feynMF: Drawing Feynman Diagrams with LaTeX and METAFONT*

Thorsten Ohl†

Technische Hochschule Darmstadt
Schloßgartenstr. 9
D-64289 Darmstadt
Germany

November 6, 2021

Abstract

feynMF is a LaTeX package for easy drawing of professional quality Feynman diagrams with METAFONT (or METAPOST). feynMF lays out most diagrams satisfactorily from the structure of the graph without any need for manual intervention. Nevertheless all the power of METAFONT (or METAPOST) is available for obscure cases.

Copying

*This is `feynmf.dtx`, version v1.08, revision 1.30, date 1996/12/02.
†e-mail: `Thorsten.Ohl@Physik.TH-Darmstadt.de`

Contents

1 Introduction

1.1 Purpose and scope

In recent years, TEX[1] [1] and LATEX[2] [2] have revolutionized the way we share information in theoretical physics (and other areas). Not only does TEX provide typographical capabilities, which transcend those of commercial "wordprocessors" substantially, TEX documents are also completely portable. Since implementations are available on essentially all computers in use in the community, documents can be shared without the usual restrictions of proprietary data formats. This has enabled us to collaborate on papers with colleagues on the other side of the globe, to replace the mailing of hard copy preprints by electronic transmission and to submit these papers electronically to the publisher.
This portability comes with a price, though. TEX (and LATEX) do not address the issue of graphical information, apart from the rudimentary (but very useful) capabilities of the LATEX `picture` environment and similar packages [3]. As an de facto standard for the inclusion of more complex graphics has emerged the inclusion of PostScript[3] files. The complete document can then be printed on any PostScript device.
Still there are areas, where complementary approaches seem worth pursuing. In particular this is the case, if the graphical information is highly formalized, like the case at hand. Feynman diagrams are specified by their topology and the type of particles connecting the vertices. Thus a given diagram can be reproduced from a very concise specification, if the software is able to choose a reasonable layout (semi-)automatically.
METAFONT[4] [4] and METAPOST[5] [5] appear to be the perfect tool for such a purpose, since

1. METAFONT is part of any (reasonable) TEX installation, thus available to all potential users,

2. both have very powerful graphics primitivs, which allow high quality output, and

3. both have builtin linear algebra, which allows us to choose a layout automatically.

Still, providing at least the basic interface in LATEX macros seems appropriate for boosting the acceptance among the less technically oriented parts of the audience. Thus `feynMF`[6] [6, 7] was conceived.
`feynMF` is unique among packages for drawing Feynman diagrams in *combining* the following features:

- Simplicity and conciseness for common diagrams. E.g. the scattering diagram in figure 1 can be specified *completely* in five lines of LATEX:

[1] TEX is a trademark of the American Mathematical Society.
[2] LATEX might be a trademark of Addison Wesley Publishing Company.
[3] PostScript is a trademark of Adobe Systems Inc.
[4] METAFONT is a trademark of Addison Wesley Publishing Company.
[5] John Hobby's METAPOST is a modified version of METAFONT which generates (encapsulated) PostScript output. METAPOST can be build trivially on top of the `web2c` version of TEX and METAFONT for UNIX. Ports to other systems should be simple.
[6] `feynMF` is not anybody's trademark.

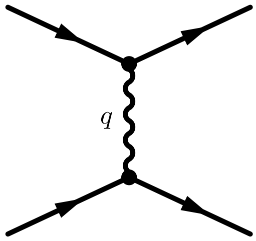


Figure 1: Simple scattering diagram.

```
\begin{fmfgraph*}(40,30) \fmfpen{thick}
  \fmfleft{i1,i2} \fmfright{o1,o2}
  \fmf{fermion}{i1,v1,o1} \fmf{fermion}{i2,v2,o2}
  \fmf{photon,label=$q$}{v1,v2} \fmfdot{v1,v2}
\end{fmfgraph*}
```

- Expressiveness for complicated diagrams (see the examples below).
- Extensibility (e.g. see section 2.8.1).
- Arbitrary TEX-labels. This point is more important than it may seem at first glance because most graphical layout systems lack the power to produce complicated mathematical expressions. Having matching fonts in text, equations and diagrams is also an important esthetical feature.

1.2 Relation to similar packages

Before we start, a couple of words about some complementary packages on the market are in order:

- Michael Levine's `feynman` package [8] is implemented on top of the standard LATEX `picture` environment [2]. This makes it completely portable (no need for a correct METAFONT installation), but it requires manual layout and the graphics output is (though very useful) less than perfect.
- Jos Vermaseren's `axodraw` package [9] uses `\special`s to access PostScript primitives for drawing diagrams. This approach is inherently not portable (the ubiquity of PostScript printers makes this a minor point, though) but as flexible as the present one. Nevertheless, it still requires manual layout for all diagrams.
- Last, but not least, I have to mention Thomas Leathrum's `mfpic` [10], which provided the inspiration for moving `feynMF`'s user interface from METAFONT to TEX. It might not have been unreasonable to design and implement `feynMF` as a package on top of `mfpic`, but closer inspection shows that `feynMF` and `mfpic` are fairly orthogonal. `mfpic` is most useful for handling simple graphical building blocks in formally unconstrained contexts. `feynMF` on the other hand excels in the formal context of Feynman diagrams (or any other labeled graphs for that matter), which can be drawn automatically from a *specification*.

1.3 Historical note

Parts of this code have a rather long history[7]. Some of the drawing primitives started in 1989 as `feynman.mf`, a library of METAFONT macros for drawing Feynman diagrams in my thesis. The layout had to be specified completely by hand, which required a long edit-process-preview cycle and made `feynman.mf` awkward to use. Nevertheless, it suited my and other's neeeds and survived for five years without major modifications. Early in 1994, I became aware of Thomas Leathrum's `mfpic` [10]. This inspired me to shift the user interface from METAFONT to LaTeX, because this allows a smoother blending of the LaTeX `picture` environment with feynMF for the purpose of labeling the graphs. While doing this and after having been taught by Tim Stelzer's and Bill Long's MADGRAPH [11] that simply minimizing the length of the graph gives much better results than I had anticipated, I added the graph manipulation and layout code.

1.4 Architecture

Even though there has never been a proper design phase in the development of feynMF, a certain structure has emerged, which is depicted in figure 2. A user who is aware of this architecture should be able to use feynMF more effectively. The most crucial aspect of the architecture is the existence of two distinct modes with different fundamental datatypes:

- *vertex mode,* which deals with graphs as structures consisting of vertices and arcs and (almost) never deals with their physical locations.
- *immediate mode,* which deals with METAFONT `path`s and `pair`s (i.e. coordinates) and allows complete control over the physical locations.

It is of course possible to mix these modes in advanced applications. Commands are provided to translate vertices and arcs to `pair`s and `path`s and vice versa. Novice users with little experience in METAFONT programming should start with vertex mode to get their job done. Later, immediate mode can be used to create more and more complex diagrams. It is possible to create most diagrams that can actually be calculated in vertex mode. Immediate mode is most useful for extending line feynMF and for drawing diagrams with fancy decorations.
A word on portability: feynMF is implemented as a LaTeX package. But it should be straightforward to adapt it to other TeX macro packages because LaTeX macros have been used for convenience only and can easily be replaced or provided in a compatibility package. The LaTeX specific *environment* construct can also easily be replaced by the equivalent construct in another macro package.

1.5 Conclusion

It goes without saying that feynMF is not perfect. There might be cases where using a graphical drawing tool with a mouse can give more pleasing results in less time. But in most cases, feynMF will give satisfactory results without any fine tuning. These will be reproducible and independent from the computer it is running on.

[7] Which is a partial explanation, if not excuse, for its slightly incoherent structure.

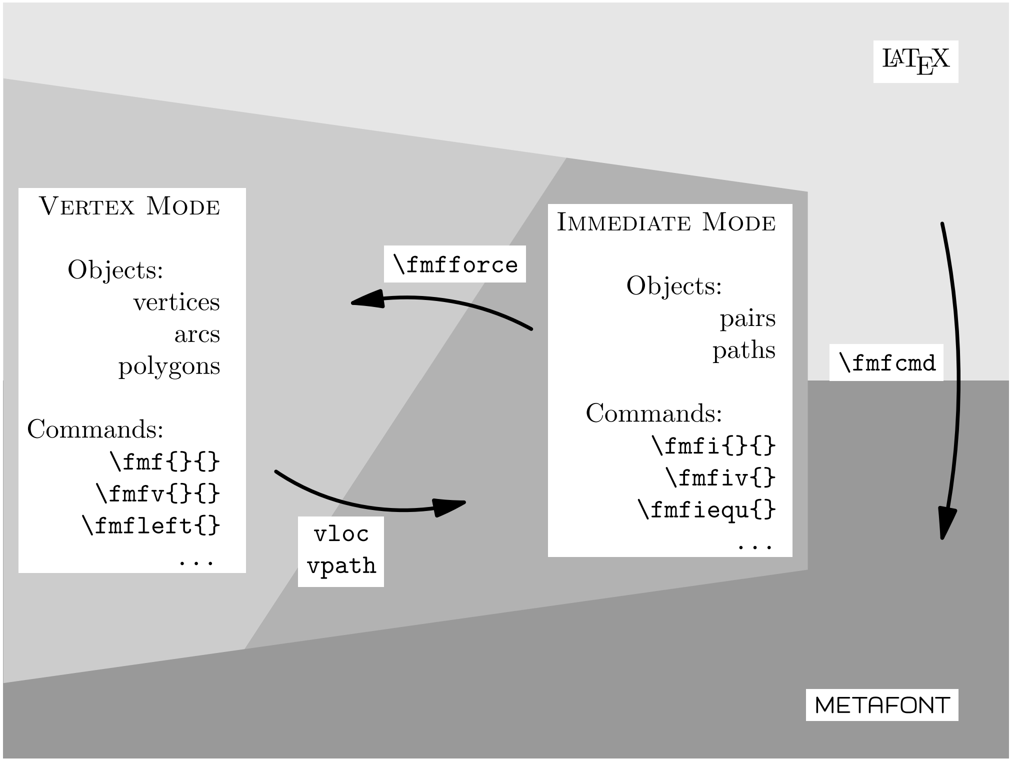


Figure 2: Architecture of `feynMF`: the two modes of `feynMF` (immediate mode and vertex mode) both interact with LaTeX and METAFONT (or METAPOST, respectively), but operate on different data types.

Early user reponses have been very encouraging. There seems to be a relatively steep learning curve for those LaTeX users that have to find out how to run METAFONT on their systems. But once this purely technical obstacle has been surmounted, users have been enthusiatic as well about the quality of the generated graphs as about the ease of use of `feynMF`.

2 Usage

In addition to this manual, there exists also a concise description of feynMF in a journal article [6], as well as a three part tutorial [7].

2.1 LaTeX package and environments

Instructing LaTeX to use feynMF is as simple as[8]

```
\usepackage{feynmf}
```

If you have METAPOST, then you can use it alternatively by placing

```
\usepackage{feynmp}
```

in your LaTeX source.[9]

feynMF has to switch interactions mode and switches to `\errorstopmode`, because it is impossible in TeX to switch back. If a different default is required (for automated preprint processing, in particular), it can be specified as a package option:

```
\usepackage[errorstop]{feynmf}
\usepackage[scroll]{feynmf}
\usepackage[batch]{feynmf}
\usepackage[nonstop]{feynmf}
```

fmffile All descriptions that should go into one METAFONT file are placed inside a `fmffile` environment which takes the name of the METAFONT file as an argument:

```
\begin{fmffile}{⟨METAFONT-file⟩}
    ...
\end{fmffile}
```

[8] As given, this applies to LaTeX. But the installation file `feynmf209.ins` allows to generate special versions `feynmf209.sty` and `feynmp209.sty` which are compatible with the obsolete LaTeX version 2.09. These files are to be used as documentstyle options

```
\documentstyle[...,feynmf209,...]{...}
```

or

```
\documentstyle[...,feynmp209,...]{...}
```

If you cannot use `epsf.sty` for including PostScript files, you can either hack `feynmp209.sty` or upgrade to LaTeX2e. Please keep in mind that feynMF has been developed for LaTeX 2_ε and the LaTeX 2.09 compatibility version will always be a retrofitted hack. I will accept bug reports for the 2.09 version, but I urge everybody to move to LaTeX2e, which is the one and only supported LaTeX right now.

[9] feynMF understands an option `pre-1.03`, that is of interest for veteran users:

```
\usepackage[pre-1.03]{feynmf}
```

or

```
\usepackage[pre-1.03]{feynmp}
```

The purpose of this option is to enable processing of old input files (pre v1.02) that use `\noexpand` in labels. Since the method for processing these files can clash (in rare cases) with LaTeX 2_ε's font loading procedure, it has been disabled by default.

