creating

MetaPost

outlines

makempy manual

Hans Hagen

1 introduction

You can use METAPOST to create graphics in a declarative manner. Although there are tools to create METAPOST graphics interactively, by nature the input is a script defining the graphic.

Plain METAPOST can handle text directly, as strings, or as pictures distilled from TEX output. In both cases, the text is composed of individual characters, and these somehow will end up in the POSTSCRIPT file generated by METAPOST.

Although one should not see METAPOST as a full blown system for doing fancy graphic, its usage, especially in a dynamic document as created by TEX, sometimes demands a more sophisticated way of handling text. The section titles in this document are an example of this.

In this document I will discuss a way to import text (as typeset by TEX) into a METAPOST graphic. Although primarily written for usage with CONTEXT, this method is quite generic and also works well with plain TEX, LATEX or others.

2 method

Including text as graphics is far from trivial. First it has to be typeset, and of course we want to use TEX for that, and in our case PDFTEX suits well. Next we need to convert the typeset text to graphics, or actually outlines. For that step we will use Wolfgang Glunz's `pstoedit`, which can produce METAPOST code from POSTSCRIPT code. This program falls back on GHOSTSCRIPT for creating the curves out of the fonts. Because (at least currently) handling PDF directly is not working as expected, we convert the PDF file to POSTSCRIPT with Derek Noonburg's `pdftops` program. This process is illustrated in figure 1.

3 tools

During a METAPOST run, the text to be processed is written to a file with the suffix `mpo`. This file, which only has TEX directives, is converted to a file with suffix `mpy` which holds METAPOST code.

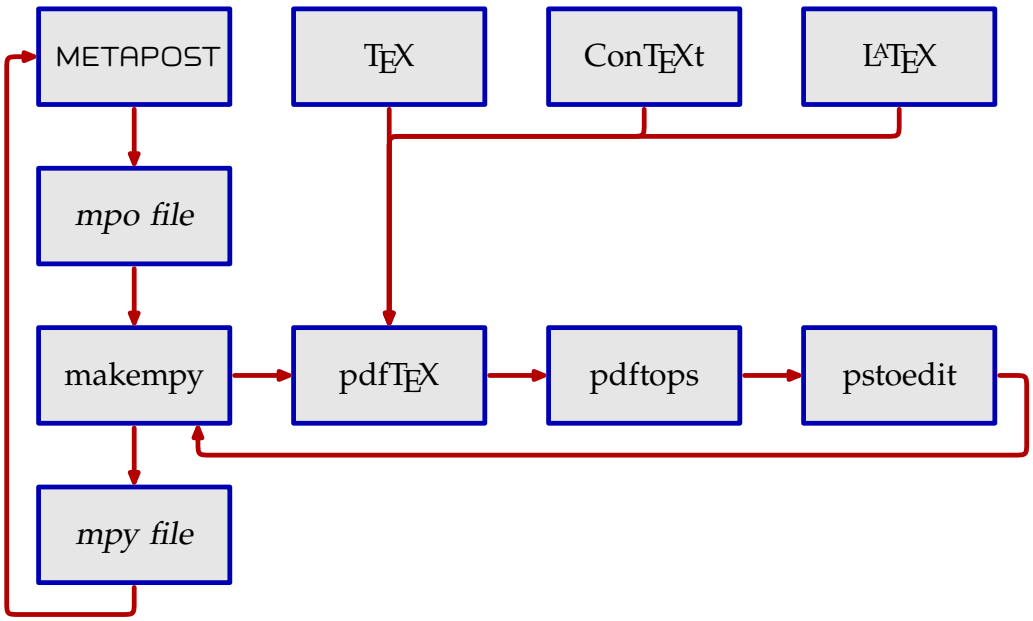


Figure 1 The process of conversion.

The conversion is taken care of by a PERL script `makempy`. This script generates some intermediate files that are fed into the programs mentioned.

In a next METAPOST run, the `mpy` file is read in each time a graphic is needed, and the curves are processed in a special way that permits us to fill and/draw them. We can apply transformations and use colors and pens. The macros that take care of this are collected in the METAPOST file `mp-grph.mp`, which is part of the CONTEXT distribution, but in itself is an independent part of the MetaFun suite.

