Zpravodaj Československého sdružení uživatelů TeXu

Karel Horák

Můj zápas s METAFONTem aneb pérovky a jiná zvěrstva (s ukázkami)

Zpravodaj Československého sdružení uživatelů TeXu, Vol. 1 (1991), No. 3, 12–23

Persistent URL: http://dml.cz/dmlcz/148796

Můj zápas s METAFONTem aneb Pérovky a jiná zvěrstva (s ukázkami)

V minulém čísle našeho TEXbulletinu byl značný prostor věnován spojení TEXu a grafiky. Rád bych v tomto nejspíše zmateném a nesouvislém příspěvku případného čtenáře přesvědčil, že není nic přirozenějšího než k tomuto účelu používat METAFONT.

Během první poloviny tohoto roku jsem postupně METAFONTem vytvořil více než tři stovky obrázků převážně z elementární geometrie (k úlohám MO, do Encyklopedie mladého matematika, kterou jsem překládal s kolegou Vrbou) a také pro sborník konference o teorii grafů, o jehož konečnou editaci v TEXu jsem byl kolegy v ústavu požádán. Po prvních nesmělých pokusech, zdlouhavých soubojích se syntaktickými pravidly a zavilými hláškami (jež se podobně jako u TEXu zapisují do souboru s příponou `.log` a ne vždy jsou jednoznačně srozumitelné), jsem zvládal obrázky stále složitější — dokonce i ty, o kterých jsem na samém začátku odmítal vůbec uvažovat.

Nejhorší je obvykle začátek. Už před rokem mi Jiří Veselý laskavě zapůjčil Knuthův METAFONTbook. Ale co dál? Program nebyl nikde v mé blízkosti instalován (na rozdíl od TEXu, kde stačilo prodělat čtvrthodinou instruktáž, jak psát soubor a jak jednotlivé programy spouštět, a bylo možno začít hned pracovat), všechny informace, které se ke mně donesly, byly většinou kusé a neurčité a jen málo povzbudivé (ještě před rokem, kdy nebyly dostupné Mattesovy ovladače obrazovky a tiskáren, byly pochybnosti, zda se vůbec dá vytisknout obrázek o velikosti několika cm).

Nakonec ale došlo ke šťastné souhře okolností. Na podzim jsem se v Praze sešel s Láďou Lhotkou, s nímž jsem, jak se ukázalo, kdysi válčil za šachovnicí, a ten mi velice pomohl svým článkem o METAFONTu, který patrně znají účastníci loňské školy v Cikháji. K jeho instruktivnímu příkladu jsem se nejednou vracel a z několika jeho maker dodnes požívám šipku (`arrow`). V tomto směru nutno konstatovat, že METAFONTbook není nejvhodnější příručkou pro začátečníka, který navíc chce „jenom“ kreslit jednoduché obrázky. Zároveň jsem v té době dostal do ruky i kompletní sadu emTEXu, jehož je METAFONT součástí, a konečně mi moc pomohlo zapůjčení počítače od kamaráda Jirky Matouška.

K práci nakonec nepotřebujeme nic víc než školní znalosti analytické geometrie, a elementární geometrie vůbec. METAFONT zná kartézské souřadnice, dovede všechna shodná zobrazení a některé podobnosti — ty ostatní ho snadno doučíme, a navíc umí i osovou afinitu. Dovede hladce spojovat body (to činí pomocí kubických křivek, tzv. Bézierových splajnů).

Svojí strukturou se velmi podobá TEXu (obejde se ovšem bez \, protože všechna slova jsou jen příkazy, proměnné apod. — žádné texty nepíšeme, pokud si chceme nějakou poznámku napsat, poslouží nám stejně jako v TEXu komentovací charakter %), jeho základní příkazy umožňují používat podmínky a cykly. Necítím se povolán zabývat se tímto programem příliš podrobně, zájemce odkazuji na Knuthův METAFONTbook, který je ostatně dostupný už i v elektronické podobě. Raději na několika příkladech ukážu něco z toho, co jsem se zatím s METAFONTem dělat naučil — to by snad mělo každému, kdo si METAFONT opatří a nainstaluje na dostupném počítači, pomoci tvořit jednoduché obrázky.