Upto 255 graphs can be placed into one METAFONT file. Currently feynMF does *not* check that the 255 graph limit per file is not overrun.[10] Note that the filename for the METAFONT file given in the argument of the fmffile environment *must not* be identical to the LaTeX source file name, because the METAFONT .log would be overwritten and LaTeX can no longer access the information in this .log file. It should be obvious that any umber of diagrams can be generated by using more than one fmffile environment with different filenames.

fmfgraph The fmfgraph environment contains the description of a single Feynman diagram which will be placed a the location of the environment. Arguments are the width and the height of the diagram, in units of \unitlength:

```
\begin{fmfgraph}(⟨width⟩,⟨height⟩)
    ...
\end{fmfgraph}
```

This environment does *not* support labels, use fmfgraph* if your diagrams contains labels.

fmfgraph* Same as fmfgraph, but enclosed in a picture environment of the same size and supporting LaTeX labels.

```
\begin{fmfgraph*}(⟨width⟩,⟨height⟩)
    ...
\end{fmfgraph*}
```

\fmfframe Allows to allocate additional space around a fmfgraph*, since the labels (or the diagram itself) might overshoot:

```
\fmfframe(⟨left⟩,⟨top⟩)(⟨right⟩,⟨bottom⟩){⟨box⟩}
```

puts an invisible frame of the given dimensions (measured in \unitlength) around ⟨*box*⟩.

2.2 Auxiliary files

feynMF needs to share information between METAFONT and LaTeX. For this task several auxiliary files are needed. The flow of information depicted in figures 3 and 4 looks much more complicated than it is. The important feature is that there a two sets of files which can be used to distribute a document:

1. Iff the recipient has a working METAFONT installation (which shouldn't be a problem, except for some impoverished commercial implementations), the document can be typset from the LaTeX source *alone*, by running LaTeX, METAFONT and LaTeX again (the latter step might have to be repeated to get cross references right).

2. Another possibility (which doesn't require METAFONT on the recipient's side), is to distribute the LaTeX source, the tfm and gf files (or pk files respectively) along with the label files with extension tn (where n as an

[10] There is also a very primitive, but (unfortunately) popular operating system, which restricts filenames to eight characters with a three character extension. On this system, only 99 graphs can be placed into one METAFONT file because auxiliary files will not be unambiguous, if more than two digits are used.

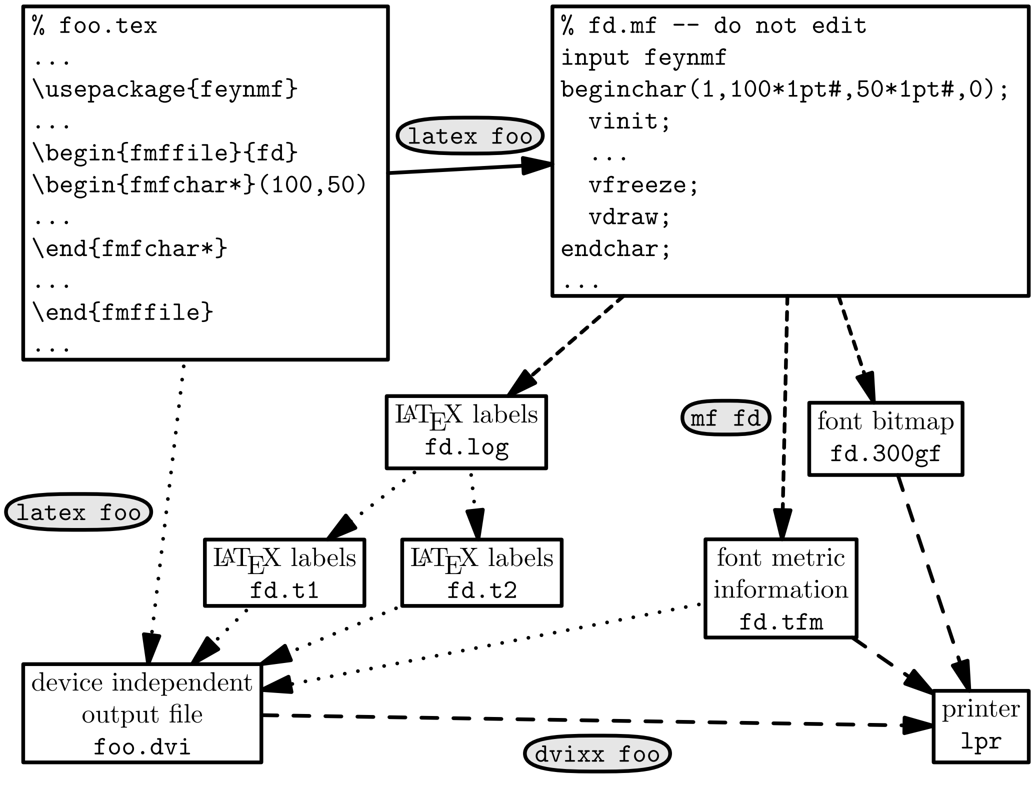


Figure 3: Interdependency of files in a `feynMF` application. The arrows show which files are updated in the two LaTeX passes, the METAFONT pass and the final dvi translation step.

integer). Distributing the METAFONT `log` files is a possible alternative for the latter, but discouraged, because these are prone to be erased accidentally.

2.3 Running METAFONT

Processing your document with LaTeX will generate one or more METAFONT files, which you will have to process with METAFONT. On UNIX[11] systems, METAFONT is invoked as

`mf '\mode:=`⟨*METAFONT-mode*⟩`; input `⟨*METAFONT-file*⟩`'`

from the shell. Here ⟨*METAFONT-file*⟩ is to be replaced by the name of the input file (which is determined by the argument to the `fmffile` environment, see section 2.1) and ⟨*METAFONT-mode*⟩ is the proper METAFONT-mode for your particular printer. Please consult your local guide or local wizards about how to run METAFONT on other systems.

[11] UNIX was a trademark of UNIX Systems Laboratory, but is rumored to have been donored to X/Open.

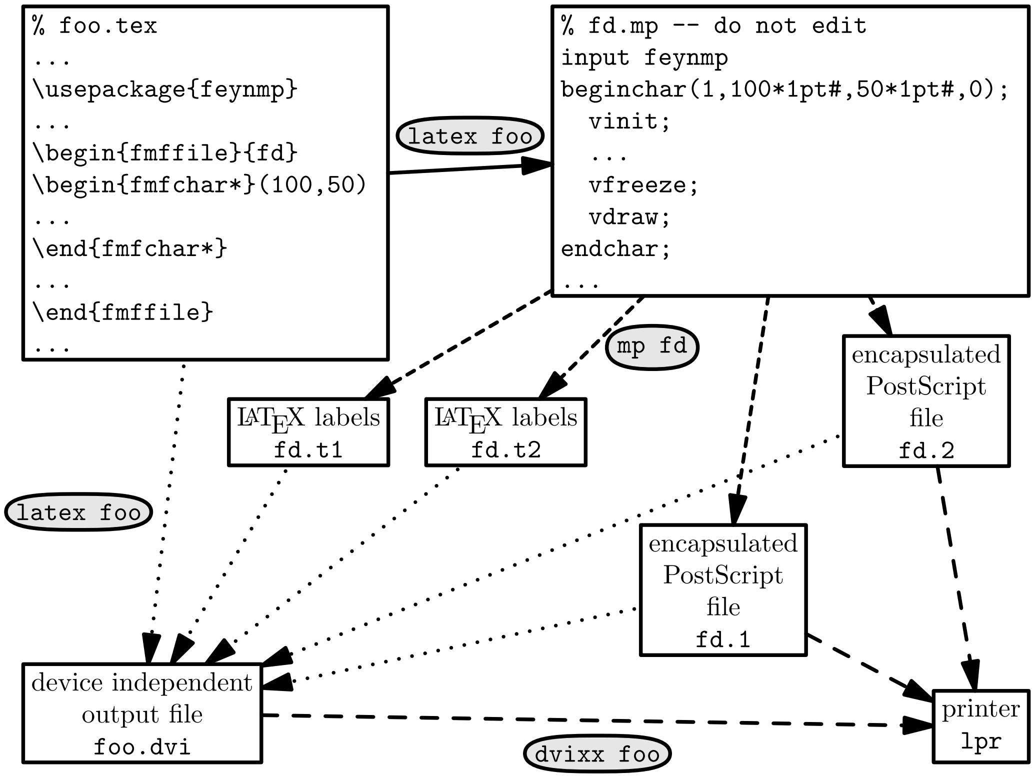


Figure 4: Interdependency of files in a feynMF application using METAPOST: The arrows show which files are updated in the two LaTeX passes, the METAPOST pass and the final dvi translation step.

Note that ⟨*METAFONT-mode*⟩ *must* be specified, otherwise METAFONT will fail or the resulting font will not be usable.[12] You can look up the correct METAFONT mode in the file `modes.mf` that comes with the METAFONT distribution. Among the more common laser printers are `laserjet` for HP Laserjets at 300dpi, `ljfour` for HP Laserjets at 600dpi, `nexthi` for NeXT laser printers at 400dpi, etc.
A non-trivial part can be instructing TeX and your favorite `dvi`-driver how to find the generated `tfm` and `gf` (resp. `pk`) files. This is highly system dependent and can be trivial (as in the standard UNIX TeX installations, where no further action is required) or almost impossible without system priviledges (as under MVS). Please consult your local guide or wizards on this point. See also section 2.9.2 for common problems with `dvi` drivers.
Some recent TeX implementations (e.g. `web2c` with `kpathsea` version 2.6 or later) are able to generate `tfm` files on the fly. Using such implementations, running LaTeX twice should suffice and METAFONT will be invoked automagically in the background. Note however, that the automagically invoked tools might also

[12]See section 2.9.1 for the typical error message and for additional information on printer modes.

install the “fonts” corresponding to the Feynman diagrams in a system directory, where they don’t belong. Adding the following lines to the `maketex.site` script will prevent this mishap in the teTEX distribution for UNIX (which is derived from `web2c`):

```
if [ -r $KPSE_DOT/$NAME.mf ]; then
  MT_PKDESTDIR=$KPSE_DOT
  MT_TFMDESTDIR=$KPSE_DOT
  MT_NAMEPART=
fi
```

The automagic tools will also *not* notice when a diagram has changed. These problems suggest that it is a generally a good idea to invoke METAFONT explicitely, instead of relying on the automagic tools.
Running METAPOST is usually trivial, because not printer specific mode is needed:

`mp` ⟨*METAPOST-file*⟩

2.4 The `feynmf` perl script

UNIX users will be able to take advantage of the `feynmf` perl script, that automates the invocation of LATEX and METAPOST. In particular it tries to guess the correct METAFONT-mode and magnification. The latter is often different from 1 in slide classes. Here is the man page of `feynmf`:

NAME

feynmf — Process **LaTeX** files using **FeynMF**

SYNOPSIS

feynmf [**-hvqncfT**] [**-t** *tfm* [**-t** *tfm* ...]] [**-m** *mode*] *file* [*file* ...]
feynmf [**-help**] [**-version**] [**-quiet**] [**-noexec**] [**-clean**] [**-force**] [**-notfm**] [**-tfm** *tfm* [**-tfm** *tfm* ...]] [**-mode** *mode*] *file* [*file* ...]

DESCRIPTION

The most complicated part of using the **FeynMF** style appears to be the proper invocation of **Metafont**. The **feynmf** script provides a convenient front end and will automagically invoke **Metafont** with the proper mode and magnification. It will also avoid cluttering system font directories and offers an option to clean them.

OPTIONS

-h, -help
Print a short help text.

-v, -version
Print the version of **feynmf**.

-q, -quiet
: Don't echo the commands being executed.

-n, -noexec
: Don't execute **LaTeX** or **Metafont**.

-c, -clean
: Offer to delete font files that have accidentally been placed in a system directory by the **MakeTeXTFM** and **MakeTeXPK** scripts (these scripts are run by **tex** (and **latex**) in the background). This option has only been tested with recent versions of UNIX TeX.

-f, -force
: Don't ask any questions.

-T, -notfm
: Don't try to prepare fake `.tfm` files for the first run.

-t, -tfm *tfm*
: Don't try guess the names of the `.tfm` files to fake for the first run and use the given *name*(s) instead. This option can be useful if our incomplete parsing of the LaTeX input files fails.

-m *mode*, -mode *mode*
: Select the METAFONT mode *mode*. The default is guessed or `localfont` if the guess fails.

file
: Main **LaTeX** input files.

