Pictures in TEX with METAFONT

mfpic.tex Version: 0.2.10.8 alfa
Program Date: Tue 28 May 1996
Principal Author: Dr Thomas E. Leathrum
Alfatest version by: Geoffrey Tobin (G.Tobin@latrobe.edu.au)
Document Date: Tue 28 May 1996

• *WHY?*

Tom got the idea for mfpic mostly out of a feeling of frustration. Different output mechanisms for printing or viewing TEX DVI files each have their own ways to include pictures. More often than not, there are provisions for including PostScript data into a DVI file using TEX \special's. However, this technique seems far from TEX's ideal of device-independence, and besides, different TEX output drivers handle these \special's in different ways. The same problems arise with including tpic \special's.

LATEX's picture environment has a hopelessly limited supply of available objects to draw—if you want to draw a graph of a polynomial curve, you're out of luck.

There is, of course, PICTEX, which is wonderfully flexible and general, but its most obvious feature is its speed—or rather lack of it. Processing a single picture in PICTEX can often take several minutes.

It occurred to Tom that it might be possible to take advantage of the fact that METAFONT is *designed* for drawing things. The result of pursuing this idea is mfpic, a set of macros for TEX and METAFONT which incorporate METAFONT-drawn pictures into a TEX file.

The nature of the macros, from the user's point of view, is very much like PICTEX. We do not pretend that mfpic has anything like the scope of PICTEX, but it should suit most purposes for drawing small graphs and including them in your TEX documents.

• *AUTHOR.*

mfpic was written primarily by Tom Leathrum (moth@bluejay.atl.ga.us) during the late (northern hemisphere) spring and summer of 1992, while at Dartmouth College. Different versions were being written and tested for nearly two years after that, during which time Tom finished his Ph.D. and took a job at Berry College, in Rome, GA. Between fall of 1992 and fall of 1993, much of the development was carried out by others. Those who helped most in this process are credited in the Acknowledgements.

• *MANIFEST.*

Eighteen files are included in this mfpic alfa test distribution:

Acknowl.tex	Some people whose work has helped mfpic
CHANGES.tex	History of changes to mfpic
CTAN.sites	A list of CTAN sites, mirrors and shadows
MANIFEST	List of these files, with sizes and dates
Makefile.dist	Distribution Makefile
NOTE	Some concerns
README2	An overview of this distribution

`grafbase.mf`	The METAFONT macros
`grafdoc.tex`	Plain TEXdocumentation for `grafbase.mf`
`header.tex`	Definitions used in the documentation files
`lapictures.tex`	A LATEX2e version of `pictures.tex`
`mf-revu.tex`	A sketchy review of METAFONT programming
`mfpic.tex`	The TEX macros
`mfpic-IS.gt`	Brief explanation of what MFpic is and does
`mfpicdoc.tex`	This document, processable in plain TEX
`objects.tex`	Contains at least one picture of each object
`pictures.tex`	A few more complicated pictures
`skip-pix.tex`	Tests effect of TEX's `\leftskip`

Information on how to set up a few specific configurations of computers, and some contributed code and METAFONT documentation, are separately available.

• *SETTING UP and PROCESSING.*

Setting up TEX and METAFONT to process these files will, to an extent, depend on your local installation. The biggest problem you are likely to have, regardless of your installation, will be convincing TEX and its output drivers to find METAFONT's output files.

To process the sample file, first run TEX on the sample file `pictures.tex`. TEX will complain that it can't find the file `pics.tfm`, but will continue processing the file anyway. When the file is finished processing, you will now have a file called `pics.mf`. This is the METAFONT file containing the descriptions of the pictures for `pictures.tex`. You need to run METAFONT on `pics.mf`, with `\mode=localfont` set up. (Read your METAFONT manual to see how to do this.) Now that you have the font and font metric files generated by METAFONT, reprocess the file `pictures.tex` with TEX. The resulting DVI file should now be complete, and you should be able to print and view it at your computer (assuming your viewer and print driver have been set up to be able to find the font generated by `pics.mf`).

These three steps of processing—processing with TEX, processing with METAFONT, and reprocessing with TEX—may not always be necessary. In particular, if you change the TeX document without making any changes at all to the pictures, then there will be no need to repeat either of the last two steps.

There is also a somewhat subtle circumstance under which you can skip the third step—if you change the picture in such a way as *not* to affect the font metric file, then you do not have to reprocess with TEX, because the original metric used for the first step will put the pictures in the right places. The only `mfpic` macros that affect the font metric file are the macros listed in the *Files and Environments* section below.

• *HOW IT WORKS.*

When you run TEX on the file `pictures.tex`, TEX generates a file `pics.mf`. This file is formed by `\write` commands in the `mfpic` macros. The user should never have to read or change the file `pics.mf` directly—the `mfpic` macros take care of it.

Be prepared for overfull hboxes, due to the fact that in `pictures.tex` the diagrams'

`\tcaption`'s are deliberately too wide for the specified widths of the `\mfpic` pictures.

The user familiar with METAFONT will notice, by looking at the `mfpic` macros, that the `mfpic` drawing macros translate almost directly into simple METAFONT `draw` commands. The `\tlabel`'s and `\tcaption`'s, however, are placed on the graph by TEX, not METAFONT.

Multiline tlabels may be specified by explicit line breaks, which are indicated by the `\\` command.

