Kapitoly o METAPOSTu

Obsah

1 Problémy s výpočty

Numerické proměnné, stejně jako body `z`⟨*index*⟩, není nutno deklarovat. Stačí napsat např. rovnici `a=5`, která určuje hodnotu proměnné `a`. 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 `beginfig` a `endfig`. 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 MetaPost používá operandy uvedené v tabulce:

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`	$\sqrt{a^2-b^2}$
`abs`	`abs a`	$\lvert a\rvert$
`sind`	`sind a`	$\sin a$
`cosd`	`cosd a`	$\cos a$
`sqrt`	`sqrt a`	$\sqrt{a}$
`mexp`	`mexp a`	$e^{a/256}$
`mlog`	`mlog a`	$256\ln a$
`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

Při sestavování výrazů je třeba počítat s tím, že prostá negace, násobení, dělení a umocňování mají v MetaPostu stejnou prioritu. Proto `3*a**2` znamená $(3a)^2$, `-a*2` znamená $(-a)^2$, `2/3**2` znamená $(2/3)^2$. Kde je situace nepřehledná, nestydíme se použit závorky.

Při násobení číslem lze použít zkrácený zápis, např `2i` místo `2*i` nebo `50mm` místo `50*mm`.

Hodnotu numerických výrazů MetaPost vždy zakrouhluje na nejbližší násobek čísla $\frac{1}{65\,536}$. Toto nejmenší kladné číslo značíme symbolem `epsilon`. Například skutečná hodnota čísla `.1` je $\frac{6\,554}{65\,536}$ stejně jako hodnota čísla `.099 999` nebo `.10001`. Toto zaokrouhlování má za následek, že při složitějších výpočtech můžeme dostat značně nepřesný výsledek. Například hodnota výrazu `2 000*(.015**2)` je v MetaPostu 0,457 76, podobně `2.3/.002` dává 1 150,633 59 nebo `.000001/.004` dává 0, protože číslo 0,000 001 je menší než $\frac{1}{65\,536}$ a MetaPost 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`.

Další nevýhodou MetaPostu je jeho číselné omezení. Absolutní hodnoty čísel vkládaných do výpočtu a také absolutní hodnota výsledku výpočtu nesmí překročit 4 096-`epsilon`, což dává 4 095,999 98. Pro toto číslo je zaveden symbol `infinity` Při počítání hodnoty výrazu se může stát, že absolutní hodnota mezivýsledeku výpočtu překročí `infinity`. Pokud nepřesáhne $8 \times 4\,096 - $`epsilon`, tj. 32 767,999 98, 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 MetaPostu u zavedeno pythagorejské sčítání a odčítání: `400++300` dává výsledek 500, `500+-+300` dává výsledek 400.

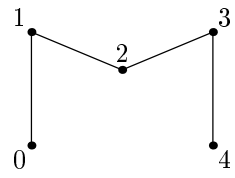
Změnou interní proměnné příkazem `warningcheck:=0` lze v případě potřeby zvětšit i maximální absolutní hodnotu vkládaných čísel a výsledků na 32 768−`epsilon`.

2 Cesty

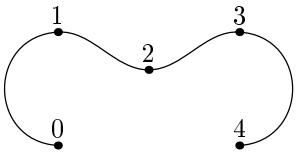
Spojité čáry vedené jedním tahem přes zvolené body nazýváme cesty. Počet zvolených bodů v jedné cestě musí být menší než 300.

MetaPost chápe cesty jako proměnné typu `path`. Pro některé operace s cestami je vhodné provést po `beginfig` jejich deklaraci, která umožní cesty označit symboly a příkazy napsat přehledným způsobem. Například deklarace `path p[]` umožní pracovat s cestami `p0, p1, p2,` atd.

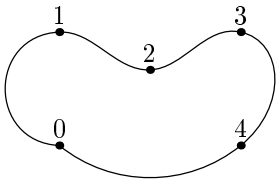
MetaPost disponuje celou řadou příkazů, kterými můžeme ovlivnit tvar cesty, což ilustrují následující ukázky. Nejčastěji používáme příkaz `--` pro nakreslení rovného úseku a příkaz `..` pro nakreslení křivého useku, který často doplňujeme příkazy {`dir` *úhel*} pro určení směru cesty v daném bodě nebo pro její zalomení.



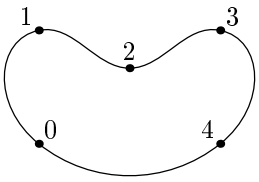
`p1=z0--z1--z2--z3--z4`



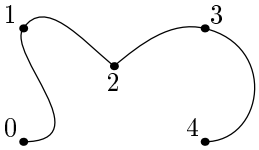
`p2=z0..z1..z2..z3..z4`



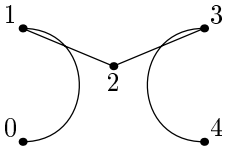
`p3=z0..z1..z2..z3..z4..z0`



`p4=z0..z1..z2..z3..z4..cycle`



`p5=z0{right}..z1..{dir-40}z2{dir40}..z3..{left}z4`



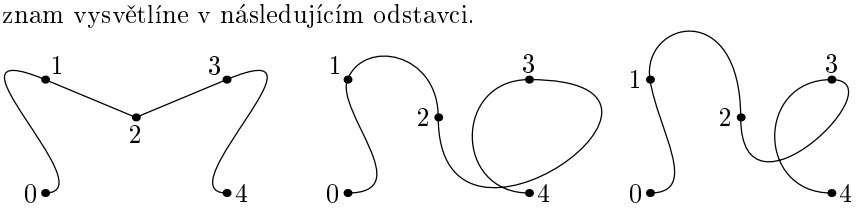
`p6=z0{right}..z1--z2--z3..{right}z4`

Průběh křivého úseku cesty můžeme ovlivnit změnou parametru `tension` v intervalu $\langle 0{,}75; \infty)$. Implicitní hodnota je 1. Menší hodnota úsek cesty „uvolňuje“, větší hodnota „napíná“. Zvolíme-li `tension infinity`, úsek je rovný a navazující křivé úseky se v těsné blízkosti vyrovnávají — jejich křivost zde spojitě přechází v nulovou. Stejného výsledku dosáhneme příkazem `---`.

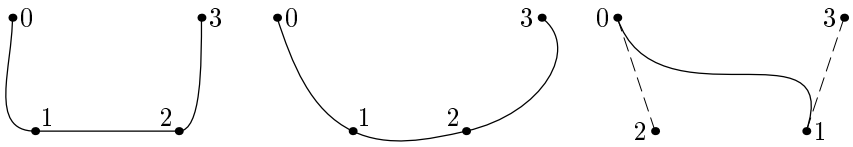
Příkaz `...` používáme tam, kde se chceme zbavit inflexního bodu. Zakřivení cesty na jejím začátku a konci můžeme ovlivnit změnou parametru `curl`.

Hodnotě 0 odpovídá nulová křivost, hodnotě ∞ maximální křivost. Implicitní hodnota je 1.

Nejúčinněji můžeme úsek cesty upravit volbou *kontrolních bodů*, jejichž význam vysvětlíne v následujícím odstavci.




```
p7=z0{right}..z1---z2---z3..{right}z4
      p8=z0{right}..z1..{down}z2..{left}z3..{right}z4
                  p9=z0{right}..z1..tension.75..z2{down}..
                          tension1.5and3..z3{left}..{right}z4
```



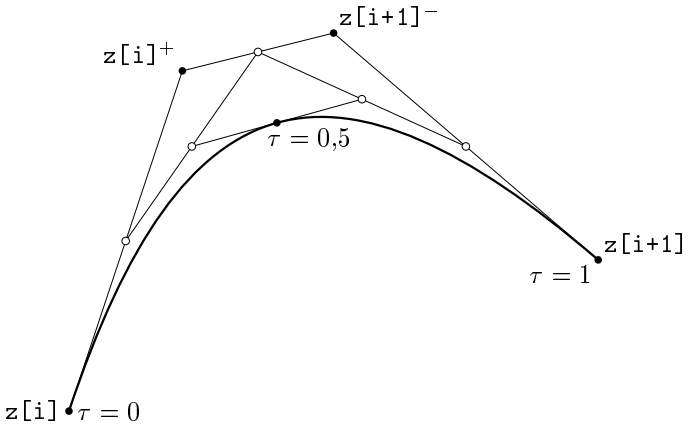

```
p10=z0{down}..z1..tension infinity..z2...{up}z3
              p11=z0{curl 0}..z1..z2..{curl infinity}z3
                                p12=z0..controls z2 and z3..z1
```

MataPost 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* $\texttt{z[i]}^+$ a $\texttt{z[i+1]}^-$. Průběh Bézierovy křivky je pak vypočítán pomocí parametrických vztahů:

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

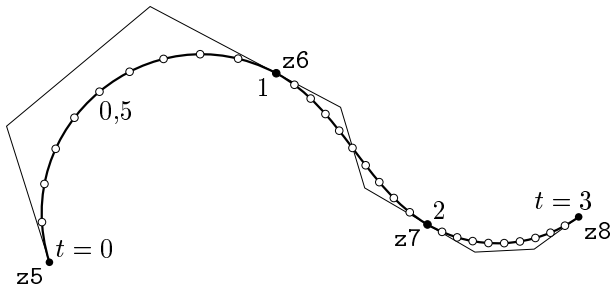
$$y(\tau+i) = (1-\tau)^3 y_i + 3\tau(1-\tau)^2 y_i^+ + 3\tau^2(1-\tau)y_{i+1}^- + \tau^3 y_{i+1}\,,$$

kde $0 \le \tau \le 1$. 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 `z0`, nakreslení jednoho úseku trvá jednotkovou dobu a na konci cesty je celkový čas `length(p)` číselně roven počtu nakreslených úseků.