***file* ...**
: Other LaTeX input files that are included by the main file.

AUTHOR

Thorsten Ohl <Thorsten.Ohl@Physik.TH-Darmstadt.de>

BUGS

The preparation of `.tfm` files is not foolproof yet, because we can parse **TeX** files only superficially.
This script has only been tested for recent **teTeX** distributions of UNIX **TeX**, though it will probably work with other versions of UNIX **TeX**. The author will be grateful for portability suggestions, even concerning **Borg** operating systems, for the benefit of those users that are forced to live with DOS or Windows.

2.5 Vertex mode

These basic features of `feynMF` are (or rather "should be") available through the LaTeX interface. No knowledge of METAFONT is necessary.

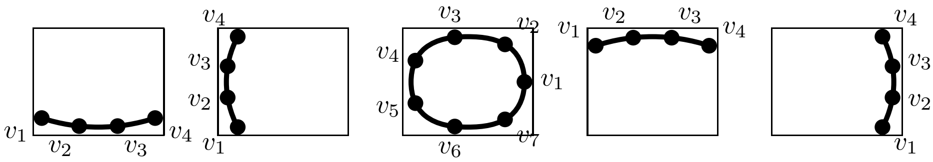


Figure 5: Curved galleries.

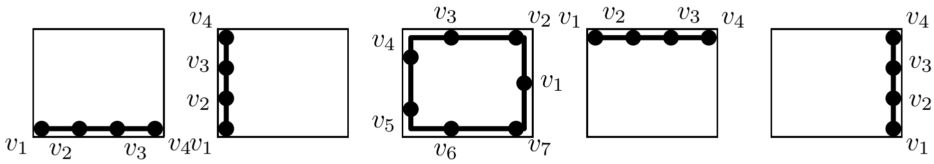


Figure 6: Straight galleries.

2.5.1 External vertices

`\fmfleft` `\fmfright` `\fmfbottom` `\fmftop` `\fmfsurround`

Positioning of external vertices has to done explicitely. The technical reason is that they would otherwise collapse with their neighbors, but practical reasons also suggest to give the user full control here. `\fmfleft{`⟨*v1*⟩`[,`...`]}` places the vertices in the comma separated list ⟨*v1*⟩,... equidistantly on a smooth path on the left side of the diagram. `\fmfright{`⟨*v1*⟩`[,`...`]}` does the same thing on the right. Similarly `\fmfbottom` and `\fmftop`, while `\fmfsurround{`⟨*v1*⟩`[,`...`]}` places its arguments on smooth path surrounding the diagram.

`\fmfcurved` `\fmfstraight`

Per default, the *galleries* on which we place the external vertices are curved as in figure 5, but straight galleries are also available. The macros `\fmfcurved` and `\fmfstraight` switch between these alternatives.

`\fmfleftn` `\fmfrightn` `\fmfbottomn` `\fmftopn` `\fmfsurroundn`

The macro `\fmfleftn` is similar to `\fmfleft`, but `\fmfleftn{`⟨*v*⟩`}{`⟨*n*⟩`}` places the vertices ⟨*v*[*1*]⟩...⟨*v*[*n*]⟩. Analogously for the macros `\fmfrightn`, `\fmfbottomn`, `\fmftopn` and `\fmfsurroundn`.

2.5.2 Arcs and internal vertices

`\fmf`

This is the the most frequently used macro in `feynMF` applications.

`\fmf{`⟨*style*⟩`[,`⟨*opt*⟩`[=`⟨*val*⟩`],`...`]}{`⟨*v1*⟩`,`⟨*v2*⟩`[,`...`]}`

connects the vertices *v1,v2,...* with a line of style ⟨*style*⟩, using a set of options ⟨*opt*⟩ with (optional) value ⟨*val*⟩. If a vertex is not known yet, it is added to the diagram. Note that the actual drawing is not done immediately, because the positions can only be calculated when *all* vertices are known. The currently available styles are collected in table 1. Most names should be self explanatory and are not discussed further. The `dashes`, `dots`, `phantom` and `plain` styles can optionally be decorated with an arrow as shown above. All styles, including `curly`, `wiggly` and `zigzag`, can be doubled. But arrows are not available for the latter three, because esthetically pleasing results can not be expected. The `phantom` style is special, because it only enters the vertices and does *not* cause a line to be drawn. This is extremely useful for advanced layout features, as explained below. If you need a line styles that is not listed in table 1, see section 2.8.1 for how to define your own line styles.

Name	Example	Parameters	Aliases
curly		curly_len	gluon
dbl_curly		curly_len	
dashes		dash_len	
dashes_arrow		dash_len	scalar
dbl_dashes		dash_len	
dbl_dashes_arrow		dash_len	
dots		dot_len	
dots_arrow		dot_len	ghost
dbl_dots		dot_len	
dbl_dots_arrow		dot_len	
phantom			
phantom_arrow			
plain			vanilla
plain_arrow			fermion, electron, quark
dbl_plain			double
dbl_plain_arrow			double_arrow, heavy
wiggly		wiggly_len	boson, photon
dbl_wiggly		wiggly_len	
zigzag		zigzag_width	
dbl_zigzag		zigzag_len	

Table 1: Available line styles

The supported options are collected in table 2[13] Note that each of the dot separated components of the options can be abbreviated. For example, `l.d` is equivalent to `label.dist`. The result of ambiguous matches is however undefined.

[13]One particulary useful further option would be `smooth`, allowing for several lines joined smoothly. Early experimentation has shown however, that the results are not always what one expects and that there is a lot of room for abuse.

Name	Explanation
`tension`	draw a tighter (> 1) or more loose (< 1) arc
`left`	draw on a halfcircle on the left
`right`	draw on a halfcircle on the right
`straight`	draw on a straight line (default)
`label`	TEX text for labeling the arc
`label.side`	force placement of the label on the `left` or `right`
`label.dist`	place label at a distance `dist`
`label.pos`	relative position of the label (not implemented yet!)
`tag`	optional tag for disambiguating arcs
`width`	width of the line
`rubout`	scale factor for crossing out lines (doesn't work properly for doubled lines yet)
`foreground`	foreground color (METAPOST only!)
`background`	background color for doubled lines (METAPOST only!)

Table 2: Available line options

Note that because the options are separated by single commata, commata inside arguments to options (`label` comes to mind) have to be doubled (similar to quotes in Fortran).[14]
Arcs that return to their origin are allowed (I will refer to them as *tadpoles*), but some options have slightly different semantics. `tension` is here a inverse scale factor for the tadpole, whose default size is 2/3 of the average distance the neighboring vertices. If `left` or `right` are specified, they give the direction (in degrees) of the preferred gap into which the tadpole is placed. By default, the largest gap is chosen for *all* tapoles at a given vertex, which will therefore overlap. This is neither a bug nor a feature, but a limitation.

`\fmfn` The macro `\fmfn` is similar to `\fmf`, but

`\fmfn{`⟨*style*⟩[`,`⟨*opt*⟩[`=`⟨*val*⟩]`,`...]`}{`⟨*v*⟩`}{`⟨*n*⟩`}`

connects the vertices ⟨*v*[*1*]⟩...⟨*v*[*n*]⟩.

`\fmfcyclen` `\fmfrcyclen` The macro `\fmfcyclen{`⟨*style*⟩`}{`⟨*v*⟩`}{`⟨*n*⟩`}` cyclically connects the vertices ⟨*v*[*1*]⟩...⟨*v*[*n*]⟩. `\fmfrcyclen` operates in reverse order.

`\fmfpen` Pick up a pen of the specified size. `\fmfpen{`⟨*weight*⟩`}` is used for changing the weight (i.e. thickness) of the lines. Predefined sizes are `thin` and `thick`.

`\fmfv` Declare vertices with options:

`\fmfv{`⟨*opt*⟩[`=`⟨*val*⟩][`,`⟨*opt*⟩[`=`⟨*val*⟩]`,`...]`}{`⟨*v1*⟩[`,`...]`}`

This is used for adding labels to a vertex and for specifying other decoration. Supported options are collected in table 3. Here the same abbreviation mech-

[14] Note that, as of version 1.03, it is no longer necessary to escape TEX control sequences in arguments. Old files will continue to work, because `noexpand` is temporarily disabled.

Name	Explanation
`label`	TeX text for labeling the vertex
`label.angle`	force placement of the label at the given angle from the vertex
`label.dist`	place label at a distance `dist`
`decoration.shape`	shape of decoration
`decoration.size`	size of decoration
`decoration.filled`	fill, shade or hatch decoration
`decoration.angle`	rotate decoration
`foreground`	foreground color (METAPOST only!)
`background`	background color (METAPOST only!)

Table 3: Available vertex options

anism as above is in effect. The available `shapes` are listed in various filling styles in table 4[15]. The tilings `gray10`, `gray25`, `gray75` and `gray90` are available in addition to `gray50`. Customized tilings can be created with the METAFONT function `tile_from_string`. It should be noted however, that tilings are gobbling up memory at high speed and should be used with discretion. The halftones[16] can be accessed by giving any number from 2 to 99, which will denote the percentage of saturation (30% and 70% here)[17]. Again, commata inside arguments to options have to be doubled.

`\fmfblob` Draw a blob of the specified diameter at the vertices. Incidentally,

`\fmfblob{`$\langle$*diameter*$\rangle$`}{`$\langle$*v1*$\rangle$`[,...]}`

is equivalent to

`\fmfv{decor.shape=circle,decor.filled=shaded,`
`decor.size=`$\langle$*diameter*$\rangle$`}{`$\langle$*v1*$\rangle$`[,...]}`

`\fmfdot` Draw a dot at the vertices given as arguments.

`\fmfdot{`$\langle$*v1*$\rangle$`[,...]}`

is equivalent to

`\fmfv{decor.shape=circle,decor.filled=full,`
`decor.size=2thick}{`$\langle$*v1*$\rangle$`[,...]}`

`\fmfvn` The macro `\fmfvn` is similar to `\fmfv`, but

`\fmfvn{`$\langle$*opt*$\rangle$`[=`$\langle$*val*$\rangle$`][,`$\langle$*opt*$\rangle$`[=`$\langle$*val*$\rangle$`],...]}{`$\langle$*v*$\rangle$`}{`$\langle$*n*$\rangle$`}`

places the vertices $\langle v[1]\rangle \ldots \langle v[n]\rangle$.

`\fmfdotn` `\fmfblobn` The macros `\fmfdotn` and `\fmfblobn` are similar to the `\fmfdot` and `\fmfblob`, but `\fmfdotn{`$\langle v\rangle$`}{`$\langle n\rangle$`}` places the vertices $\langle v[1]\rangle \ldots \langle v[n]\rangle$. Analogously for `\fmfblobn`.

[15] If the variable `feymfwizard` is `true` (e.g. after calling the `\fmfwizard` macro), it is also possible to specify any METAFONT expression that evaluates to a `path`. Naturally, this has to used with great care, because strange errors can be triggered by typos!

[16] METAPOST will give true halftones (if your printer supports them), while METAFONT tries to mimic them. The dithering algorithm of the latter will be improved in the future.

[17] The old numeric arguments in the range $-1\ldots1$ continue to work, but are considered obsolete.

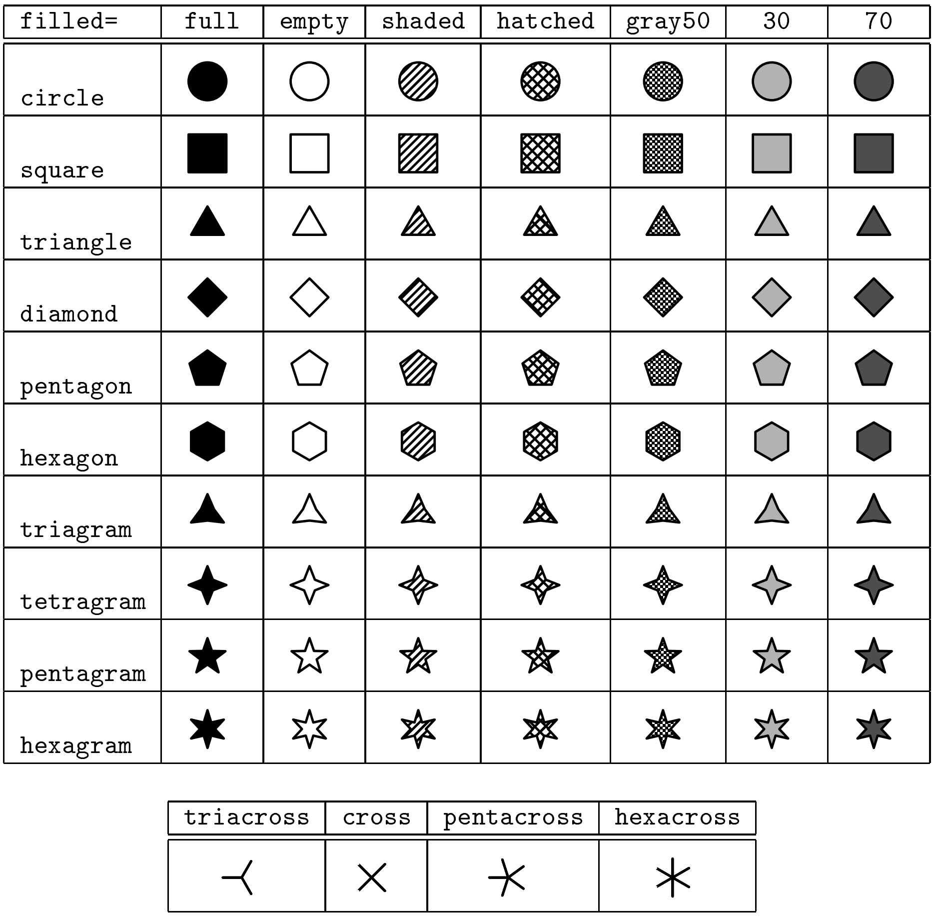


Table 4: Available vertex shapes and fill styles.

