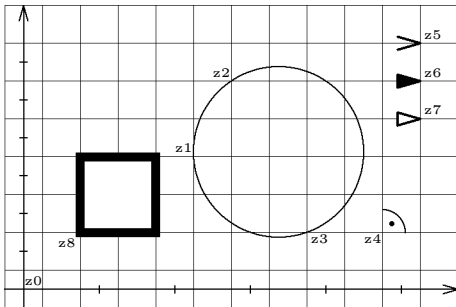
Po přeložení T<sub>E</sub>Xem vznikne soubor `obr.dvi`. Pak stačí spustit dávku  
`dvi2pcx obr.dvi`  
a obrázek se uloží do souboru `str01.pcx`.

Následující obrázek byl vytvořen METAFONTEM, pomocí dávky `dvi2pcx` převeden do formátu `pcx` a v programu Malování pod Windows doplněn jemným rastrem ve stínech. Pak teprve byl pomocí programu `bm2font` zařazen na tuto stránku.

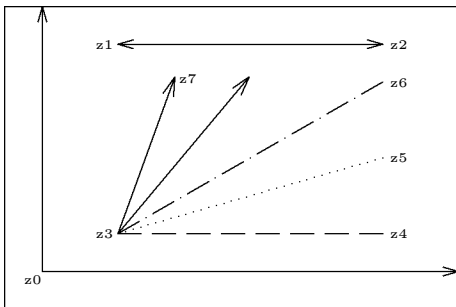


## B Makra

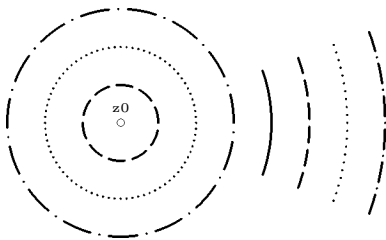
Soubor `gjkt.mf` obsahuje řadu užitečných maker různých autorů. Vedle obrázků je naznačeno jejich použití. Následuje úplný výpis souboru.



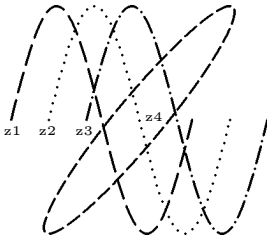
```
penc.1mm;
net(5mm);
penc.2mm;
axis(z0,2mm);
scale(z0,1mm,10mm,5mm);
circ(z1,z2,z3)
R(z4,0);
penc.3mm;
arrow(3mm,0,z5);
farrow(3mm,0,z6);
earrow(3mm,0,z7);
pens1.2mm;
overdraw(unitsquare
scaled 10mm shifted z8);
```



```
penc.2mm;
frame;
halfaxis(z0,2mm);
dim(z1,z2,2mm);
vector(z3,z7mm,2.5mm,50);
vect(z3,z7,2.5mm);
dashline(z3,z4,7);
dottedline(z3,z5,35);
dashdottedline(z3,z6,7);
```



```
penc.1mm;
odot(z0,1mm);
penc.3mm;
dashcircle(z0,5mm,0,10);
dotcircle(z0,10mm,0,80);
dashdotcircle(z0,15mm,0,15);
arc(z0,20mm,-20,20);
dasharc(z0,25mm,-20,20,5);
dotarc(z0,30mm,-20,20,20);
dashdotarc(z0,35mm,-20,20,5);
```



```

path p[];
p1=(origin for i=1 upto 12:..(i,sind(30i))
      endfor) xscaled 2mm yscaled 15mm;
p2=fullcircle
      xscaled 8mm yscaled 30mm slanted .8;
penc.3mm;
dashpath(p1 shifted z1,18,2.5);
dotpath(p1 shifted z2,61);
dashdotpath(p1 shifted z3,18,3);
dashcyclepath(p2 shifted z4,30,2.5);

```

---

```

%% gjkt.mf

```

```

%% poslední uprava 11.1.1998

```

```

def penc = pickup pencircle scaled enddef; %% pera

```

```

def pens = pickup pensquare scaled enddef;

```

```

def overdraw(expr p)=erase fill p; draw p; enddef; %% překreslení

```

```

def frame=draw unitsquare xscaled w yscaled h; enddef; %% rámeček

```

```

def axis(expr bod,delka)= %% osy
  draw (0,ypart bod)--(w,ypart bod);
  draw (xpart bod,0)--(xpart bod,h);
  arrow(delka,90,(xpart bod,h));
  arrow(delka,0,(w,ypart bod));
enddef;

```

```

def halfaxis(expr bod,delka)= %% půlosy
  draw bod--(w,ypart bod); draw bod--(xpart bod,h);
  arrow(delka,90,(xpart bod,h));
  arrow(delka,0,(w,ypart bod));
enddef;

```

```

def scale(expr bod,delka,dw,dh)= %% měřítka
  for i=1 upto ((w-3mm-xpart bod)/dw):
    draw ((0,delka/2)--(0,-delka/2)) shifted (bod+i*(dw,0));
  endfor;
  for i=1 upto ((xpart bod)/dw):
    draw ((0,delka/2)--(0,-delka/2)) shifted (bod-i*(dw,0));
  endfor;
  for i=1 upto ((h-3mm-ypart bod)/dh):
    draw ((delka/2,0)--(-delka/2,0)) shifted (bod+i*(0,dh));
  endfor;
  for i=1 upto ((ypart bod)/dh):