- *THE MACROS.*

- *Beware!*

Due to the current method by which line breaks in the TEX file are preserved in the METAFONT file, it is necessary to commence the argument of a command on the same line as the command, for example THIS works:

```
    \cyclic { point1, point2 }
```

AS does this:

```
    \cyclic { point1,
            point2 }
```

AND this:

```
    \cyclic {
            point1,
            point2 }
```

BUT this does NOT (it causes the whole argument to be omitted from the METAFONT file) :

```
    \cyclic
            { point1,
            point2 }
```

- *Files and Environments.*

`\opengraphsfile{`⟨*font*⟩`}`...`\closegraphsfile`

These macros open and close the METAFONT file which will contain the pictures to be included in this document. The name of the file will be ⟨*font*⟩`.mf`. If the ⟨*font*⟩ parameter is changed, you will have to reprocess the TEX file after processing ⟨*font*⟩`.mf`.

`\mfpic[`⟨*xscale*⟩`][`⟨*yscale*⟩`]{`⟨*xneg*⟩`}{`⟨*xpos*⟩`}{`⟨*yneg*⟩`}{`⟨*ypos*⟩`}`...`\endmfpic`

These macros open and close the `mfpic` environment in which the rest of the macros below make sense. The `\mfpic` macro also sets up the local coordinate system for the picture. The ⟨*xscale*⟩ and ⟨*yscale*⟩ parameters establish the length of a coordinate system unit, as a multiple of the TEX dimension `\mfpicunit`. At least one scale parameter must be specified, but if only one is specified, then they are assumed to be equal. The ⟨*xneg*⟩ and ⟨*xpos*⟩ parameters establish the lower and upper (resp.) bounds for the x-axis coordinates; similarly, ⟨*yneg*⟩ and ⟨*ypos*⟩ establish the bounds for the y-axis. These bounds are expressed in local units—in other words, the actual width of the picture will be (⟨*xpos*⟩−⟨*xneg*⟩)·⟨*xscale*⟩ times `\mfpicunit`, its height (⟨*ypos*⟩−⟨*yneg*⟩)·⟨*yscale*⟩ times `\mfpicunit`, and its depth zero. These scales and bounds are used primarily to establish the metric for the character containing the picture described within the environment. If any of these parameters are changed, the ⟨*font*⟩`.tfm` file will be affected, so you will have to reprocess the TEX file after processing ⟨*font*⟩`.mf`.

• *Using* `mfpic` *with* LATEX.

In LATEX, instead of using the `\mfpic` and `\endmfpic` macros, you may prefer to use `\begin{mfpic}` and `\end{mfpic}`. Due to the way that LATEX has been designed, `\begin{command}` effectively means `\command`, and `\end{command}` effectively means `\endcommand`, for any TEX command.

A LATEX version of the sample file, `lapictures.tex`, has also been provided.

Be prepared for overfull hboxes, due to the fact that the diagrams' `\tcaption`'s are deliberately too wide for the specified widths of the `\mfpic` pictures.

Note that the `\opengraphsfile` and `\closegraphsfile` macros should be used under those names in LATEX too, as they do not quite possess a `\command`...`\endcommand` structure.

This version of `mfpic` should be compatible with the LaTeX `center` environment.

The rest of the `mfpic` macros do not affect the font metric file (⟨*font*⟩`.tfm`), and so if these commands are changed or added in your document, you will not have to repeat the third step of processing (reprocessing with TEX) to complete your TEX document.

For the remainder of the macros, the numerical parameters are expressed in the units of the local coordinate system specified by `\mfpic`, unless otherwise indicated.

• *Figures.*

METAFONT *Pairs.*

Since many of the arguments of the `mfpic` drawing commands are sent to METAFONT to be interpreted, it's useful to know something about METAFONT concepts.

In particular, METAFONT has `pair` objects, which may be constants or variables. Pair constants have the form `(`x`,`y`)`. Pairs are two-dimensional rectangular (cartesian) quantities, and are clearly useful for representing both points and vectors on the plane.

To shorten the descriptions of `mfpic` macros, we herein often represent each pair by a brief name, such as p, v or c, the meanings of which are usually obvious in the context of the macro. The succinctness of this notation also helps us to think geometrically rather than only of coordinates.

Points, Lines, and Rectangles.

`\point[`⟨*ptsize*⟩`]` [⟨p_0⟩,⟨p_1⟩,...]

Draws small disks centered at the points ⟨p_0⟩, ⟨p_1⟩, and so on. If the optional argument ⟨*ptsize*⟩ is present, it determines the diameter of the disks, which otherwise equals the TEX dimension `\pointsize`. The default value of `\pointsize` is 2 points. The disks have a filled interior if `\pointfilled` is true, otherwise their interior is erased.

`\polyline{`⟨p_0⟩`,`⟨p_1⟩`,...}` `\lines{`⟨p_0⟩`,`⟨p_1⟩`,...}`

Draws the line segment with endpoints at ⟨p_0⟩ and ⟨p_1⟩, then the line segment with endpoints at ⟨p_1⟩ and ⟨p_2⟩, etc. The result is an open polygonal path through the specified points, in the specified order.

`\polygon{`$\langle p_0\rangle$`,`$\langle p_1\rangle$`,...}` Draws a closed polygon with vertices at the specified points.

`\rect{`$\langle p_0\rangle$`,`$\langle p_1\rangle$`}`

Draws the rectangle specified by the points $\langle p_0\rangle$ and $\langle p_1\rangle$, these being any two opposite corners of the rectangle.

