MetaPost Command List

Shengjun Pan

July 24, 2005

Syntax	Explanation
`numeric, pair, path, transform, color, string, boolean, picture, pen.`	available data types.

`beginfig{Number};`
`commands`
`endfig;`
	`Number: output to file.Number`

`draw (x1,y1)--(x2,y2)--(x3,y3) [--cycle]`
`draw (x1,y1)..(x2,y2)..(x3,y3) [..cycle]`
`[withcolor <color>]`
	`--: straight line.`
	`..: curve.`
	`(combination allowed)`
	`cycle: closed`
	`default unit: bp`

`draw z1..controls x1 and x2 .. z2`	`each curve betwwen z1 and z2 has two control points x1,x2.`
`draw z1..{dir D1}z2{dir D2}..z3`	`predefined directions:` `left, right, up, down`
`draw z1..tension # .. z2`	`#: set tension, default = 1`
`draw z0{curl c}..z1..{curl c}z2`	`c: set curl, default =1`
`z1=1/3[z2,z3]`	`equivalent to` `z1=z2+1/3*(z3-z2)`
`z0=whatever[z1,z3]=whatever[z2,z4]`	`whatever: anonymous variable,` `useful in solving equations.`
`pickup <pen expression>`	`choose pen style`

`a++b`
`a+-+b`
`z1 dotprod z2`
	$a++b=\sqrt{a^2+b^2}$
	$a+-+b=\sqrt{a^2-b^2}$
	dotprod: dot product

`"abc" & "de"` `substring (pos1,pos2) of "abc"`	`&: string concatenation` `substring: positions (pos1,pos2]`

Syntax	Explanation
p=(xpart p, ypart p) c=(redpart c, greenpart c, bluepart c) z1=(x1,y1) z2r=(x2r,y2r) z.a=(x.a,y.a)	some equivalence.
pair p[]; numeric p[]q[];	array declaration.
label<suffix>(content,pos) dotlabel<suffix>(content,pos) dotlabels<suffix>(suffixes)	suffix: lft, rt, top, bot, ulft, llft, urt, lrt (a dot '.' before suffix) suffixes: of z, separated by ','
btex <TeX Commands> etex	not LaTeX. fraction is "a\over b". data type: picture.
label("text" infont .....,pos) char(n) infont fontname	infont: followed by font vars. n: n-th char in fontname.
bbox <pic> setbounds <pic> to <path>	bbox: bounding box, closed path, ll-> lr -> ur -> ul -> cycle
fill <closed path> fill <closed path> withcolor <color>	
buildcycle(path1,path2,...)	choose intersection to be as late as possible on path$i$, and as early as possible on path$(i+1)$
a intersectiontimes b	early time gets priority. total time = # points -1
length<path> arclength <path> arctime <arclength> of <path>	t = arctime a of p <=> arclength subpath(0,t) of p=a
point <time> of <path> direction <time> of <path> directiontime <vector> of <path> directionpoint <vector> of <path>	
reverse <path> subpath (t1,t2) of <path> path1 cutbefore path2 path1 cutafter path2	$t_1 \leqslant t_2$

Syntax	Explanation
(x,y) shifted (a,b) rotated θ slanted a scaled a xscaled a yscaled a zscaled (a,b) transformed $(t_x,t_y,t_{xx},t_{xy},t_{yx},t_{yy})$	(x,y) slanted $a = (x+ay,y)$; (x,y) zscaled $(a,b) = (ax-by,bx+ay)$, product of complex numbers (x,y) transformed T $= (t_x+t_{xx}x+t_{xy}y, t_y+t_{yx}x+t_{yy}y)$

`transform T,T';`
`T= indentity xscaled -1 .......`
`T'= inverse T;`
	T' : inverse transform of T

`reflectedabout(p,q)`	reflect about line defined by p and q.
`rotatedaround(p,theta)`	counter-clockwise
`xxpart T = yypart T;` `yxpart T = -xypart T;`	shape preserving.
`draw <path> dashed <dash pattern>` `dashpattern(on <size> off <size> ...)`	predefined dash pattern: **withdots , evenly** dashpattern: argument repeated. dash pattern is treated as a picture, can be drawn or transformed, and any picture can be used as a dash pattern.
`lincap:= rounded | butt | squared`	default = rounded. butt: cut the ends off

