MetaPost for Beginners

Hartmut Henkel, Oftersheim, Germany
N 49°22.275′ E 008°35.553′

2[nd], 2008, Bohinj
N *xx*°*xx*.*xx*′ W *xx*°*xx*.*xx*′

August 2008

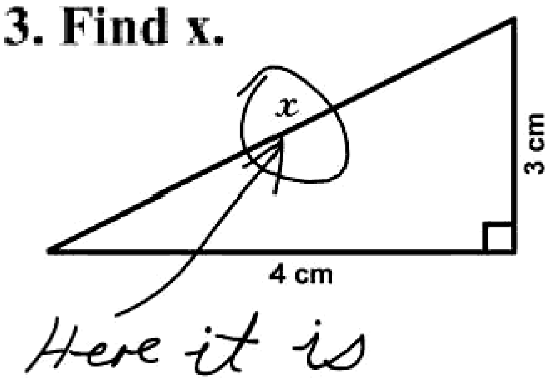
Introduction
What is MetaPost?

MetaPost — a picture drawing language and compiler with vector output.

What is MetaPost good for?

- Production of scientific and technical drawings.
- Results of highest typographic standards (not automatically, but. . .)
- Works perfectly together with LaTeX, TeX, and friends.
- Powerful macro language, extensible.
- Fun, even MetaFun :-)

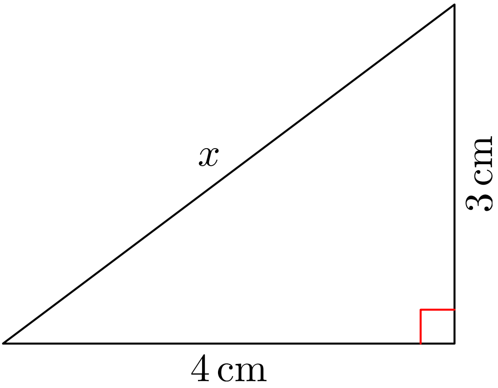
This is *not* MetaPost (from an exam). . .



Introduction

What is MetaPost?

. . . , but this is:



Shouldn't x be rotated? Graphics design questions. . .

Introduction

Tutorial Overview

Tutorial of $\approx$ 90 minutes:

- Introduction
- Workflow
- Showstoppers for beginners
- Basic concepts
- Macros
- Text inclusion
- Examples

Introduction

History

A short history of MetaPost:

- 1984: METAFONT Version 0 by D. E. Knuth
- 1990: MetaPost by John D. Hobby, based on METAFONT Version 1.9, Copyright 1990 – 1995 by AT&T Bell Laboratories.
- 1995: MetaPost Version 0.63
- Version 0.641 for long time, bugs accumulating

Major overhaul by Taco Hoekwater, pending bugs removed, functionality extended.

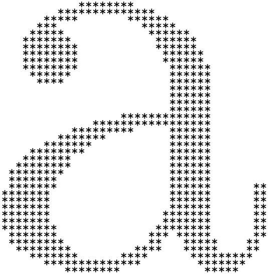
- Now (July 2007): Version 1.000
- Active development; next: Linkable MetaPost library. . .

Introduction

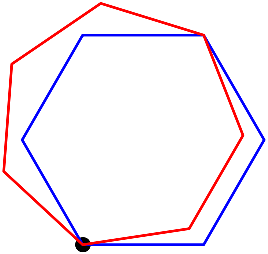
MetaPost vs. METAFONT

How is MetaPost related to METAFONT?

METAFONT
Raster output (GF = Generic Font)

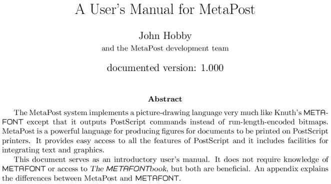


MetaPost
Vector output (PostScript)



Introduction

MetaPost info where?


A User's Manual for MetaPost

John Hobby
and the MetaPost development team

documented version: 1.000

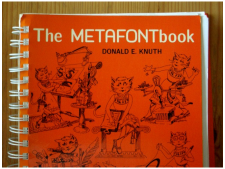
Abstract

The MetaPost system implements a picture-drawing language very much like Knuth's METAFONT except that it outputs PostScript commands instead of run-length-encoded bitmaps. MetaPost is a powerful language for producing figures for documents to be printed on PostScript printers. It provides easy access to all the features of PostScript and it includes facilities for integrating text and graphics.

This document serves as an introductory user's manual. It does not require knowledge of METAFONT or access to *The METAFONTbook*, but both are beneficial. An appendix explains the differences between MetaPost and METAFONT.

- "A User's Manual for MetaPost" by John D. Hobby (extended by the MetaPost Team)

Other indispensable source:



- "The METAFONTbook" by D. E. Knuth

MetaPost homepage:

- http://tug.org/metapost

Current development hosted at

- http://foundry.supelec.fr/projects/metapost/
  Check for new releases, maybe even participate in development. . .

Mailinglist:

- http://tug.org/mailman/listinfo/metapost

Introduction

MetaPost info where?

Another important information source:

- MetaPost macro package files.
  The fundamental macros are here:
  `/usr/local/texlive/2008/texmf-dist/metapost/base/plain.mp`

MetaPost input files typically have extension `.mp`

Where is `plain.mp`?
Try: `kpsewhich plain.mp`

A real treasure trove for MetaPost fans:

- MetaFun package with documentation, from Hans Hagen.