Ještě snad stručný aktuální komentář k Mattesově verzi METAFONTu 2.0. Náhodou se mi během setkání ve Skalském dvoře dostala do ruky i poněkud novější verze

sbMETAFONTu 2.7 (konvergence k číslu e se začíná nebezpečně zrychlovat). Stačil jsem tedy zjistit, že tato verze je podstatně rychlejší zejména při generování kompletní sady jednotlivých řezů (fontů). U mých obrázků to už nebylo tak patrné, nepodařilo se mi ale sbMETAFONT přinutit k okamžitému výstupu v grafickém módu,[1] přestože jsem ho několikrát znovu inicializoval pro různé grafické režimy (`plain.mf` umožňuje definovat počet řádků a sloupců na obrazovce). A navíc je sbMETAFONT mnohem náročnější na operační paměť, což si sám testuje, a není-li spokojen, dál se s vámi nebaví.

Takže teď doporučuji, abyste si další text jen zběžně prohlédli a utíkali si opatřit METAFONT (spolu s mnoha pomocnými programy je součástí veřejně dostupného balíku označeného jako emTEX, který je k dispozici prostřednictvím našeho sdružení). Pro začátek bude stačit, když si nainstalujete do adresáře `C:\TEX` programy `mf.exe`, `gftodvi.exe` (zejména máte-li k dispozici jen grafickou kartu Hercules), `gftopk.exe` a do podadresáře `MFBASES` soubory `plain.bas` a `mf.poo`, které plní obdobnou funkci jako formát `plain.fmt` a soubor `tex.poo` u TEXu. Na obvyklé místo přidejte příslušné cesty

```
SET MFINPUT=C:\TEX\MFINPUT
SET MFBAS=C:\TEX\MFBASES
SET MFJOB=C:\TEX\MFJOB
```

a můžete se pustit do experimentování.

Libovolným editorem vytvořte soubor např. `pokus.mf`, který bude obsahovat následující záhlaví:

```
mode=localfont;
mode_setup;
u#:=1mm#;
define_pixels(u);
screenstrokes;
```

První dva řádky nejsou při prvních pokusech nutné, jejich význam budeme potřebovat znát až při definitivní přípravě obrázků do fontu. Poslední řádek umožňuje velmi působivý efekt: METAFONT na obrazovce postupně předvádí jednotlivé tahy, takže během několika sekund se vám postupně poskládá celý obrázek.

Třetí řádek znamená, že mojí oblíbenou jednotkou je 1 mm (trochu to souvisí i s tím, že pro náčrt používám milimetrový papír — pro zkušeného TEXperta není ovšem problém vyrobit si třeba „cicerový papír“ a měřit souřadnice v cicerech). V dalším řádku si METAFONT tuto moji oblíbenou jednotku přepočte na svoji vnitřní jednotku 1 pixel, v níž všechno počítá (přitom už bere do úvahy, jestli vámi preferované výstupní zařízení pracuje se čtvercovými nebo obdélníkovými pixely). Tady ještě trochu záleží na tom, co v konkrétní instalaci znamená „`mode=localfont`“. Ve skutečnosti ovšem během ladění jednotlivých obrázků dávám přednost jen polovině mm (`u#:=.5mm#`), aby se obrázky pohodlně vešly na obrazovku a METAFONT pracoval rychleji. Teprve při konečné přípravě fontu s obrázky se vrátím k 1 mm; pak je možno i vyřadit příkaz `screenstrokes`, což zase METAFONT trochu urychlí, jeho ponechání však umožňuje poslední kontrolu před spuštěním `gftopk.exe`, protože změna

[1] Teď už vím, že grafického výstupu lze dosáhnout změnou konfigurace programu `mf.exe` programem `sbmfset.exe` (doplněno při korektuře).

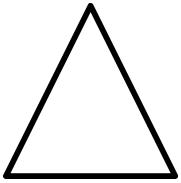
měřítka může někdy vést k nečekaným efektům (stačí, aby některá hodnota vzrostla nad únosnou mez např. při nevhodném pořadí násobení a dělení).

Uvedené příkazy nejsou pro naši práci důležité. Sám jsem je v podstatě opsal zejména z Láďova příkladu a dlouho jsem o jejich významu neměl ani tušení. Ještě ale musím upozornit na jednu zvláštnost verze METAFONTu, která je obsažena v emTEXu a jež dovede hned zpočátku zkazit chuť do další pokusů. Zapomenete-li stisknout `ENTER` po napsání závěrečného `end` na konci svého souboru, odmění se vám METAFONT úděsnou hláškou

```
! METAFONT capacity exceeded, sorry [buffer size=500].
l.132
      end^^?^^?^^?^^?^^?^^?^^?^^?^^?^^?^^?^^?^^?^^?^^?^^?^^?^^?^^?...
If you really absolutely need more capacity,
you can ask a wizard to enlarge me.
```