Axes and Axis Marks.

`\axes`

Draws the x and y axes for the coordinate system. The axes extend the full width and height of the `mfpic` environment. The length of the arrowhead on each axis is determined by the TeX dimension `\axisheadlen`. The default value of `\axisheadlen` is 5 points. The shape of the arrowhead is determined as in the `\arrow` macro.

`\xmarks{`$\langle x_0\rangle$`,`$\langle x_1\rangle$`,...}` and `\ymarks{`$\langle y_0\rangle$`,`$\langle y_1\rangle$`,...}`

These macros place hash marks on the x and y axes (resp.) at the places indicated by the values in the list. The length of the hash marks is determined by the TeX dimension `\hashlen`. The default value of `\hashlen` is 4 points.

Circles and Ellipses.

`\circle{`$\langle c\rangle$`,`$\langle r\rangle$`}`

Draws a circle centered at the point $\langle c\rangle$ and with radius $\langle r\rangle$.

`\ellipse[`$\langle\theta\rangle$`]{`$\langle c\rangle$`,`$\langle r_x\rangle$`,`$\langle r_y\rangle$`}`

Draws an ellipse with the x radius $\langle r_x\rangle$ and y radius $\langle r_y\rangle$, centered at the point $\langle c\rangle$. The optional parameter $\langle\theta\rangle$ provides a way of rotating the ellipse by $\langle\theta\rangle$ degrees counterclockwise around its center.

Curves.

`\curve{`$\langle p_0\rangle$`,`$\langle p_1\rangle$`,...}`

Draws a METAFONT Bézier path through the specified points, in the specified order.

`\cyclic{`$\langle p_0\rangle$`,`$\langle p_1\rangle$`,...}`

Draws a cyclic (i.e., closed) METAFONT Bézier curve through the specified points, in the specified order.

Circular Arcs.

`\arc[`⟨*format*⟩`]{...}`

Draws a circular arc specified as determined by the ⟨*format*⟩ optional parameter—this macro is unusual in that the optional ⟨*format*⟩ parameter determines the format of the other parameter, as indicated below:

`\arc[s]{`$\langle p_0\rangle$`,`$\langle p_1\rangle$`,`⟨*sweep*⟩`}`

(*Point-Sweep Form —this is the default format.*) Draws a circular arc starting from the point $\langle p_0\rangle$, ending at the point $\langle p_1\rangle$, and covering an arc angle of ⟨*sweep*⟩ degrees, measured counterclockwise around the center of the circle. If, for example, the points $\langle p_0\rangle$ and $\langle p_1\rangle$ lie on a horizontal line with $\langle p_0\rangle$ to the left, and ⟨*sweep*⟩ is between 0 and 180 (degrees), then the center of the circle will be above the horizontal line (in order for the angle to be counterclockwise). Negative values of $\langle sweep\rangle$ give arcs curving in the other direction.

`\arc[t]{`$\langle p_0\rangle$`,`$\langle p_1\rangle$`,`$\langle p_2\rangle$`}`
(*Three-Point Form.*) Draws the circular arc which passes through all three points given.

`\arc[p]{`$\langle c\rangle$`,`$\langle r\rangle$`,`$\langle \theta_1\rangle$`,`$\langle \theta_2\rangle$`}`
(*Polar Form.*) Draws the circular arc with center $\langle c\rangle$ and radius $\langle r\rangle$, starting at the angle $\langle \theta_1\rangle$ and ending at the angle $\langle \theta_2\rangle$, where both angles are measured counterclockwise from the positive x axis.

`\arc[c]{`$\langle c\rangle$`,`$\langle p_1\rangle$`,`$\langle \theta\rangle$`}`
(*Center-Point Form.*) Draws the circular arc with center $\langle c\rangle$, starting at the point $\langle p_1\rangle$, and sweeping an angle of $\langle \theta\rangle$ around the center from that point. (This is actually `mfpic`'s internal way of handling arcs—all other formats are translated to this format before drawing.)

Polar Coordinates.

`\plr{(`$\langle r_0\rangle$`,`$\langle \theta_0\rangle$`), (`$\langle r_1\rangle$`,`$\langle \theta_1\rangle$`), ...}`
Replaces the specified list of polar coordinate pairs by the equivalent list of rectangular (cartesian) coordinate pairs. Through `\plr`, commands designed for rectangular coordinates can be applied to data represented in polar coordinates—and to data containing both rectangular and polar coordinate pairs.

Other Figures.

`\turtle{`$\langle p_0\rangle$`,`$\langle v_1\rangle$`,`$\langle v_2\rangle$`,...}`
Draws a line segment, starting from the point $\langle p_0\rangle$, and extending along the (2-dimensional vector) displacement $\langle v_1\rangle$. It then draws a line segment from the previous segment's endpoint, along displacement $\langle v_2\rangle$. This process continues for all listed displacements, similarly to "turtle graphics".

`\sector{`$\langle c\rangle$`,`$\langle r\rangle$`,`$\langle \theta_1\rangle$`,`$\langle \theta_2\rangle$`}`
Draws the sector, from the angle $\langle \theta_1\rangle$ to the angle $\langle \theta_2\rangle$ inside the circle with center at the point $\langle c\rangle$ and radius $\langle r\rangle$, where both angles are measured in degrees counterclockwise from the direction parallel to the x axis. The sector forms a closed path.