2.5.3 Polygons

Complex vertices with arcs attached at the corners can be contructed with polygons, which share some characteristics with arcs and vertices.

`\fmfpoly` The macro

$$\texttt{\textbackslash fmfpoly\{}\langle style\rangle\texttt{[,}\langle opt\rangle\texttt{[=}\langle val\rangle\texttt{],...]\}\{}\langle v1\rangle\texttt{,}\langle v2\rangle\texttt{[,...]\}}$$

places the vertices $\langle v1\rangle$, $\langle v2\rangle$, ... on the corners of a regular polygon. The orientation of the polygon is fixed to be mathematically positive (i.e. counter clockwise). Note that this can have strange results if the orientation is opposite to the orientation of the vertices the corners are connected to. The available options are collected in table 5. The effects of some options are depicted in table 6.

Note that is is technically *impossible* to fix the size of a polygon. The size can be controlled by specifying a `tension` for the edges, which acts like the `tension` of normal arcs.

`\fmfpolyn` `\fmfrpolyn` The macro `\fmfpolyn{`$\langle style\rangle$`}{`$\langle v\rangle$`}{`$\langle n\rangle$`}` is similar to `\fmfpoly` but connects the vertices $\langle v[1]\rangle \ldots \langle v[n]\rangle$. `\fmfrpolyn` operates in reverse order.

Name	Explanation
`filled`	fill, shade or hatch interior
`phantom`	don't draw anything
`empty`	draw outline
`shade`	shade interior
`hatched`	hatch interior
`full`	fill interior
`pull`	pull edges in (< 0) or out (> 0)
`tension`	tension of the edges
`smooth`	draw smoth corners
`label`	TEX text for labeling the polygon
`label.angle`	force placement of the label at the given angle from the vertex
`label.dist`	place label at a distance `dist`
`foreground`	foreground color (METAPOST only!)
`background`	background color (METAPOST only!)

Table 5: Available polygon options

2.5.4 Color

If METAPOST is used for rendering the diagrams, vertices and arcs can be colored. The corresponding options give a warning message under METAFONT and are otherwise ignored.
Two colors are available for vertices and arcs: `foreground` and `background`. Both can either be specified as a linear combination of the predefined colors `white`, `black`, `red`, `green` and `blue` or as RBG triples `(`⟨*red*⟩`,`⟨*green*⟩`,`⟨*blue*⟩`)`. Note that as always commata have to be doubled in option arguments. Therefore both `foreground=(1,,0,,1)` and `foreground=red+blue` are valid options setting the foreground color to magenta. For arcs, the background color is used only for the interior of doubled lines.
While the color feature is rarely used in papers or books, it can be very useful for jazzing up your transparencies. See page 27 for a not very serious example.

2.5.5 Examples

After the main features of the vertex mode have been introduced, it is time for a couple of illustrative examples that are taken from *The Real World*.

> As a first example, consider drawing a straightforward box diagram, familiar from K-$\bar{K}$, D-$\bar{D}$, and B-$\bar{B}$ mixing. The commands for the labels are not shown here, they are discussed in section 2.5.6
>
> Let us start the diagram and pick up a thick pen:

```
\begin{fmfgraph}(40,25)
  \fmfpen{thick}
```

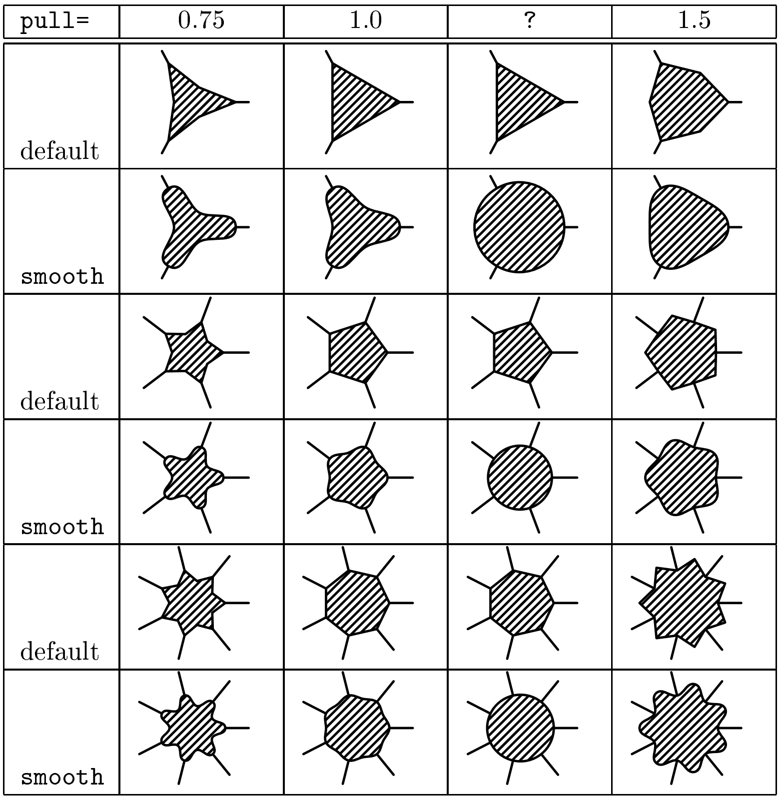
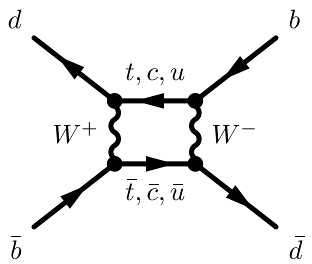


Table 6: Some of the available polygon shapes. Note that `pull=1.0` is identical to `pull=?` for straight lines, but very different for `smooth` lines.

The incoming and outcoming vertices are placed on the left and right hand side, respectively:

```
  \fmfleft{i1,i2}
  \fmfright{o1,o2}
```



Now we tell `feynMF` how the arcs are connected.

```
  \fmf{fermion}{i1,v1,v3,o1}
  \fmf{fermion}{o2,v4,v2,i2}
  \fmf{photon}{v1,v2}
  \fmf{photon}{v3,v4}
```

Finally we tell `feynMF` to draw dots at the vertices and we're done.

```
  \fmfdotn{v}{4}
\end{fmfgraph}
```

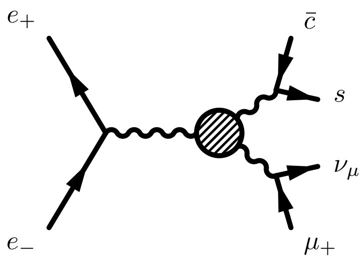
With a little effort the layout of this diagram can actually be improved by enlarging the inner box, see page 29 below.

Here is the resonant s-channel contribution to $e^+e^- \to 4f$. (From now on, we do no longer display the

```
\begin{fmfgraph}(40,25)
  \fmfpen{thick}
    ...
\end{fmfgraph}
```

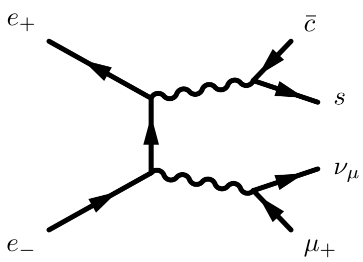
environment surrounding all pictures.)

```
\fmfleftn{i}{2}
\fmfrightn{o}{4}
  \fmf{fermion}{i1,v1,i2}
\fmf{photon}{v1,v2}
\fmfblob{.15w}{v2}
  \fmf{photon}{v2,v3}
    \fmf{fermion}{o1,v3,o2}
  \fmf{photon}{v2,v4}
    \fmf{fermion}{o4,v4,o3}
```



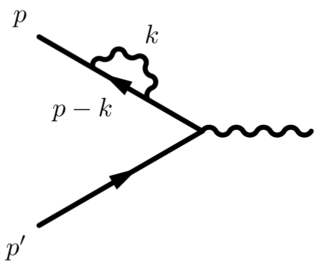
And the resonant t-channel contribution:

```
\fmfleftn{i}{2}
\fmfrightn{o}{4}
\fmf{fermion}{i1,v1,v2,i2}
  \fmf{photon}{v1,v3}
    \fmf{fermion}{o1,v3,o2}
  \fmf{photon}{v2,v4}
    \fmf{fermion}{o4,v4,o3}
```



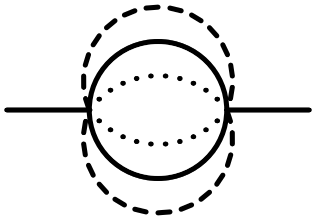
Two point loop diagrams pose another set of problems. We must have a way of specifying that one or more of the lines connecting the two vertices are *not* connected by a straight line. The options `left`, `right` and `straight` offer the possibility to connect two vertices by a semicircle detour, either on the left or on the right. Since by default all lines contribute to the tension between two vertices, the `tension` option allows us to reduce this tension. The next examples shows both options in action. The lower fermion line is given an tension of 1/3 to make is symmetrical with the upper line with consists of three parts. The loop photon is using a detour on the right and does not contribute any tension.

```
\fmfleft{i1,i2}
\fmfright{o1}
\fmf{fermion,tension=1/3}{i1,v1}
\fmf{plain}{v1,v2}
\fmf{fermion}{v2,v3}
\fmf{photon,right,tension=0}{v2,v3}
```



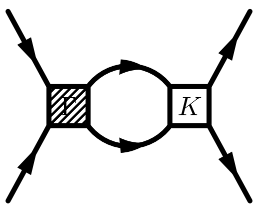

```
\fmf{plain}{v3,i2}
\fmf{photon}{v1,o1}
```

The optional argument to `left` and `right` can be used to deform the corresponding contour as in the following example. The default value of `left` and `right` is 1.



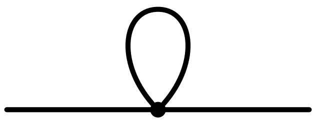
```
\fmfleft{i} \fmfright{o}
\fmf{plain}{i,v1} \fmf{plain}{v2,o}
\fmf{dots,left=.5,tension=0.3}{v1,v2,v1}\fmffreeze
\fmf{plain,left}{v1,v2,v1}
\fmf{dashes,left=1.5}{v1,v2,v1}
```

Polygons are particulary useful for depicting non-perturbative contributions:



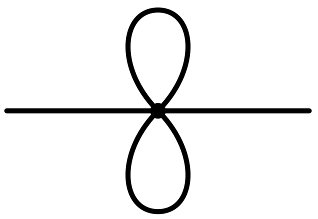

```
\fmfpen{thick}
\fmfleftn{l}{2}\fmfrightn{r}{2}
\fmfrpolyn{shaded,label=$\Gamma$}{G}{4}
\fmfpolyn{empty,label=$K$}{K}{4}
\fmf{fermion}{l1,G1}\fmf{fermion}{l2,G2}
\fmf{fermion}{K1,r1}\fmf{fermion}{K2,r2}
\fmf{fermion,left=.5,tension=.5}{G3,K3}
\fmf{fermion,right=.5,tension=.5}{G4,K4}
```

To conclude this first picture show, here's a self energy in scalar ϕ^4-theory showing the simplicity of the tadpole feature:



```
\fmfpen{thick}
\fmfleft{i}
\fmfright{o}
\fmf{plain}{i,v,v,o}
\fmfdot{v}
```

Scalar ϕ^6-theory needs a little manual intervention to force the second on the opposite side:



```
\fmfpen{thick}
\fmfleft{i}
\fmfright{o}
\fmf{plain}{i,v,v,o}
\fmf{plain,left=90}{v,v}
\fmfdot{v}
```

2.5.6 Labels

Let us now come back to the examples on page 21 and discuss how to add the labels.