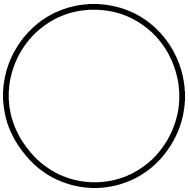
A můžeme začít tvořit! Napíšeme třeba

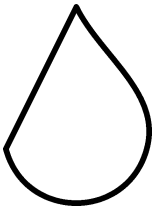
```
beginchar(1,30u#,30u#,0);
z1=(0,0); z2=(30u,0); z3=(1/2[x1,x2],30u);
pickup pencircle scaled u;
draw z1--z2--z3--z1;
endchar;
```



a dostaneme rovnoramenný trojúhelník. Příkaz `beginchar` anuluje případné předchozí deklarace a v závorce obsahuje následující údaje o vytvářeném charakteru: jeho umístění v ASCII tabulce, jeho šířku, výšku a hloubku (referenční bod každého charakteru je vždy v počátku `(0,0)=origin` souřadné soustavy. Každý zadávaný bod je označen proměnnou `z` s pořadovým číslem (připustím-li, že mohou být i desetinná, je jich tolik, že s tím pro začátek určitě vystačíme). Každý bod `z`k má dvě souřadnice označené příznačně `(x`k`,y`k`)` a METAFONT si je dokáže sám spočítat ze zadaných lineárních vztahů. Zápis `1/2[x1,x2]` souřadnice `x3` znamená, že je aritmetickým průměrem souřadnic `x1` a `x2` (zápis λ`[z1,z2]` označuje lineární kombinaci bodů `z1` a `z2`, přičemž hodnoty λ nemusí ležet jen v intervalu $\langle 0, 1\rangle$).

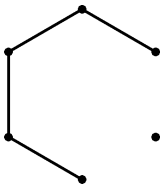
Příkaz `draw` je asi srozumitelný každému, kdo se smířil s tím, že u klávesnice počítače se bez znalosti angličtiny neobejdeme. Uvedený příklad ukazuje ovšem jen tu nejjednodušší možnost, jak lze zadané body spojovat. Příkaz `pickup pencircle` (`scaled u`) říká, že chceme něco malovat a že zatím vystačíme s jednoduchým perem s kruhovým hrotem. Údaj v závorce ilustruje možnost volby libovolně silných čar (pokud nepoužijeme příkaz `scaled` ke zvětšení průměru, použije METAFONT svoji oblíbenou hodnotu 0,4 pt). Pro zajímavost ještě zkusme, co se stane, když `z1--z2--z3--z1` (spojení jednotlivých bodů úsečkami) nahradíme výrazem `z1..z2..z3..z1`, který znamená, že chceme body spojit hladkou křivkou bez dalších podmínek. Obrázek vedle naopak ukazuje, co se stane, když u některých bodů navíc požadujeme, aby nakreslená křivka vycházela ve směru předepsaného vektoru `z1..z2..{z3-z2}z3{z1-z3}..z1`.





V dalším příkladu sestrojíme jednoduchý graf s uzly ve vrcholech pravidelného šestiúhelníku.

```
beginchar(2,30u#,30u#,0);
x1=x4=.5[x6,x2]; y1=0; y4=30u; z2-z1=15u*dir 30;
y3-y2=y5-y6=15u; x2=x3;  x5=x6=0; y6=y2;
pickup pencircle scaled .5u;
draw z1--z6--z5--z4--z3;
pickup pencircle scaled 1.5u;
drawdot z1; drawdot z2; drawdot z3;
drawdot z4; drawdot z5; drawdot z6;
endchar;
```

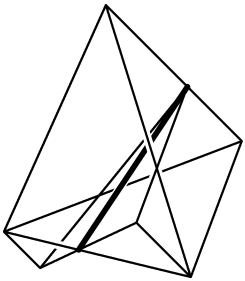


Nikoho asi nepřekvapí, že METAFONT pro násobení používá `*`; výraz `dir30` označuje jednotkový vektor, který svírá s osou x úhel 30°. METAFONT dává ve svém slovníku přednost klasické stupňové míře, takže i jeho operátory `sind` a `cosd` označují funkce sin a cos ve stupňové míře (je tedy `dir30=(cosd30,sind30)`. Příkaz `drawdot z`k je asi každému jasný. METAFONT naštěstí umožňuje v případě velkého počtu bodů i jednodušší zápis `for k=1 upto 6: drawdot z[k]; endfor`.