• *Shape-Modifier Macros.*

Some `mfpic` macros operate as *shape-modifier* macros—for example, if you want to put an arrowhead on a line segment, you could write: `{\arrow\lines{(0,0),(1,0)}}`. The example illustrates two modifiers that are *switches*; these apply from when they are used until the end of the innermost enclosing TeX scope. All but one of the `mfpic` modifier macros are described here.

For the purposes of these macros, a distinction must be made in the figure macros between "open" and "closed" paths. Note that a path that merely returns to its starting point is *not* automatically closed; such a path is open, and must be explicitly closed, for example by `\closed` (see below). (On the METAFONT level, path closure is achieved by some variant of `..cycle`). The (already) closed paths are: `\rect`, `\circle`, `\ellipse`, `\cyclic`, `\plrregion` and `\btwnfcn` (below).

Closure of Paths.

`\lclosed`

Makes each open path into a closed path by adding a line segment between the endpoints of the path.

`\bclosed`
`\sclosed`
`\cbclosed`

These macros are similar to `\lclosed`, except that they close each open path by drawing a Bézier, or a smooth curve (as in the smooth case in the `\curve` macro), or a cubic B-spline, respectively, between the path's endpoints.

Reversal, Accumulation and Connection of Paths.

`\reverse`...

Turns a path around, reversing its orientation. This will affect both the direction of arrows (e.g. bi-directional arrows can be coded with `\arrow\reverse\arrow`... —here the first `\arrow` modifier applies to the *reversed* path), and the order of endpoints for a `\connect`...`\endconnect` environment (below).

`\patharr{`⟨*pv*⟩`}`...`\endpatharr`

This pair of macros, acting as an environment, accumulate all enclosing paths, in order, into a path array named ⟨*pv*⟩.

Note: In LaTeX, this pair of macros can be used in the form of a LaTeX-style environment called `patharr` —as in `\begin{patharr}`...`\end{patharr}`.

`\connect`...`\endconnect`

This pair of macros, acting as an environment, add line segments from the trailing endpoint of one open path to the leading endpoint of the next path, in the given order. The result is a connected, *open* path.

Note 1: `\connect` and `\endconnect` are jointly implemented using the the `patharr` environment with a METAFONT path array named '`nexus`'.

Note 2: In LaTeX, this pair of macros canbe used in the form of a LaTeX-style environment called `connect` — as in `\begin{connect}`...`\end{connect}`.

Drawing.

`\draw`...

Draws the subsequent path using a solid outline. When `mfpic` is loaded, this is the initial way in which a path is rendered by default, if no rendering prefix is given. (See the description of `\setrender` below.)

`\dashed`...

Draws dashed segments along the path specified in the next command. The length of the dashes is the value of the TeX dimension `\dashlen`. The space between the dashes is the value of the TeX dimension `\dashspace`. Adjusts the space between the dashes by as much as $\frac{dashspace}{n}$, where n is the number of spaces appearing in the curve, in order to have the proper dashes at the ends. The dashes at the ends are half of `\dashlen` long.

`\dotted`...

Draws dots along the specified path. The size of the dots is the value of the TEX dimension `\dotsize`. The space between the dots is the value of the TEX dimension `\dotspace`.

Arrows.

`\arrow[l`⟨*headlen*⟩`] [r`⟨*rotate*⟩`] [b`⟨*backset*⟩`]`...

Draws an arrowhead at the endpoint of the open path (or at the last key point of the closed path) that follows. The optional parameter ⟨*headlen*⟩ determines the length of the arrowhead. The default is the value of the TEX dimension `\headlen`. The optional parameter ⟨*rotate*⟩ allows the arrowhead to be rotated counterclockwise around its point an angle of ⟨*rotate*⟩ degrees. The default is 0 degrees. The optional parameter ⟨*backset*⟩ allows the arrowhead to be "set back" from its original point, thus allowing e.g. double arrowheads. This parameter is in the form of a TEX dimension—its default value is 0 points. If an arrowhead is both rotated and set back, the rotation affects the direction in which the arrowhead is set back. The optional parameters may appear in any order, but the indicated key character for each parameter must always appear.

Shading, Filling, Erasing, Hatching.

The shading macros can all be used to shade the interior of closed paths, even if the paths cross themselves. Shading an open curve is technically an error, but the METAFONT code in `grafbase.mf` responds by drawing the path and not doing any filling.

`\gfill`...

Fills in the subsequent closed path.

`\gclear`...

Erases everything inside the subsequent closed path.

`\shade[`⟨*shadesp*⟩`]`...

Shades the interior of the subsequent closed path with dots. The diameter of the dots is set by the macro `\shadewd`. The optional argument specifies the space between dots, which defaults to the TEX dimension `\shadespace`. If `\shadespace` is 0 points (or less), the closed path is filled, as with `\gfill`.

`\thatch[`⟨*hatchsp*⟩`,`⟨*angle*⟩`]`...

Shades a closed path using lines at the specified angle. The thickness of the lines is set by the macro `\hatchwd`. In the optional argument, ⟨*hatchsp*⟩ specifies the space between lines, which defaults to the TEX dimension `\hatchspace`. If `\hatchspace` is 0 points (or less), the closed path is filled, as with `\gfill`. The ⟨*angle*⟩ defaults to 0 degrees. Either both optional arguments must be present, or both must be absent.

`\lhatch[`⟨*hatchsp*⟩`]`...