```

```

    draw ((delka/2,0)--(-delka/2,0)) shifted (bod-i*(0,dh));
endfor;
enddef;

def net(expr dilek)= %% síť
  for i=0 upto w/dilek+.1:
    draw (i*dilek,0)--(i*dilek,h);
  endfor;
  for i=0 s upto h/dilek+.1:
    draw (0,i*dilek)--(w,i*dilek);
  endfor;
enddef;

def arrow(expr delka,smer,bod)= %% šipka
  begingroup
  save x,y;
  z1=(0,0); x2=x3=-delka; -y2=y3=2/7delka;
  draw (z2--z1--z3) rotated smer shifted bod;
  endgroup
enddef;

def farrow(expr delka,smer,bod)= %% plná šipka
  begingroup
  save x,y;
  z1=origin; x2=x3=x1-delka; y1-y2=y3-y1=2/7delka;
  filldraw ((z3--z2--z1--cycle) rotated smer shifted bod);
  endgroup
enddef;

def earrow(expr delka,smer,bod)= %% prazdná šipka
  begingroup
  save x,y;
  z1=origin; x2=x3=x1-delka; y1-y2=y3-y1=2/7delka;
  overdraw((z3--z2--z1--cycle) rotated smer shifted bod);
  endgroup
enddef;

def vector(expr bod,velikost,delka,smer)= %% vektor
  begingroup
  save x,y;
  z1=bod; z2=(velikost,0) rotated smer shifted bod;
  draw z1--z2;
  arrow(delka,smer,z2);
  endgroup
enddef;

def vect(expr boda,bodb,delka)= %% vektor určený počátkem a koncem
  begingroup
  save x,y;

```

```

    z1=boda; z2=bodb; smer:=angle(z2-z1);
    draw z1--z2; arrow(delka,smer,z2);
endgroup
enddef;

def dim(expr boda,bodb,delka)= %% kóta
begingroup
save x,y;
z1=boda; z2=bodb; smer:=angle(z2-z1);
draw z1--z2;
arrow(delka,smer,z2); arrow(delka,smer+180,z1);
endgroup
enddef;

def R(expr bod,uhel)= %% značka pro pravý úhel
draw quartercircle scaled 6mm rotated uhel shifted bod;
drawdot 1.7mm*dir45 rotated uhel shifted bod
withpen currentpen scaled 3;
enddef;

def circ(expr boda,bodb,bodc)= %% kružnice určená třemi body
begingroup
save x,y,r;
z1=boda; z2=bodb; z3=bodc; z12=1/2[z1,z2]; z23=1/2[z2,z3];
z0-z12=whatever*((z1-z2) rotated 90);
z0-z23=whatever*((z3-z2) rotated 90);
r=length(z1-z0);
draw fullcircle scaled 2r shifted z0;
endgroup
enddef;

def dashline(expr boda,bodb,n)= %% čárkovaná úsečka
for t=0 upto n:
draw (3t/(3n+2))[boda,bodb]--((3t+2)/(3n+2))[boda,bodb];
endfor;
enddef;

def dotline(expr boda,bodb,n)= %% tečkovaná úsečka
for t=0 upto n: drawdot (t/n)[boda,bodb]; endfor;
enddef;

def dashdotline(expr boda,bodb,n)= %% čerchovaná úsečka
for t=0 upto n:
draw (5t/(5n+3))[boda,bodb]--((5t+3)/(5n+3))[boda,bodb];
endfor;
for t=0 upto (n-1):
drawdot ((5t+4)/(5n+3))[boda,bodb];
endfor;
enddef;

```



```

def odot(expr bod,r)= %% kroužek
  overdraw(fullcircle scaled 2r shifted bod);
enddef;

def dashcircle(expr stred,r,uhel,n)= %% čárkovaná kružnice
  %% parametr uhel určuje polohu začátku první čárky
  for t=0 upto (n-1):
    arc(stred,r,t*360/n+uhel,(t+2/3)*360/n+uhel);
  endfor;
enddef;

def dashdotcircle(expr stred,r,uhel,n)= %% čerchovaná kružnice
  for t=0 upto (n-1):
    arc(stred,r,t*360/n+uhel,(t+3/5)*360/n+uhel);
    arc(stred,r,(t+4/5)*360/n+uhel,(t+4/5)*360/n+uhel);
  endfor;
enddef;

def dotcircle(expr stred,r,uhel,n)= %% tečkovaná kružnice
  for t=0 upto (n-1):
    arc(stred,r,t*360/n+uhel,t*360/n+uhel);
  endfor;
enddef;

def arc(expr stred,r,uhela,uhelb)= %% oblouk
  begingroup
    save x,y;
    z1=stred+dir(uhela)*r; z2=stred+dir(uhelb)*r;
    z3=stred+dir((uhelb+uhela)/2)*r;
    draw z1{dir(uhela+90)}..z3..{dir(uhelb+90)}z2;
  endgroup
enddef;

def dasharc(expr stred,r,uhela,uhelb,n)= %% čárkovaný oblouk
  for t=0 upto n:
    arc(stred,r,t*(uhelb-uhela)/(n+2/3)+uhela,
      (t+2/3)*(uhelb-uhela)/(n+2/3)+uhela);
  endfor;
enddef;

def dashdotarc(expr stred,r,uhela,uhelb,n)= %% čerchovaný oblouk
  for t=0 upto n:
    arc(stred,r,5*t*(uhelb-uhela)/(5*n+3)+uhela,
      (5*t+3)*(uhelb-uhela)/(5*n+3)+uhela);
  endfor;
  for t=0 upto (n-1):
    arc(stred,r,(5*t+4)*(uhelb-uhela)/(5*n+3)+uhela,
      (5*t+4)*(uhelb-uhela)/(5*n+3)+uhela);
  endfor;
enddef;