Další příklad jednak ukazuje, jak dostat do lineárních vztahů pro určení jednotlivých bodů *neznámou*, kterou nazývá vtipně `whatever`, jednak skvělé možnosti, které dává „gumování“. METAFONT zná příkazy jako `undraw` a `erase draw`, které se kromě jiného jemně liší v tom, do jaké hloubky mazání probíhá — v prvním případě METAFONT nevymaže bod, který jsme nakreslili dvakrát (např. průsečík dvou úseček), zatímco v druhém případě nebere na počet předchozích průchodů pera ohledy. To nám umožňuje nakreslit hrany čtyřstěnu způsobem, z kterého je patrno, které hrany se protínají a které jsou mimoběžné.

```
beginchar(3,50u#,45u#,0);
z1=(0,8u);z2=(33u,0);z3=(42u,24u);z4=(18u,48u);
z12=.4[z1,z2];z34=.6[z4,z3];
z20-z2=z34-z3;whatever*(z4-z3)=z1-z10;z10=whatever[z12,z20];
z3'=1/5[z1,z3];
pickup pencircle scaled 0.4u;
draw z1--z3;
 pickup pencircle scaled 1.5u;
 erase draw z12--z34; erase draw z20--z34;
pickup pencircle scaled 0.4u;
draw z10--z34;
```

```
 pickup pencircle scaled 1.5u;
 erase draw z1--z2;
pickup pencircle scaled 1u;
draw z12--z34;
 pickup pencircle scaled 1.5u;
 erase draw z2--z4;
pickup pencircle scaled 0.4u;
draw z2--z4;draw z20--z34; draw z1--z3';
draw z1--z2--z3--z4--z1;
draw z1--z10--z20--z2;
endchar;
```

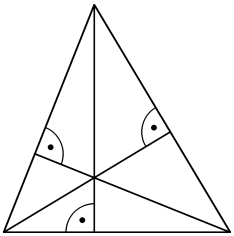


Ještě si snad všimněme, že zadány jsou jen vrcholy daného čtyřstěnu. Ostatní body jsme zadali „projektivně“, takže při případné úpravě vzhledu požadovaného čtyřstěnu se už o další body nemusíme starat.

Ještě vtipnější by bylo definovat i tyto body pomocí parametrů `w` (width — šířka), `h` (height — výška) a `d` (depth — hloubka), za které `plain.mf` dosazuje hodnoty zadané v `beginchar`. Pak bychom tvar obrázku mohli měnit jen vhodnou volbou rozměrů výsledného charakteru, aniž bychom cokoli měnili na vlastním programu.

Protože METAFONT dovede shodná zobrazení, není těžké sestrojit výšky trojúhelníku např. pomocí otočení o 90° (`rotated 90`). Funkce `angle` přiřazuje každému vektoru jeho úhel v intervalu $(-180°, 180°)$ — na to se nevyplácí zapomenout: jednou mi příšerně dlouho trvalo, než jsem pochopil, proč se mi dvě osy úhlů v trojúhelníku neprotínají, když v trojúhelníku se, jak známo, protínají dokonce všechny tři.

```
beginchar(4,50u#,45u#,0);
z1=(0,0);z2=(40u,0);x12=x3=16u;y3=40u;y12=0;
z23=whatever[z2,z3];z31=whatever[z3,z1];
z2-z31=whatever*(z1-z3) rotated 90;
z1-z23=whatever*(z3-z2) rotated 90;
pickup pencircle scaled .3u;
draw z1--z2--z3--z1; draw z2--z31; draw z3--z12; draw z1--z23;
 pickup pencircle scaled .2u;
  draw quartercircle scaled 10u rotated 90 shifted z12;
  draw quartercircle scaled 10u
       rotated angle (z3-z2) shifted z23;
  draw quartercircle scaled 10u
       rotated (angle(z1-z3)+90) shifted z31;
 pickup pencircle scaled 1u;
  drawdot 3u*dir45
       rotated 90 shifted z12;
  drawdot 3u*dir45
       rotated  angle (z3-z2) shifted z23;
  drawdot 3u*dir45
       rotated  (angle(z1-z3)+90) shifted z31;
endchar;
```



Příkaz `shifted` je asi rovněž srozumitelný — znamená posunutí o příslušný vektor (podobně jako ve vektorové algebře nerozlišujeme body a vektory). Nezdržuji se příliš syntaktickými pravidly jednotlivých příkazů — při jejich porušení nám to METAFONT ovšem vytkne. Zde bych jen upozornil na to, že každý příkaz ukončujeme středníkem a že je rozdíl mezi `draw z1--z2 shifted z0` a `draw (z1--z2) shifted z0`.

METAFONT samozřejmě dovede kreslit i kružnice. Podíváme-li se do souboru `plain.mf`, jak je kružnice definována, zjistíme, že se skládá z dvou polokružnic a polokružnice ze dvou čtvrtkružnic (`quartercircle`). Čtvrtkružnice má střed v počátku (referenční bod každého charakteru) a průměr jednoho pixelu, takže se bez zvětšení (`scaled`) obvykle neobejdeme.