Draws lines shading in the subsequent closed path in a "left-oblique hatched" (upper left to lower right) pattern. The thickness of the lines is set by the macro `\hatchwd`. The optional ⟨*hatchsp*⟩ argument is as in `\thatch`.

`\rhatch[`⟨*hatchsp*⟩`]`...

Draws lines shading in the subsequent closed path in a "right-oblique hatched" (lower left to upper right) pattern. The thickness of the lines is set by the macro `\hatchwd`. The optional ⟨*hatchsp*⟩ argument is as in `\thatch`.

`\hatch[`⟨*hatchsp*⟩`]`...
`\xhatch[`⟨*hatchsp*⟩`]`...

Draws lines shading in the subsequent closed path in a "cross-hatched" pattern. The thickness of the lines is set by the macro `\hatchwd`. The optional ⟨*hatchsp*⟩ argument is as in `\thatch`.

Changing the Default Rendering.

Rendering is the process of converting a geometric description into a drawing. In METAFONT, this means producing a bitmap (METAFONT calls this a `picture`), either by stroking (drawing) a path using a particular pen), or by filling a closed path.

`\setrender{`⟨*TEX commands*⟩`}` Initially, `mfpic` uses the `\draw` command (stroking) as the default operation when a figure is to be rendered. However, this can be changed to any combination of (rendering and/or other!) TEX commands, by using the `\setrender` command. This is a local redefinition, so it can be enclosed in braces to restrict its range.

Examples.

It may be instructive, for the purpose of seeing how the syntax of *shape-modifier switches* works, to consider two examples:

{\ shade\ draw\ lclosed\ lines[...]}

which shades inside a polygon and draws its outline; and

{\ shade\ lclosed\ draw\ lines[...]}

which shades inside the polygon, and draws all of the outline *except* the line segment supplied by `\lclosed`.

• *Affine Transforms.*

Coordinate transformations that keep parallel lines in parallel are called **affine transforms**. These include translation, rotation, reflection, scaling and skewing (slanting). For the METAFONT coordinate system only—that is, for paths, but not for `\tlabel`'s (let alone `\tcaption`'s)—`mfpic` provides the ability to apply METAFONT affine transforms. Transforms can be localised to any group of METAFONT paths (this is implemented using a METAFONT path array, `paths`).

Rotation of Paths.

`\rotatepath{(`⟨x⟩`,`⟨y⟩`),`⟨θ⟩`}`

Rotates the following path by ⟨θ⟩ degrees about point `(`⟨x⟩`,`⟨y⟩`)`.

Affine Transforms of the METAFONT *Coordinate System.*

`\coords...\endcoords`

All affine transforms are restricted to the innermost enclosing `\coords...\endcoords` pair. If there is *no* such enclosure, then the transforms will apply globally, and *even across* `mfpic` *environments*, so be careful to remember to enclose transforms!

Note: In LaTeX, a `coords` environment may be used.

`\applyT{`⟨*transformer*⟩`}`

Apply the METAFONT ⟨*transformer*⟩ to the current coordinate system. For example, the `mfpic` TeX macro `\zslant#1` is implemented as `\applyT{zslanted #1}` where the argument `#1` is a METAFONT pair, such as (x, y).

Transforms provided by `mfpic`.

`\rotate{`θ`}`	Rotates around origin by θ degrees
`\rotatearound{`p`}{`θ`}`	Rotates around point p by θ degrees
`\turn[`p`]{`θ`}`	Rotates around point p (origin is default) by θ degrees
`\mirror{`p_1`}{`p_2`}`	
`\reflectabout{`p_1`}{`p_2`}`	Reflects about the line $p_1 \ldots p_2$
`\shift{`p`}`	Shifts origin by the vector p
`\scale{`s`}`	Scales uniformly by a factor of s
`\xscale{`s`}`	Scales only X by a factor of s
`\yscale{`s`}`	Scales only Y by a factor of s
`\zscale{`p`}`	Scales uniformly by magnitude of p, and rotates by angle of p
`\xslant{`s`}`	Skew in X direction by the multiple s of Y
`\yslant{`s`}`	Skew in Y direction by the multiple s of X
`\zslant{`s`}`	See `zslanted` in `grafdoc.tex`
`\boost{`χ`}`	Special relativity boost by χ
`\xyswap`	Reflects in the line $Y = X$

• *Functions and Plotting.*

`\fdef{`⟨*fcn*⟩`}(`⟨*param1*⟩,⟨*param2*⟩,. . .)[⟨*mf-expr*⟩]

Defines a METAFONT function ⟨*fcn*⟩ of the parameters ⟨*param1*⟩, ⟨*param2*⟩, . . ., by the METAFONT expression ⟨*mf-expr*⟩ in which the only free parameters are those named. The return type of the function is the same as the type of the expression.

The expression ⟨*mf-expr*⟩ is passed directly into the corresponding METAFONT macro and interpreted there, so METAFONT's rules for algebraic expressions apply.

Operations available include `+`, `-`, `*`, `/`, and `**` (`x**y`= x^y), with `(` and `)` for grouping. Functions available include `round`, `floor`, `ceiling`, `abs`, `sqrt`, `sind`, `cosd`, `mlog`, and `mexp`. (*Notes:* The METAFONT trigonometric functions `sind` and `cosd` take arguments in degrees; `mlog(x)`= $256 \ln x$, and `mexp` is its inverse. There are other operations and functions available, but these are the most useful for plotting purposes.) You can also define the function ⟨*fcn*⟩ by cases, using the METAFONT conditional expression

`if` ⟨*boolean*⟩`:` ⟨*expr*⟩ `else:` ⟨*expr*⟩ `fi`.