Workflow

Tools for playing with MetaPost…

What you need:

- MetaPost engine "`mpost`", helper programs, macro files…
  These are core components of any current TEX distribution.
  (e. g. TEX Live 2008).
- A text editor (`vi`, `emacs`, …).
  MetaPost requires text input (no window interface).
- Some PostScript viewer, e. g. GhostScript (`gs`).
- Or some PDF viewer, e. g. `xpdf`, `acroread`.
- Pen and paper.

Workflow

Very first simple drawing example

Create file fig.mp with editor (% starts comment):

```
prologues := 3;     % set up MetaPost for EPS generation
beginfig(1)         % begin figure no. 1
draw (0,0)--(3,4);  % actual drawing command(s)
endfig;             % end figure
end                 % end of MetaPost run
```

No TeX backslash '\'. Commands are separated by semicolon ';'!
Units: PostScript Points ($1/72\,\mathrm{in} = 0.352777\ldots\ \mathrm{mm}$)
Command line call:

```
mpost fig
```

And here is our first drawing, file `fig.1`: 
We see: 2-dimensional Cartesian coordinate system (right, up).

Workflow

Very first simple drawing example

`mpost` produces selfstanding Encapsulated PostScript file `fig.1`:

```
%!PS-Adobe-3.0 EPSF-3.0
%%BoundingBox: -1 -1 4 5
%%HiResBoundingBox: -0.25 -0.25 3.25 4.25
%%Creator: MetaPost 1.000
%%CreationDate: 2007.07.17:0158
%%Pages: 1
%%BeginProlog
%%EndProlog
%%Page: 1 1
 0 0 0 setrgbcolor 0 0.5 dtransform truncate idtransform
 setlinewidth pop [] 0 setdash 1 setlinecap 1 setlinejoin
 10 setmiterlimit
newpath 0 0 moveto
3 4 lineto stroke
showpage
%%EOF
```

Workflow

How to use MetaPost output in TEX workflow

Workflow with TEX/LATEX and `dvips`:

- `mpost fig.mp` → `fig.1`
- Include with `\includegraphics{fig.1}`,
  `latex` and `dvips` → EPS file

With pdfTEX/pdfLATEX:

- `mpost fig.mp` → `fig.1`
- Include with `\includegraphics{fig.1}`,
  `pdflatex` → PDF file
  This converts the EPSF output from `mpost` directly into PDF, using a parser from the ConTEXt package.

Workflow

How to use MetaPost output in TeX workflow

Other way with pdfTeX/pdfLaTeX, via PDF file:

- `mptopdf -raw fig.mp` → `fig-1.pdf`
  `mptopdf` gives selfstanding PDF output, versatile!
- Include with `\includegraphics{fig-1.pdf}`,
  `pdflatex` → PDF file

Yet another way via PDF file, using `mpost` with `prologues := 3`:

- `mpost fig.mp` → `fig.1` (selfstanding EPS file!)
- `epstopdf --outfile=fig-1.pdf --hires fig.1`
  → `fig-1.pdf`
  `epstopdf` uses GhostScript.
- Include with `\includegraphics{fig-1.pdf}`,
  `pdflatex` → PDF file

Workflow

Steps for graphics design with MetaPost

At the beginning often very helpful:

- Make sketch by hand (visualize problem).
- Mark key points in sketch.

Actual graphics programming and refinement:

- Write MetaPost program.
- Identify things that can be put into macros.
- Refine program using macros.
- If macros are used for several graphics, maybe consider creation of a MetaPost macro package.

Basic concepts

Variable types

Back to our first drawing command: `draw (0,0)--(3,4);`
There are...

- two points `(0,0)` and `(3,4)` → `pair` (one of MetaPost's variable types)
- straight line inbetween → `path` (a MetaPost variable type)
- (implicit) pen for stroking → `pen` (a MetaPost variable type)

In fact we can write:

```
beginfig(2)
pair a,b; path p; pen mypen;
a = (0,0); b = (3,4);
p = a--b;
mypen = pencircle scaled 1;
pickup mypen; draw p;
endfig;
```

Basic concepts

Variable types

All MetaPost variable types:

Type	Example
numeric	(default, if not explicitly declared)
pair	`pair a; a := (2in,3mm);`
boolean	`boolean v; v := false;`
path	`path p; p := fullcircle scaled 5mm;`
pen	`pen r; r := pencircle;`
picture	`picture q; q := nullpicture;`
transform	`transform t; t := identity rotated 20;`
color	`color c; c := (0,0,1);` (blue)
cmykcolor	`cmykcolor k; k := (1,0.8,0,0);` (some blue)
string	`string s; s := "Hello";`

Showstoppers for beginners

Watch out for these...

- The semicolon ;
- Assignments := vs. equations =
- Variable suffixes
- Pairs vs. the z macro

Showstoppers for beginners

The semicolon ;

In general: Each command must be ended by a semicolon.
But: MetaPost uses an interesting “expansion” concept.

```
beginfig(1)
pair a[]; a0=(0,0); a1=(1,0); a2=(1,1); a3=(0,1);
draw                 % no ; here!
  for i=0 upto 3:
    a[i]--           % no ; here!
  endfor             % no ; here!
cycle;
endfig;
```

This is in effect similar to following:

```
beginfig(1)
pair a[]; a0=(0,0); a1=(1,0); a2=(1,1); a3=(0,1);
draw a0--a1--a2--a3--cycle;
endfig;
```

Showstoppers for beginners

Assignments := vs. equations =

MetaPost has an integrated solver for linear equations and even equation systems! So we have:

- Assignments, like `a := 3;`
- Equations, like `3 = 4b;`

Know when to use := and when to use =.