`\fmflabel` The macro

`\fmflabel{`⟨*label*⟩`}{`⟨*v*⟩`}`

is equivalent to

`\fmfv{label=`⟨*label*⟩`}{`⟨*v*⟩`}`

and adds the label ⟨*label*⟩ to the vertex ⟨*v*⟩. In the current implementation, there can be only a single label for each vertex. Thus earlier calls to `\fmflabel` for the same vertex will be overwritten. ⟨*label*⟩ will be placed with the `\put` command of the LaTeX `picture` environment.[18] Note that the `fmfgraph*` environment must be used to use labels, they will silently disappear in `fmfgraph`.
`\fmflabel` gives the user *no* control on the placement of the the label (use the `\fmfv` macro for a more fine-grained control). The label is placed using the following algorithm:

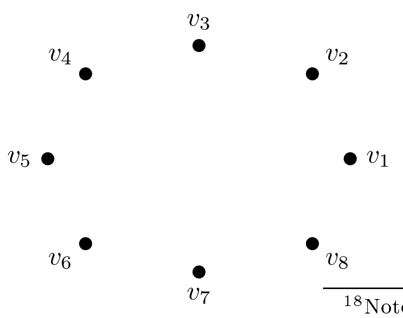
1. The reference point of the box containing ⟨*label*⟩ is placed at the distance `3thick` on the continuation of the straight line connecting the center of the picture with the vertex ⟨*v*⟩.

2. The reference point of the box is chosen such that the contents of the box is on the outside of the vertex (with respect to the center of the diagram). It is chosen from the four corners and the four midpoints of the sides.

Therefore the four external particles in the B-$\bar{B}$ mixing diagram on page 21 are labelled simply by:

```
\fmflabel{$\bar{b}$}{i1}
\fmflabel{$d$}{i2}
\fmflabel{$\bar{d}$}{o1}
\fmflabel{$b$}{o2}
```

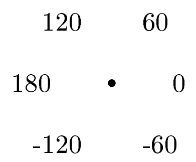
Here is a more systematical demonstration of the default placement of labels:




```
\fmfsurroundn{v}{8}\fmfdotn{v}{8}
\fmflabel{$v_1$}{v1}
\fmflabel{$v_2$}{v2}
\fmflabel{$v_3$}{v3}
\fmflabel{$v_4$}{v4}
\fmflabel{$v_5$}{v5}
\fmflabel{$v_6$}{v6}
\fmflabel{$v_7$}{v7}
\fmflabel{$v_8$}{v8}
```

[18] Note again that, as of version 1.03, it is no longer necessary to escape TeX control sequences in arguments. Old files will continue to work, because `noexpand` is temporarily disabled.

And here is a demonstration of the explicit placement of labels[19]:




```
\fmfiv{d.sh=circle,d.f=1,d.si=2thin}{c}
\fmfiv{l=-120,l.a=-120,l.d=.2w}{c}
\fmfiv{l=-60,l.a=-60,l.d=.2w}{c}
\fmfiv{l=0,l.a=0,l.d=.2w}{c}
\fmfiv{l=60,l.a=60,l.d=.2w}{c}
\fmfiv{l=120,l.a=120,l.d=.2w}{c}
\fmfiv{l=180,l.a=180,l.d=.2w}{c}
```

There is no equivalent to `\fmflabel` for arcs. Here options to the `\fmf` command have to be used. The default placement rules put the label on the outside at the midpoint of a curved arc. If the arc is straight, one should use the `label.side` option to push the label either to the `left` or to the `right`. This `label.dist` option is treated analogously to same option for vertices.
Therefore the four internal particles in the B-$\bar{B}$ mixing diagram on page 21 are labelled simply by adding options to the `\fmf` commands:

```
\fmf{fermion,label=$\bar{t},,\bar{c},,\bar{u}$,
            label.side=right}{v1,v3}
\fmf{fermion,label=$t,,c,,u$,label.side=right}{v4,v2}
\fmf{photon,label=$W^+$,label.side=left}{v1,v2}
\fmf{photon,label=$W^-$,label.side=right}{v3,v4}
```

2.5.7 Manipulating the layout

The automatic layout algorithms of `feynMF` are rather simple, therefore it is sometimes necessary to allow for manual intervention from time to time.

`\fmffreeze` Calculate the positions of the vertices based on the arcs which are defined up to this point. Usually this calculation is performed automatically at the end of the `fmfgraph` environment. Calling it explicitely is useful for later adding arcs that should not enter the calculation . The layout is chosen to minimize the overall length of all arcs. The length of each arc is weighted with the `tension` option, whose default value is 1. See section 2.5.8 for more information on `\fmffreeze`.

`\fmfforce` `\fmfforce{`⟨*pos*⟩`}{`⟨*v1*⟩[,...]`}` forces the position ⟨*pos*⟩ of the vertices ⟨*v1*⟩..., bypassing and overwriting the automatic layout. In all arguments that are METAFONT `pairs` (i.e. points), you can use the variable `w` and `h`, which are predefined to the width and the height of the whole diagram respectively. E.g. the center is specified as `(.5w,.5h)` and the lower right corner as `(w,0)`. The center and the four corners of the current subgraph (see the `fmfsubgraph` environment on page 29) are available as `c`, `nw`, `ne`, `sw` and `se` (for *north-west* etc).

`\fmfshift` `\fmfshift{`⟨*dist*⟩`}{`⟨*v1*⟩[,...]`}` shifts the position of the vertices ⟨*v1*⟩... by ⟨*dist*⟩ from the automatic layout. This command is only useful *after* a `\fmffreeze` of the corresponding vertex.

`\fmffixed` `\fmffixed{`⟨*dist*⟩`}{`⟨*v1*⟩[,...]`}` fixes the distance between subsequent vertices in the list ⟨*v1*⟩... to ⟨*dist*⟩. This command should be used with care, because

[19]Don't be confused by the `\fmfiv` command. It is described below (see section 2.7.2) and takes the same arguments as the `\fmfv` command. We use it here for convenience to place multiple vertices at the same point, i.e. the center.

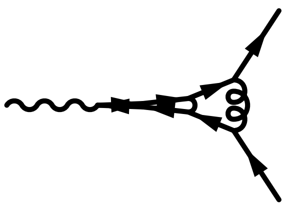
it is possible to overconstrain the layout of the graph and the error messages will be obscure for a novice user.

`\fmffixedx` `\fmffixedy` `\fmffixedx{`⟨*dx*⟩`}{`⟨*v1*⟩`[,`...`]}` is identical to `\fmffixed{(`⟨*dx*⟩`,whatever)}{`⟨*v1*⟩ `[,`...`]}` and `\fmffixedy{`⟨*dy*⟩`}{`⟨*v1*⟩`[,`...`]}` is identical to `\fmffixed{(whatever,`⟨*dy*⟩`)}{`⟨*v1*⟩ `[,`...`]}`. These commands can be used to fix relative positions in one coordinate, while allowing movement in the other coordinate.

2.5.8 Skeletons

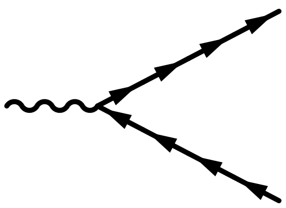
The single most powerful concept for adjusting `feynMF`'s layout decisions is the use of *skeletons*. By issuing a `\fmffreeze` after specifying a subgraph (skeleton), we can fix the location of the skeleton *as if* the other arcs were not there. We can then successively add more subgraphs whose layout will be chosen with the skeleton remaining fixed. Similar effects can be achieved by giving some arcs a vanishing `tension`.

Consider the following example: suppose we want to draw a ladder diagram contributing to the quark form factor. Simply linking in the gluons does not produce a satisfactory result:



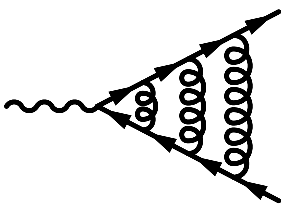
```
\fmfleft{i1} \fmfright{o1,o2}
\fmf{photon}{i1,v4}
\fmf{quark}{o1,v1,v2,v3,v4,v5,v6,v7,o2}
\fmf{gluon}{v1,v7}
\fmf{gluon}{v2,v6}
\fmf{gluon}{v3,v5}
```

What went wrong? Obviously the gluons are bonding the quark lines too strongly. The fix is simple: just create a skeleton excluding the gluons



```
\fmfleft{i1} \fmfright{o1,o2}
\fmf{photon}{i1,v4}
\fmf{quark}{o1,v1,v2,v3,v4,v5,v6,v7,o2}
```

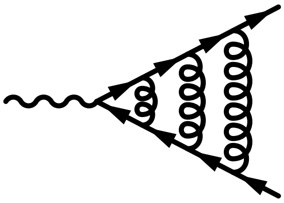
and add the gluons later:



```
\fmfleft{i1} \fmfright{o1,o2}
\fmf{photon}{i1,v4}
\fmf{quark}{o1,v1,v2,v3,v4,v5,v6,v7,o2}
\fmffreeze
\fmf{gluon}{v1,v7}
\fmf{gluon}{v2,v6}
\fmf{gluon}{v3,v5}
```

Alternatively, we can use a vanishing `tension`, which will effectively exclude the gluons from the layout decisions:

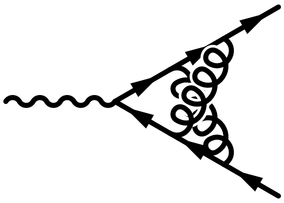
```
\fmfleft{i1} \fmfright{o1,o2}
\fmf{photon}{i1,v4}
```



```
\fmf{quark}{o1,v1,v2,v3,v4,v5,v6,v7,o2}
\fmf{gluon,tension=0}{v1,v7}
\fmf{gluon,tension=0}{v2,v6}
\fmf{gluon,tension=0}{v3,v5}
```

Which method is more intuitve is largely a matter of taste and at the discretion of the user.

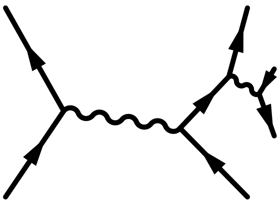
The "crossed" version of this diagram is best drawn using the `rubout` option:



```
\fmfleft{i1} \fmfright{o1,o2}
\fmf{photon}{i1,v3}
\fmf{quark}{o1,v1,v2,v3,v4,v5,o2}
\fmffreeze
\fmf{gluon}{v1,v4}
\fmf{gluon,rubout}{v2,v5}
```

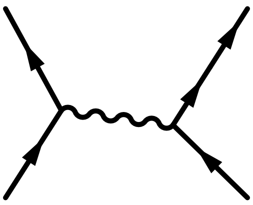
If `rubout` is selected, it defaults to 2, but it can be changed to any (reasonable) value > 1. The current implementation should work for almost all user-defined line styles. However, it does not work correctly for the predefined `double` line styles yet.

Another instructive example is the following: imagine you want to draw a typical non-resonant contribution to $e^+e^- \to 4f$. The obvious solution doesn't look right:



```
\fmfleft{i1,i2}
\fmfright{o1,o2,o3,o4}
\fmf{fermion}{i1,v1,i2}
\fmf{boson}{v1,v2}
\fmf{fermion}{o1,v2,v3,o4}
\fmf{boson}{v3,v4}
\fmf{fermion}{o3,v4,o2}
```

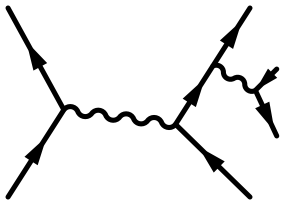
Again, we should start with a skeleton



```
\fmfleft{i1,i2}
\fmfright{o1,o2,o3,o4}
\fmf{fermion}{i1,v1,i2}
\fmf{boson}{v1,v2}
\fmf{fermion}{o1,v2,v3,o4}
```

and add the second W later:

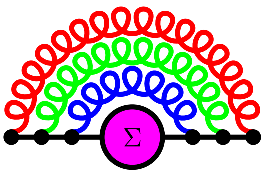
```
\fmfleft{i1,i2}
\fmfright{o1,o2,o3,o4}
\fmf{fermion}{i1,v1,i2}
\fmf{boson}{v1,v2}
\fmf{fermion}{o1,v2,v3,o4}
```



```
\fmffreeze
\fmf{boson}{v3,v4}
\fmf{fermion}{o3,v4,o2}
```

Here's another example that uses stretchable arcs. Diagrams of this kind are known as *rainbow* diagrams. If you're using METAPOST and are watching this on a color device, you'll see why.

```
\fmfpen{thick}
\fmfleft{i1,d1}
\fmfright{o1,d2}
\fmfn{plain}{i}{4}
\fmf{plain}{i4,v,o4}
\fmfn{plain}{o}{4}
\fmffreeze
\fmf{gluon,left,fore=red}{i1,o1}
\fmf{gluon,left,fore=green}{i2,o2}
\fmf{gluon,left,fore=blue}{i3,o3}
\fmfdotn{i}{3}
\fmfdotn{o}{3}
\fmfv{d.sh=circle,d.f=empty,d.si=.2w,b=(1,,0,,1),
      l=$\Sigma$}{v}
```



Experience has shown that the method advocated in this section is more effective than fuzzing around with fractional `tension` parameters. Using `\fmfshift` or `\fmfforce` should be a last resort only.