Relations available for the ⟨*boolean*⟩ part of the expression include `=`, `<`, `>`, `<=`, and `>=`.

Complicated functions can be defined by a compound expression, which is a series of METAFONT statements, followed by an expression, all enclosed in the METAFONT commands `begingroup` and `endgroup`. METAFONT functions can call METAFONT functions, recursively.

The plotting macros take two or more arguments. They have an optional first argument, ⟨*spec*⟩, which determines whether a function is drawn smooth (as a METAFONT Bézier curve), or polygonal (as line segments)—if ⟨*spec*⟩ is `s` , the function will be smooth; if ⟨*spec*⟩ is `p`, the function will be polygonal; the default ⟨*spec*⟩ depends on the purpose of the macro.

One compulsory argument contains three values ⟨*min*⟩, ⟨*max*⟩ and ⟨*step*⟩. The independent variable of a function starts at the value ⟨*min*⟩ and steps by ⟨*step*⟩ until reaching ⟨*max*⟩.

There are one or more subsequent arguments, denoted ⟨*fcn*⟩, each of which is a METAFONT function.

`\function[`⟨*spec*⟩`]{`⟨*xmin*⟩`,`⟨*xmax*⟩`,`⟨*step*⟩`}{`⟨*fcn*⟩`}`

Plots ⟨*fcn*⟩, a METAFONT numeric function of one numeric argument. The default value for the optional parameter ⟨*spec*⟩ is `s`.

`\parafcn[`⟨*spec*⟩`]{`⟨*tmin*⟩`,`⟨*tmax*⟩`,`⟨*step*⟩`}{`⟨*pfcn*⟩`}`

Plots the parametric path determined by $(x(t), y(t))$ =⟨*pfcn(t)*⟩, where ⟨*pfcn*⟩ is a METAFONT function of one numeric argument, returning a METAFONT *pair* (such as (x, y)). The default value for the optional parameter ⟨*spec*⟩ is `s`.

`\plrfcn[`⟨*θmin*⟩...

`\plrfcn[`⟨*spec*⟩`]{`⟨*θmin*⟩`,`⟨*θmax*⟩`,`⟨*θstep*⟩`}{`⟨*fcn*⟩`}`

Plots the polar function determined by r =⟨*fcn*⟩(θ), where ⟨*fcn*⟩ is a METAFONT numeric function of one numeric argument, and θ varies from ⟨*θmin*⟩ to ⟨*θmax*⟩ in steps of ⟨*θstep*⟩. Each θ value is interpreted as an angle measured in *degrees*. The default value for the optional parameter ⟨*spec*⟩ is `s`.

`\btwnfcn[`⟨*spec*⟩`]{`⟨*xmin*⟩`,`⟨*xmax*⟩`,`⟨*step*⟩`}{`⟨*fcn0*⟩`}{`⟨*fcn1*⟩`}`

Draws the region between the two METAFONT functions ⟨*fcn0*⟩ and ⟨*fcn1*⟩, these being numeric functions of one numeric argument. The region is bounded also by the vertical lines at ⟨*xmin*⟩ and ⟨*xmax*⟩. Unlike the previous function macros, the default value for ⟨*spec*⟩ is `p` —this macro is intended to be used for shading between functions, a task for which smoothness is usually unnecessary.

`\plrregion[`⟨*spec*⟩`]{`⟨*θmin*⟩`,`⟨*θmax*⟩`,`⟨*θstep*⟩`}{`⟨*fcn*⟩`}`

Plots the polar region determined by r =⟨*fcn*⟩(θ), where ⟨*fcn*⟩ is a METAFONT numeric function of one numeric argument. The θ values are angles (measured in *degrees*), varying from ⟨*θmin*⟩ to ⟨*θmax*⟩ in steps of ⟨*θstep*⟩. The region is also bounded by the angles ⟨*θmin*⟩ and ⟨*θmax*⟩, i.e. by the line segments joining the origin to the endpoints of the function. The default value for ⟨*spec*⟩ is `p` —this macro is intended to be used for shading the region, a task for which smoothness is usually unnecessary.

• *Labels and Captions.*

The next two macros do not affect the METAFONT file (⟨*font*⟩`.mf`) at all, but are added to the picture by TEX. Therefore, if these macros are changed or added in your document, there is no need to repeat either of the last two steps (processing with METAFONT or reprocessing with TEX) in order to complete your TEX document.

`\tlabel[`⟨*trans*⟩`](`⟨*x*⟩`,`⟨*y*⟩`){`⟨*label text*⟩`}`

Places a TEX label on the graph. (Not to be confused with LaTeX's `\label` command.) Without the `[`⟨*trans*⟩`]` parameter, the tlabel is placed with the lower left-hand corner of the tlabel at the point `(`⟨*x*⟩`,`⟨*y*⟩`)`.

Note the different format of the point specification from the other `mfpic` macros—this is because `\tlabel` does not use METAFONT to place the tlabel, but instead uses TEX `\kern` commands.