Showstoppers for beginners

Assignments := vs. equations =

Assignment examples (variable on left side gets new value):

- `a := 3;` $\longrightarrow$ *a* gets the value 3
- `a := a + 1;` $\longrightarrow$ increment a

Forbidden (gives error), e. g.:

- `3 := a;`
- `(a,b) := (3,4);`

But `(a,b) = (3,4);` is ok (two variables can't be *assigned* simultaneously).

There can't go much wrong with exclusively using assingments, but you would miss MetaPost's powerful equation solver.

Showstoppers for beginners

Assignments `:=` vs. equations `=`

Equation examples:

- `a = b; b = 2-a;` $\rightarrow a = 1, b = 1$
- `(2,a) = (b,3)` $\rightarrow a = 3, b = 2$

Inconsistent equations give errors, e. g.:

- `a = b; a = b+1;` $\rightarrow$ Error message:
  `! Inconsistent equation (off by 1)`

Showstoppers for beginners

Variable suffixes

Variable names are made from “tags” (generic names) & suffixes.
Suffixes can be a mix of alpha/numeric/other tokens.
E. g., all these refer to the same variable:

- `a3` `a[3]` `a3.` `a[3.]` `a3.00` `a03.00`

Danger: The dot `.` is used in two cases:

- as decimal point in numeric suffix parts
- as separator between tags and alpha suffixes

This can lead to confusion:

- `a[foo]` refers to variable indexed by variable `foo`
- `a.foo` refers to variable with fixed suffix `foo`
- `a.7` refers to variable with suffix 0.7
- a7 refers to variable with suffix 7

Showstoppers for beginners

Variable suffixes

To be safe:

- If using suffixes composed from dots and numbers, think in real numbers.
- If in doubt, use square brackets [] around numeric suffixes.
- Learn by playing with suffixes...

Showstoppers for beginners

Pairs vs. the z macro

The pair variables `z` with suffix are special: They can only be calculated by equations, *not assigned* a pair value.
E. g., this gives an error:

- `z3 := (10mm,12mm);` → Error:
  `! Improper ‘:=’ will be changed to ‘=’.`

This is ok:

- `z3 = (10mm,12mm);`

Basic concepts

The special variables `x`, `y`, and `z`

The special pair variables `z` with suffix consist of `x` and `y` coordinate variables with similar suffix. E. g.:

```
z1 = (1,0);
x2 = 3; 4 = y2;
draw z1--z2;
```

In MetaPost z_k stands for (x_k, y_k), when k is any type of suffix. This is very handy!

- Use `z` variables wherever possible.

Basic concepts

The special variables `x`, `y`, and `z`

How to access *x*- and *y*-parts from ordinary pair variables? By `xpart` and `ypart`, e. g.:

```
pair a; a = (1,2);
x1 = 2 * xpart a;
y1 = 3 * ypart a;
```

Or, shorter:

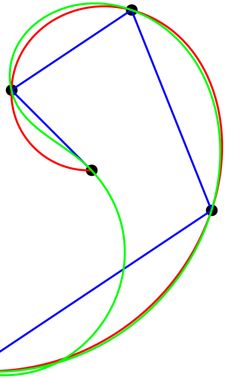
```
pair a; a = (1,2);
z1 = (2xpart a, 3ypart a);
```

Basic concepts

Straight and curved paths

Straight and curved paths, extending over 2 or more points:

```
beginfig(3)
z0 = origin; % short form for (0,0)
z1 = (60,40); z2 = (40,90);
z3 = (10,70); z4 = (30,50);
pickup pencircle scaled 1mm;
draw z0; draw z1; draw z2;
draw z3; draw z4;
pickup defaultpen;
draw z0--z1--z2--z3--z4 withcolor blue;
draw z0..z1..z2..z3..z4 withcolor red;
draw z0..z1..z2..z3..z4..z0 withcolor green;
endfig;
```



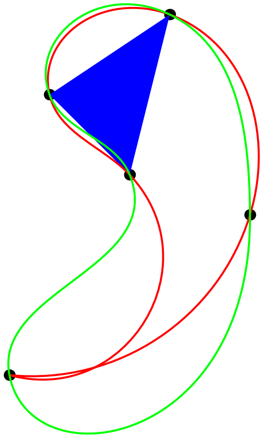
Color works!