```

```

    endfor;
enddef;

def dotarc(expr stred,r,uheľa,uheľb,n)= %% tečkovaný oblouk
  for t=0 upto n:
    arc(stred,r,t*(uheľb-uhela)/n+uhela,t*(uheľb-uhela)/n+uhela);
  endfor;
enddef;

def dashpath(expr p,pocet_carek,pomer)= %% čárkovaná křivka
  begingroup
    %% pomer=délka_čarky/délka_mezery
    save t,tz,krok,s,d_carky,d_mez;
    krok:=0.001; t:=0; s:=0;
    forever:
      t:=t+krok;
      s:=s+length((point t of p)-(point (t-krok) of p));
      exitif t>length(p);
    endfor;
    d_carky:=s*pomer/(pocet_carek*pomer+pocet_carek-1);
    d_mez:=s/(pocet_carek*pomer+pocet_carek-1);
    t:=0;
    forever:
      tz:=t; s:=0;
      forever:
        t:=t+krok;
        s:=s+length((point t of p)-(point (t-krok) of p));
        exitif (s>=d_carky) or (t>length(p));
      endfor;
      draw subpath(tz,t) of p;
      s:=0;
      forever:
        t:=t+krok;
        s:=s+length((point t of p)-(point (t-krok) of p));
        exitif (s>=d_mez) or (t>length(p));
      endfor;
      exitif t>length(p);
    endfor;
  endgroup
enddef;

def dashdotpath(expr p,pocet_carek,pomer)= %% čerchovaná křivka
  begingroup
    %% pomer=délka_čarky/délka_mezery
    save t,tz,krok,s,d_carky,d_mez;
    krok:=0.001; t:=0; s:=0;
    forever:
      t:=t+krok;
      s:=s+length((point t of p)-(point (t-krok) of p));
      exitif t>length(p);
    endfor;
  endgroup
enddef;

```

```

endfor;
d_carky:=s*pomer/(pocet_carek*pomer+2pocet_carek-2);
d_mez:=s/(pocet_carek*pomer+2pocet_carek-2);
t:=0;
forever:
  tz:=t; s:=0;
  forever:
    t:=t+krok;
    s:=s+length((point t of p)-(point (t-krok) of p));
    exitif (s>=d_carky) or (t>length(p));
  endfor;
draw subpath(tz,t) of p;
s:=0;
forever:
  t:=t+krok;
  s:=s+length((point t of p)-(point (t-krok) of p));
  exitif (s>=d_mez) or (t>length(p));
endfor;
drawdot point t of p;
s:=0;
forever:
  t:=t+krok;
  s:=s+length((point t of p)-(point (t-krok) of p));
  exitif (s>=d_mez) or (t>length(p));
endfor;
exitif t>length(p);
endfor;
endgroup
enddef;

def dashcyclepath(expr p,pocet_carek,pomer)=
  begingroup
  save t,tz,krok,s,d_carky,d_mez,poc;
  krok:=0.001; t:=0; s:=0;          %% čárkovaná uzavřená křivka
  forever:                          %% pomer=délka_čarky/délka_mezery
    t:=t+krok;                      %% první a poslední čárka je dvakrát kratší
    s:=s+length((point t of p)-(point (t-krok) of p));
    exitif t>length(p);
  endfor;
  d_carky:=s*pomer/(pocet_carek*pomer+pocet_carek);
  d_mez:=s/(pocet_carek*pomer+pocet_carek);
  t:=0; poc:=0;
  forever:
    tz:=t; s:=0; poc:=poc+1;
    forever:
      t:=t+krok;
      s:=s+length((point t of p)-(point (t-krok) of p));
      if ((poc=1) or (poc=pocet_carek+1)):
        exitif (s>=d_carky/2) or (t>length(p));
      endif;
    endfor;
  endfor;
enddef;

```

```

    fi;
    exitif s>=d_carky;
endfor;
draw subpath(tz,t) of p;
exitif t>length(p);
s:=0;
forever:
    t:=t+krok;
    s:=s+length((point t of p)-(point (t-krok) of p));
    exitif (s>=d_mez) or (t>length(p));
endfor;
exitif t>length(p);
endfor;
endgroup
enddef;

def dotpath(expr p,pocet_tecek)=      %% tečkovaná křivka
begingroup
save t,krok,s,d_mez;
krok:=0.001; t:=0; s:=0;
forever:
    t:=t+krok;
    s:=s+length((point t of p)-(point (t-krok) of p));
    exitif t>length(p);
endfor;
d_mez:=s/(pocet_tecek-1);
t:=0;
drawdot(point t of p);
forever:
    s:=0;
    forever:
        t:=t+krok;
        s:=s+length((point t of p)-(point (t-krok) of p));
        exitif (s>=d_mez) or (t>length(p));
    endfor;
    drawdot(point t of p);
    exitif t>length(p);
endfor;
endgroup
enddef;

```

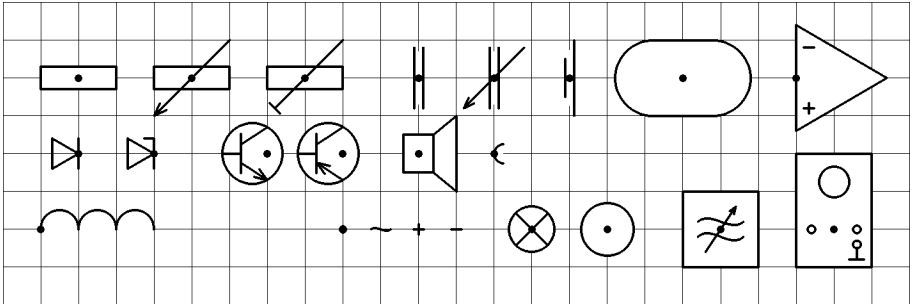
## C Elektrotechnické značky

V tomto dodatku je soubor maker pro kreslení elektrotechnických značek v jednoduchých schématech. Všechny značky nakreslené na obrázku mají základní velikost a nulový úhel otočení. Černé tečky vyznačují referenční body. V následujícím seznamu jsou uvedeny povinné parametry určující polohu značky ve

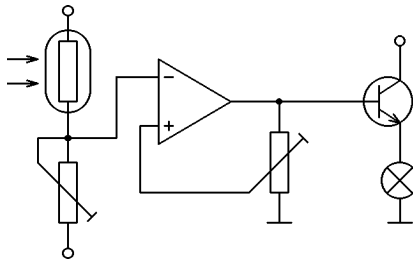
schématu. Jsou to: poloha referenčního bodu ve schématu, úhel otočení (jen u některých značek) a u cívky počet obloučků.