Na následujícím obrázku je nakreslena cesta složená ze tří Bézierových křivek definovaná příkazem `p=z5..z6..z7..z8`. Na křivce je vyznačen čas po 0,1 jednotky. Lomená čára spojuje kontrolní body, které si program sám vypočítal.[1]) Je zřejmé, že každá trojice bodů `z[i]`$^-$, `z[i]` a `z[i]`$^+$ leží v jedné přímce.



Jako parametr cesty můžeme kromě času použít i její délku. Celková délka cesty v `bp` je rovna hodnotě výrazu `\arclength` *cesta*. Délku cesty v milimetrech zjistíme příkazem

`show arclength` *cesta*`/mm`.

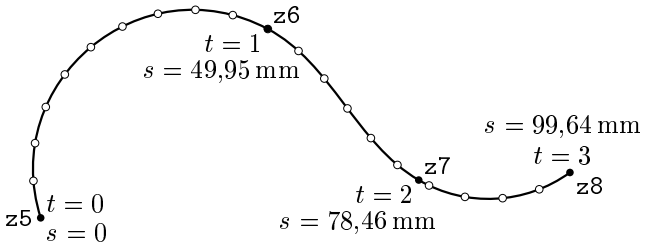
[1]Podrobnou informaci o kontrolních bodech získáme příkazem `show` *cesta*. Objeví se v souboru s příponou `log`.

Chceme-li určit polohu bodu, ve kterém délka čáry dosáhne hodnoty s, určíme nejprve jeho čas jako `arctime` s `of` *cesta*. Hledaný bod je tedy

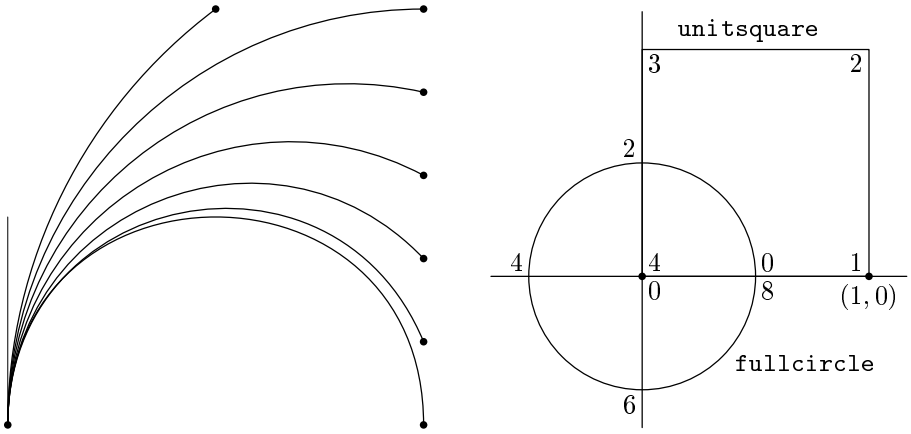
`point arctime` s `of` *cesta* `of` *cesta*.

Cestu z předcházející ukázky změříme a opatříme značkami po 5 milimetrech pomocí příkazů:

```
l=arclength p;
show l,l/mm;
show arclength subpath(0,1) of p1/mm;
show arclength subpath(0,2) of p1/mm;
for i=1 upto l/5mm:
 unfill fullcircle scaled u shifted point arctime i*5mm of p of p;
 draw fullcircle scaled u shifted point arctime i*5mm of p of p;
endfor;
```



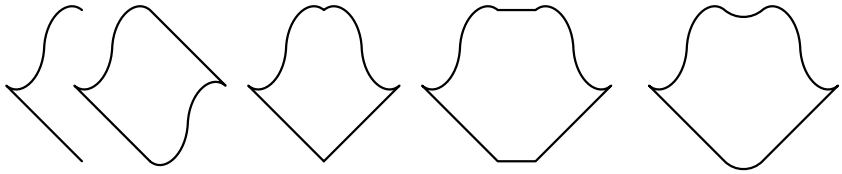
Jednoduché cesty definované příkazem typu `p=z1{dir` *úhel*`}..z2` nebo příkazem `z1..z2..z3` jsou propočítány MetaPostem tak, aby se co nejméně lišily od kruhového oblouku. To platí zejména pro oblouky se středovým úhlem menším než 45°, kde jsou odchylky hluboko pod rozlišovací schopností obrazovek 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 bp. Jsou tvořeny oblouky se středovým úhlem 45°. 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 bp 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.

3 Operace s cestami

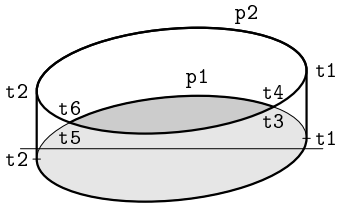
Složitější cesty můžeme získat složením z několika jednodušších. Pokud na sebe jednotlivé úseky bezprostředně navazují, použijeme operátor `&`, jinak použijeme rovné spojky `--` nebo plynulé napojení pomocí `..`:



```
path p;
p=(origin--(-2,2){dir-40}..(-1,3)..{dir-40}(0,4))scaled 5mm;
pickup pencircle scaled .3mm;
draw p;
draw p shifted(9mm,0) & p rotated 180 shifted (9mm,20mm);
draw p shifted(32mm,0) & reverse p reflectedabout(up,down) shifted
(32mm,0);
draw p shifted(55mm,0)--reverse p reflectedabout(up,down) shifted
(60mm,0)--cycle;
draw p shifted(85mm,0)..reverse p reflectedabout(up,down) shifted
(90mm,0)..cycle;
```

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

```
beginfig(2);
path p[]; numeric t[];
p1=fullcircle scaled 35mm yscaled .4 slanted .5;
p2=p1 shifted (0,9mm);
t1=directiontime up of p1;
t2=directiontime down of p1;
(t3,t4)=p1 intersectiontimes p2;
(t5,t6)=reverse p1 intersectiontimes reverse p2;
show t1,t2,t3,t4,t5,t6;
fill p1 withcolor .9white;
fill subpath(t3,length(p1)-t5) of p1--
  subpath(length(p2)-t6,t4) of p2--cycle withcolor .8white;
pickup pencircle scaled .15mm;
draw(-20mm,0)--(20mm,0);
draw (left--right) scaled .5mm shifted point t1 of p1;
draw (left--right) scaled .5mm shifted point t2 of p1;
draw p1;
pickup pencircle scaled .3mm;
draw subpath(t3,length(p1)-t5) of p1;
draw subpath(t2,8+t1) of p1--subpath(t1,t2) of p2--cycle;
draw p2;
endfig;
```



Pomocí `directiontime z~of p` určíme čas, kdy má cesta `p` směr daný vektorem `z`. V ukázce tedy časy `t1` a `t2` určují polohy bodů na cestách `p1` a `p2`, kde mají jejich tečny svislý směr.

Operací `(t3,t4)=p1 intersectiontimes p2` získáme vektor, jehož souřadnice `t3`, `t4` jsou časy na cestách `p1` a `p2` v jejich průsečíku.

Proměnná `length(p)` představuje celkový čas cesty.

Cesta `subpath(t1,t2) of p` je částí cesty `p` mezi časy `t1` a `t2`. Pokud na uzavřené cestě překročíme její počátek, musíme k druhému času přičíst celkový čas cesty — např. v ukázce jsme použili `subpath(t1,8+t2) of p1`. Číslo 8 je celkový čas cesty `fullcircle` a tedy i `length(p1)`.

Makro `reverse p` nám cestu `p` otočí, tj. změní 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.