Basic concepts

Closed paths, filling

Paths are closed by `cycle`:

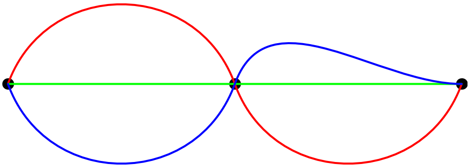
```
beginfig(4)
z0 = origin;
z1 = (60,40); z2 = (40,90);
z3 = (10,70); z4 = (30,50);
pickup pencircle scaled 1mm;
draw z0; draw z1;
draw z2; draw z3; draw z4;
pickup defaultpen;
fill z2--z3--z4--cycle withcolor blue;
draw z0..z1..z2..z3..z4..cycle withcolor red;
draw z0..z1..z2..z3..z4..z0 withcolor green;
endfig;
```



Basic concepts

Specifying path direction

Specifying path direction:



```
beginfig(5)
z0 = origin;
z2 = (40mm,0);
z1 = 0.5(z0+z2); % multiplication ‘*’ not required
pickup pencircle scaled 1mm;
draw z0; draw z1; draw z2;
pickup defaultpen;
draw z0..z1{dir -70}..z2 withcolor red;
draw z0..z1{dir 0}..z2 withcolor green;
draw z0..z1{dir 70}..{right}z2 withcolor blue;
endfig;
```

Basic concepts

Pre-defined vectors

Handy pre-defined vectors (macros):

`origin`	`(0,0)`
`right`	`(1,0)`
`left`	`(-1,0)`
`up`	`(0,1)`
`down`	`(0,-1)`

Their definitions are in file `plain.mp`.

Practical MetaPost functions regarding directions:

- `dir` x is the unit vector with direction x (in degrees)
- `angle`(x, y) gives numeric angle of pair z

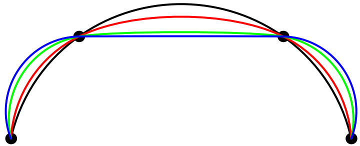
You will barely need sine and cosine (`sind`, `cosd`).

Basic concepts

Tension

Fine-tuning of curves in the middle by `tension`:

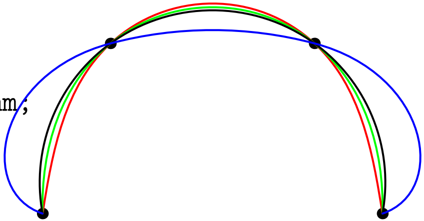
```
beginfig(6)
z0 = origin;
z3 = right*30mm; % same as (30mm,0)
x1 = 0.2[x0,x3]; % mediation ‘on the way between’
x2 = 0.8[x0,x3];
y1 = y2 = 0.3x3;
pickup pencircle scaled 1mm;
draw z0; draw z1;
draw z2; draw z3;
pickup defaultpen;
draw z0..z1.. tension 1   ..z2..z3;
draw z0..z1.. tension 1.2 ..z2..z3 withcolor red;
draw z0..z1.. tension 2   ..z2..z3 withcolor green;
draw z0..z1.. tension 5   ..z2..z3 withcolor blue;
endfig;
```



Basic concepts

Curl

Fine-tuning of curves in the end by `curl`



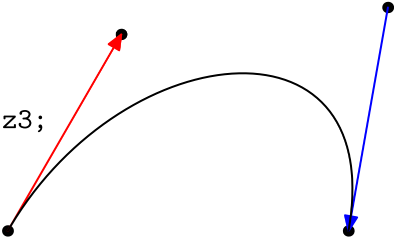
```
beginfig(7)
z0 = origin;
z3 = right*30mm;
x1 = 0.2[x0,x3];
x2 = 0.8[x0,x3];
y1 = y2 = 0.5x3;
pickup pencircle scaled 1mm;
draw z0; draw z1;
draw z2; draw z3;
pickup defaultpen;
draw z0{curl 0}  ..z1..z2..{curl 0}  z3 withcolor red;
draw z0{curl 0.5}..z1..z2..{curl 0.5}z3 withcolor green;
draw z0{curl 1}  ..z1..z2..{curl 1}  z3;
draw z0{curl 10} ..z1..z2..{curl 10} z3 withcolor blue;
endfig;
```

Underlying `{dir x}`, `tension`, `{curl x}`: Bézier Cubic Curves

2 control points for each point: `precontrol`, `postcontrol`

Curve may be also specified by curve points and control points:

```
beginfig(8)
z0 = origin;
z3 = right*30mm;
z1 = z0 + 20mm*dir 60;
z2 = z3 + 20mm*dir 80;
pickup pencircle scaled 1mm;
draw z0; draw z1; draw z2; draw z3;
pickup defaultpen;
drawarrow z0--z1 withcolor red;
drawarrow z2--z3 withcolor blue;
draw z0 .. controls z1 and z2 .. z3;
endfig;
```

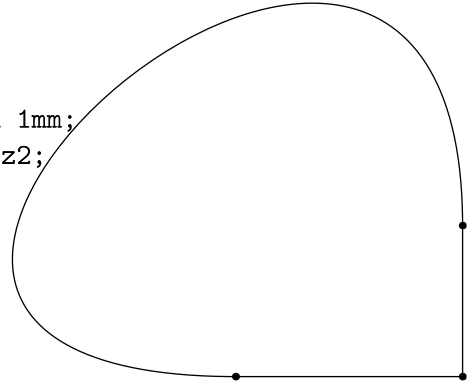


Basic concepts

Connecting paths by &

Paths can be connected, but only if they 'touch' (share a common endpoint):

```
beginfig(9)
path p,q,r;
z0 = origin;
z1 = right*30mm;
z2 = z1 + up*20mm;
pickup pencircle scaled 1mm;
draw z0; draw z1; draw z2;
pickup defaultpen;
p = z0--z1;
q = z1..z2;
r = p & q & cycle;
draw r;
endfig;
```