Velikost jednotlivých značek můžeme upravit přidáním nepovinného parametru *transformace* zapsaného ve tvaru *scaled číslo*. Potřebujeme-li značku zrcadlově převrácenou, připišeme *reflectedabout(left,right)* nebo *reflectedabout(up,down)*. Velikost všech značek můžeme upravit najednou pomocí parametru *scalingfactor*.

Vše vidíme v ukázce na následující stránce. Schéma bylo navrženo na milimetrovém papíře se značkami v základní velikosti. Zmenšení na 80 % bylo dosaženo volbami *u#=.8mm#* a *scalingfactor:=.8*.



rez(*b,u*) pot(*b,u*) trimr(*b,u*) kond(*b,u*) lkond(*b,u*) zdroj(*b,u*)  
 oval(*b,u*) oz(*b,u*)  
 dioda(*b,u*) zdioda(*b,u*) npn(*b,u*) pnp(*b,u*) rep(*b,u*) zdirka(*b,u*)  
 civka(*n*)(*b,u*) spoj(*b*) vlnka(*b*) plus(*b*) minus(*b*) zarovka(*b*)  
 meridlo(*b*) tongen(*b*) oscil(*b*)



```
mode_setup; u#:=.8mm#; define_pixels(u);
input gjkt.mf; input znacky1.mf;
```

```
beginchar(2,70u#,40u#,0);
scalingfactor:=.8;
penc .3u;
oval((10u,30u),90)scaled .75;
draw((10,40)--(10,0))scaled u;
draw((35,25)--(65,25))scaled u;
```

```

draw((65,5)--(65,35))scaled u;
draw((45,5)--(45,25))scaled u;
draw((5,15)--(5,19)--(18,19)--(18,29)--(25,29))scaled u;
draw((25,21)--(22,21)--(22,10)--(40,10))scaled u;
draw(left--right)scaled 2u shifted (45u,5u);
draw(left--right)scaled 2u shifted (65u,5u);
odot((10u,0),.8u); odot((10u,40u),.8u); odot((65u,35u),.8u);
vector((0,28u),5u,1.8u,0); vector((0,32u),5u,1.8u,0);
rez((10u,30u),90); trimr((10u,10u),90);
trimr((45u,15u),90) reflectedabout (up,down);
oz((25u,25u),0); npn((65u,25u),0);
zarovka((65u,12u)); spoj((10u,19u)); spoj((45u,25u));
endchar;
end.

```

Ve zdrojovém textu maker je použit poněkud jiný mechanismus zpracování parametrů — ukažme si to na příkladě. Makro *dioda* očekává parametry ve tvaru (*bod*, *úhel*) *transformace*; . Tyto parametry však samo nezpracovává ani nevyhodnocuje, ale předává je jako *text* makru *uloz*. To teprve vyhodnotí parametry *bod*, *úhel*, *scalingfactor* a *transformace*. Parametr *transformace* sloužící ke zvětšení, případně ke stranovému převrácení značky, je nepovinný — proto je zaveden jako *text*. Všechny parametry slouží k vytvoření transformace, která umístí značku ve správné velikosti a orientaci do zvolené polohy.

Pokud makro vyžaduje pouze parametry (*bod*) *transformace*; , nepředáváme je makru *uloz*, ale *vloz*.

Návrat k původní transformaci po nakreslení značky obstarává makro *obnov*.

```
%% znacky.mf
```

```
%% posledni uprava 19. 12. 1997
```

```

newinternal scalingfactor; %% promenna urcujici velikost zvetseni
scalingfactor:=1;

transform now;

def uloz(expr bod,uhel) text tr= %% makro pro nastaveni transformaci
  now:=currenttransform;
  currenttransform:=identity tr scaled scalingfactor rotated uhel
  shifted bod;
  begingroup
enddef;

def vloz(expr bod) text tr= %% makro pro nastaveni transformaci
  now:=currenttransform;
  currenttransform:=identity tr scaled scalingfactor shifted bod;
  begingroup
enddef;

def obnov= %% pomocna procedura pro obnoveni transformaci
  endgroup;
  currenttransform:=now;
enddef;

```

```

def civka(expr p) text params=
  uloz params;
  save i;
  erase draw (0,0)--(p*5mm,0);
  for i=1 upto p:
    draw halfcircle scaled 5mm shifted (i*5mm-2.5mm,0);
  endfor;
  obnov;
enddef;

def dioda text params=
  uloz params;
  draw (origin--dir150--dir-150--origin--(0,ypart(dir150))
    --(0,ypart(dir-150))) scaled 4mm;
  obnov;
enddef;

def zdioda text params=
  dioda params;
  uloz params;
  draw (0,2.05mm)--(-1.3mm,2.05mm);
  obnov;
enddef;

def kond text params=
  uloz params;
  erase fill unitsquare xscaled 1.2mm yscaled 7mm shifted (-0.6mm,-3.5mm);
  draw (-0.6mm,-4mm)--(-0.6mm,4mm); draw (0.6mm,-4mm)--(0.6mm,4mm);
  obnov;
enddef;

def lkond text params=
  kond params;
  uloz params;
  draw (-4mm,-4mm)--(4mm,4mm);
  draw (-4mm,-4mm)+1.5mm*dir25--(-4mm,-4mm)--(-4mm,-4mm)+1.5mm*dir65;
  obnov;
enddef;

def rez text params=
  uloz params;
  erase fill unitsquare xscaled 10mm yscaled 3mm shifted (-5mm,-1.5mm);
  draw unitsquare xscaled 10mm yscaled 3mm shifted (-5mm,-1.5mm);
  obnov;
enddef;

def pot text params=
  rez params;