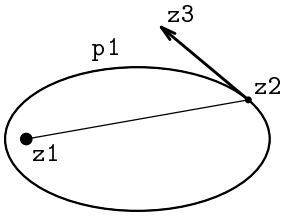
V druhé ukázce je znázorněna dráha komety obíhající okolo Slunce, okamžitá poloha komety a vektor okamžité rychlosti po průchodu afelem v okamžiku, kdy průvodič svírá s hlavní osou elipsy úhel 10°.

```
beginfig(3);
path p[];  numeric t[];
a:=17.5mm; b:=9.5mm; e:=a+-+b;
p1=fullcircle xscaled 2a yscaled 2b;
z1=(-e,0);
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;
uhel:=angle direction t1 of p1;
z3=z2+15mm*dir uhel;
draw z2--z3;
draw (dir165--origin--dir195) scaled 2mm rotated uhel shifted z3;
pickup pencircle scaled .2mm;
draw p1;
draw z1--z2;
filldraw fullcircle scaled 1.5mm shifted z1;
filldraw fullcircle scaled .75mm shifted z2;
label.top(btex{\tt p1}etex,point 2.3 of p1);
label.urt(btex{\tt z2}etex,z2);
label.urt(btex{\tt z3}etex,z3);
label.lrt(btex{\tt z1}etex,z1);
endfig;
```

Je použita pomocná směrová úsečka `p2`, jejíž délku volíme tak, aby určitě proťala elipsu `p1`. Výsledkem

```
p1 intersectionpoint p2
```

je bod, kde se cesty protínají.



Oproti minulé ukázce jsme navíc použili makro `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. Jednotkový směrový vektor získáme konstrukcí

```
dir angle direction t of p
```

nebo

```
unitvector 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`.

Část cesty můžeme odstřihnout jinou cestou pomocí operace

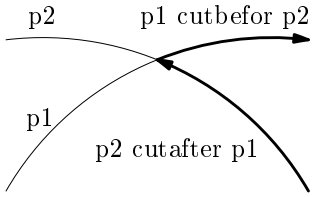
1. cesta `cutafter` *2. cesta*,

která odstraní část 1. cesty za průsečíkem s 2. cestou, nebo

1. cesta `cutbefor` *2. cesta*,

která odstraní část 1. cesty před průsečíkem s 2. cestou:

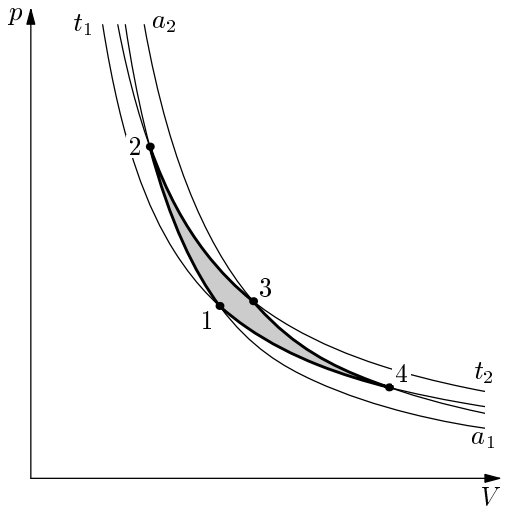
```
beginfig(4)
path p[];
p1=(0,0){dir60}..(40u,20u);
p2=(40u,0){dir120}..(0,20u);
penc.1u;
draw p1; draw p2;
penc.4u;
ahangle:=30; ahlength:=2u;
drawarrow p2 cutafter p1;
drawarrow p1 cutbefore p2;
label.llft(btex p2 cutafter p1 etex,point.2 of p2);
label.top(btex p1 cutbefor p2 etex,point.77 of p1+.5u*up);
label.top(btex p2 etex,point.9 of p2);
label.lft(btex p1 etex,point.25 of p1);
endfig;
```



Uzavřenou křivku tvořenou na sebe navazujícími úseky několika cest získáme jednoduše pomocí makra

`buildcycle`(*výčet cest*).

Například Carnotův cyklus se skládá ze dvou izotermických a dvou adiabatických dějů a v diagramu $p - V$ je zobrazen uzavřenou křivkou tvořenou úseky dvou izoterm t_1, t_2 a dvou adiabat a_1, a_2:



```
beginfig(5)
path t[], a[], c;
t0=(1,1){1,-1} for i=2 upto 7:..(i,1/i)endfor;
t1=t0 xscaled 9.5u yscaled 60u;
t2=t0 xscaled 11.5u yscaled 60u;
a0=(1,1){1,-1.4} for i=2 upto 5:..(i,1/i**1.4)endfor;
a1=a0 xscaled 12.5u yscaled 60u;
a2=a0 xscaled 15u yscaled 60u;
c=buildcycle(t1,a2,t2,a1);
pickup pencircle scaled .2u;
draw t1; draw t2; draw a1; draw a2;
clip currentpicture to unitsquare xscaled 60u yscaled 60u;
drawarrow(0,0)--(0,62u); drawarrow(0,0)--(62u,0);
pickup pencircle scaled .4u;
fill c withcolor .8white;
draw c;
label.lft(btex$t_1$etex,point 0 of t1);
label.rt(btex$a_2$etex,point 0 of a2);
label(btex$a_1$etex,(60u,5u));
label(btex$t_2$etex,(60u,14u));
label.lft(btex$p$etex,(0u,61u));
label.bot(btex$V$etex,(61u,0));
dotlabel.llft(btex 1 etex,t1 intersectionpoint a1);
dotlabel.urt(btex 3 etex,t2 intersectionpoint a2);
dotlabel(btex etex,t1 intersectionpoint a2);
dotlabel(btex etex,t2 intersectionpoint a1);
picture v[];
v1=thelabel.lft(btex 2 etex,t2 intersectionpoint a1);
v2=thelabel.urt(btex 4 etex,t1 intersectionpoint a2);
bboxmargin:=1pt;
unfill bbox v1; draw v1; unfill bbox v2; draw v2;
endfig;
```

4 Pera, štětce, barvy

Chceme-li nakreslit bod nebo cestu, musíme nejprve zvolit pero příkazem

`pickup` *druh pera* `scaled` *zvětšení*;

MetaPost nabízí tři základní druhy pera:

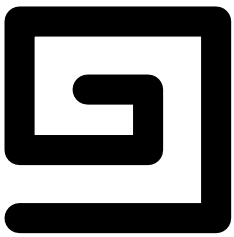
`pencircle` ... kruhové pero o průměru 1 bp,

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

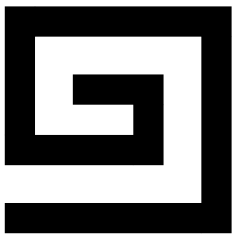
`penrazor` ... vodorovná úsečka nulové tloušťky a délky 1 bp.

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:

Zvolené pero se ukládá do proměnné `pen` jako hodnota `currentpen`.



`pencircle scaled 4mm`



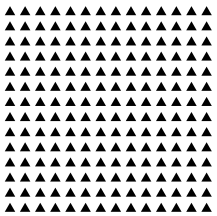
`pensquare scaled 4mm`

`penrazor scaled 4mm rotated 30`

Příkaz `draw`, kterým kreslíme cesty, můžeme použít i pro kreslení jednotlivých bodů, lepšího výsledku však dosáhneme při použití příkazu `drawdot`. Ten ale klade větší nároky na paměť počítače.

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

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

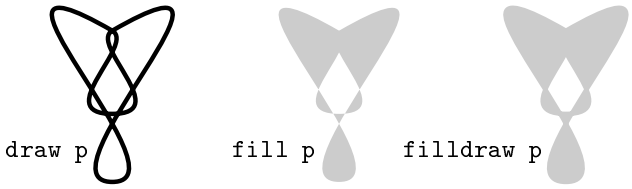


MetaPost dovede vybarvit oblast ohraničenou uzavřenou cestou. Používá se k tomu příkaz `fill` *cesta*. Cesta, která omezuje vyplňovanou oblast musí být uzavřena pomocí příkazu `cycle`. Nestačí tedy, aby první bod cesty byl současně jejím koncem, tedy:

nikoli `p=z1--z2--z3--z1`, ale `p=z1--z2--z3--cycle`.

Příkazem `filldraw` *cesta* uzavřenou plochu vybarvíme a obtáhneme perem `currentpen`. MetaPost si poradí i s komplikovanými cestami, které se několikrát kříží:

```
beginfig(17)
path p;
p=((0,20){dir30}..(0,0)..(0,20){dir-30}..(0,9)..cycle)scaled mm;
pickup pencircle scaled .6mm;
draw p;
fill p shifted (30mm,0) withcolor .8white;
filldraw p shifted (60mm,0) withcolor .8white;
label.lft(btex{\tt draw p}etex,(-2mm,4mm));
label.lft(btex{\tt fill p}etex,(28mm,4mm));
label.lft(btex{\tt filldraw p}etex,(58mm,4mm));
endfig;
```



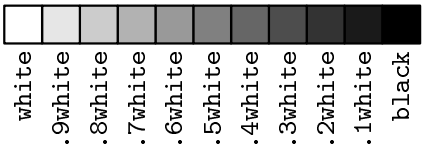
Pro kreslení čar, vyplňování ploch a psaní popisek je po příkazu `beginfig` implicitně nastavena černá barva. Můžeme ji však změnit na libovolnou barvou z palety `rgb`, která je založena na aditivním skládání barev. Stačí, když příkazy `draw`, `fill` a `label` doplníme na

`draw` *cesta* `withcolor` *barva*, `fill` *cesta* `withcolor` *barva*,

nebo `label(` ... `)withcolor` *barva*.

Barvu definujeme jako proměnnou typu `color` určenou třemi souřadnicemi `redpart, greenpart, bluepart`. Hodnoty souřadnic leží v intervalu $\langle 0, 1\rangle$. Zadáme-li větší číslo než 1, platí jako 1; menší čísla než 0 platí jako 0.) Předdefinované barevné konstanty jsou `white=(1,1,1)`, `black=(0,0,0)`, `red=(1,0,0)`, `green=(0,1,0)` a `blue=(0,0,1)`. Barvy můžeme jako vektory sčítat nebo násobit reálným číslem.

Šedou škálu získáme násobením barvy `white` čísly z intervalu $\langle 0, 1\rangle$.

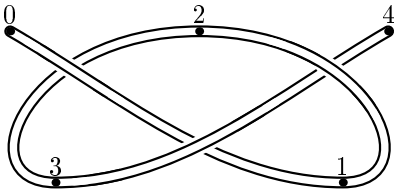


Zvláštním případem je kreslení čáry nebo vyplnění plochy původní barvou podkladu `background`, která je předdefinována jako `white`. Tato operace, která danou čáru nebo plochu vymaže, se provádí příkazy

`undraw` *cesta*, `unfill` *cesta* nebo `unfilldraw` *cesta*.

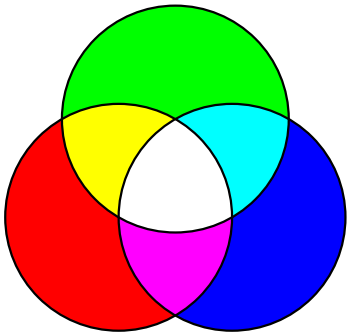
V následující ukázce je použit příkaz `undraw` pro nakreslení uzlu dvojitou čárou stálé šířky a pro vyznačení viditelnosti vymazáním úseků, které se mají jevit jako spodní:

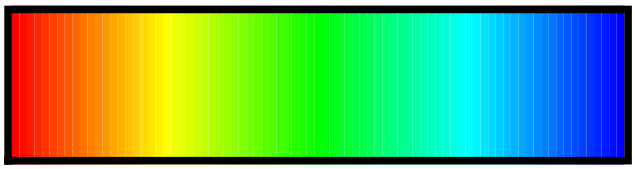
```
beginfig(16)
path p;
z0=(0,20mm); z1=(44mm,0); z2=(25mm,20mm); z3=(6mm,0); z4=(50mm,20mm);
p=z0{dir-30}..z1{right}..z2..{right}z3..{dir30}z4 ;
pickup pencircle scaled 1.6mm;
draw p;
pickup pencircle scaled mm;
undraw p;
pickup pencircle scaled 2.5mm;
undraw subpath(.15,.25)of p;
undraw subpath(1.6,1.7)of p;
undraw subpath(3.2,3.5)of p;
pickup pencircle scaled 1.6mm;
draw subpath(.1,.3)of p;
draw subpath(1.5,1.8)of p;
draw subpath(3.1,3.6)of p;
pickup pencircle scaled mm;
undraw subpath(.05,.35)of p;
undraw subpath(1.45,1.85)of p;
undraw subpath(3.05,3.65)of p;
dotlabels.top(0,1,2,3,4);
endfig;
```



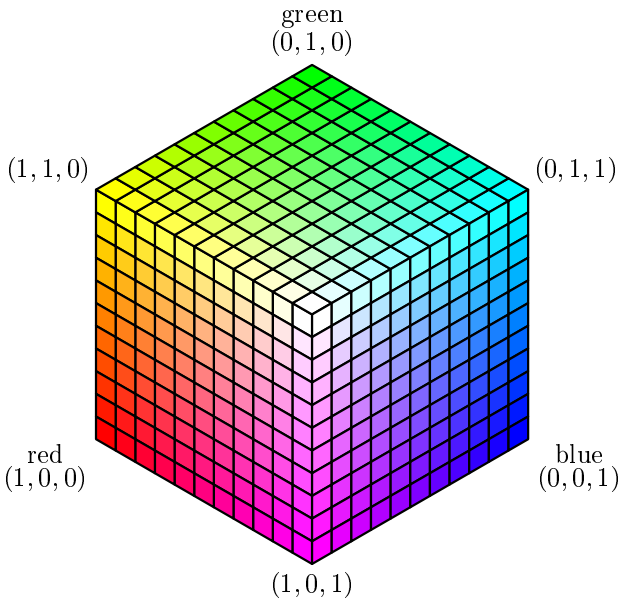
Další dvě stránky příkladů jsou určeny pro barevnou tiskárnu.

```
beginfig(8)
path p[];
p1=fullcircle scaled 30mm;
p2=p1 shifted(15mm,0);
p3=p1 shifted (15mm*dir60);
p13=buildcycle(p1,p3);
p12=buildcycle(subpath(6,10)of p1,p2);
p23=buildcycle(p2,p3);
p123=buildcycle(p1,p2,p3);
fill p1 withcolor red;
fill p3 withcolor green;
fill p2 withcolor blue;
fill p13 withcolor (red+green);
fill p23 withcolor (green+blue);
fill p12 withcolor (red+blue);
fill p123 withcolor (red+green+blue);
penc .3u;
for i=1 upto 3: draw p[i]; endfor;
endfig;
```





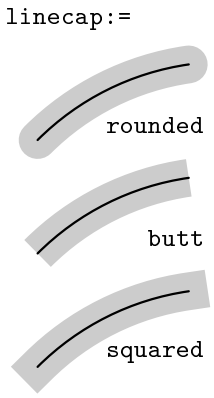
```
beginfig(9)
pickup pensquare scaled 1mm;
for i=-40 upto 40:
 draw (i*mm,0)--(i*mm,20mm) withcolor(-.05i,2-abs(.05i),.05i);
endfor;
draw (-41mm,0)--(41mm,0)--(41mm,20mm)--(-41mm,20mm)--cycle;
endfig;
```


green
(0, 1, 0)
(1, 1, 0)
(0, 1, 1)
red
(1, 0, 0)
blue
(0, 0, 1)
(1, 0, 1)

```
beginfig(15)
penc.3mm;
for i= 0 upto 10: for j= 0 upto 10:
 fill((0,0)--dir150--(dir150+down)--down--cycle)
   shifted(j*dir150+i*down) scaled 3mm withcolor(1,1-.1i,1-.1j);
 draw((0,0)--dir150--(dir150+down)--down--cycle)
   shifted(j*dir150+i*down) scaled 3mm;
endfor; endfor;
for i= 0 upto 10: for j= 0 upto 10:
 fill((0,0)--dir30--(dir30+dir150)--dir150--cycle)
   shifted(i*dir30+j*dir150) scaled 3mm withcolor(1-.1i,1,1-.1j);
 draw((0,0)--dir30--(dir30+dir150)--dir150--cycle)
   shifted(i*dir30+j*dir150) scaled 3mm;
endfor; endfor;
for i= 0 upto 10: for j= 0 upto 10:
 fill((0,0)--dir30--(dir30+down)--down--cycle)
   shifted(i*dir30+j*down) scaled 3mm withcolor(1-.1i,1-.1j,1);
 draw((0,0)--dir30--(dir30+down)--down--cycle)
   shifted(i*dir30+j*down) scaled 3mm;
endfor; endfor;
label.llft(btex$\hbox{red}\atop\displaystyle(1,0,0)$etex,
   33mm*dir-150);
label.ulft(btex$(1,1,0)$etex,33mm*dir150);
label.top(btex$\hbox{green}\atop\displaystyle(0,1,0)$etex,
   33mm*dir90);
label.urt(btex$(0,1,1)$etex,33mm*dir30);
label.lrt(btex$\hbox{blue}\atop\displaystyle(0,0,1)$etex,
   33mm*dir-30);
label.bot(btex$(1,0,1)$etex,33mm*dir-90);
endfig;
```

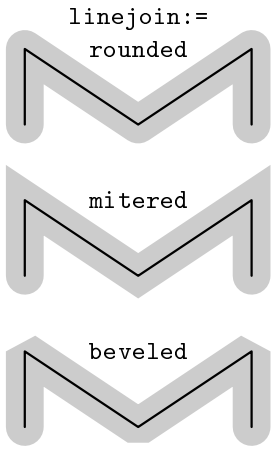
Vraťme se ještě ke kreslení čar. Zakončení čáry můžeme ovlivnit volbou parametru `linecap`, který může nabývat hodnot `rounded`, `butt` a `squared`. Předvolená hodnota je `rounded`:

```
beginfig(10)
path p[];
for i=1 upto 3:
  z[i]=(0,-i*15mm); z[i+3]=(20mm,(10-15i)*mm);
  p[i]=z[i]{dir45}..z[i+3];
endfor;
pickup pencircle scaled 5mm;
  draw p1 withcolor .8white;
  linecap:=butt; draw p2 withcolor .8white;
  linecap:=squared; draw p3 withcolor .8white;
linecap:=rounded;
pickup pencircle scaled .3mm;
for i = 1 upto 3: draw p [i]; endfor;
label(btex{\tt linecap:=}etex,(4mm,y1+16mm));
label.lft(btex{\tt rounded}etex,(x4+3mm,y4-8mm));
label.lft(btex{\tt butt}etex,(x4+3mm,y5-8mm));
label.lft(btex{\tt squared}etex,(x4+3mm,y6-8mm));
endfig;
```



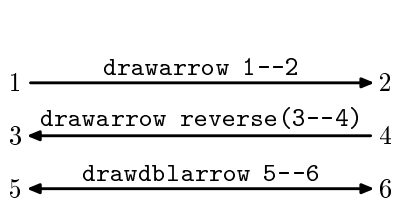
Obdobně můžeme ovlivnit vykreslení ostrých zlomů na čáře volbou parametru `linejoin`, který může nabývat hodnot `rounded`, `mitered` a `beveled`. Předvolená hodnota je `rounded`:

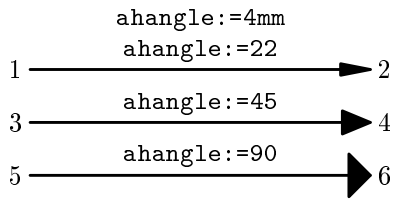
```
beginfig(11)
path p[];
p1=((0,0)--(0,10)--(15,0)--(30,10)--(30,0))
  scaled mm;
p2=p1 shifted (0,-20mm);
p3=p1 shifted (0,-40mm);
pickup pencircle scaled 5mm;
drawoptions(withcolor .8white);
  draw p1;
  linejoin:=mitered; draw p2;
  linejoin:=beveled; draw p3;
drawoptions();
  linejoin:=rounded;
pickup pencircle scaled .3mm;
for i = 1 upto 3: draw p [i]; endfor;
label(btex{\tt linejoin:=}etex,(15mm,14mm));
label(btex{\tt rounded}etex,(15mm,10mm));
label(btex{\tt mitered}etex,(15mm,-10mm));
label(btex{\tt beveled}etex,(15mm,-30mm));
endfig;
```



V předcházející ukázce si všimněte příkazu `drawoptions(`*nepovinná volba*`)`, který nahrazuje opakované připisování téže nepovinné volby (tentokrát je to `withcolor .8white`) za každý jednotlivý příkaz `draw`, `fill` nebo `label`. Zrušení nepovinné volby provedeme příkazem `drawoptions()`, jinak k němu dojde automaticky u `endfig`.

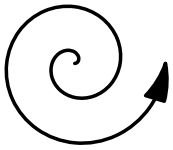
Velmi jednoduše je v MetaPostu vyřešeno kreslení šipek. Na konec cesty můžeme umístit plnou šipku příkazem `drawarrow` *cesta*. Předvolená délka ramen šipky `ahlength` je `4bp` a předvolený úhel ramen `ahangle` je 45°. Příkazem `drawarrow reverse` *cesta* umístíme šipku na začátek cesty a příkazem `drawdblarrow` *cesta* na začátek i konec. Parametry šipek můžeme ovšem změnit. Nová volba pak zůstává v platnosti i po `endfig` při kreslení následujících obrázků.





Ramena šipky vzniknou otočením konce cesty v délce `ahlength` a polovinu úhlu `ahangle` v kladném a záporném smyslu. Je-li tedy konec cesty křivý, promítne se to i do tvaru šipky:

```
beginfig(12)
path p;
p=(0,0){right}
  for i = 1 upto 24:..i*.5mm*dir(30i) endfor;
pickup pencircle scaled .5mm;
ahangle:=30; ahlength:=5mm;
drawarrow p;
endfig;
```

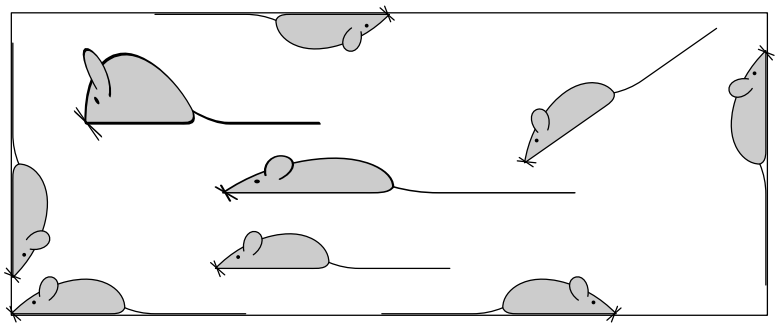


5 Skládání obrázků

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. S takto připraveným detailem můžeme provést libovolné afinní transformace a přidávat ho 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.

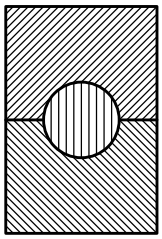


```
beginfig(1);
w:=100mm; h:=40mm;
picture mouse;
pickup pencircle scaled .2u;
fill((0,.4)---(25,.4)..{dir70}(30,2){up}..(26,8)..(15,9)
  ..{dir226}cycle)scaled .5u withcolor .8white;
draw ((0,.4)---(25,.4)..{dir70}(30,2){up}..(26,8)..(15,9)
  ..{dir226}cycle)scaled .5u;
draw ((30,2)..(38,.4)---(62,.4))scaled .5u;
fill ((7.5,5){dir100}..(11,10)..{dir215}(10,4)--cycle)scaled .5u
  withcolor .8white;
draw ((7.5,5){dir100}..(11,10)..{dir215}(10,4))scaled .5u;
draw (3dir-35--2dir130)shifted(0,.4) scaled .5u;
draw (2.5dir-60--2dir100)shifted(0,.4) scaled .5u;
fill fullcircle scaled .5u shifted (3u,1.7u);
mouse=currentpicture;
addto currentpicture also mouse shifted (27u,6u);
addto currentpicture also mouse xscaled 1.5 shifted (28u,16u);
addto currentpicture also mouse yscaled 2 slanted -.5
  shifted (10u,25u);
```

```
addto currentpicture also mouse rotated 180 shifted (50u,h);
addto currentpicture also mouse reflectedabout ((40u,0),(40u,h));
addto currentpicture also mouse rotated 35 shifted(68u,20u);
addto currentpicture also mouse reflectedabout(origin,dir45)
  shifted (0,5u);
addto currentpicture also mouse reflectedabout(origin,dir-45)
  shifted (w,35u);
draw unitsquare xscaled w yscaled h;
endfig;
```

Část obrázku ohraničenou uzavřenou cestou vystřihneme příkazem

`clip currentpicture to` *cesta*.



Toho využíváme například při šrafování ploch. V ukázce byl nejprve připraven vyšrafovaný kruh, který byl uložen do proměnné v typu *picture* a dočasně vymazán z pracovní plochy příkazem `clearit`. Pak byl nakreslen vyšrafovaný obdélník a nakonec do jeho středu umístěn vyšrafovaný kruh. Aby se zakrylo šrafování obdélníku, museli jsme kruh před jeho šrafováním „vybílit“ příkazem `unfill`.

```
beginfig(2);
picture v;
unfill fullcircle scaled 10mm;
pickup pencircle scaled .2mm;
for i=-4 upto 4: draw(i*mm,-10mm)--(i*mm,10mm);endfor;
clip currentpicture to fullcircle scaled 10mm;
pickup pencircle scaled .4mm;
draw fullcircle scaled 10mm;
v=currentpicture; clearit;
pickup pencircle scaled .2mm;
for i=-14 upto 19:
  draw ((0,0)--dir45)scaled 25u shifted(i*u,0);
  draw ((0,0)--dir-45)scaled 25u shifted(i*u,0);
endfor;
clip currentpicture to unitsquare xscaled 20u yscaled 30u
  shifted(15mm*down);
pickup pencircle scaled .4mm;
draw unitsquare xscaled 20u yscaled 30u shifted (15u*down);
draw(0,0)--(20mm,0);
addto currentpicture also v shifted (10mm,0);
endfig;
```

6 Afinní transformace

MetaPost 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)

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

stejnolehlost se středem v počátku a koeficientem a

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

a násobné zvětšení ve směru osy x

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

a násobné zvětšení ve směru osy y

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

zkosení ve směru osy x

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

otočení okolo počátku o úhel ϑ

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

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

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

Jen o málo složitější jsou transformační příkazy se dvěma parametry:

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

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

otočení okolo bodu (a,b) o úhel ϑ

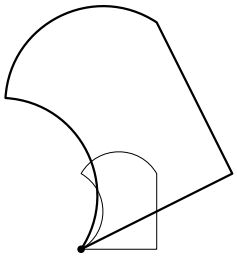
$$(x,y)\ \texttt{rotatedaround}((a,b),\vartheta))\,.$$

Následující obrázky znázorňují jednoduché transformace cesty

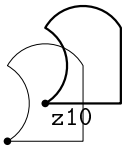
```
p0=origin--(10mm,0)--(10mm,10mm){dir120}..
                          {dir240}(0,10mm){dir330}..origin;
```

p0

p0 slanted 1/3

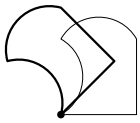


p0 zscaled (2,1)

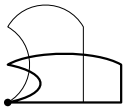


p0 shifted z10

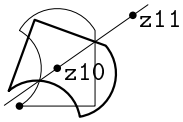
p0 scaled .5



p0 rotated 45



p0 xscaled 1.5 yscaled .5

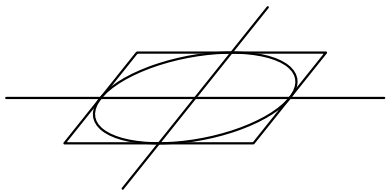


p0 reflectedabout (z10,z11)

p0 rotatedaround (z10,45)

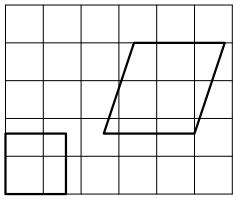
Základní transformace můžeme skládat v libovolném pořadí za sebou a vytvořit tak kterékoli afinní zobrazení. Používáme-li takovou složenou transformaci vícekrát, vyplatí se deklarovat pro ni zvláštní označení. Deklaraci složené transformace je nutno zahájit předdefinovanou transformací `identity`, která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