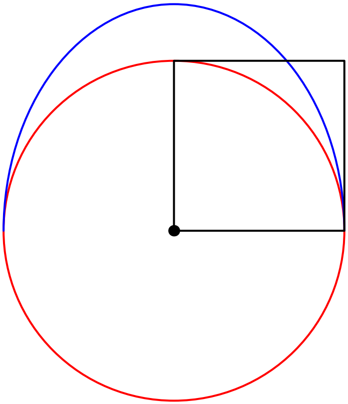


Basic concepts

Predefined paths

Predefined standard paths (macros, see `plain.mp`): quartercircle, halfcircle, fullcircle, unitsquare

```
beginfig(10)
pickup pencircle scaled 1mm;
draw origin;
pickup defaultpen;
draw fullcircle scaled 30mm
  withcolor red;
draw halfcircle xscaled 30mm
  yscaled 40mm withcolor blue;
draw unitsquare scaled 15mm;
endfig;
```



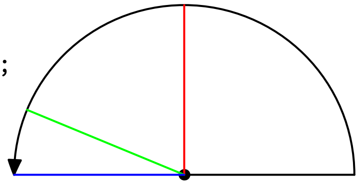
Paths can be `transformed`, e. g. `scaled`, `rotated`...

Basic concepts

Length of a path

Paths have a "length" and can be accessed parametrically:

```
beginfig(11)
path p;
draw origin
  withpen pencircle scaled 1mm;
p = halfcircle scaled 30mm;
drawarrow p;
draw origin--point 0 of p;
draw origin--point 2 of p withcolor red;
draw origin--point 3.5 of p withcolor green;
draw origin--point infinity of p withcolor blue;
endfig;
```



A `halfcircle` is made from 4 Bézier segments.

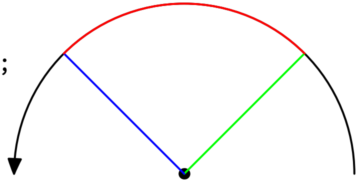
Dont mix with `arclength`; this gives the geometerical path length.

Basic concepts

Subpaths

Subpaths can be cut out from paths, given start and end parameters:

```
beginfig(12)
path p,q;
draw origin
  withpen pencircle scaled 1mm;
p = halfcircle scaled 30mm;
pickup defaultpen;
drawarrow p;
q = subpath(1,3) of p;
draw origin--point 0 of q withcolor green;
draw origin--point infinity of q withcolor blue;
draw q withcolor red;
endfig;
```

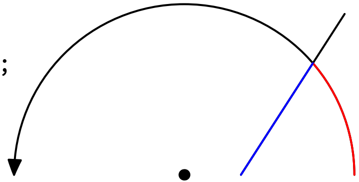


Basic concepts

Intersections between paths

Intersections between paths can be found.
`intersectiontimes` gives the parametric locations on both paths:

```
beginfig(13)
path p,q;
draw origin
  withpen pencircle scaled 1mm;
p = halfcircle scaled 30mm;
q = right*5mm--dir45*20mm;
drawarrow p; draw q;
z1 = p intersectiontimes q;
draw subpath (0, x1) of p withcolor red;
draw subpath (0, y1) of q withcolor blue;
endfig;
```



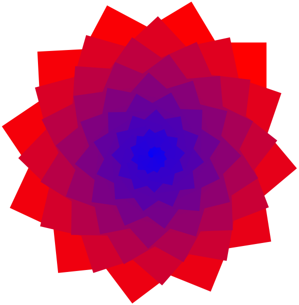
We get `z1=(-1,-1)` if there is no intersection.
There is also `intersectionpoint`, giving the point of intersection.

Basic concepts

for-loops

MetaPost loops: E. g. running over numeric range

```
beginfig(14)
for i=0 upto 100:
  fill unitsquare
    scaled ((100-i)*0.1mm)
    rotated 31i
    withcolor (0.01i)[red,blue];
endfor;
endfig;
```



Expression after `scaled` needs parenthesis.

Expression before `[red,blue]` needs parenthesis.

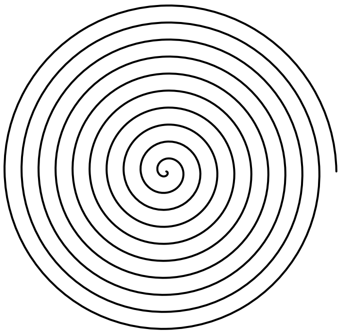
`31i` is ok, else it must be `(31*i)`

Basic concepts

Expansion of for-loops

A glimpse on expansion...

```
beginfig(15)
pair a;
a = right*15mm;
draw a
for i=30 step 30 until 3600:
  .. a rotated i
  scaled ((3600-i)/3600)
endfor;
endfig;
```



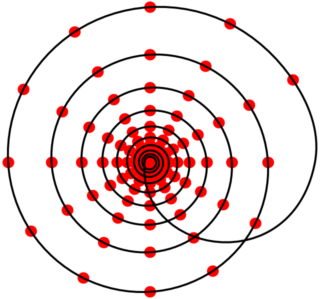
Points can be transformed like paths.
for used with step.
No semicolon inside for-loop here!