Její definice v `plain.mf` je poučná:

```
quartercircle=(right{up}..(right+up)/sqrt2..up{left}) scaled .5;
```

Slova `up`, `down`, `right`, `left` zastupují příslušné jednotkové vektory.

METAFONT stejně jako TEX má nedocenitelnou výhodu vytváření maker, jejichž srozumitelné pojmenování záleží ovšem už na nás. Jako jednoduchý příklad uvedu definici značky pro pravý úhel ve standardních geometrických obrázcích, jaký byl ten předchozí.

```
def R(expr bod,smer)=      %% smer prvni poloprimky v kladnem smeru
   draw quartercircle scaled 10u rotated smer shifted bod;
   drawdot 3u*dir 45 rotated smer shifted bod
           withpen currentpen scaled 5;
enddef;
```

Pak už můžeme místo předešlých několika řádků napsat jen

```
R(z12,z3-z12);R(z23,z3-z2);R(z31,z2-z31);
```

Ještě než váš zrak sklouzne k následujícím řádkům, zkuste si rozmyslet, jak jednoduše naprogramovat kružnici opsanou třem daným bodům. Jedna možná konstrukce následuje, stačí ji vyzkoušet. Ještě by se hodilo přidat podmíněný příkaz, který by vytvořil chybovou hlášku v případě, že zadané body leží v přímce (příkaz `save` tu zaručuje, že použité proměnné nebudou kolidovat s budoucími programy).

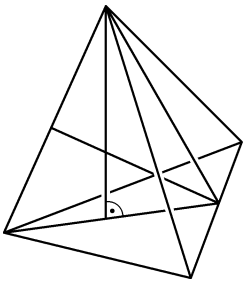
```
def circ(expr boda,bodb,bodc)=    %% kružnice určená třemi body
    begingroup
    save x,y,R;
    z1=boda;z2=bodb;z3=bodc;
    z12=1/2[z1,z2];z23=1/2[z2,z3];
    z0-z12=whatever*((z1-z2) rotated 90);
    z0-z23=whatever*((z3-z2) rotated 90);
    R=length(z1-z0);
    draw fullcircle scaled 2R shifted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

V novinách, kde využívají různé poměrně dokonalé, ale typograficky méně dobré systémy, se často pyšní zaoblenými rámečky. Ty ostatně pomocí fontů `lcircle` dovede i TEX, resp. LATEX. Pomocí METAFONTu si ovšem snadno vyrobíme sadu elipsovitých rámečků přímo na míru. Stačí vědět, že příkaz `scaled` lze rozdělit do směrů os x a y (`xscaled` a `yscaled`). Jako TEXové cvičení si zkuste vytvořit i jednoduché makro pro vytvoření např. následujícího nápisu, který používá elipsu vytvořenou METAFONT během několika sekund přímo na míru.

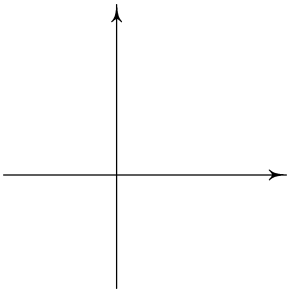
```
beginchar("O",50u#,15u#,0);
pickup pencircle scaled u;
draw fullcircle xscaled w yscaled h shifted ((w,h)/2);
endchar;
```

Další obrázek je poučný tím, že se mnoho neliší od jednoho z předchozích obrázků. Stačilo několik drobných úprav, a je tu docela jiný obrázek. A je vyděláno. Tohle nedovedou ani ta nejdokonalejší péra od firmy Faber-Castell. Poučeni tím, jak METAFONT dělá čtvrtkružnici, můžeme vylepšit dojem ze značky kolmosti u tělesové výšky čtyřstěnu.