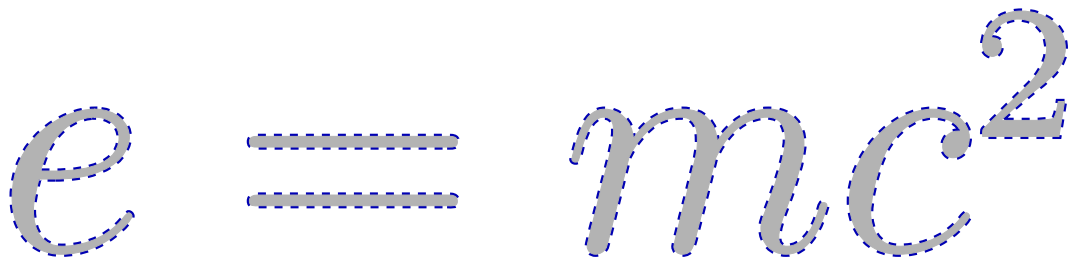
We will now demonstrate this mechanism using a typical TEX example.

```
input mp-grph ;

beginfig(1) ;
  graphictext
    "$e=mc^2$"
    scaled 10
```

```
    dashed evenly
    withdrawcolor .7blue
    withfillcolor .7white
    withpen pencircle scaled 2pt ;
endfig ;
```

This example introduces a METAPOST macro `graphictext`. The first argument of this macro is a string. After this string you can specify transform operations like a scaling factor, a shift, a rotation or a combination of these. In addition to the normal color, pen and dash specifications, there are two special color operators. When `fillcolor` is provided, the graphic will be also be filled, otherwise it will be an outline.



In this example, the string to be processed by TEX is given between `""`. When using METAPOST this way, you need to be aware of the fact that there can be no line feeds in a string, so a more complicated formula has to be specified as follows:

```
graphictext
  ("$$\pmatrix{D_x&D_y&1\cr} =" &
   "  \pmatrix{U_x&U_y&1\cr}  " &
   "  \pmatrix{s_x&r_x&0\cr   " &
   "           r_y&s_y&0\cr   " &
   "           t_x&t_y&1\cr}$$")
  ....
```

So, we split the string into lines, which we paste with `&` and surround by `()`. However, when in CONTEXT you want to include such more complicated TEX code in a graphic that is defined in the source, you need to be aware of unwanted expansion. This is why in CONTEXT we can best define the string separately, which also keeps the graphic more readable. There we have:

```
\setMPtext{formula}
  {$$\pmatrix{D_x&D_y&1\cr} =
     \pmatrix{U_x&U_y&1\cr}
     \pmatrix{s_x&r_x&0\cr
              r_y&s_y&0\cr
              t_x&t_y&1\cr}$$}
```

This string can be used as follows:

```
\startMPcode
  graphictext
    \MPstring{formula}
    scaled 2
    withfillcolor white
    withdrawcolor .7blue
    withpen pencircle scaled .75pt ;
\stopMPcode
```

As you can see from the graphic below, you need to adapt the size of the pen to the scale.

$$\begin{pmatrix} D_x & D_y & 1 \end{pmatrix} = \begin{pmatrix} U_x & U_y & 1 \end{pmatrix} \begin{pmatrix} s_x & r_x & 0 \\ r_y & s_y & 0 \\ t_x & t_y & 1 \end{pmatrix}$$

In some cases a glyph is composed of multiple outlines. Therefore, by default, the fill is applied after the draw. As a result, the perceived pen is smaller than the specified one. One can reverse drawing and filling by supplying the keyword `reversefill` as demonstrated in the next example.

```
\startMPcode
  graphictext
    \MPstring{formula}
    scaled 2
    reversefill
    withcolor .7blue
    withpen pencircle scaled .25pt ;
\stopMPcode
```

As you can see clearly here, big delimiters are composed of pieces.

$$\begin{pmatrix} D_x & D_y & 1 \end{pmatrix} = \begin{pmatrix} U_x & U_y & 1 \end{pmatrix} \begin{pmatrix} s_x & r_x & 0 \\ r_y & s_y & 0 \\ t_x & t_y & 1 \end{pmatrix}$$

In addition to the default method and the reversed fill, we also have an outline fill. The differences show up clearly if we use a dashed line:

```
\startMPcode
  graphictext "MP" scaled 8
    withdrawcolor .7blue withfillcolor .7white
    dashed evenly withpen pencircle scaled 2pt ;
\stopMPcode
```

In the reversed fill, we will get a thinner line, since the fill covers half the line.

```
\startMPcode
  graphictext "MP" scaled 8 reversefill
    withdrawcolor .7blue withfillcolor .7white
    dashed evenly withpen pencircle scaled 1pt ;
\stopMPcode
```

The outline fill method will extend the fill to the line, which sometimes gives nicer results with dashed shapes.

```
\startMPcode
  graphictext "MP" scaled 8 outlinefill
    withdrawcolor .7blue withfillcolor .7white
    dashed evenly withpen pencircle scaled 2pt ;
\stopMPcode
```