Basic concepts

Hiding stuff

Calculate and draw stuff without affecting main path:

```
beginfig(16)
pair a; a = right*15mm;
draw a
for i=30 step 30 until 3420:
  hide(a := 0.97a;
    draw a rotated i
    withpen pencircle
    scaled 1mm withcolor red)
  .. a rotated i
endfor .. cycle;
endfig;
```



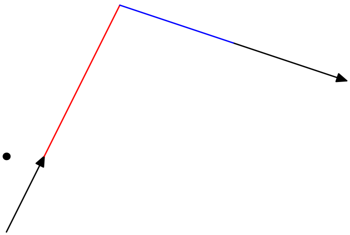
No semicolon after `hide()`, after `rotated i`, and after `endfor`!
And mind the `:=`

Basic concepts

Anonymous variables, whatever

Anonymous variables whatever to find point on a line:

```
beginfig(18)
draw origin withpen pencircle scaled 1mm;
z1 = down * 10mm; z2 = right * 5mm;
z3 = (30mm,15mm); z4 = (45mm,10mm);
z5 = whatever[z1,z2]
   = whatever[z3,z4];
drawarrow z1--z2;
drawarrow z3--z4;
draw z2--z5 withcolor red;
draw z3--z5 withcolor blue;
endfig;
```



Similar to writing e. g.: `z5 = n[z1,z2] = m[z3,z4];`

BTW, `intersectionpoint` won't work here (no intersection)!

Macros

Simple macros

Simplify expressions for repeated use or typical cases, e. g.:

```
for i=0 upto 100: endfor
```

. . . contains a simple parameterless macro:

```
def upto = step 1 until enddef;
```

So `for i=0 upto 100: endfor`
is same as: `for i=0 step 1 until 100: endfor`

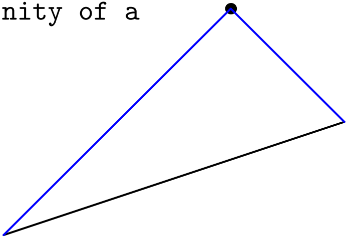
Other example:

```
def -- = {curl 1}..{curl 1} enddef;
```

Check out file `plain.mp` for more examples.

Simple macros with parameters

```
beginfig(19)
def sides(expr a,b) =
point 0 of a -- b -- point infinity of a
enddef;

path p;
p = origin--(30mm,10mm);
z1 = (20mm,20mm);
draw p;
draw z1 withpen pencircle scaled 1mm;
draw sides(p,z1) withcolor blue;
endfig;
```

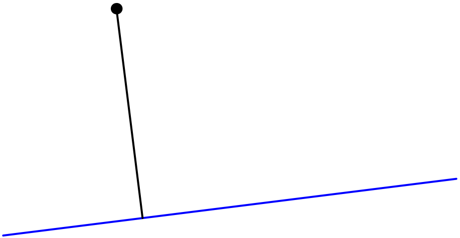


Macros

vardef

Vardef macros allow to do calculations and expand only to the result. E. g., the perpendicular through a point onto a given line.

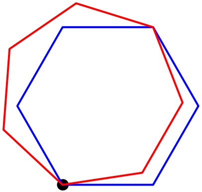
```
beginfig(20)
vardef perpendicular(expr a,b,c) =
pair p;
p = whatever[a,b] = c + whatever*((b-a) rotated 90);
p -- c
enddef;

path p;
z1 = origin;
z2 = (40mm,5mm);
z3 = (10mm,20mm);
draw z1--z2 withcolor blue;
draw z3 withpen pencircle scaled 1mm;
draw perpendicular(z1,z2,z3); endfig;
```



Macros

Macros with Suffixes

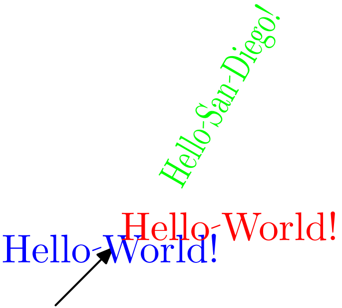
```
vardef setgon@#(expr c) =
  for i := 2 upto (c - 1):
    z@#[i]-z@#[i-1] = (z@#[i-1]-z@#[i-2]) rotated (360/c);
  endfor; ngon_@#=c;
enddef;
vardef gon@# =
  for i=0 upto ngon_@#-1: z@#[i] -- endfor cycle
enddef;

beginfig(21)
z.a0=z.b0=origin; z.a1=8mm*right;
setgon.a(6); setgon.b(7);
z.b3=z.a3;
draw z.a0 withpen pencircle scaled 1mm;
draw gon.a withcolor blue;
draw gon.b withcolor red; endfig;
```



Text

PostScript text

PostScript text (just simple text, fast)
A new data type: “`string`”



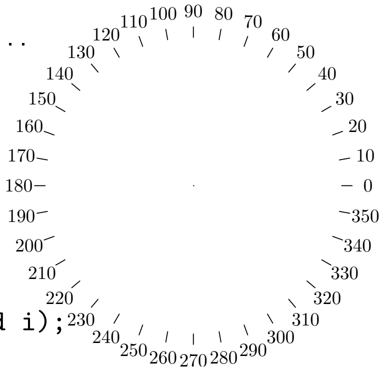

```
beginfig(22)
z1 = (5mm,5mm);
drawarrow origin--z1;
label("Hello World!", z1) withcolor blue;
label.urt("Hello World!", z1) withcolor red;
draw thelabel.rt("Hello" & " " & "San Diego!", origin)
  xscaled 0.7
  rotated 60 shifted 2z1 withcolor green;
endfig;
```

See string concatenation by use of &.
`thelabel` produces a “`picture`”, yet another data type.

Text

PostScript text

Another example of PostScript text...