```
transform T;
T=identity xscaled 25mm yscaled 12mm
  slanted .8;
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 =(13mm,8mm);
(8mm,0) transformed T =(25mm,8mm);
(0,8mm) transformed T =(17mm,20mm);
draw unitsquare scaled 8mm;
draw unitsquare scaled 8mm transformed T;
```



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

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

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

$$(t_x, t_y, t_{xx}, t_{xy}, t_{yx}, t_{yy}) = (36{,}85033; 22{,}67712; 1{,}5; 0{,}5; 0; 1{,}5)\,.$$

Souřadnice tohoto vektoru můžeme použít samostatně jako

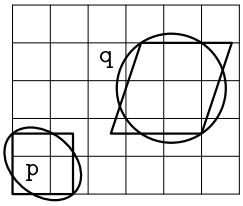
`xpart T`, `ypart T`, `xxpart T`, `xypart T`, `yxpart T`, `yypart T`.

K zavedené transformaci umí MetaPost příkazem `inverse` provést transformaci inverzní. Jestliže pro cesty `p` a `q` platí

```
q = p transformed T,
```

platí také

```
p = q transformed inverse T.
```



7 Podmínky, cykly

Syntaxe zápisu podmínky v MetaPostu 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. MetaPost 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 MetaPost 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 Metapostu 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. Program

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

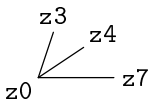
způsobí velmi husté vytečkování „ořezané“ sinusoidy:

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

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

Tento zápis MetaPost 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 řídící 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 1×1 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).

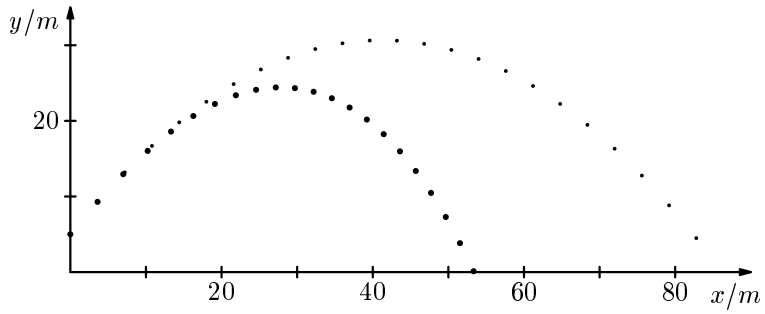
Poslední typ cyklu, který zde uvedeme, je nekonečný cyklus:

`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, MetaPost okamžitě přeruší zpracování tohoto cyklu a pokračuje textem za `endfor`. Následující příklad zobrazuje šikmý vrh ve vakuu a ve vzduchu. Cyklus je použit jako prostředek numerického modelování:

```
beginfig(4);
x:=0; y:=5; t:=0; dt:=.2; g:=9.8;           % vrh ve vakuu
v:=28; al:=50; vx:=v*cosd al; vy:=v*sind al;
pickup pencircle scaled .5mm;
forever:
  drawdot(x*mm,y*mm);
  ax:=0;  ay:=-g;
  x:=x+vx*dt; y:=y+vy*dt;
  vx:=vx+ax*dt; vy:=vy+ay*dt;
```

```
  exitif y<0;
endfor;
x:=0; y:=5; t:=0;                          % vrh ve vzduchu
vx:=v*cosd al; vy:=v*sind al; b:=.01;
pickup pencircle scaled .8mm;
forever:
  drawdot(x*mm,y*mm);
  ax:=-b*vx*(vx++vy);  ay:=-g-b*vy*(vx++vy);
  x:=x+vx*dt; y:=y+vy*dt;
  vx:=vx+ax*dt; vy:=vy+ay*dt;
  exitif y<0;
endfor;
pickup pencircle scaled .3mm;
ahangle:=30; ahlength:=1.5mm;
drawarrow (0,0)--(90mm,0); drawarrow (0,0)--(0,35mm);
for i=1 upto 3:
  draw(left--right) scaled .7mm shifted(0,i*10mm);
endfor;
for i=1 upto 8:
  draw(up--down) scaled .7mm shifted(i*10mm,0);
endfor;
label.bot(btex$x/m$etex,(88mm,-.4mm));
label.bot(btex 20 etex,(20mm,-.4mm));
label.bot(btex 40 etex,(40mm,-.4mm));
label.bot(btex 60 etex,(60mm,-.4mm));
label.bot(btex 80 etex,(80mm,-.4mm));
label.lft(btex$y/m$etex,(-.4mm,33mm));
label.lft(btex 20 etex,(-.4mm,20mm));
endfig;
```



8 Grafy funkcí

V matematice a ve fyzice často potřebujeme nakreslit graf funkce, u které známe její analytický předpis.

Při kreslení grafu využijeme toho, že Bézierova křivka, kterou proložíme dostatečně velkým počtem bodů, jejichž polohu vypočteme podle funkčního předpisu, odpovídá v mezích grafického rozlišení skutečnému grafu funkce. V první ukázce nakreslíme graf funkce $y = \sin 2x + 2\sin x$ v intervalu $\langle 0, 2\pi\rangle$.

Křivku grafu nejprve naprogramujeme jako cestu `p0` v takovém měřítku, aby zápis programu byl co nejjednodušší. V našem případě bohatě postačí, když zvolíme krok po deseti stupních. To znamená, že náš interval $\langle 0, 2\pi\rangle$, tj. $\langle 0°, 360°\rangle$, rozdělíme na 36 úseků.

Začneme v počátečním bodě grafu (v našem případě je to bod o souřadnicích (0,0)). Pak pomocí `for` cyklu s parametrem i vypočítáme polohy dalších bodů a současně jimi proložíme Bézierovu křivku:

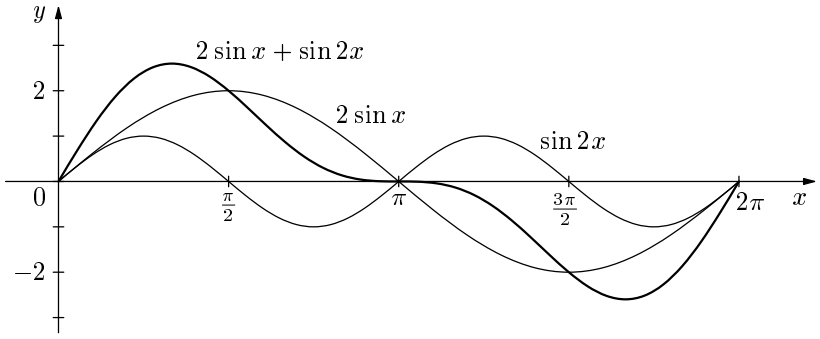
```
p0=(0,0) for i=1 upto 36:..(i,2*sind (10i)+sind(20i)) endfor;
```

Příkazem

```
p1=p0 xscaled 2.5u yscaled 6u;
```

ji pak roztáhneme ve směrech os na potřebnou velikost. Interval $\langle 0, 2\pi\rangle$ je na ose x zobrazen úsečkou délky 90 mm a jednotka na ose y má velikost 6 mm.

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.