```

```

uloz params;
draw (-5mm,-5mm)--(5mm,5mm);
draw (-5mm,-5mm)+1.8mm*dir25--(-5mm,-5mm)--(-5mm,-5mm)+1.8mm*dir65;
obnov;
enddef;

def trimr text params=
  rez params;
  uloz params;
  draw (-4mm,-4mm)--(5mm,5mm); draw (-4.7mm,-3.3mm)--(-3.3mm,-4.7mm);
  obnov;
enddef;

def zdroj text params=
  uloz params;
  erase fill (-0.6mm,-2.5mm)--(-0.6mm,2.5mm)--(0.6mm,5mm)
    --(0.6mm,-5mm)--cycle;
  draw (0.6mm,-2.5mm)--(-0.6mm,2.5mm); draw (0.6mm,-5mm)--(0.6mm,5mm);
  obnov;
enddef;

def tranzist=
  save x,y,R;
  z1=(-6mm,0); z2=(0,3.5mm); z3=(0,-3.5mm);
  z12=1/2[z1,z2];
  z0-z12=whatever*((z1-z2) rotated 90); y0=0;
  R=length(z1-z0);
  erase fill fullcircle scaled 2R shifted z0;
  draw fullcircle scaled 2R shifted z0;
  z4=(-3.5mm,0); z5=(-3.5mm,-2.6mm); z6=(-3.5mm,2.6mm);
  z7=(-3.5mm,-1.2mm); z8=(-3.5mm,1.2mm);
  draw z1--z4; draw z5--z6; draw z7--z3; draw z8--z2;
enddef;

def npn text params=
  uloz params;
  save x,y;
  tranzist;
  z3=(0,-3.5mm); z7=(-3.5mm,-1.2mm);
  draw z3+1.8mm*dir(angle(z7-z3)+20)--z3--z3+1.8mm*dir(angle(z7-z3)-20);
  obnov;
enddef;

def pnp text params=
  uloz params;
  save x,y;
  tranzist;
  z3=(0,-3.5mm); z7=(-3.5mm,-1.2mm);
  draw z7+1.8mm*dir(angle(z3-z7)+20)--z7--z7+1.8mm*dir(angle(z3-z7)-20);

```



```

    obnov;
enddef;

def plus text params=
    vloz params;
    draw (right--left) scaled .7mm; draw (up--down) scaled .7mm;
    obnov;
enddef;

def minus text params=
    vloz params; draw (right--left) scaled .7mm; obnov;
enddef;

def oz text params=
    uloz params;
    save x,y;
    z1=(12mm,0); z2=(0,7mm); z3=(0,-7mm); z4=(1.7mm,4mm);
    z5=(1.7mm,-4mm);
    erase fill z1--z2--z3--cycle; draw z1--z2--z3--cycle;
    draw (right--left) scaled .7mm shifted z4;
    draw (right--left) scaled .7mm shifted z5;
    draw (up--down) scaled .7mm shifted z5;
    obnov;
enddef;

def zarovka text params=
    vloz params;
    erase fill fullcircle scaled 6mm; draw fullcircle scaled 6mm;
    draw (dir45--dir-135) scaled 3mm; draw (dir-45--dir135) scaled 3mm;
    obnov;
enddef;

def meridlo text params=
    vloz params;
    erase fill fullcircle scaled 7mm; draw fullcircle scaled 7mm;
    obnov;
enddef;

def zdirka text params=
    uloz params;
    erase fill (halfcircle--cycle) rotated 90 shifted (.5,0) scaled 2.5mm;
    draw halfcircle rotated 90 shifted (.5,0) scaled 2.5mm;
    obnov;
enddef;

def rep text params=
    uloz params;
    erase fill (2mm,-2.5mm)--(5mm,-5mm)--(5mm,5mm)--(2mm,2.5mm)--
        (-2mm,2.5mm)--(-2mm,-2.5mm)--cycle;

```

```

draw unitsquare xscaled 4mm yscaled 5mm shifted (-2mm,-2.5mm);
draw (2mm,-2.5mm)--(5mm,-5mm)--(5mm,5mm)--(2mm,2.5mm)--cycle;
obnov;
enddef;

def vlna=
draw (left{dir45}..origin..{dir45}right)
enddef;

def vlnka text params=
vloz params; vlna scaled 1.3mm; obnov;
enddef;

def tongen text params=
vloz params;
erase fill unitsquare scaled 10mm shifted (-5mm,-5mm);
draw unitsquare scaled 10mm shifted (-5mm,-5mm);
vlna scaled 3mm shifted (0,1mm); vlna scaled 3mm shifted (0,-1mm);
draw (-2mm,-3mm)--(2mm,3mm);
draw (2mm,3mm)+mm*dir(angle(-2,-3)+20)--(2mm,3mm)
--(2mm,3mm)+mm*dir(angle(-2,-3)-20);
obnov;
enddef;

def oscil text params=
vloz params;
erase fill unitsquare xscaled 10mm yscaled 15mm shifted (-5mm,-5mm);
draw unitsquare xscaled 10mm yscaled 15mm shifted (-5mm,-5mm);
draw fullcircle scaled 4mm shifted (0,6.25mm);
draw fullcircle scaled mm shifted (-3mm,0);
draw fullcircle scaled mm shifted (3mm,0);
draw (3mm,-4mm)--(3mm,-2mm); draw (2mm,-4mm)--(4mm,-4mm);
erase fill fullcircle scaled mm shifted (3mm,-2mm);
draw fullcircle scaled mm shifted (3mm,-2mm);
obnov;
enddef;

def spoj text params=
vloz params; filldraw fullcircle scaled .8mm; obnov;
enddef;

def oval text params=
uloz params;
save p;
path p;
p=halfcircle scaled 10mm rotated 270 shifted (4mm,0)--
halfcircle scaled 10mm rotated 90 shifted (-4mm,0)--cycle;
erase fill p;
draw p;
obnov;
enddef;