```
beginchar(5,50u#,45u#,0);
z1=(0,8u);z2=(33u,0);z3=(42u,24u);z4=(18u,48u);
z12=.4[z1,z2];z34=.6[z4,z3];z23=.55[z2,z3];
z20-z2=z34-z3;whatever*(z4-z3)=z1-z10;z10=whatever[z12,z20];
z40=whatever[z1,z23]; x40=x4;
z14=whatever[z1,z4]; z14-z23=whatever*(z4-z1) rotated 90;
z401-z40=3u*dir(angle(z23-z1)); z402-z40=3u*dir 90;
pickup pencircle scaled 0.4u;
draw z1--z3;
 pickup pencircle scaled 1.5u;
 erase draw z4--z40; erase draw z4--z23;
pickup pencircle scaled 0.4u;
draw z1--z23--z14;
 pickup pencircle scaled 1.5u;
 erase draw z2--z4;
pickup pencircle scaled 0.4u;
draw z2--z4;draw z1--z2--z3--z4--z1;
draw z4--z23;draw z4--z40;
pickup pencircle scaled 0.2u;
draw z401{z4-z40}..{z40-z23}z402;
 pickup pencircle scaled .8u;
 drawdot 9/5u*dir50 shifted z40;
endchar;
```



U geometrických obrázků dost často potřebujeme souřadné osy, které se ovšem neobejdou bez pěkných šipek (jako d*k* jsem si označil několik „standardních“ tlouštěk kreslených čar).



```
beginchar(6,50u#,50u#,0);
z1000=(20u,20u);
z100=(0,y1000);z200=(w,y1000);
z300=(x1000,0);z400=(x1000,h);
pickup pencircle scaled d1;
 draw z100--z200; draw z300--z400;
 filldraw arrow(3u,0) shifted z200;
 filldraw arrow(3u,90) shifted z400;
endchar;
```

Jak už jsem se zmínil v úvodu, příkaz `arrow` jsem si i s názvem vypůjčil od Ládi Lhotky.

```
def arrow(expr delka,smer)=
    begingroup
    save x,y;  turningcheck := 0;
    z1=(0,0); x2=x3=-delka; -y2=y3=2/7delka; z4=(-3/4delka,0);
      (z2{dir45}..{right}z1&z1{left}..{dir135}z3&z3..
            controls z4..z2&z2..cycle) rotated smer
    endgroup
enddef;
```

Máme-li před sebou více podobných obrázků, vyplatí se dát do souvislosti rozměry obrázku a umístění obou os charakterizovat pouze zadáním jejich průsečíku — počátku souřadného systému:

```
def osy expr bod=
    begingroup
    save x,y;
    x1=y3=0; x2=w; x3=x4=xpart bod;
    y4=h; y1=y2=ypart bod;
    pickup pencircle scaled d1;
    draw z1--z2; draw z3--z4;
    filldraw arrow(3u,0) shifted z2;
    filldraw arrow(3u,90) shifted z4;
    endgroup
enddef;
```

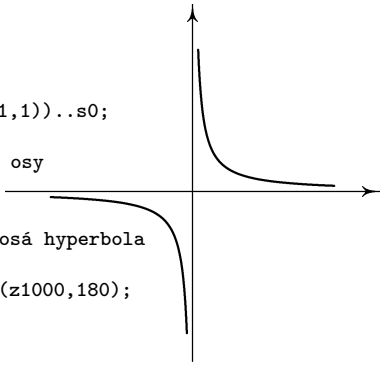
i když se musím přiznat, že jsem dlouho vystačil s opakovaným kopírováním sedmi řádků v popisu charakteru č. 6 ze souboru do souboru.

Když zjistíme, jak METAFONT krásně dovede dělat křivky, už těžko odoláme pokušení — a naše další možnosti podstatně vzrostou.

První, co člověka asi napadne, když si chce třeba narýsovat jednoduchou funkci $y = \frac{1}{x}$ (tedy hyperbolu), je určit několik bodů uvažované křivky, které METAFONT

„hladce“ spojí. Většinou se ale vyplatí přidat k jednotlivým bodům směrnici tečny dané křivky. A někdy se bez ní dokonce neobejdeme. Vynecháme-li ji např. v následujícím obrázku, dostaneme poněkud pichlavou „hyperbolu“ (doporučuji vyzkoušet — stačí vyhodit `{(1,-1)}` v definici cesty (`path`) `s0`.