```
beginfig(1)
path p[];
p0=(0,0) for i:=1 upto 36:..(i,2*sind(10i)+sind(20i)) endfor;
pickup pencircle scaled .3mm;
p1=p0 xscaled 2.5mm yscaled 6mm;
draw p1;
p10=(0,0) for i:=1 upto 20:..(i,sind (18i)) endfor;
p11=p10 xscaled 4.5mm yscaled 12mm;
p12=p10 xscaled 2.25mm yscaled 6mm;
p13=p10 xscaled 2.25mm yscaled 6mm shifted (45mm*right);
pickup pencircle scaled .2mm;
ahangle:=30; ahlength:=1.5mm;
drawarrow (-7mm,0)--(100mm,0);
drawarrow (0,-20mm)--(0,23mm);
for i=1 upto 4:
  draw (up--down) scaled .7mm shifted (22.5mm*i*right);
endfor;
for i=-3 upto 3:
  draw (left--right) scaled .7mm shifted (6mm*i*up);
endfor;
for i=11,12,13: draw p[i]; endfor;
label.bot(btex$x$etex,(98mm,-.5mm));
label.bot(btex$\pi$etex,(45mm,-.5mm));
label.bot(btex$2\pi$etex,(91.5mm,-.5mm));
label.bot(btex$\pi\over2$etex,(22.5mm,-.5mm));
label.bot(btex$3\pi\over2$etex,(67mm,-.5mm));
label.urt(btex$2\sin x+\sin2x$etex,point 7 of p1);
label.urt(btex$2\sin x$etex,point 8 of p11);
label.urt(btex$\sin2x$etex,point 8 of p13);
label.lft(btex$y$etex,(-.5mm,22mm));
label.lft(btex$2$etex,(-.5mm,12mm));
label.lft(btex$0$etex,(-.5mm,-2mm));
label.lft(btex$-2$etex,(-.5mm,-12mm));
endfig;
```

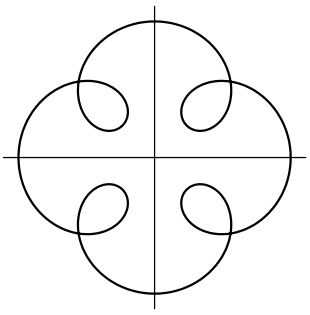
Podobným způsobem postupujeme i při kreslení křivek zadaných parametricky. Epicykloidu popsanou rovnicí

$$z = R\mathrm{e}^{\mathrm{i}\Omega t} + r\mathrm{e}^{\mathrm{i}\omega t},$$

nebo soustavou rovnic

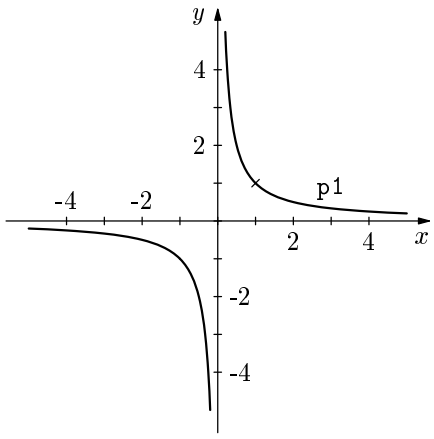
$$x = R\cos(\Omega t) + r\cos(\omega t),$$
$$y = R\sin(\Omega t) + r\sin(\omega t),$$

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



```
draw (18mm,0)
   for i=1 upto 179:..12mm*dir 2i+6mm*dir 10 endfor..cycle;
```

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



Nejprve vytvoříme graf funkce v intervalu $\langle 1, 5\rangle$ (cesta `p1`). Přihlédneme i k počátečnímu sklonu $-45°$. Zrcadlením cesty `p1` podle osy prvního a třetího kvadrantu a podle osy druhého a čtvrtého kvadrantu a jejím otočením okolo počátku soustavy souřadnic o $180°$ dostaneme zbývající části grafu.

```
beginfig(2)
path p[];
pickup pencircle scaled .2u;
ahangle:=30; ahlength:=1.5mm;
drawarrow(-28mm,0)--(28mm,0);
drawarrow(0,-28mm)--(0,28mm);
for i=-4 upto 4:
  draw (up--down) scaled .5u shifted (5u*i*right);
  draw (left--right) scaled .5u shifted (5u*i*up);
endfor;
p0=(1,1){dir-45} for i:=2 upto 5:..(i,1/i) endfor;
p1=p0 scaled 5u;
draw(dir-135--dir45)scaled .7u shifted (5u,5u);
pickup pencircle scaled .3mm;
draw p1;
draw p1 reflectedabout (origin,dir 45);
draw p1 reflectedabout (origin,dir -45);
draw p1 rotatedaround (origin,180);
label.top(btex{\tt p1}etex,point 2 of p1);
```

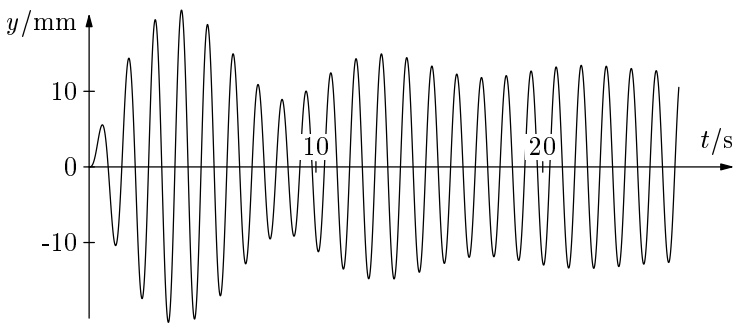
⋮

Je-li výpočet funkčních hodnot složitější, ukládáme výsledky nejprve do číselného pole a teprve po skončení výpočtů kreslíme graf. V následující ukázce je zobrazen časový průběh vynucených kmitů popsaných diferenciální rovnicí

$$\ddot{y} = -Ky - B\dot{y} + Q\sin\Omega t$$

s počátečními podmínkami $y_0 = 0$, $v_0 = 0$ a zadanými hodnotami koeficientů K, B, Q a Ω. Hodnoty funkce $y(t)$ jsou určeny postupným numerickým výpočtem. Aby byl výpočet dostatečně přesný, je třeba zvolit malý časový krok a pracovat s velkým počtem bodů. V takovém případě proložíme vypočtenými body lomenou čáru opakováním kreslícího příkazu `--`. Kdybychom chtěli použít hladkou Bézierovu křivku nakreslenou příkazem `..`, museli bychom graf rozdělit na úseky s méně než 300 body, abychom nepřekročili kapacitu MetaPostu, a nakreslit je samostatně.

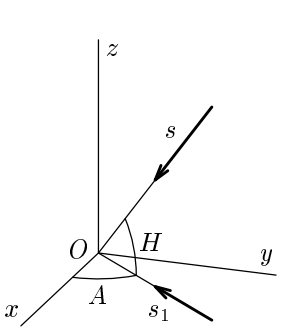
```
beginfig(4);
warningcheck:=0;
t:=0; y:=0; v:=0; dt:=.01;
K:=25; B:=.3; Q:=1; T:=1.1;
Om:=360/T;
for i:=0 upto 2600:
  y[i]:=y;
  a:=-K*y-B*v+Q*sind(Om*t);
  v:=v+a*dt;  y:=y+v*dt;  t:=t+dt;
endfor;
pickup pencircle scaled .2mm;
draw(0,y0*100mm)
    for i=1 upto 2600:--(i*.03mm,y[i]*100mm) endfor;
ahangle:=30; ahlength:=1.5mm;
drawarrow(0,0)--(85mm,0);
drawarrow(0,-20mm)--(0,20mm);
for i=-1 upto 1:
  draw (left--right) scaled .7mm shifted (0,i*10mm);
endfor;
label.lft(btex$y/{\rm mm}$etex,(-.5mm,19mm));
label.lft(btex -10 etex,(-.5mm,-10mm));
label.lft(btex 0 etex,(-.5mm,0));
label.lft(btex 10 etex,(-.5mm,10mm));
label.top(btex$t/{\rm s}$etex,(83mm,.5mm));
picture v[]; bboxmargin:=1.5;
v1=thelabel.top(btex 10 etex,(30mm,.5mm));
v2=thelabel.top(btex 20 etex,(60mm,.5mm));
for i=1 upto 2:
  draw (up--down) scaled .7mm shifted (i*30mm,0);
  unfill bbox v[i]; draw v[i];
endfor;
endfig;
```

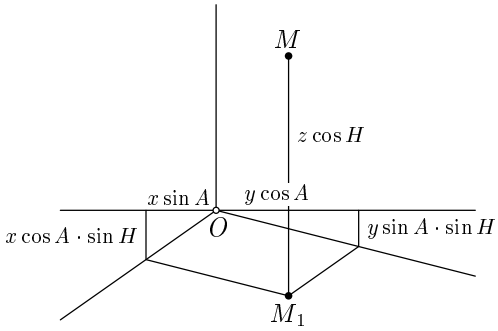


9 Axonometrické a kosoúhlé zobrazení

Axonometrii můžeme určit zadáním *azimutu* A a *úhlové výšky* H nekonečně vzdáleného bodu, ze kterého přicházejí promítací paprsky. Umístíme-li axonometrický průmět počátku O kartézské souřadnicové soustavy do referenčního bodu obrázku, dostaneme axonometrický průmět bodu $M = [x, y, z]$ pomocí následujícího makra, do kterého za `kx`, `ky` a `kz` dosadíme kartézské souřadnice x, y a z.