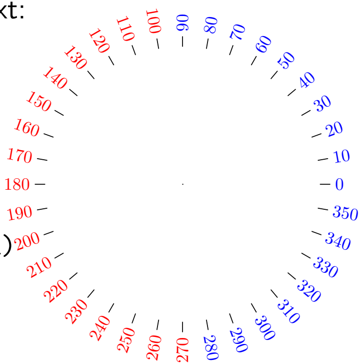

```
beginfig(23)
z1 = right*28mm;
z2 = right*30mm;
z3 = right*33mm;
draw origin;
for i=0 step 10 until 350:
  label(decimal(i),z3 rotated i);
  draw (z1--z2) rotated i;
endfor;
endfig;
```

decimal converts numeric type into string type.

Text

PostScript text

Yet another example of PostScript text:



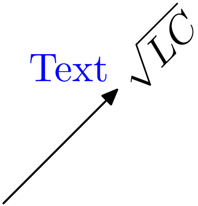

```
beginfig(24)
z1=right*28mm; z2=right*30mm;
draw origin;
for i=0 step 10 until 350:
  if (i < 100) or (i > 270):
    label.rt(decimal(i),origin)
      shifted z2 rotated i
      withcolor blue;
  else:
    label.lft(decimal(i),origin)
      rotated 180 shifted z2 rotated i withcolor red;
  fi;
  draw (z1--z2) rotated i;
endfor;
endfig;
```

Text

TEX text

Text between btex and etex is typeset by the TEX engine, and converted into a picture.

```
beginfig(25)
picture p;
z1 = (10mm,10mm);
drawarrow origin--z1;
label.ulft(btex Text etex, z1)
  withcolor blue;
p := btex $\sqrt{LC}$ etex;
label.rt(p, origin)
  rotated angle z1 shifted z1;
endfig;
```



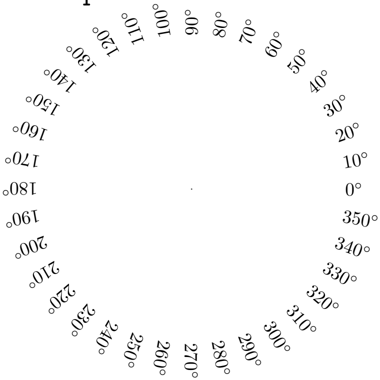
Slow, but with all typographic capabilities of TEX.

Text

TEX.mp macro file

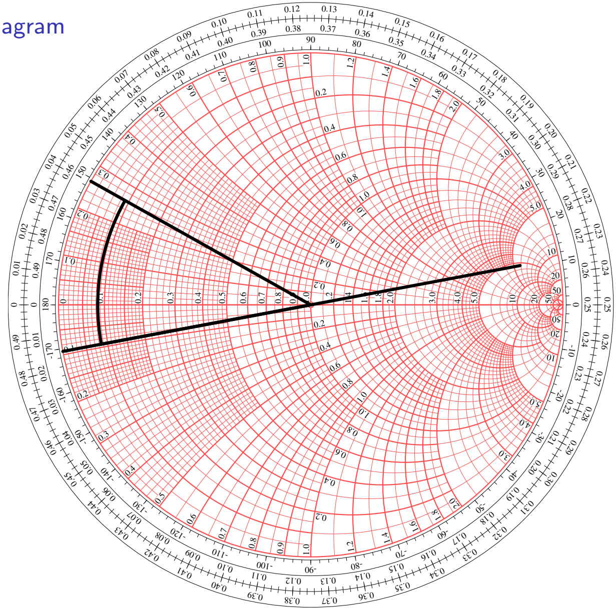
Dynamic TEX text requires to write string to temporary file (mptextmp.mp) and re-scan. Needs TEX.mp macro file. This is very slow, but most versatile.

```
input TEX; % loading macros



beginfig(26)
z1 = right*28mm;
draw origin;

for i=0 step 10 until 350:
  label.rt(TEX("$" & decimal(i) & "^\circ$"),origin)
    shifted z1 rotated i;
endfor;
endfig;
```