2.5.9 Pulling strings

If you add to any arc one or more `phantom` arcs they will cause a tighter bonding between the vertices involved

```
\fmf{fermion}{v1,v2}
\fmf{phantom}{v1,v2}
```

which is equivalent to

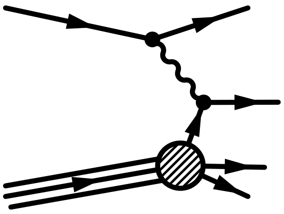
```
\fmf{fermion,tension=2}{v1,v2}
```

The `phantom` arc has to be added *before* any `\fmffreeze` involving these vertices, of course.

Here is an example from deep inelastic scattering[20]:

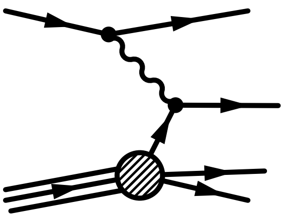
```
\fmfleft{ip,il}
\fmfright{oq1,oq2,d1,oq3,d2,d3,ol}
\fmf{fermion}{ip,vp,vq,oq3}
```

[20] Don't be confused by the `\fmfi` command. It is described below (see section 2.7.1) and takes the same arguments as the `\fmfv` command. We use it here for adding to more lines parallel to the incoming proton line. They do not enter the layout decisions.



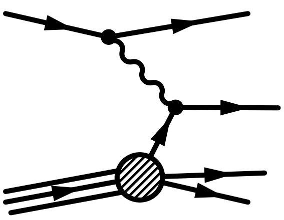
```
\fmf{fermion}{vp,oq1}
\fmf{fermion}{vp,oq2}
\fmf{photon}{vl,vq}
\fmf{fermion}{il,vl,ol}
\fmfblob{.15w}{vp}
\fmfdot{vq,vl}
\fmffreeze
\fmfi{plain}{vpath (__ip,__vp) shifted (thick*(0,2))}
\fmfi{plain}{vpath (__ip,__vp) shifted (thick*(1,-2))}
```

As it stands, all vertices come out too far to the right, because the greater number of outgoing lines pulls them over. Adding `\fmf{phantom}` makes the bond between the incoming vertices and the interactions tighter and produces a better balanced picture:



```
\fmfleft{ip,il}
\fmfright{oq1,oq2,d1,oq3,d2,d3,ol}
\fmf{fermion}{ip,vp,vq,oq3}
\fmf{phantom}{ip,vp}
\fmf{fermion}{vp,oq1}
\fmf{fermion}{vp,oq2}
\fmf{photon}{vl,vq}
\fmf{fermion}{il,vl,ol}
\fmf{phantom}{il,vl}
\fmfblob{.15w}{vp}
\fmfdot{vq,vl}
\fmffreeze
\fmfi{plain}{vpath (__ip,__vp) shifted (thick*(0,2))}
\fmfi{plain}{vpath (__ip,__vp) shifted (thick*(1,-2))}
```

Equivalently, we could add `tension` to the lines in question and we will get the same result:

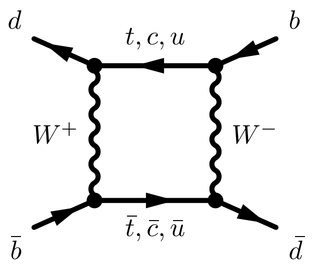


```
\fmfleft{ip,il}
\fmfright{oq1,oq2,d1,oq3,d2,d3,ol}
\fmf{fermion,tension=2}{ip,vp}
\fmf{fermion}{vp,vq,oq3}
\fmf{fermion}{vp,oq1}
\fmf{fermion}{vp,oq2}
\fmf{photon}{vl,vq}
\fmf{fermion,tension=2}{il,vl}
\fmf{fermion}{vl,ol}
\fmfblob{.15w}{vp}
\fmfdot{vq,vl}
\fmffreeze
\fmfi{plain}{vpath (__ip,__vp) shifted (thick*(0,2))}
\fmfi{plain}{vpath (__ip,__vp) shifted (thick*(1,-2))}
```

Conversely, specifing a `tension` < 1 will make the corresponding arcs more loose.

Reconsider the box graph on page 20 and reduce the tension on the inner lines[21]

```
\fmfleft{i1,i2}
\fmflabel{$\bar{b}$}{i1}
\fmflabel{$d$}{i2}
\fmfright{o1,o2}
\fmflabel{$\bar{d}$}{o1}
\fmflabel{$b$}{o2}
\fmf{fermion}{i1,v1}
\fmf{fermion,tension=.5,label=$\bar{t},,\bar{c},,\bar{u}$,
             l.side=right}{v1,v3}
\fmf{fermion}{v3,o1}
\fmf{fermion}{o2,v4}
\fmf{fermion,tension=.5,label=$t,,c,,u$,l.side=right}{v4,v2}
\fmf{fermion}{v2,i2}
\fmf{photon,tension=.2,label=$W^+$,l.side=left}{v1,v2}
\fmf{photon,tension=.2,label=$W^-$,l.side=right}{v3,v4}
\fmfdotn{v}{4}
```



This result is much nicer than the original.

2.6 Miscellaneous commands

2.6.1 Graphs in graphs

`fmfsubgraph` The `fmfsubgraph` environment contains a subgraph, for which the galleries will be placed inside the rectangle of width ⟨*width*⟩ and height ⟨*height*⟩, with lower left corner at (⟨*x*⟩, ⟨*y*⟩):

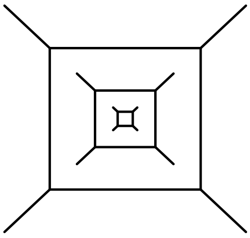
```
\begin{fmfsubgraph}(⟨x⟩,⟨y⟩)(⟨width⟩,⟨height⟩)
    ⟨body⟩
\end{fmfsubgraph}
```

The center and four corners are available as `c`, `nw`, `ne`, `sw` and `se` (for *north-west* etc). Because of the restrictions on the overall size of the diagram in METAFONT, this environment will, mainly be useful for preparing transparencies with METAPOST.

Here is a not very serious application of this feature:

```
\def\subgraphsample#1{%
  \fmfleftn{#1i}{2}%
  \fmfrightn{#1o}{2}%
  \fmf{plain}{#1i1,#1v1}%
  \fmf{plain}{#1o1,#1v2}%
  \fmf{plain}{#1o2,#1v3}%
  \fmf{plain}{#1i2,#1v4}%
  \fmfcyclen{plain,tension=0.3}{#1v}{4}}
\begin{fmfgraph}(40,30)
  \subgraphsample{a}
```

[21] Now that you know, I have also displayed the `label` options used.



```
    \begin{fmfsubgraph}(.3w,.3h)(.4w,.4h)
      \subgraphsample{b}
    \end{fmfsubgraph}
    \begin{fmfsubgraph}(.45w,.45h)(.1w,.1h)
      \subgraphsample{c}
    \end{fmfsubgraph}
  \end{fmfgraph}
```

More serious applications will use this feature to group together related diagrams and use other METAFONT drawing commands for graphical elements that are not available in the LaTeX `picture` environment.

2.6.2 Reusing diagrams

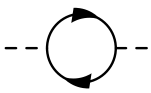
`\fmfkeep` `\fmfreuse` After saving a pointer with `\fmfkeep{⟨name⟩}` inside a `fmfgraph` or `fmfgraph*` environment, the diagram can be reused later by `\fmfreuse{⟨name⟩}`. This works across `fmffile` environments.

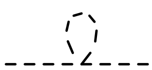
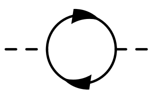
Here's an example that uses centered `\parbox`es to place diagrams into equations. The celebrated cancelation of quadratic divergencies in scalar selfenergies in supersymmetric field theories can be written as:

```
\begin{eqnarray}
  \parbox{20mm}{\begin{fmfgraph}(20,15)\fmfkeep{boson}
    \fmfleft{i} \fmfright{o} \fmf{dashes}{i,v,v,o}
  \end{fmfgraph}} & \propto & \Lambda^2 \\
  \parbox{20mm}{\begin{fmfgraph}(20,15)\fmfkeep{fermion}
    \fmfleft{i} \fmfright{o} \fmf{dashes}{i,v1} \fmf{dashes}{v2,o}
    \fmf{fermion,left,tension=.3}{v1,v2,v1}
  \end{fmfgraph}} & \propto & \Lambda^2 \\
  \parbox{20mm}{\fmfreuse{boson}} \quad + \quad
  \parbox{20mm}{\fmfreuse{fermion}} & \propto & \ln\Lambda^2
\end{eqnarray}
```

The result is:

 $\propto \quad \Lambda^2$ (1)

 $\propto \quad \Lambda^2$ (2)

 $+$  $\propto \quad \ln\Lambda^2$ (3)

2.6.3 Grouping

`fmfgroup` Make the enclosed `\fmfset` assignments local to this environment.

Parameter	Default	Semantics
`thin`	1pt	thin arcs
`thick`	1.5`thin`	thicker arcs
`arrow_len`	4mm	length of arrow head
`arrow_ang`	15	opening angle of arrow head
`curly_len`	3mm	length of one curl
`dash_len`	3mm	length of one dash
`dot_len`	2mm	distance of two dots
`wiggly_len`	4mm	length of one wiggle
`wiggly_slope`	60	inclination of wiggles
`zigzag_len`	2mm	length of a zig-zag period
`zigzag_width`	2`thick`	width of zig-zag lines
`decor_size`	5mm	default size of vertex decors
`dot_size`	4`thick`	diameter of dots

Table 7: Available style parameters.

2.6.4 Changing parameters

`\fmfset` This command can be used to change the parameters in table 7 as follows:

`\fmfset{`⟨*parameter*⟩`}{`⟨*value*⟩`}`

Note that these parameters are not stored in the graph data structure for the individual vertices and arcs. Instead the current values at the time of `\fmfdraw` are used.

2.6.5 Shrinking

`fmfshrink` Shrink the linewidths and similar parameters in the enclosed section.

2.6.6 Debugging

`\fmftrace` `\fmfnotrace` Enable and disable tracing of the layout decisions. This is not necessarily printed in an intuitive format, but can be helpful for debugging.

`\fmfdisplay` `\fmfstopdisplay` Enable online displays. `\fmfstopdisplay` will halt METAFONT everytime a graph is complete.