```
def ax(expr kx,ky,kz)=
 (ky*cosd A-kx*sind A,kz*cosd H-ky*sind A*sind H-kx*cosd A*sind H)
enddef;
```

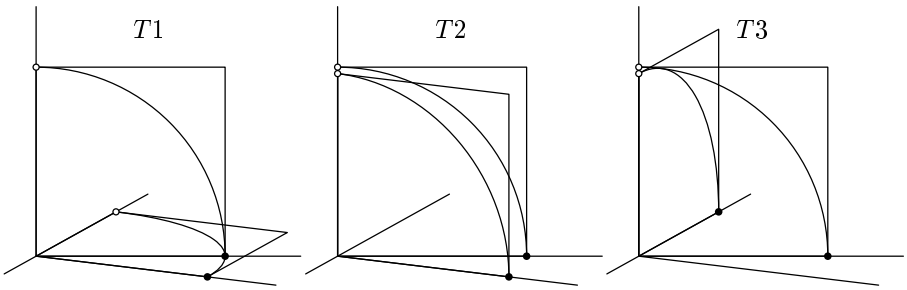




Například příkazem `z10=ax(10mm,20mm,30mm)` nalezneme axonometrický průmět bodu o kartézských souřadnicích $x = 10$ mm, $y = 20$ mm a $z = 30$ mm, kterému přiřadíme označení `z10`.

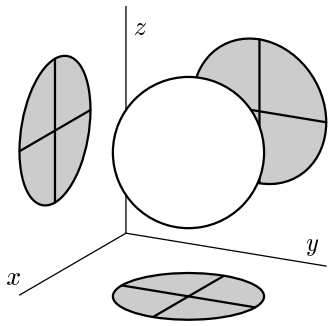
Následujícím makrem zavedeme transformace, pomocí kterých získáme tvary axonometrických průmětů cest ležících v rovinách rovnoběžných s půdorysnou, nárysnou nebo bokorysnou:

```
def axonometrie=
transform T[];
(0,0) transformed T1 =(0,0);    %% axonometrický půdorys
(1,0) transformed T1 =ax(0,1,0); (0,1) transformed T1 =ax(-1,0,0);

(0,0) transformed T2 =(0,0);    %% axonometrický nárys
(1,0) transformed T2 =ax(0,1,0); (0,1) transformed T2 =ax(0,0,1);

(0,0) transformed T3 =(0,0);    %% axonometrický bokorys
(1,0) transformed T3 =ax(-1,0,0); (0,1) transformed T3 =ax(0,0,1);
enddef;
```



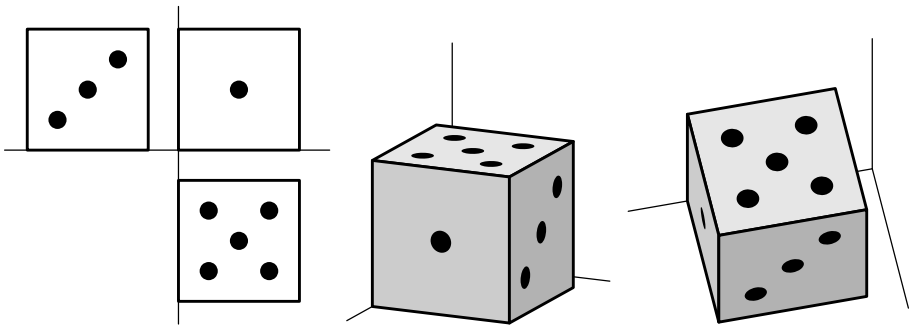
V ukázce je zobrazena koule o poloměru r se středem $[2r, 2r, 2r]$ a její průměty do rovin souřadných os. Průmět koule do půdorysny se zobrazuje jako elipsa, jejíž hlavní osa je kolmá k obrazu osy z a má velikost $2r$ a vedlejší osa má velikost $2r \sin H$. Obdobné vlastnosti mají obrazy průmětů koule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

```
beginfig(6)
z0=(0,0);
A:=28;H:=18;
r:=10u;
axonometrie;
penc.2u;
draw(z0--(0,-3r))transformed T1;
draw(z0--(3r,0))transformed T1;
draw z0--(0,3r);
z10=ax(2r,2r,2r);
z1=ax(2r,2r,0); z2=ax(0,2r,2r);
z3=ax(2r,0,2r);
penc.3u;
for i=1 upto 3:
  fill fullcircle scaled 2r transformed T[i] shifted z[i]
     withcolor .8white;
  draw fullcircle scaled 2r transformed T[i] shifted z[i];
  draw ((-r,0)--(r,0)) transformed T[i] shifted z[i];
  draw ((0,-r)--(0,r)) transformed T[i] shifted z[i];
endfor;
unfill fullcircle scaled 2r shifted z10;
draw fullcircle scaled 2r shifted z10;
label.ulft(btex$x$etex,(0,-2.8r)transformed T1);
label.top(btex$y$etex,(2.8r,0)transformed T1);
label.rt(btex$z$etex,(0,2.7r));
endfig;
```



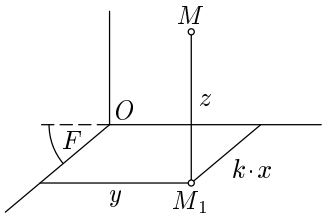
V další ukázce vidíme sdružené pravoúhlé průměty hrací kostky a její axonometrické průměty při volbě $A = 25°$, $H = 15°$ (prostřední obrázek) a při

volbě $A = 78°$, $H = 55°$ (obrázek vpravo). Zdrojový text se týká pouze prostředního obrázku. Pro pravý obrázek stačí změnit řádek se zadáním azimutu a výšky.



```
beginfig(8)
A:=25;H:=15;
z0=(0,0);
z1=(1,1);z2=(3,1);z3=(2,2);z4=(1,3);z5=(3,3);
axonometrie;
penc.2u;
draw z0--ax(33u,0,0); draw z0--ax(0,23u,0); draw z0--ax(0,0,30u);
z201=ax(25u,0,20u); z202=ax(25u,0,0); z203=ax(25u,20u,0);
pickup pencircle scaled.4u;
for i=1 upto 3:
  fill unitsquare scaled 20u transformed T[i] shifted z[200+i]
    withcolor (1-.1i)*white;
  draw unitsquare scaled 20u transformed T[i] shifted z[200+i];
endfor;
for i=1 upto 5:
  fill fullcircle scaled 3u transformed T1
    shifted (z201+5u*z[i]transformed T1);
endfor;
fill fullcircle scaled 3u transformed T2
    shifted (z202+5u*z3transformed T2);
for i=1,3,5:
  fill fullcircle scaled 3u transformed T3
    shifted (z203+5u*z[i]transformed T3);
endfor;
endfig;
```

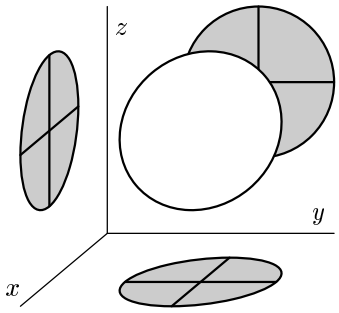
Kosoúhlé zobrazení určujeme směrem průmětu osy kolmé k nákresně a jeho zkrácením. I zde je výhodné zavést makro pro pro výpočet polohy kosoúhlého průmětu bodu o daných kartézských souřadnicích a transformace pro vytvoření kosoúhlých průmětů půdorysu, nárysu a bokorysu:




```
def kos(expr kx,ky,kz)=
  (ky-kx*k*cosd F,kz-kx*k*sind F)
enddef;

def kosouhle=
transform T[];
(0,0) transformed T1 =(0,0);    %% kosoúhlý půdorys
(1,0) transformed T1 =(1,0); (0,1) transformed T1 =kos(-1,0,0);

T2=identity;                    %% kosoúhlý nárys

(0,0) transformed T3 =(0,0);    %% kosoúhlý bokorys
(1,0) transformed T3 =kos(-1,0,0); (0,1) transformed T3 =(0,1);
enddef;
```

Kosoúhlé zobrazení koule a jejích průmětů do půdorysny, nárysny a bokorysny bylo získáno programem, který se téměř neliší od toho, který jsme použili v axonometrii:

```
beginfig(10)
z0=(0,0);
F:=40; k:=.5;
r:=10u;
kosouhle;
penc.2u;
draw kos(3r,0,0)--z0--(3r,0);
draw z0--(0,3r);
z10=kos(2r,2r,2r);
z1=kos(2r,2r,0); z2=kos(0,2r,2r);
z3=kos(2r,0,2r);
penc.3u;
for i=1 upto 3:
```



```
  fill fullcircle scaled 2r transformed T[i] shifted z[i]
     withcolor .8white;
  draw fullcircle scaled 2r transformed T[i] shifted z[i];
  draw ((-r,0)--(r,0)) transformed T[i] shifted z[i];
  draw ((0,-r)--(0,r)) transformed T[i] shifted z[i];
endfor;
unfill fullcircle scaled 2r xscaled sqrt(1+k**2)
  rotated f shifted z10;
draw fullcircle scaled 2r xscaled sqrt(1+k**2)
  rotated f shifted z10;
label.ulft(btex$x$etex,(0,-2.8r)transformed T1);
label.top(btex$y$etex,(2.8r,0)transformed T1);
label.rt(btex$z$etex,(0,2.7r));
endfig;
```