```

## Literatura a internetové odkazy

- [1] Knuth, D. E.: The  $\text{\TeX}$ book. Addison-Wesley, Reading, Massachusetts, 1994.
- [2] Knuth, D. E.: The  $\text{\METAFONT}$ book. Addison-Wesley, Reading, Massachusetts, 1994.
- [3] Olšák, P.: Typografický systém  $\text{\TeX}$ . Praha, 1995.
- [4] Rybička, J.:  $\text{\LaTeX}$  pro začátečníky. Konvoj, Brno, 1995.
- [5] Horák, K.: Můj zápas s  $\text{\METAFONTem}$  aneb pérovky a jiná zvěřstva (s ukázkami). Zpravodaj Československého sdružení uživatelů  $\text{\TeXu}$ , **1** (3), strana? (1991), Praha
- [6] Balvínová, A., Bílý M. Textové informační systémy, sázecí systém  $\text{\LaTeX}$ . ČVUT, Praha, 1994.
- [7] bm2font — <ftp://ftp.muni.cz/pub/tex/CTAN/graphics/bm2font/>
- [8] GWS — <http://www.mindworkshop.com/alchemy/alchemy.html>
- [9] Paint Shop Pro — <http://www.jsac.com>
- [10] scr2pcx — <http://www.kolej.mff.cuni.cz/~broz/kreslime/>
- [11] pcxview — <http://www.kolej.mff.cuni.cz/~broz/kreslime/>
- [12] Screen Thief — <http://www.nildram.co.uk/>

## Rejstřík

# — `em#:=1/3pt#`; jednotka `em` nezávisí na rozlišení  
& — `draw p1 & p2`; volné navázání cest `p1`, `p2`  
“ ” — `s=""`; prázdný řetězec  
[] — `z3=.3[z1,z2]`; lineární kombinace bodů, bod `z3` leží v 3/10 mezi body `z1`, `z2`  
+ — `c=a+b`; sčítání  
++ — `c=a++b`;  $\sim \sqrt{a*a+b*b}$  Pythagorejské sčítání 10  
+-+ — `c=a+-+b`;  $\sim \sqrt{a*a-b*b}$  Pythagorejské odčítání 10  
- — `c=a-b`; odečítání  
-- — `z1--z2`; rovná čára 16  
--- —  $\sim$  `..tension infinity`.. křivka s maximálním napětím, hladký přechod úsečky na křivku  
\* — `c=a*b`; násobení  
\*\* — `c=a**b`; umocňování 10  
/ — `c=a/b`; dělení  
:= — `a:=b`; přiřazení  
= — `a=b`; porovnání, lineární rovnice  
.. — `z1{dir40}..z2..z3`; volná křivka, z bodu `z1` vychází pod úhlem 40° 16  
... —  $\sim$  `..tension atleast 1..` křivka s napětím větším než 1 18  
abs — `abs z`; absolutní hodnota vektoru 7  
addto — `addto currentpicture also V`; k aktuálnímu obrázku přičte obrázek `V` 18  
also — viz `addto` 18  
and — logický součin 25  
angle — `angle(z1-z2)`; úhel vektoru v intervalu (-180°, 180°) 7  
beginchar — `beginchar("A",11pt#,11pt#,0)`; začátek definice znaku, umístění v ASCII ("A" nebo 65), šířka, výška nad účarím, hloubka pod účarím (uloží se do proměnných `w`, `h`, `d`) 4, 9, 11  
begingroup — `begingroup` začátek skupiny 28  
*Beziérový křivky* — 12  
boolean — `boolean b`; logická proměnná 25  
bot — `bot z1=(0,0)`; zarovnání na dolní okraj podle tloušťky pera  
ceiling — `ceiling x`; zaokrouhlování nahoru 10  
clearit — `clearit`;  $\sim$  `currentpicture:=nullpicture`; vymazání obrázku  
clear\_pen\_memory — `clear_pen_memory`; smazání dříve uchovaných per  
controls — `draw z1..controls z3 and z4..z2`; křivka s pomocnými body `z3`, `z4`  
cosd — `cosd40`;  $\cos 40^\circ$  7, 10  
cull —  $\sim$  `cullit`, viz `keeping` 23  
cullit — `cullit`; záporné hodnoty bodu nahradí 0, kladné hodnoty 1  
currentpicture — `V:=currentpicture`; aktuální obrázek 18, 19, 23  
cutdraw — `cutdraw p`;  $\sim$  `draw p`; `cutoff(point 0 of p, 180+angle direction 0 of p)`; `cutoff(point infinity of p, angle direction infinity of p)`;

**cutoff** — **cutoff**( $z, \Phi$ ); v bodě  $z$  smaže polovinu bodů pera **currentpen** se směrem v intervalu  $(\Phi - 90^\circ, \Phi + 90^\circ)$

**cycle** — **fill**  $z1..z2..z3..$ **cycle**; uzavření cesty 11, 16

**def** — **def** **arrow**(**expr**  $x, y, uhel$ )= začátek definice makra 27

**define\_pixels** — **define\_pixels**(**u**); přepočítá jednotky na pixely podle parametru  $\backslash mode=$ ,  $u=u\#*h\text{ppp}$ , kde **hppp** je počet pixelů na bod v horizontálním směru, další viz soubor **plain.mf**