These three definitions result in the following graphics. The third alternative is the least efficient because the paths are duplicated.

default fill reverse fill outline fill

In the first example of this section, we use plain TEX, while the second and following examples depend on CONTEXT. You can specify a TEX back-end using the following directive:

```
graphictextformat := "latex" ;
```

In CONTEXT you can specify environments in the normal way[1] like:

[1] For more information on what is normal, you can consult the MetaFun manual.

```
\startMPenvironment
\setupbodyfont[pos,11pt]
\stopMPenvironment
```

In a stand-alone METAPOST graphic, you can pass information to TEX using `graphictextdirective`:

```
graphictextformat := "latex" ;
graphixtextdirective "\documentclass[]{article}" ;
```

You don't have to provide document structuring commands, since these are added by `makempy`.

You can also use `pstoedit` independently, in which case each page becomes a figure. You can load such a figure using the `loadfigure` macro, as in:

```
loadfigure("filename.suffix", 4) ;
```

This macro makes sure that the content of figure 4, when present, is added to `currentpicture`. You can also use this macro to include figures defined in other METAPOST files, so that you can build libraries.

The PERL script `makempy` uses the following programs:

gs	a general purpose POSTSCRIPT processor
pdftops	a PDF to POSTSCRIPT converter from the xpdf suite
pstoedit	a POSTSCRIPT to whatever vector format converter (often comes with GHOSTVIEW)

When set, the environment variables `GS` and `GS_PROG` will be used to identify GHOSTSCRIPT. Of course you need to have PERL on your system to run `makempy` as well as PDFTEX (which nowadays is part of most TEX distributions). If needed, the names of the other programs that are used can be set with `PDFTOPS`, `ACROREAD` and `PSTOEDIT`.

The `makempy` script does its work without user intervention. When it deduces that there are no changes, it will not run at all, otherwise it will report its steps and/or encountered error. The script takes one argument:

```
makempy filename
```

A suffix, when provided, is stripped. The following switches are recognized:

`--help`	returns some basic help information
`--silent`	don't report status messages
`--force`	always process the file (no checksum test)
`--noclean`	don't remove temporary files when finished
`--acrobat`	use ACROBAT for conversion (only unix)
`--pdftops`	use PDFTOPS for conversion
`--ghostscript`	use GHOSTSCRIPT for conversion

The last few switches are handy when you encounter errors or get unexpected results. In that case you can do as follows:

```
mpost   yourfile
makempy yourfile --force --noclean
```

after that you can take a look at the files whose name start with `mpy-yourfile`.

```
mpy-yourfile.tex | .pdf | .log | .pos | .tmp
```

When using CONTEXT, keep in mind that TEXEXEC uses files with names as `mpgraph` which can be prefixed by `yourfile`, so there we get:

```
mpy-yourfile-mpgraph.tex | .pdf | .log | .pos | .tmp
```

At this moment we always use PDFTEX combined with `pdftops` since it guarantees that we get everything we need in the file. Experiments with `dvips` and `dvipsone` were unsatisfying with respect to page dimensions and font conversion.

In order to achieve optimal results, a few tricks are used, some of which don't deserve a beauty price.

Normally a typeset text will use font sizes between 10 and 30 points, because that's the way TEX systems are set up. When such small shapes as font glyphs are converted to curves, we loose quite some accuracy and the results will be quite awful. For this reason, behind the screens, we scale the graphic up and down by a factor 10.

Because `pdftops` has problems with non standard paper sizes, we specify a pretty large working area when converting the PDF file to POSTSCRIPT. The `pstoedit` program generates tight code, so there we will loose unwanted white space.

In order to demonstrate how well TEX and METAPOST can work together, I will give an example that uses a typeset paragraph. This example uses CONTEXT, but you can of course make an independent METAPOST figure instead. First we define a graphic:

```
\startuseMPgraphic{quotation}
  graphictext
    \MPstring{text}
    scaled 2
    withdrawcolor .7blue
    withfillcolor .7white
    withpen pencircle scaled 1pt ;
  picture one ; one := currentpicture ; currentpicture := nullpicture ;
  graphictext
    \MPstring{author}
    scaled 4
    withdrawcolor .7red
    withfillcolor .7white
    withpen pencircle scaled 2pt ;
  picture two ; two := currentpicture ; currentpicture := nullpicture ;
  currentpicture := one ;
  addto currentpicture also two
    shifted  lrcorner one
    shifted - 1.125 lrcorner two
    shifted (0, - 1.250 * ypart urcorner two)  ;
  setbounds currentpicture to boundingbox currentpicture enlarged 12pt ;
\stopuseMPgraphic
```