`drawarrow <path>`
`drawdblarrow <path>`
`internal vars for arrows:`
`ahlength, ahangle`	

`filldraw <closed path> <options>`	draw the path and fill the area closed by the path.
`undraw ...` `unfilldraw ...`	`erase, equivalent to` `draw/filldraw withcolor background`
`drawoptions(<text>)` `drawoptions()`	With drawoptions, we don't need to type options each time we draw. Empty list returns all options to default.

`pickup <pen>`
`makepen <closed path>`
`maekpath <pen>`
	<pen> is essentially a picture, can be transformed.
	defaultpen = pencircle scaled 0.5bp

Syntax	Explanation
`clip <pic> <closed path>`	erase all the part of <pic> outside <closed path> use **currentpicture** to clip the whole picture.

`def <symbol>[(datatype var1,...)]`
`= <replacement> enddef`
`def <symbol>(varlist)(varlist)`
`= <replacement> enddef`
	datatype:
	expr means var1,... can be arbitary expressions.
	text is more general.
	suffix var1,... can only be variables.
	The second def can be called by using one pair of ().

`begingroup`
`...`
`save var1,...`
`interim <intervar>:= <expr>`
`...`
`endgroup`
	save: make variables local.
	interim: make an internal variable local yet without forgetting internality.

`if <boolean>: ... else: ... fi`
`if <boolean>: ...`
`elseif <boolean>: ...`
`else: ...`
`fi`	

`vardef a[]b(expr p) = <replacement> enddef` `vardef <macrname>@# = <replacement> enddef`	In the first vardef, when the macro name is **a2b**, @ = b, #@ = a2 In the second vardef, when the macro name is **z.a1**, @# = a1.

`vardef <name> primary <arg> =...`
`def <name> expr <arg>=...`
`def <name> expr <arg1> of <arg2>=...`
`def <name>(arg list) expr <arg2>=...`
`primarydef <arg1> <name> <arg2> = ...`
	These are different types of macro definitions.
	primarydef could be replaced by:
	secondarydef or **tertiarydef**.

`<primary> -> <variable>`
`| (<expression>)`
`| <nullary op>`
`| <of operator><expression>of<primary>`
`| <unary op><primary>`
`<secondary> -> <primary>`
`| <secondary><primary binop><primary>`
`<tertiary> -> <secondary>`
`| <tertiary><secondary binop><secondary>`
`<expression> -> <tertiary>`
`| <expression><tertiary binop><tertiary>`
	The overall syntax rules for expressions. Understanding the rules is important in **vardef** and **def**.

Syntax	Explanation

`for <symbol> = <expr> upto <expr>:`
`<loop>`
`endfor`
 	
`for i=a step b until c:`
`<loop>`
`endfor`
 	
`for t=<List>:`
`<loop>`
`endfor`
 	
`forsuffixes <symbol> = <List>:`
`<loop>`
`endfor`
 	
`forever:`
`...`
`exitif<boolean>`
`exitunless<boolean>`
`...`
`endfor`
	different versions of loops.

`boxit.<suffix>(<pic>)`
`drawboxed(<suffix list>)`
`  boxjoin(<equation text>)`
`drawunboxed(...)`
`drawboxes(...)`
	If boxit.bb(<pic>) is used,
	bb.c: center,
	bb.dx, bb.dy
	(defaultdx, defaultdy),
	bb.n, bb.s, bb.w, bb.e,
	bb.nw, bb.ne, bb.se, bb.sw

`bpath<box name>` `pic<box name>`	both **bpath.**<**boxname**> and **bpath** <**boxname**> are legal. Same for **pic**.
`circleit<boxname>(<box contents>)`	If circleit.cc(<pic>) is used, cc.c: center, cc.dx, cc.dy (defaultdx, defaultdy), bb.n, bb.s, bb.w, bb.e,