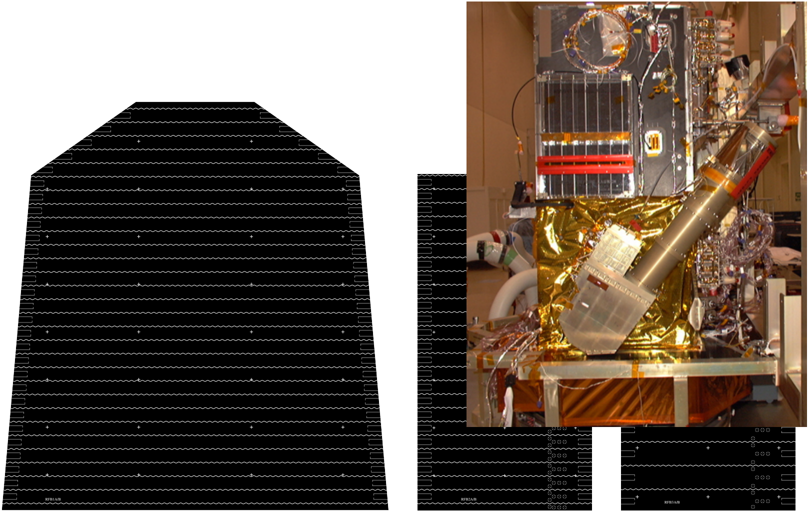
Examples

Smith-Chart diagram



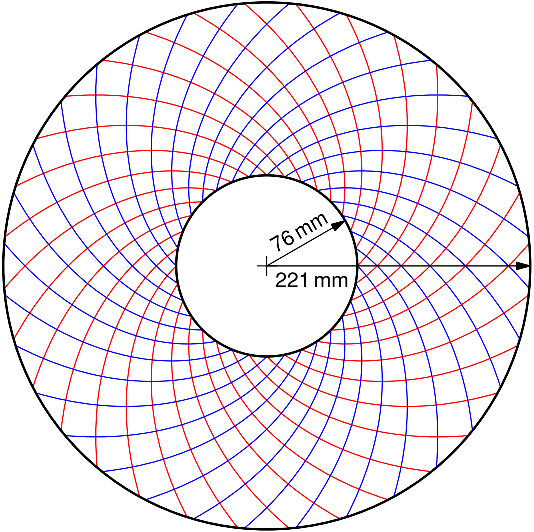
Examples

Printed circuit boards for ion optics (CIDA by vH&S)



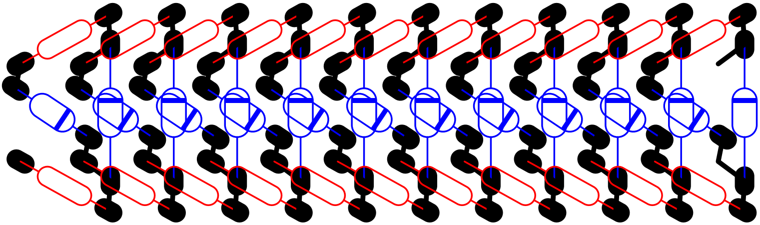
Examples

Logarithmic spirals (dust trajectory sensor)



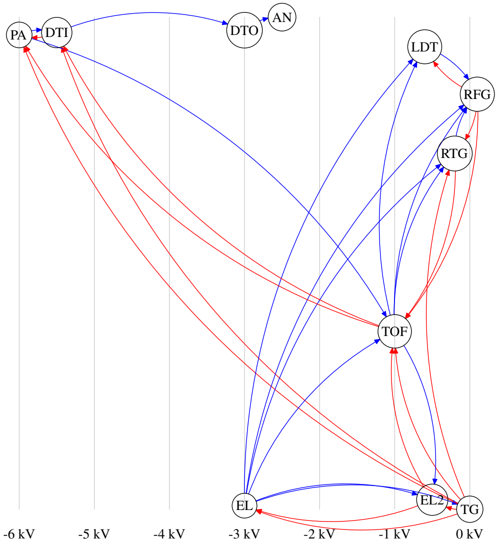
Examples

High-voltage cascade layout



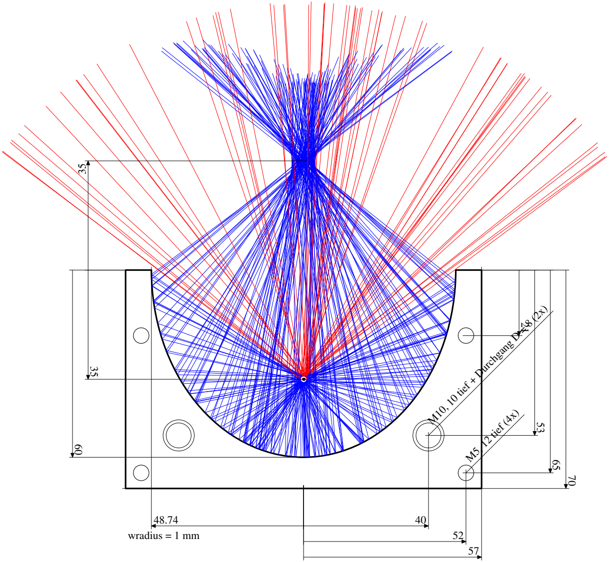
Examples

Potential plot with scatter ions (uses `boxes.mp`)



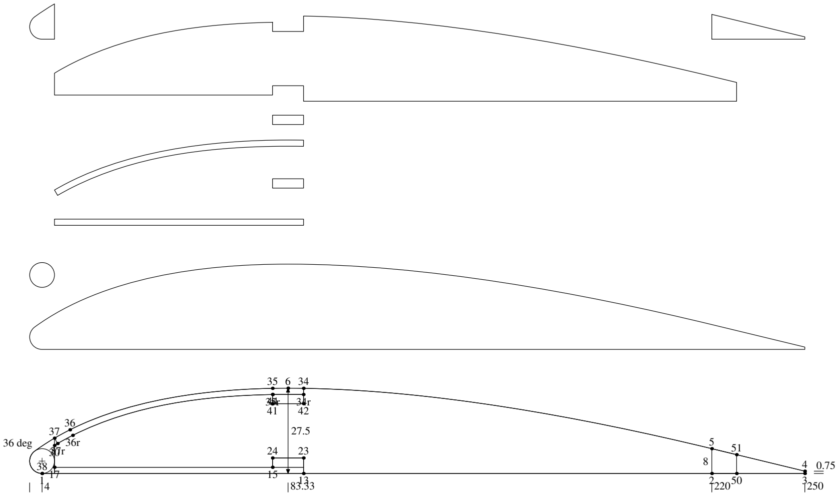
Examples

Raytracing of ellipsoid mirror



Examples

NACA wing design for RC plane model (Jörg Henkel)



Examples

Tiling ('Arabesque'), after Folke Hanfeld