2.6.7 Multiple vertices and arcs

`fmffor` The environment

```
\begin{fmffor}{⟨var⟩}{⟨from⟩}{⟨step⟩}{⟨to⟩}
    ⟨body⟩
\end{fmffor}
```

executes ⟨*body*⟩ multiple times, setting ⟨*var*⟩ to ⟨*from*⟩, ⟨*from*⟩ + ⟨*step*⟩, . . . , ⟨*to*⟩. An application of this above `feynMF` feature is shown in figure 7, which is generated by calling the TEX macro

```
\def\EulerHeisenberg#1{%
```

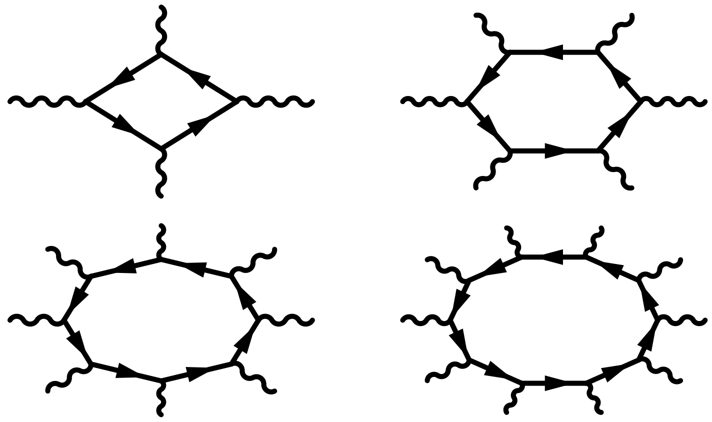


Figure 7: Higher order terms in the Euler-Heisenberg lagrangian.

```
\begin{fmfgraph}(40,25)
  \fmfpen{thick}
  \fmfsurroundn{e}{#1}
  \begin{fmffor}{n}{1}{1}{#1}
    \fmf{photon}{e[n],i[n]}
  \end{fmffor}
  \fmfcyclen{fermion,tension=#1/8}{i}{#1}
\end{fmfgraph}}
```

with the arguments 4, 6, 8, and 10, respectively.
Similarly, we can draw the diagrams from many particle physics in figures 8 and 9:

```
\def\PPRing#1{%
  \begin{fmfgraph}(20,20)
    \fmfsurroundn{v}{#1}
    \fmfdotn{v}{#1}
    \fmfcyclen{fermion,right=0.25}{v}{#1}
    \fmfcyclen{fermion,left=0.25}{v}{#1}
  \end{fmfgraph}}
\def\PHRing#1{%
  \begin{fmfgraph}(20,20)
    \fmfsurroundn{v}{#1}
    \fmfdotn{v}{#1}
    \fmfcyclen{fermion,right=0.25}{v}{#1}
    \fmfrcyclen{fermion,right=0.25}{v}{#1}
  \end{fmfgraph}}
```

2.7 Immediate mode

In addition to the automatic layout of vertices, feynMF features an immediate mode, in which feynMF's drawing commands operate directly on METAFONT's

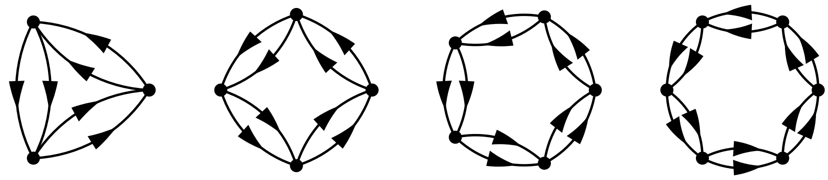


Figure 8: Particle-particle ring diagrams

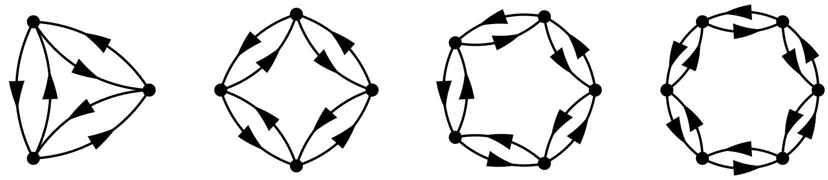


Figure 9: Particle-hole ring diagrams

pairs and paths. You might want to consult The METAFONT Book [4] or the METAPOST manual [5] for further information on the available path expressions.

2.7.1 Arcs

`\fmfi` Immediate mode's brother of `\fmf`.

`\fmfi{`⟨*style*⟩[`,`⟨*opt*⟩[`=`⟨*val*⟩]`,`...]`}{`⟨*p*⟩`}`

draws a line of style ⟨*style*⟩ on path ⟨*p*⟩. Use the `vpath` function in ⟨*p*⟩ (*after* `\fmffreeze`!) to access the METAFONT `path` connecting two vertices: `vpath`[⟨*tag*⟩]`(`⟨*from*⟩`,`⟨*to*⟩`)`. The optional numeric ⟨*tag*⟩ can be used together with a matching `tag` option to `\fmf` to disambiguate arcs that connect the same vertices. You have to prepend each name of a vertex in `vpath`'s arguments with two underscores (e.g. `v1` becomes `__v1`). This is necessary for avoiding name-clashes with some reserved words in METAFONT (*sparks* in DEK's terminology).

2.7.2 Vertices

`\fmfiv` Immediate mode's brother of `\fmfv`.

`\fmfiv{`⟨*shape*⟩[`=`⟨*val*⟩][`,`⟨*opt*⟩[`=`⟨*val*⟩]`,`...]`}{`⟨*v*⟩`}`

draws a vertex at position ⟨*v*⟩. Note that here ⟨*v*⟩ is a METAFONT `pair` and *not* a `feynMF` vertex name. The former's equivalent of the latter can be accessed (*after* `\fmffreeze`!) with the `vloc` function: `vloc(`⟨*vertex*⟩`)`. Again, you have to prepend the name of the vertex in `vloc`'s argument with two underscores (e.g. `v1` becomes `__v1`).

2.7.3 Declarations

`\fmfipath` The argument(s) are declared METAFONT `path`s.
`\fmfipair` The argument(s) are declared METAFONT `pair`s (coordinates).

2.7.4 Assignments

`\fmfiequ` Establish equality for the two arguments, i.e. `\fmfiequ{lval}{rval}` translates to `lval=rval`.

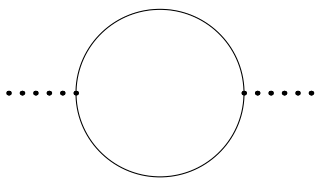
`\fmfiset` Assign the second argument to the first, i.e. `\fmfiset{lval}{rval}` translates to `lval:=rval`.

Specifying equality of two variables is a very different operation from assignment in METAFONT. See The METAFONT Book [4] for details on METAFONT's builtin equation solver.

2.7.5 Examples

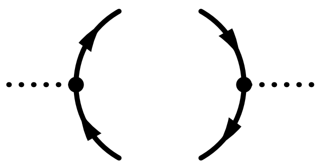
Here is a non-trivial example of immediate mode, which shows some useful tricks. The non-trivial aspect of the diagram in question is that it has lines broken in two, denoting particles coupling to a condensate.

We start the diagram with a skeleton (the `phantom` lines are shown as thin lines for clarity):



```
\fmfleft{i}
\fmfright{o}
\fmf{dots}{i,v1}
\fmf{dots}{v2,o}
\fmf{phantom,left,tension=0.2,tag=1}{v1,v2}
\fmf{phantom,left,tension=0.2,tag=2}{v2,v1}
\fmfdot{v1,v2}
\fmfposition
```

Add the fermions to the skeleton



```
\fmfipath{p[]}
\fmfiset{p1}{vpath1(__v1,__v2)}
\fmfiset{p2}{vpath2(__v2,__v1)}
\fmfi{fermion}{subpath (0,length(p1)/3) of p1}
\fmfi{fermion}{subpath (2length(p1)/3,length(p1)) of p1}
\fmfi{fermion}{subpath (0,length(p2)/3) of p2}
\fmfi{fermion}{subpath (2length(p2)/3,length(p2)) of p2}
```

Add condensates and a gluon

```
\def\cond#1#2{%
  \fmfiv{d.sh=cross,d.ang=#1,d.siz=5thick}{#2}}
\cond{30}{point length(p1)/3 of p1}
\cond{-30}{point 2length(p1)/3 of p1}
\cond{30}{point length(p2)/3 of p2}
\cond{-30}{point 2length(p2)/3 of p2}
\fmfi{gluon}{point length(p1)/10 of p1
               -- point 11length(p1)/12 of p1}
\def\vert#1{%
  \fmfiv{d.sh=circle,d.f=1,d.siz=2thick}{#1}}
\vert{point length(p1)/12 of p1}
\vert{point 11length(p1)/12 of p1}
```

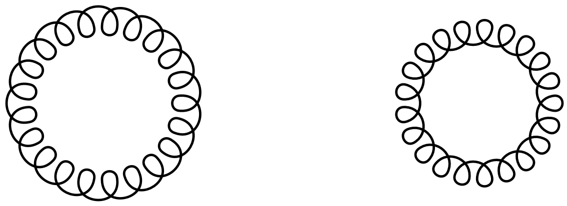
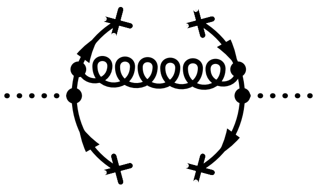
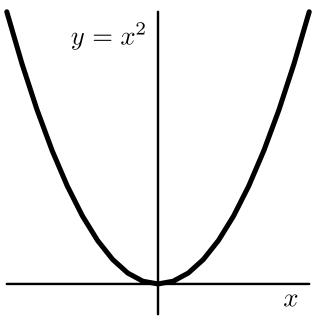


Figure 10: Circular gluons.



Here's an interesting abuse of feynMF (see the next section for \fmfcmd):

```
\begin{fmfgraph*}(40,40)
  \fmfipair{o,xm,xp,ym,yp}
  \fmfiequ{o}{(.5w,.1h)}
  \fmfiequ{xm}{(0,.1h)}
  \fmfiequ{xp}{(w,.1h)}
  \fmfiequ{ym}{(.5w,0)}
  \fmfiequ{yp}{(.5w,h)}
  \fmfiv{l=$x$,l.a=-135,l.d=2mm}{xp}
  \fmfiv{l=$y=x^2$,l.a=-135,l.d=2mm}{yp}
  \fmfpen{thin}
  \fmfcmd{draw xm--xp; draw ym--yp;}
  \fmfpen{thick}
  \fmfiequ{xs}{xpart(xp-o)}
  \fmfiequ{ys}{ypart(yp-o)}
  \fmfcmd{draw (o + (-xs,ys)) for n = -9 upto 10:
      --(o + (xs*(n/10),ys*((n/10)**2)))
    endfor;}
\end{fmfgraph*}
```



Finally, for the curious, here is how to draw the circular gluons in figure 10:

```
\fmfi{gluon}{fullcircle scaled .5w shifted (.5w,.5h)}
\fmfi{gluon}{reverse fullcircle scaled .5w shifted (.5w,.5h)}
```

2.8 Raw METAFONT

Some more advanced features of feynMF are more conveniently accessed through raw METAFONT commands. This can either be achieved by preparing a METAFONT input file or by using \fmfcmd extensively. The latter apprach is usally more convenient.

\fmfcmd The \fmfcmd macro writes its argument into the METAFONT input file generated by feynMF. While some experience in using METAFONT doesn't hurt here, this approach can simplify the production of complex diagrams considerably. Note that *no* semicolon is appended, the user has to provide it explicitely.

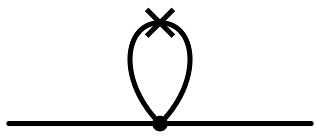
2.8.1 Extending feynMF

A prominent example for using raw METAFONT is provided by the option to add new styles for arcs. There is of course always one more style that *must* be added to the default list. But increasing this list without bounds will eventually slow down feynMF and increase its memory requirements. It is therefore better to allow users to define their own styles. This is done with the METAFONT macro `style_def`, which defines a macro that will be called to do the drawing and registers this macro with feynMF so that it can be used in the first argument to `\fmf`.

The macro takes one argument of type `path` and is responsible for drawing the arc on this path. If METAPOST's color functionality is to be used, the color aware functions `cdraw`, `cfill`, `cfilldraw`, `ccutdraw` and `cdrawdot` should be used instead of `draw`, etc.

After the following `style_def`, a new style `crossed` will be available:

```
\fmfcmd{%
  vardef cross_bar (expr p, len, ang) =
   ((-len/2,0)--(len/2,0))
      rotated (ang + angle direction length(p)/2 of p)
      shifted point length(p)/2 of p
  enddef;
  style_def crossed expr p =
    cdraw p;
    ccutdraw cross_bar (p, 5mm,  45);
    ccutdraw cross_bar (p, 5mm, -45)
  enddef;}
```



it can be used just like any other style:

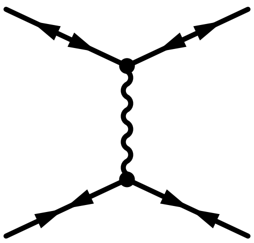
```
\fmfleft{i}
\fmfright{o}
\fmf{plain}{i,v,o}
\fmf{crossed}{v,v}
\fmfdot{v}
```

And here is an (esthetically questionable!) attempt at wiggly lines with arrows:

```
\fmfcmd{%
   style_def wiggly_arrow expr p =
     cdraw (wiggly p);
     shrink (2);
       cfill (arrow p);
     endshrink;
   enddef;}
```

Note how the `shrink` macro (which is the METAFONT equivalent of the `fmfshrink` environment) is used to temporarily double the dimensions of the arrowhead which is constructed by the `arrow` macro.

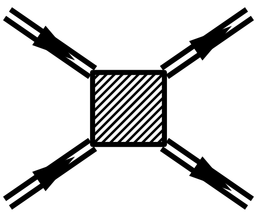
In particular theorists beyond the standard model are likely to need *a lot* of different line styles. The aficionados of majorana neutrinos might find the following two useful:



```
\fmfcmd{%
  style_def majorana expr p =
    cdraw p;
    cfill (harrow (reverse p, .5));
    cfill (harrow (p, .5))
  enddef;
  style_def alt_majorana expr p =
    cdraw p;
    cfill (tarrow (reverse p, .55));
    cfill (tarrow (p, .55))
  enddef;}
```

Note the use of the `harrow` and `tarrow` functions which return an arrowhead on the given fraction of the path, with reference points at the head (`harrow`) or tail (`tarrow`). The `arrow` function used above is equivalent to `marrow(p,.5)`, which has the reference point at the center of the arrowhead. Having the three different reference point available is important for supporting arcs of substantially differing lengths.