In this graphic, we have two text fragments, the first one is a text, the second one the name of the author. We link the quotation and author into this graphic using the following definitions:

```
\setMPtext{text}  {\vbox{\hsize 8.5cm \input zapf }}
\setMPtext{author}{\hbox{\sl Hermann Zapf}}
```

These definitions assume that the file `zapf.tex` is present on the system. The graphic, which is shown on the last page, can now be typeset using the following call:

```
\useMPgraphic{quotation}
```

For those not familiar with CONTEXT: the graphic data is written to a file, and processed either directly or between TEX runs. If you want to know more about the way CONTEXT can interact with METAPOST, you may want to take a look at the MetaFun manual (which also discusses METAPOST in detail).

Of course you can also process this quotation as stand alone graphic, in which case the METAPOST code goes between `beginfig-endfig`.

For CONTEXT users who want to know how the section headers in this document are defined, I let their definition follow here.

```
\startuseMPgraphic{text}
  graphictext \MPstring{string} scaled 4
    withdrawcolor \MPcolor{blue} withfillcolor \MPcolor{gray}
    withpen pencircle scaled 2pt ;
\stopuseMPgraphic

\startuseMPgraphic{number}
  graphictext \MPstring{string} scaled 10
    withdrawcolor \MPcolor{red} withfillcolor \MPcolor{gray}
    withpen pencircle scaled 2pt ;
\stopuseMPgraphic

\def\TextCommand#1%
  {\setMPtext{string}{#1}%
   \framed
     [frame=off,offset=overlay,height=3cm,width=\hsize]
     {\useMPgraphic{text}}}

\def\NumberCommand#1%
  {\expanded{\setMPtext{string}{\currentheadnumber}}%
   \hsmash
     {\framed
        [frame=off,offset=overlay,height=3cm,width=\hsize]
        {\useMPgraphic{number}}}}

\setuphead
  [section]
  [distance=0pt,
```

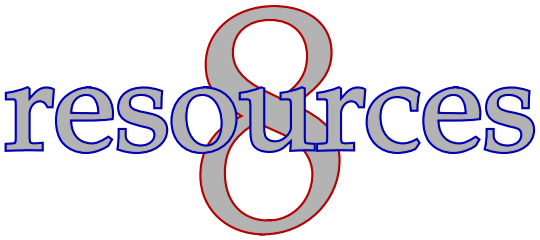
```
    textcommand=\TextCommand,
    numbercommand=\NumberCommand]
```

There are two points worth noticing. We define two hooks into the section header command. In the macro that deals with the section number, we don't use the number as provided by the argument `#1`, but use the raw number instead. We do so, because the content of this argument contains label related macros, which don't operate that well when processed independently from this document. We need to expand the number, because the graphic is processed in an independent run.

The second trick concerns the `hsmash`. Because the number and text are placed after each other, we explicitly have to set the distance to zero, as well as makes sure that the number does not consume space. By using the `\framed` macro, we get both text and number nicely centered.

We use default font size but pass the colors (red, blue and gray) from this particular document. The bodyfont environment is passed on to METAPOST's TEX run by saying:

```
\startMPenvironment
  \setupbodyfont[loc,ppl]
\stopMPenvironment
```



The PERL script `makempy` as well as the METAPOST module `mp-grph.mp` are part of the CONTEXT distribution. You can find the latest version in the zipped archive that can be fetched from the download page at:

`www.pragma-ade.com`

or one of the mirrors. You can also find CONTEXT in the main stream TEX distributions. If you encounter problems, you may contact:

Hans Hagen
PRAGMA-ADE, Hasselt, NL
j.hagen@pragma-ade.com

But for support you can best go to the CONTEXT mailing list:

`ntg-context@ntg.nl`

This manual is typeset at October 28, 2001.

Coming back to the use of typefaces in electronic publishing: many of the new typographers receive their knowledge and information about the rules of typography from books, from computer magazines or the instruction manuals which they get with the purchase of a PC or software. There is not so much basic instruction, as of now, as there was in the old days, showing the differences between good and bad typographic design. Many people are just fascinated by their PC's tricks, and think that a widely–praised program, called up on the screen, will make everything automatic from now on.

Hermann Zapf

I sincerely hope that Hermann Zapf will forgive me for mis-using his quotation —from an article on *HZ*-optimization— as well as for manipulating his Palatino font in this way.