**dir** — **dir**40; jednotkový vektor svírající úhel  $40^\circ$  s osou  $x$ ; 7, 11

**direction** — **direction**  $t$  of  $p$ ; směr cesty  $p$  v čase  $t$  18

**directiontime** — **directiontime**  $z$  of  $p$ ; čas  $t$ , pro který má tečna v bodě křivky směr vektoru  $z$  17

**dotprod** —  $z1$  **dotprod**  $z2$ ; skalární součin

**down** —  $z1:=$ **down**; bod  $(0, -1)$  7

**draw** — **draw**  $p$ ; nakreslení cesty  $p$  22, 30

**drawdot** — **drawdot**  $zk$ ; vykreslení tečky 22

**end** — **end**; konec souboru, stisknout  $\langle \text{Enter} \rangle$  5

**endchar** — **endchar**; konec definice znaku 5, 9

**enddef** — **enddef**; konec definice 27

**endfor** — **endfor** konec **for** cyklu

**endgroup** — **endgroup**; konec skupiny 28

**epsilon** —  $\epsilon = \frac{1}{65536}$  nejmenší číslo větší než nula 10

**erase** — **erase** **draw**  $p$ ;  $\sim$  **cullit**; **undraw**  $p$ ; **cullit**; 22

**exitif** — **exitif**  $b=true$ ; podmínka pro ukončení **for** cyklu

**expr** — **def**(**expr**  $x$ )= parametr libovolného typu 27

**fill** — **fill**  $p$ ; vyplnění oblasti ohraničené cestou  $p$  22

**filldraw** — **filldraw**  $p$ ;  $\sim$  **draw**  $p$ ; **fill**  $p$ ; 18

**flex** — **flex**( $z1, z2, \dots, zn$ )  $\sim z1..z2\{zn-z1\}.....zn$

**floor** — **floor**  $x$ ; celá část  $x$  10

**fontdimen** — **fontdimen** 3: 2.5, 6.5, 0, 4x v  $\text{T}_\text{E}\text{X}$ u se tyto hodnoty volají příkazy  $\backslash fontdimen3$  až  $\backslash fontdimen6$ , v **plain**u je  $\backslash fontdimen1$  až 7 rezervováno

**for** — **for**  $x=x1$  upto  $x2$ : **text**( $x$ ); **endfor**;  
 nebo **for**  $x=x1$  step  $x2$  until  $x3$ : **text**( $x$ ); **endfor**;  
 nebo **for**  $k=1, 2, 3$ : **text**( $x[k]$ ); **endfor**; cyklus **for** 8, 25, 29

**forever** — **forever**: **text**; **exitif**  $b=true$ ; **endfor**; cyklus **repeat** until 26

**fullcircle** — **draw** **fullcircle**; kružnice o jednotkovém průměru, střed  $(0, 0)$  13

**halfcircle** — **draw** **halfcircle**; půlkružnice o jednotkovém průměru, střed  $(0, 0)$  13

**identity** —  $t=$ **identity**; identické zobrazení 15

**if**: **elseif**: **else**: **fi** — **if**  $podm_1: text_1$  **elseif**  $podm_2: text_2$  **else**:  $text_3$  **fi**; podmínka 25

**infinity** —  $z1=$ **point** **infinity** of  $p$ ; konečný bod cesty  $p$ ; 10

**input** — **input** **makra.mf**; vložení souboru **\*.mf**

**interim** — **interim**  $x:=0.4$ ; po **endgroup** (součást **endchar**) bude mít  $x$  hodnotu jako před **interim**

**intersectionpoint** —  $(x,y):=p1$  **intersectionpoint**  $p2$ ; bod  $(x,y)$ , kde se protínají cesty  $p1, p2$  18  
**intersectiontimes** —  $(t1,t2)=p1$  **intersectiontimes**  $p2$ ; časy  $t1, t2$ , kdy se protínají cesty  $p1, p2$  17  
**keeping** — **cull currentpicture keeping**(2,4); vymaže body, které nejsou nakresleny  $2\times, 3\times$  nebo  $4\times$  23  
**known** — **known**  $x$ ; logický výraz, true pokud je  $x$  definováno  
**label** — **label**(1,2,3); popis bodů  $z1, z2, z3$  pro program GFtoDVI  
**left** — **draw**  $z1\{left\}..z2$ ; bod  $(-1,0)$  7  
**length** — **length**( $z2-z1$ ); **length**  $s$ ; **length**  $p$ ; délka vektoru, počet znaků, počet bodů na cestě zmenšený o 1 17, 25  
**lft** — **lft**  $z1=(0,y)$ ; zarovnání na levý okraj podle tloušťky pera  
**ligtable** — **ligtable** "A":"V" **kern** -2pt; kerning, ligatury, vyskytne-li se V po A, posune se o 2pt doleva  
**makepen** — **makepen**  $p$ ; vytvoření pera podle cesty  $p$  22  
**makelabel** — **makelabel**( $z1$ , "bod 1"); popis bodu  $z1$  pro program GFtoDVI, automatické umístění, **makelabel.top.nodot**(); umístění nahoře, bez tečky  
**max** — **d:=max**( $a,b,c$ ); maximum z množiny 10  
**message** — **message** "text"; výpis textu na obrazovku při překladu (např. spolu s **show**)  
**mexp** — **mexp**  $x$ ;  $\sim e^{x/256}$  10  
**min** — **d:=min**( $a,b,c$ ); minimum z množiny 10  
**mlog** — **mlog**  $x$ ;  $\sim 256x$  10  
**mode** — **mode=proof**; nastavení módu (přehled viz soubor **local.mf**)  
**mode\_def** — **mode\_def laserjet=** definice módu  
**mode\_setup** — **mode\_setup**; nastaví hodnoty jednotek podle parametrů **\mode=**, **\mag=4**  
**normaldeviate** — **x:=normaldeviate**; náhodné číslo s normálním rozdělením 32  
**nullpen** — **pickup nullpen**; pero neviditelný bod, **beginchar** nastaví aktuální pero **currentpen:=nullpen**  
**numeric** — **numeric**  $a$ []; nepovinná deklarace proměnné typu číslo 25  
**or** — logický součet 25  
**origin** — **z1=origin**; bod  $(0,0)$  7  
**pair** — **pair**  $z$ ; nepovinná deklarace bodu 25  
**path** — **path**  $p$ []; deklarace proměnné typu cesta, řada bodů 11, 25  
**pencircle** — **pickup pencircle**; pero jednotková kružnice 5, 21  
**penlabels** — **penlabels**(1,2); popis bodů  $z1, z2, z1r, z2r, z1l, z2l$  ve výstupu programu GFtoDVI  
**penpos** — **penpos1**( $a,b$ )  $\sim z1=.5[z1l, z1r]$ ;  $z1r=z1l+(a,0)$  **rotated**  $b$ ; deklarace  $z1l, z1r$ , viz **penstroke**  
**penrazor** — **pickup penrazor**; pero nejtenčí jednotková úsečka v ose  $x$  se středem v počátku 21  
**pensquare** — **pickup pensquare**; pero jednotkový čtverec 21