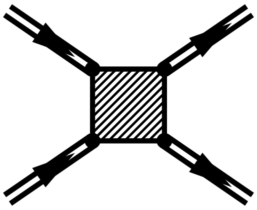
A problem of the current implementation is that the endpoints of double lines don't match smoothly at vertices:



```
\fmfpen{thick}\fmfleftn{l}{2}\fmfrightn{r}{2}
\fmfpolyn{shaded}{z}{4}
\fmf{dbl_plain_arrow}{l2,z3}
\fmf{dbl_plain_arrow}{l1,z4}
\fmf{dbl_plain_arrow}{z1,r1}
\fmf{dbl_plain_arrow}{z2,r2}
```

One way around is to add dots of the right size at the vertices *after* the arcs have been drawn.

```
\fmfpen{thick}\fmfleftn{l}{2}\fmfrightn{r}{2}
\fmfpolyn{shaded}{z}{4}
\fmf{dbl_plain_arrow}{l2,z3}
\fmf{dbl_plain_arrow}{l1,z4}
\fmf{dbl_plain_arrow}{z1,r1}
\fmf{dbl_plain_arrow}{z2,r2}
\fmffreeze\fmfdraw
\fmfvn{d.siz=2thick,d.sh=circle}{z}{4}
```



Without the `\fmffreeze\fmfdraw`, the arcs would "know" about the dots and would be shortened.

A more elegant solution is to define line styles with dots at the head,

```
\fmfcmd{vardef endpoint_dot expr p =
  save oldpen; pen oldpen;
  oldpen := currentpen;
  pickup oldpen scaled 3;
  cdrawdot p;
  pickup oldpen;
enddef;}
\fmfcmd{style_def hd_double expr p =
  draw_double p;
  endpoint_dot point infinity of p;
enddef;}
\fmfcmd{style_def hd_dbl_plain_arrow expr p =
  draw_hd_double p;
  shrink (1.5);
    cfill (arrow p);
  endshrink;
enddef;}
```

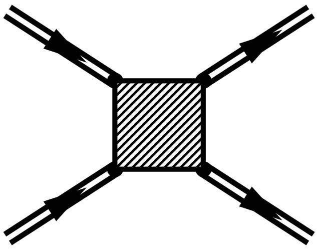
tail

```
\fmfcmd{style_def td_double expr p =
  draw_double p;
  endpoint_dot point 0 of p;
enddef;}
\fmfcmd{style_def td_dbl_plain_arrow expr p =
  draw_td_double p;
  shrink (1.5);
    cfill (arrow p);
  endshrink;
enddef;}
```

or both

```
\fmfcmd{style_def htd_double expr p =
  draw_double p;
  endpoint_dot point 0 of p;
  endpoint_dot point infinity of p;
enddef;}
\fmfcmd{style_def htd_dbl_plain_arrow expr p =
  draw_htd_double p;
  shrink (1.5);
    cfill (arrow p);
  endshrink;
enddef;}
```

which can be used as follows to give an equivalent result:



```
\fmfpen{thick}\fmfleftn{l}{2}\fmfrightn{r}{2}
\fmfpolyn{shaded}{z}{4}
\fmf{hd_dbl_plain_arrow}{l2,z3}
\fmf{hd_dbl_plain_arrow}{l1,z4}
\fmf{td_dbl_plain_arrow}{z1,r1}
\fmf{td_dbl_plain_arrow}{z2,r2}
```

2.9 Common traps, trouble shooting and frequently asked questions (FAQs)

2.9.1 `! Value is too large`

The following will be one of the most frequent errors:

```
! Value is too large (4097).
<recently read> ;

beginchar->...rdp:=(EXPR3);w:=hround(charwd*hppp);
                                                 h:=...
l.685 ...char(64, 40*2.84526pt#, 25*2.84526pt#, 0)
                                                 ;
?
```

It can have one of two causes:

- METAFONT was invoked without specifying the `mode`. This case is easily fixed by looking up the correct METAFONT mode in the file `modes.mf` that comes with the METAFONT distribution. This mode *must* be specified on the command line as `\mode:=laserjet` for HP Laserjets at 300dpi, as `\mode:=ljfour` for HP Laserjets at 600dpi, as `\mode:=nexthi` for NeXT laser printers at 400dpi, etc, just to name three of the more common laser printers in the physics community.
- The diagrams are too large for the printer at hand. This case is actually not very likely, because even at 1200dpi the diagrams can be as large as 86mm. For the popular laserprinter resolution of 300dpi, even 346mm are possible. Last time I checked, the diagrams for this manual could be generated for a *Linotype Linotronic 300* at 2540dpi (`mode:=linosuper`), but failed in the standard `proof` mode at 2601.72dpi.[22] In fact, in the current `modes.mf` file, the *Chelgraph IBX* at 9600dpi and the *Alphatype CRS* at $5333 + 1/3$dpi are the only typesetters that can not be used to typeset this manual.[23]

[22] It would be trivial to shrink the diagrams by 1% to make them work in `proof` mode (accidentally, the largest diagram is 40mm wide, while 4096/2601.72dpi corresponds to 39.99mm). However, I prefer METAFONT to give an error message if the user forgot to specify the mode. It is much more obscure when METAFONT works without errors but the `dvi` driver fails to find the generated bitmap file.

[23] If someone wants to use `feynMF` with one of these high end typesetters, I would be glad to try to help them out with kludges.

2.9.2 Diagrams in the document are never updated

There are two known reasons why diagrams may not be updated the document after the source file has been changed:

- Some `dvi` file previewers (e.g. `xdvi(1)` under UNIX) do *not* reread font information if the `tfm` or `pk` files have changed, even though they reread the `dvi` file if it has changed. Therefore you have to restart such previewers if you have made changes in diagrams to see these changes on the screen.
- Some `dvi` drivers (e.g. `dvips(1)` under UNIX) do not work with the `gf` files directly, but convert them with an external program to `pk` format first. On later occasions, the `dvi` driver will then use the `pk` file which is out of date with respect to the sources and the `gf` file. The only known fix is to delete the `pk` fils before running the `dvi` driver.

2.9.3 Disgrams show up in the wrong spot

If you are using feynMF with LaTeX's `\includeonly` feature, you should watch out for the following situation:

```
\includeonly{bar1,bar3}
\begin{fmffile}{foograph}
\include{bar1}
\include{bar2}
\include{bar3}
\end{fmffile}
```

where `bar1.tex` defines graph #1, `bar2.tex` graph #2 and `bar3.tex` defines graph #3. If you now proceeded to add graphs to `bar1.tex`, you will notice that a second graph is accepted, but instead of the new third graph, the old graph #3 appears. What happens is that LaTeX stores the value of the counter for `fmfgraph`s in each `.aux` file so that because `bar2.tex` is not processed, this counter is always reset to 3 at the beginning of `bar3.tex`.
Even though this situation appears to be contrived, it actually occured in real life applications and the resulting error is very confusing.
The only "fix" for this problem would be to use a private counter behind LaTeX's back. Unfortunately, it appears that this will violate the *principle of minimal surprise* even more. It is therefore usually a good idea to reprocess the complete document when the number of graphs has changed in an `\include`d file. The other solution is to have a separate `fmfgraph` environment for each `\include`d file.

2.9.4 Spurious labels show up

If spurious labels show up in your diagrams, this is most likely caused by old label files (e.g. `foo.t`$\langle n\rangle$) still lying around. Just delete these files and rerun TeX and METAFONT (or METAPOST respectively).

2.10 Known bugs

2.10.1 Chaotic manual

This is being worked on. It should probably be rewritten from scratch, but I don't have enough time at the moment (this is a spare time activity).

2.10.2 Delayed error messages

This can't be fixed. The problem is that errors can manifest themselves only a long time after the corresponding source line has been read. Since TEX doesn't allow to access the current source line number, there is no way to store this information along with the other information on the graph. I can only hope to have enough sanity checks in place some day that error messages from METAFONT won't occur.

2.10.3 Multiple tadpoles

Currently, `feynMF` will not layout multiple tadpoles at a single vertex automatically. This could be fixed in principle, but these fixes would cause other problems which are more inconvenient than having to lay out tadpoles manually.

2.10.4 Hard limits

Currently the most severe limitation lies in the size of the generated pictures. The largest number METAFONT can represent internally is 4095.99998 and this is also the largest value any coordinate measured in pixels can assume. At the most popular laserprinter resolution of 300 dots per inch (dpi), this corresponds to a horizontal and vertical extension of about 346mm, which is plenty and we're more likely to hit the internal limits on the complexity of a picture. However, at the proof mode resolution of 2601.72dpi, this is reduced to slightly less than 40mm and we're running the risk of arithmetic overflow in internal calculations much earlier.
There are two potential solutions of different scope and complexity:

- Since John Hobby's METAPOST is now available without a non-disclosure agreement from AT&T, one solution is to replace METAFONT by METAPOST, which doesn't suffer from the size limitations. This comes with a small price paid in reduced portability of the generated output, but as already stated above in the case of `axodraw`, the ubiquity of PostScript printers (and the free GhostScript interpreter) makes this a minor point.
- The more ambitious solutions would be *virtual graphs*, i.e. graphs which are larger than the current limit enforced by numeric overflow at higher resolutions. This could be implemented by calculating the layout of a miniature graph and afterwards distributing the full graph among several METAFONT characters.

Acknowledgements

I am most grateful to Wolfgang Kilian, who pushed `feynMF`'s predecessor `feynman.mf` to its limits [12]. Discussions with him triggered a lot of good

ideas. Thanks also to my students and the people on *The Net* for suggestions, portability fixes and for volunteering as guinea pigs.

References

[1] Donald E. Knuth, *The TEXbook*, Addison-Wesley, Reading MA, 1986.

[2] Leslie Lamport, *LATEX — A Documentation Preparation System*, Addison-Wesley, Reading MA, 1985.

[3] Michel Goosens, Frank Mittelbach, and Alexander Samarin, *The LATEX Companion*, Addison-Wesley, Reading MA, 1994.

[4] Donald E. Knuth, *The METAFONTbook*, Addison-Wesley, Reading MA, 1986.

[5] John D. Hobby, *A User's Manual for METAPOST*, Computer Science Report #162, AT&T Bell Laboratories, April 1992.

[6] Thorsten Ohl, Comp. Phys. Comm. **90** (1995) 340.

[7] Thorsten Ohl, CERN Computer Newsletter **220** (1995) 22; **221** (1995) 46; **222** (1996) 24.

[8] Micheal J. S. Levine, Comp. Phys. Comm. **58** (1990) 181.

[9] Jos Vermaseren, Comp. Phys. Comm. **83** (1994) 45. `axodraw` is available from CTAN (cf. p. 42), in the `graphics` directory.

[10] Thomas E. Leathrum, `mfpic`, available from CTAN (cf. p. 42), in the `graphics` directory.

[11] Tim Stelzer and Bill Long, Comp. Phys. Comm. **81** (1994) 357.

[12] Wolfgang Kilian, Doctoral Thesis, Technical University Darmstadt, 1994.

[13] Alan Jeffrey, *Lists in TEX's Mouth*, TUGboat 199?.

Distribution

`feynMF` is available by anonymous internet ftp from any of the Comprehensive TEX Archive Network (CTAN) hosts

`ftp.shsu.edu`, `ftp.tex.ac.uk`, `ttp.dante.de`

in the directory

`macros/latex/contrib/supported/feynmf`

It is also available from the host

`crunch.ikp.physik.th-darmstadt.de`

in the directory

```
pub/ohl/feynmf
```

Unsupported snapshots of my work in progress are provided as

```
pub/ohl/feynmf.versions/feynmf-current.tar.gz
```

There are two mailing lists

```
feynmf-announce@crunch.ikp.physik.th-darmstadt.de
feynmf-bugs@crunch.ikp.physik.th-darmstadt.de
```

open for subscription. The former should carry only important announcements, of new versions in particular. To subscribe, send mail to the (electronic) mailing list manager

```
majordomo@crunch.ikp.physik.th-darmstadt.de
```

and *not* to the lists itself. The following commands (on a line in the body of the mail, not in the subject) are useful:

```
subscribe feynmf-announce
unsubscribe feynmf-announce
help
```

Index

Numbers written in italic refer to the page where the corresponding entry is described; numbers underlined refer to the code line of the definition; numbers in roman refer to the code lines where the entry is used.