The optional parameter `[`⟨*trans*⟩`]` specifies the relative placement of the tlabel with respect to the point `(`⟨*x*⟩`,`⟨*y*⟩`)` — ⟨*trans*⟩ is a two-character sequence where the first character is one of `t` (top), `c` (center), or `b` (bottom), to specify vertical placement, and the second character is one of `l` (left), `c` (center), or `r` (right), to specify horizontal placement. The default translation is equivalent to specifying `[bl]`.

If the tlabel goes beyond the bounds of the graph in any direction, the box containing the graph is expanded to make room for the tlabel.

`\tcaption[`⟨*maxwd*⟩`,`⟨*linewd*⟩`]{`⟨*caption text*⟩`}`

Places a TEX caption at the bottom of the graph. (Not to be confused with LaTeX's similar `\caption` command.) The macro will automatically break lines which are too much wider than the graph —if the tcaption line exceeds ⟨*maxwd*⟩ times the width of the graph, then lines will be broken to form lines at most ⟨*linewd*⟩ times the width of the graph. The default settings for ⟨*maxwd*⟩ and ⟨*linewd*⟩ are 1.2 and 1.0, respectively.

If the tcaption and graph have different widths, the two are centered relative to each other. If the tcaption takes multiple lines, then the lines are both left- and right-justified (except for the last line), but the first line is not indented.

In a tcaption, Explicit line breaks may be specified by using the `\\` command.

• *Saving an* `mfpic` *picture.*

Summary: `\savepic\foo` causes the next `\mfpic` picture to be saved in the box `\b\foo`. (The `\savepic` command should not be issued before the previous `\mfpic` picture ends!) The picture is placed by`\foo`; this also empties the box. To place the same picture twice, copy it with `\copypic\foo`.

`\savepic`⟨*\foo*⟩

Allocates a box `\b\foo` (the second `\` is part of the name) and defines `\foo` to expand to `\box\b\foo`.

`\copypic`⟨*bsl foo*⟩

Effects a `\copy\b\foo`, to copy the `mfpic` picture that's been saved in the box `\b\foo` by a `\savepic`⟨*\foo*⟩ command.

• *Parameters.*

There are many parameters in `mfpic` which the user can modify to obtain different effects, such as different arrowhead size or shape. Most of these parameters have been described already in the context of macros they modify, but they are all described together here.

Most of the parameters are stored by TEX as dimensions, and so are available globally, even if there is no METAFONT file open; changes to them are subject to the usual TEX rules of scope. Some parameters, however, are stored by METAFONT, so the macros to change them will have no effect unless a METAFONT file is open, and the changes are subject to METAFONT's rules of scope—to the `mfpic` user, this means that changes inside the `\mfpic` ... `\endmfpic` environment are local to that environment, but other TEX groupings have no effect on scope.

`\mfpicunit`
This TEX dimension stores the basic unit length for `mfpic` pictures—the x and y scales in the `\mfpic` macro are multiples of this unit. The default value is 1 point.

`\pointsize`
This TEX dimension stores the diameter of the circle drawn by the `\point` macro. The default value is 2 points.

`\pointfilled`
This TEX boolean value determines whether the circle drawn by `\point` will be filled (if `true`) or open (outline drawn, background erased). The default value is `true`.

`\pen{`⟨*drawpensize*⟩`}`
Establishes the width of the normal drawing pen. The default, at the start of each mfpic environment, is 0.5 points. The ⟨*drawpensize*⟩ is stored by METAFONT. The shading dots and hatching pen are unaffected by this.

`\shadewd{`⟨*dotdiam*⟩`}`
Sets the diameter of the dots used in the shading macro. The drawing and hatching pens are unaffected by this.

`\hatchwd{`⟨*hatchpensize*⟩`}`
Sets the line thickness used in the hatching macros. The drawing pen and shading dots are unaffected by this.

`\headlen`
This TEX dimension stores the length of the arrowhead drawn by the `\arrow` macro. The default value is 3 points.

`\axisheadlen`
This TEX dimension stores the length of the arrowhead drawn by the `\axes` macro. The default value is 5 points.

`\headshape{`⟨*hdwdr*⟩`}{`⟨*hdten*⟩`}{`⟨*hfilled*⟩`}`
Establishes the shape of the arrowhead drawn by the `\arrow` and `\axes` macros. The value of ⟨*hdwdr*⟩ is the ratio of the width of the arrowhead to its length; ⟨*hdten*⟩ is the tension of the Bézier curves; and ⟨*hfilled*⟩ is a METAFONT boolean value indicating whether the arrowheads are to be filled (if `true`) or open. The default values are 1, 1, `false`, respectively. The ⟨*hdwdr*⟩, ⟨*hdten*⟩ and ⟨*hfilled*⟩ values are stored by METAFONT. Setting ⟨*hdten*⟩ to "`infinity`" will make the sides of the arrowheads straight lines.

`\dashlen`, `\dashspace`
These TEX dimensions store, respectively, the length of dashes and the length of spaces between dashes, for lines drawn by the `\dotted` macro. The `\dotted` macro may adjust

the space between the dashes by as much as $\frac{dashspace}{n}$, where n is the number of spaces appearing in the line segment, in order not to have partial dashes at the ends. The default values are both 4 points.

`\dashlineset`, `\dotlineset`

These macros provide convenient standard settings for the `\dashlen` and `\dashspace` dimensions. The macro `\dashlineset` sets both values to 4 points; the macro `\dotlineset` sets `\dashlen` to 1 point and `\dashspace` to 2 points.

`\hashlen`

This TEX dimension stores the length of the axis hash marks drawn by the `\xmarks` and `\ymarks` macros. The default value is 4 points.