penstroke — penstroke  $z1e..z2e$ ;  $\sim$  fill  $z1l..z2l--z2r..z1r--$ cycle;  
viz penpos

**pickup** — pickup pencircle xscaled 2pt yscaled 1pt rotated 30;  
nastavení aktuálního pera 5, 21

picture — picture V[]; deklarace proměnné typu obrázků 18, 25

point — point t of p; bod na cestě p v čase t 17

postcontrol —  $z1:=$ postcontrol t of p; kontrolní bod po bodu s časem t

precontrol —  $z1:=$ precontrol t of p; kontrolní bod před bodem s časem t

*přířazovací příkazy* — 9

quartercircle — draw quartercircle; čtvrtkružnice s jednotkovým průměrem, střed (0,0) 13

randomseed — randomseed:=2; reprodukovatelná posloupnost náhodných čísel 32

**reflectedabout** — (x,y) reflectedabout((a,b),(c,d)); osová souměrnost podle vektoru (a,b)-(c,d) 14, 16, 31

reverse — reverse p; body cesty v opačném pořadí 16

right — draw  $z1\{\text{right}\}..z2$ ; bod (1,0) 7

**rotated** — (x,y) rotated  $\Phi$ ;  $\sim$  ( $x \cos \Phi - y \sin \Phi, x \sin \Phi + y \cos \Phi$ ) otočení okolo (0,0) 14, 18

**rotatedaround** — (x,y) rotatedaround((a,b), $\Phi$ ); otočení (x,y) o úhel  $\Phi$  okolo bodu (a,b) 14, 31

round — round x; zaokrouhlování 10

rt — rt  $z1=(x,y)$ ; zarovnání na pravý okraj podle tloušťky pera

sind — sind40;  $\sin 40^\circ$  7, 10

save — save x; po endgroup bude mít x hodnotu jako před begingroup 28

savepen — p:=savepen; uchování aktuálního pera

**scaled** — (x,y) scaled a;  $\sim$  (ax, ay) násobení v obou složkách 14, 16

screenchars — screenchars; showit po endchar

screenstrokes — screenstrokes; showit po draw, fill

**shifted** — (x,y) shifted (a,b);  $\sim$  (x+a, y+b) posunutí 14, 16

show — show a,b; výpis proměnných na obrazovku 9, 12

showdependencies — showdependencies; výpis proměnných s neznámou hodnotou, na kterých jiné proměnné závisí

showit — showit; vykreslení aktuálního obrázku

showvariable — showvariable s; výpis pole proměnných s1, s2 ...

slanted — (x,y) slanted a;  $\sim$  (x+sy, y) zešikmení 14

special — special "labelfontat" numspecial 20; nastavení velikosti popisu pro GFtoDVI

$\backslash$ special — 40

sqrt — b:=sqrt(a); odmocnina 10

subpath — subpath (t1,t2) of p; část cesty p mezi časy t1, t2 17

superellipse — superellipse(right,top,left,bottom,superiness); ovál

tension — draw  $z1..tension$  1 and 2.5.. $z2$ ; napětí křivky u bodu  $z1$  a  $z2$  11

*tiskový bod* — 4

top — top  $z1=(0,y)$ ; zarovnání na horní okraj podle tloušťky pera

**transform** — **transform t**; proměnná typu transformace 15, 25  
**transformed** — **currentpicture transformed t**; transformace 15  
**undraw** — **undraw p**; inverze **draw**, sníží počet nakreslení bodů 24  
**undrawdot** — **undrawdot z1**; inverze **drawdot**  
**unfill** — **unfill p**; inverze **fill** 22  
**unfilldraw** — **unfilldraw p**; inverze **filldraw**  
**uniformdeviate** — **uniformdeviate x**; náhodné číslo od 0 do **x** 32  
**unitsquare** — **draw unitsquare**; jednotkový čtverec, levý dolní roh (0,0) 13  
**unitvector** — **unitvector(x,y)**; jednotkový vektor daného směru 7  
**unknown** — **if unknown mag: mag:=1; fi**; logický výraz, false pokud je **mag** defino-  
váno  
**up** — **z1:=up**; bod (0,1) 7  
**whatever** — **z1=z2+whatever\*(z3-z4)**; proměnná, která nabývá hodnoty při použití,  
body **z1, z2** leží na přímce se směrnici **z3-z4** 8  
**xpart** — **xpart z**; *x*-ová část 7, 16  
**xscaled** — **(x,y) xscaled a**;  $\sim (ax, y)$  násobení souřadnice *x* číslem 14, 30  
**ypart** — **ypart z**; *y*-ová část 7, 16  
**yscaled** — **(x,y) yscaled a**;  $\sim (x, ay)$  násobení souřadnice *y* číslem 14, 30  
**zscaled** — **(x,y) zscaled (a,b)**;  $\sim (xa - yb, xb + ya)$  násobení vektorů jako komplex-  
ních čísel 14