A short introduction to METAPOST

K. Höppner
Haardtring 230 a, 64295 Darmstadt, Germany
`klaus.hoeppner@gmx.de`

1 History

When Knuth developed TEX, he also created a set of new fonts, Computer Modern. For this, he created his own font description language and the program METAFONT, that converts a METAFONT source into a bitmap, usually stored in a file with the extension `.pxl` or more often `.pk`. The major feature of METAFONT is that paths may be described with a set of linear equations, that determines how the single points of the path are related, and this equation set is solved by the METAFONT program. Additionally, Knuth extensively used parameters within these equations, so different font series (e. g. bold and medium) could be produced from the same equations by changing some parameters.

John D. Hobby created METAPOST as a system using (nearly) the same programming language, but with PostScript output. It was presented first in TUGboat [1], while the first public versions were released in the early 1990s. Some new features were added to the Meta language, e. g. the ability to include stuff typeset by TEX into a METAPOST drawing (something, that wasn't needed in METAFONT for creating glyphs of a font, but is very useful to put text labels into graphics).

Meanwhile METAPOST is maintained by the METAPOST team, with Taco Hoekwater as chief developer. Since then, many improvements were made. For the future, they plan to release MPlib, a component library that contains the METAPOST engine and can be reused in other applications.

2 Basics

As mentioned before, METAPOST defines its own programming language. It consists of the following elements:

- points,
- pens,
- paths,
- numbers,
- colors (originally RGB only, meanwhile CMYK is supported)

Points are normally named by the letter z, represented by a pair (x, y).

Paths may contain geometrical elements (e. g. **`fullcircle`**) or may consist of points that are connected by lines or Bézier curves.

Colors are tuples of three (in case of RGB) or four (in case of CMYK) numbers.

For a short example let's have a look on the following example:

Listing 1: 1st example

```
filenametemplate "%j-%3c.mps";
beginfig(1);
pickup pencircle scaled 1bp;
draw origin--(1cm,2cm)
     ..(0.5cm,1cm)..cycle;
endfig;
end
```

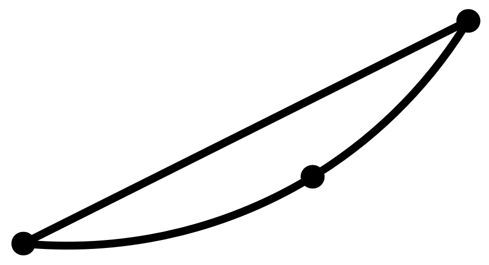


Figure 1: Example figure, as defined in listing 1

It shows that each METAPOST figure is put between **`beginfig`** and **`endfig`**, with a number identifying the figure. So, a METAPOST source may contain several figures. Originally, when processing the source (e. g. `ex.mp`) with METAPOST (`mpost ex`), the figure numbers were used as file extension for the resulting PostScript files. In later releases, the command `filenametemplate` was introduced, that uses a syntax like the `printf` command in C. In the example above, we would get a PostScript file with the name `ex-001.mps` (and if we add a figure with number 2, the PostScript output would be written into `ex-002.mps`). Since pdfTEX recognizes files with the extension `.mps` as METAPOST output, the file could be easily used in a LATEX document with `\includegraphics{ex-001.mps}` and the document may be processed both by pdfTEX and, in the original workflow, by compiling to DVI and using dvips.

As in C, all statements may span multiple lines and are finished by the "`;`" character.

The example figure itself shows a straight line from the origin to the point (1cm, 2cm). Then, the path is closed by a Bézier curve via the point located at (1.3cm, .3cm). For drawing, a round pen with diameter of one PostScript point is used. METAPOST knows the same units as TEX, like `bp` for PS points, cm, mm. The result is shown in fig. 1.

3 Defining points by linear equations

While there is nothing exciting about the first example, we will see what makes METAPOST special in a moment. Assume you want to draw a simple rectangle. Then you know it consists of four corners (e. g. with the lower left one in the origin), that we will describe by the following equations:

Listing 2: Rectangle

```
path p[];
z0 = origin;
x0 = x3;
x1 = x2;
y1 = y0;
y3 = y2;
x1-x0 = 3cm;
y3-y0 = 2cm;
p0 = z0--z1--z2--z3--cycle;
fill p0 withcolor blue;
draw p0 withpen pencircle scaled 1bp;
```

You see, all corners besides z_0 aren't defined directly as (x, y) pairs but described by their relations. While describing a rectangle with linear equations seems like a bit of overkill, this METAPOST feature becomes really powerful for the construction of complex paths.

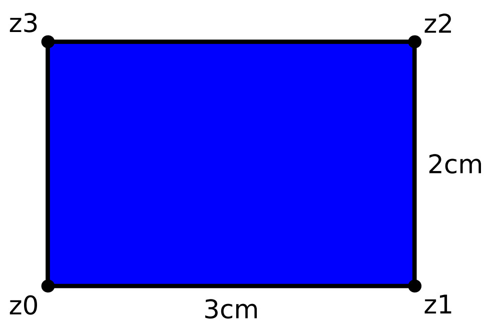


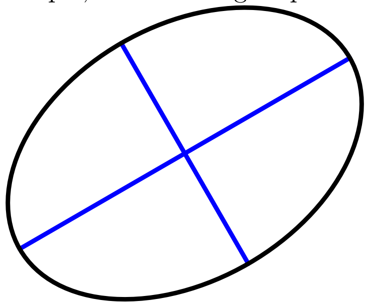
Figure 2: Rectangle, resulting from code in listing 2

4 Transformations of paths

METAPOST supports the following transformations of paths:

- Translations
  `p0 shifted (x1, x2)`
- Rotation
  `p0 rotated alpha`
- Scaling (in both directions, or in x or y direction individually)
  `p0 scaled factor`
  `p0 xscaled xfactor`
  `p0 yscaled yfactor`
- Slanting
  `p0 slanted alpha`

For example, the following ellipsis



is the output of the code

```
draw fullcircle xscaled 3cm
     yscaled 2cm rotated 30;
```

5 Intersection points

Finding the intersection points of paths is another nice METAPOST feature.