```
beginchar(7,72u#,75u#,0);
path s[];
z1000=(35u,35u);
z1=(2u,35u);z2=(68u,35u);z3=(35u,5u);z4=(35u,68u);
for k=11 upto 15:
 z[k]=5u*(k-10,1/(k-10));
 endfor
s0=z11{(1,-1)}..z12..z13..z14..z15;
s1=reverse s0 reflectedabout((0,0),(1,1))..s0;
pickup pencircle scaled d1;
 draw z1--z2; draw z3--z4;        %% osy
 filldraw arrow(3u,0) shifted z2;
 filldraw arrow(3u,90) shifted z4;
pickup pencircle scaled d3; %% rovnoosá hyperbola
 draw s1 shifted z1000;
 draw s1 shifted z1000 rotatedaround(z1000,180);
endchar;
```

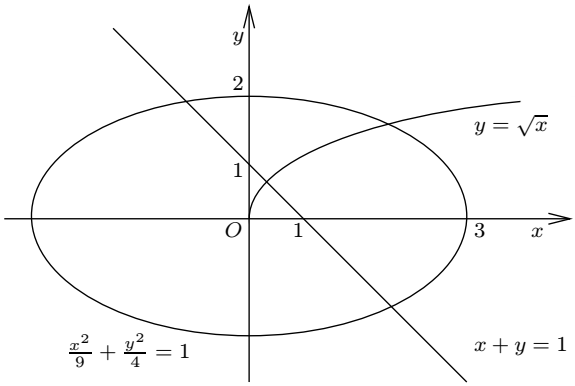


Samozřejmě si šetříme práci a využíváme souměrnosti, kde můžeme, takže kreslíme jen jednu část hyperboly v 1. kvadrantu, druhou část dostaneme souměrností podle osy 1. kvadrantu (`reflectedabout`) a otočením kolem počátku (`rotatedaround` s údajem ve stupních) vznikne druhá větev. Příkaz `reverse` před proměnnou typu `path` mění smysl průběhu dané křivky. Kdybychom na něj zapomněli, dostali bychom poněkud zvláštní křivku.

Popis obrázku lze zařídit různými způsoby (tomu byl ostatně věnován i jeden z článků v minulém bulletinu). Mně se nejvíce osvědčil popis jednoduchým makrem

```
\newdimen\unit \unit=1truemm
\def\bod#1, #2 #3 {\rlap{\kern#2\unit\raise#3\unit\hbox{$#1$}}}
```

které umožňuje ve zdrojovém textu popisek kdykoli opravit či doplnit, aniž bychom museli cokoli měnit v programu příslušného obrázku. Následující obrázek



byl popsán pomocí několika řádků:

```
\line{\hfil\OBR\bod 0, 31 21.5 \bod 1, 42.5 21.5 \bod 3, 66 21.5
\bod x, 75 21.5
\bod y, 32 55 \bod 2, 32 46 \bod y=\sqrt x, 67 40 \bod x+y=1, 66 6 \bod
\dfrac{x^2}9+\dfrac{y^2}4=1, 4 3 \bod 1, 32 32 \char8\rm\hfil}
```

Přitom souřadnice bodů, které jsme číselně nezadali, může METAFONT zapsat do informačního souboru `pokus.log`, zadáme-li příkaz `show z`*k*`/u`.

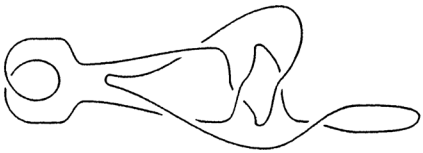
Myslím, že jako ukázka možností METAFONTu by to zatím mohlo stačit. Samozřejmě není nic obtížného vyrábět tečkované, přerušované (na to stačí následující makro)

```
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;
```

nebo dokonce čerchované čáry; grafisty zas zcela jistě okouzlí naprosto pravidelné kroužky v uzlech grafů kreslené pomocí příkazu `odot` (takového výsledku v LaTeXu nedosáhnete ani pomocí TEXcadu)

```
def odot expr bod=
   erase fill fullcircle scaled 2d3 shifted bod;
   draw fullcircle scaled 2d3 shifted bod;
enddef;
```

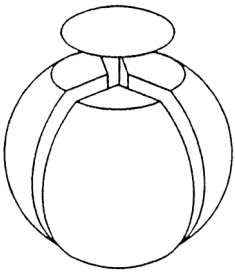
nebo propletence na následujícím obrázku.



Jako lahůdku na konec ještě ukázka toho, co se dá vytvořit pomocí jednoduchého příkazu