`\shadespace`

This TEX dimension establishes the spacing between dots drawn by the `\shade` macro. The default value is 1 point.

`\darkershade`, `\lightershade`

These macros both multiply the `\shadespace` dimension by constant factors, $\frac{5}{6}$ and $\frac{6}{5}$ respectively, to provide convenient standard settings for several levels of shading.

`\hatchspace`

This TEX dimension establishes the spacing between lines drawn by the `\hatch` macro. The default value is 3 points.

• *For Power Users Only.*

`\mfsrc{`⟨*metafont code*⟩`}`

Writes the ⟨*metafont code*⟩ directly to the METAFONT file, using a TEX `\write` command. This can have some rather bizarre consequences, though, so using it is not recommended to the unwary.

`\noship`

This modifier macro turns off character shipping (by METAFONT to the TFM and GF files) for the duration of the innermost enclosing group (eg, for the `mfpic` environment). This is useful if all one wishes to do in the current `mfpic` environment is to make *tiles* (see below).

`\store{`⟨*path variable*⟩`}{`⟨*path*⟩`}`

Store the following ⟨*path*⟩ in the specified METAFONT ⟨*path variable*⟩ (any valid METAFONT variable name will do) for later processing by the `\mfobj` macro.

`\mfobj{`⟨*path expression*⟩`}`

Use the METAFONT ⟨*path expression*⟩ as a path.

Examples of use of `\store` and `\mfobj`:

```
\store{f}{\circle{...}}
\dotted\mfobj{f}
\hatch\mfobj{f}
\store{f}{\curve{...}}
```

```
\store{g}{\curve{...}}
\store{h}{\mfobj{f..g..cycle}}
\dotted\mfobj{f}
\dotted\mfobj{g}
\shade\mfobj{h}
```

`\tile{`⟨*tilename*⟩`,`⟨*unit*⟩`,(`⟨*wd*⟩`,`⟨*ht*⟩`),`⟨*clip*⟩`}...\endtile`

In this environment, all drawing commands contribute to a *tile*. A *tile* is a rectangular picture which may be used to fill the interior of closed paths. The units of drawing are given by ⟨*unit*⟩, the tile's horizontal dimensions are 0 to ⟨*wd*⟩, its vertical dimensions 0 to ⟨*ht*⟩, and if ⟨*clip*⟩ is `true` then all drawing is clipped to be within the tile's boundary.

By using this macro, you can design your own fill patterns (to use them, see the `\tess` macro below), but please take some care with the æsthetics!

`\tess{`⟨*tilename*⟩`}...`

Tile the interior of each closed path with a tesselation comprised of *tiles* of the type specified by ⟨*tilename*⟩. There is *no* default ⟨*tilename*⟩; you must make all your own tiles. Tiling an open curve is technically an error, but the METAFONT code responds by drawing the path and not doing any tiling.

`\mftitle{`⟨*title*⟩`}`

Write the string ⟨*title*⟩ to the METAFONT file, and use it as a METAFONT message. (See `The` METAFONT`book`, chapter 22 `Strings`, page 187, for two uses of this.)

`\tmtitle{`⟨*title*⟩`}`

Write the text ⟨*title*⟩ to the TEX document, and to the log file, and use it implicitly in `\mftitle`.

`\newfdim{`⟨*fdim*⟩`}`

Create a new global font dimension, named ⟨*fdim*⟩, which can be used almost like an ordinary TEX dimension. The exception is that the TEX commands `\advance`, `\multiply` and `\divide` cannot be applied directly to font dimensions; however, the font dimension can be copied to a temporary TEX dimension register, which can then be manipulated and copied back. Also beware that `\newfdim` uses font dimensions from a single font, the `dummy` font, which most TEX systems ought to have. (You'll know if yours doesn't, because `mfpic` will fail upon loading!) Also, implementations of TEX differ in the number of font dimensions allowed per font. Hopefully, `mfpic` won't exceed your local TEX's limit.

• *ACKNOWLEDGEMENTS.*

Tom would like to thank all of the people at Dartmouth as well as out in the network world for testing `mfpic` and sending him back comments. He would particularly like to thank:

Geoffrey Tobin (G.Tobin@latrobe.edu.au) for his many suggestions, especially about cleaning up the METAFONT code, enforcing dimensions, fixing the dotted line computations, and speeding up the shading routines (through this process, Geoffrey and Tom managed to teach each other many of the subtleties of METAFONT), and for keeping track of `mfpic` for nearly a year while Tom finished his thesis;

Bryan Green (bgreen@sanjuan.uvic.ca) for his many suggestions, some of which (including his rewriting the `\tcaption` macro) ultimately led to the current version's ability to put graphs in-line or side-by-side; and

Uwe Bonnes (bon@lte.e-technik.uni-erlangen.de) and Jaromir Kuben (vabo@muni.cz), who worked out re-writes of `mfpic` during Tom's working hiatus and who each contributed several valuable ideas.

Some credit also belongs to Anthony Stark (ajs@merck.com), whose work on a FIG to METAFONT converter has had a serious impact on the development of many of `mfpic`'s capabilities.

Finally, Tom would like to thank Alan Vlach, the other TEXnician at Berry College, for helping him decide on the format of many of the macros, and for helping with testing.

- *CHANGES HISTORY.*

See the file `CHANGES.tex` for the history of changes to `mfpic`.