```
def overdraw expr c = erase fill c; draw c enddef;
```

převzatého z METAFONTbooku.

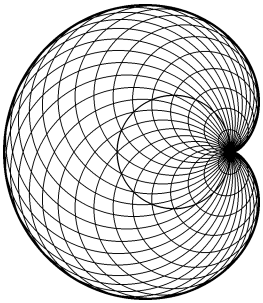


Všechny předvedené postupy a definice nejsou samozřejmě jediné možné a ani zdaleka nejlepší. Mají jedinou dobrou vlastnost: fungují. Teprve podrobným studiem METAFONTbooku odhalíte lepší či jednodušší možnosti. Tato kniha podobně jako TEXbook není rozhodně určena jen na jedno čtení. A její studium spolu s obrázky poměrně vysoké kvality jak na laserové, tak i na běžné jehličkové tiskárně přináší nejedno potěšení. Během přípravy tohoto článku jsem např. zjistil, že jediným příkazem `screenstrokes` lze dosáhnout toho, že okamžitě tah po tahu můžeme na monitoru sledovat vznik obrázku. To ocení jen ten, kdo byl předtím šest měsíců s grafickou kartou Hercules odkázán na program `gftodvi` a `preview`, aby pak zjistil, že výsledek se podstatně liší od záměru.

Poslední příklad, který ilustruje známou vlastnost srdcovky (srdcovku dostaneme jako obálku kružnic, jejichž středy leží na dané kružnici a jež všechny procházejí jedním jejím bodem), využívá možností TEXu (na rozdíl od jiných sázecích programů tiskne na pixel přesně!) ke zmírnění nedostatků METAFONTu. Vzhledem k poměrně velkému počtu kružnic trvá digitalizace obrázku nepřiměřeně dlouho (zvláště nemáme-li dost prostornou operační paměť). Pomůžeme si tedy tím, že obrázek rozdělíme do dvou charakterů, které pak přesadíme přes sebe (např. pomocí příkazu `\rlap{ }`). Přitom digitalizace obou charakterů je mnohonásobně kratší.

```
beginchar(10,60u#,60u#,0);
z1000=(50u,h/2); z0=(40u,h/2);
path s[];
for k=1 upto 6: z[k]=20u*right*(1-cosd(30k))
      rotated(30k) shifted z1000; endfor
s0=z1000{right}..z1..z2..z3..z4..z5..{down}z6;
s1=s0..(reverse s0 reflectedabout(z0,z1000));
```

```
s2= fullcircle scaled 20u shifted z0;
pickup pencircle scaled d1/2;
for k=1 upto 16: z[10+k]=point k/4 of s2;
 draw fullcircle scaled (2abs(z1000-z[10+k]))
          shifted z[10+k]; endfor
pickup pencircle scaled d2;
 draw s1;
pickup pencircle scaled d1;
 draw s2;
endchar;
```



Když jsem s METAFONTem začínal, nebylo to lehké. Tento článek ale už vznikal tak, jak by si průměrně náročný uživatel TEXu asi představoval: jednotlivé obrázky „vznikaly“ průběžně při psaní textu (používám upravenou konfiguraci menu šířeného prostřednictvím CSTUGu), generace fontu obsahujího nový obrázek, netrvala nikdy více než 20–30 vteřin. Ty uvozovky jsem tu napsal jen proto, že většinu obrázků jsem si samozřejmě vypůjčil ze svých bohatých zásob a uveřejňuji je bez větších zásahů.

Jisté je, že dokonalé technické pérovky s interaktivními programy typu *paintbrush*e nedostanete. Ten se hodí zas na jiné grafické věci, potřebujete-li obrázek ozvláštnit různými rastry apod. S programem `pcltopic` pak takové obrázky snadno dostanete do TEXového dokumentu. Nevidím ovšem důvod, proč nevyužít každou možnost — někdy se dokonce může hodit kombinace obou či více metod v jediném obrázku.

(*Karel Horák*)
e-mail: `horakk@csearn`

Dedham — 12th Annual Meeting of TUG

Přiznávám, že když jsem poprvé četl oznámení o tomto setkání, nevěnoval jsem mu pozornost. Pak jsem dostal nezávisle pozvání do USA a tak jsem oznámení opět vyhledal. Blízko Bostonu leží Dedham a trm se nalézá hotel Hilton — v něm se setkání konalo. Díky podpoře TUGu jsem se ho mohl zúčastnit.

Mezi „chudým v Hiltonu“ a „nahým v trní“ je jistá podoba, pocity totiž nejsou někdy o nic lepší, podařilo se mi však soustavně se půvabným číšnicím s třetinkami (pro Čecha mizerného) piva za 4.50 US $ vyhýbat. Konference byla lepší, než jsem čekal — oficialit bylo naštěstí minimálně a program byl velmi zajímavý. Byl věnován ve velké míře publikační činnosti na bázi TEXu.

K mému překvapení se zahajovalo naprosto elementárním úvodem pro nováčky. Mnoho (lépe: příliš mnoho) času se věnovalo povídání o dalších možných úpravách TEXu. Přednášky běžely v sekcích a občas docházelo ke konfliktu zájmů. Vydavatelský svět se jevil značně konservativně a až na vyjímky zaujímal k publikování pomocí TEXu zdrženlivé stanovisko; těmito vyjímkami jsou např. Springer Verlag, Addison–Wesley a do jisté míry Elsevier.