Assume you have a triangle. Theory says, if you draw three lines, each of them from one corner of the triangle to the mid point of the opposite side, all these lines will intersect at the same point.

The following code shows how this can be demonstrated in a METAPOST drawing:

Listing 3: Triangle 1

```
pickup pencircle scaled 1bp;
path p[];
z0 = origin;
z1 - z0 = 3cm*right;
z2 - z0 = 2.7cm*dir(40);
p0 = z0--z1--z2--cycle;
p1 = .5[z0,z1]--z2;
p2 = .5[z1,z2]--z0;
p3 = .5[z2,z0]--z1;
draw p1 withcolor blue;
draw p2 withcolor blue;
draw p3 withcolor blue;
draw p1 intersectionpoint p2
```

```
     withpen pencircle scaled 3bp;
draw p0;
```

This code is rather simple. It consists of three parts.

First, the three points $z_0 \ldots z_2$ are defined and path p_0 is defined as the triangle with these points as corners.

Second, the paths $p_1 \ldots p_3$ are defined. Each consists of a line from one corner to the mid point of the opposite side. This may be easily expressed in METAPOST, since e. g. the statement `.5[z1,z2]` is just the point on half the line from z_1 to z_2.

Finally, after drawing all the paths defined above, we mark the intersection point of p_1 and p_2. This is directly given by the command

```
p1 intersectionpoint p2
```

It may be a bit more complicated if two paths have more than one intersection point.

The result of this drawing is shown in fig. 3.

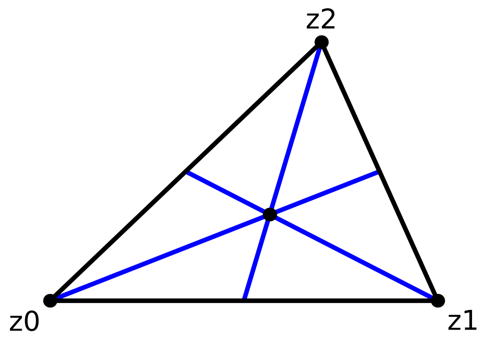


Figure 3: Construction of a triangle

6 Whatever it is ...

Coming back to the triangle in the latest example, another interesting task is the following: draw a line from one corner that is orthogonal to the opposite side.

Let's assume we want to draw the line from point z_2, so the opposite side is just the base line of our triangle. Now we know two things:

1. The line is orthogonal to the base line (connection of $z0$ and z_1).
2. The starting point is in z_2, the end point shall be on the base line.

This may be directly expressed in METAPOST:

Listing 4: Triangle 2

```
z10-z2= whatever*((z1-z0) rotated 90);
z10 = whatever[z0,z1];
```

In the code above the end point of the line is named z_{10}.

Here we see both conditions listed before: first, the distance vector between z_{10} and z_2 is given by the distance vector between z_1 and z_0 (i. e. the base line), rotated by 90 degrees, *scaled by an arbitrary factor*.

Second, z_{10} is located *somewhere* on the line defined by the points z_0 and z_1.

In both cases, I used a numerical value named **`whatever`**. This may become an arbitrary number. In fact, the value may change from statement to statement, since the variable **`whatever`** is encapsulated per statement.

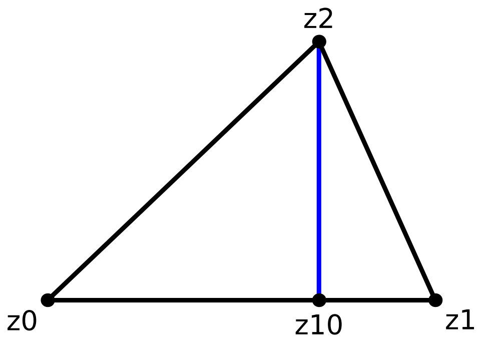


Figure 4: Triangle 2

As shown in fig. 4, METAPOST finds the correct position for z_{10} as starting point of a perpendicular line to the base line, with z_2 as end point.

7 Time variables

A path in METAPOST may be imagined as the travel of a vehicle. Paths are parameterized by a time variable (what might be a bit misleading, since of course the drawing is static). So a path has a start and end time, and any point is correlated to a time in between (and vice versa).

Here is some example, where time variables are used:

Listing 5: Time variables and subpaths

```
pickup pencircle scaled 1bp;
path p[];
p0 = origin{up}..(3cm,2cm);
p1 = (-5mm,2cm)--(3cm,5mm);
draw p0 dashed withdots;
draw p1 dashed withdots;
(t0,t1) = p0 intersectiontimes p1;
draw subpath (0,t0) of p0
  -- subpath (t1,length(p1)) of p1;
```

We have two paths, p_0 and p_1: a Bézier curve from lower left to upper right, and a straight line from upper left to lower right, drawn with dotted lines.

Now we are not only interested in the intersection point of these paths, but we want to combine the subpath of p_0 before the intersection point with the subpath of p_1 after this point, drawn with a solid line in fig. 5.

In this case, we need the time values of both paths in the intersection point. For this, the statement `p0 intersectiontimes p1` is used. The result of this is a pair (in fact a point), with the time value of p_0 in the intersection point as first part (x-part) and the time value of p_1 as second part (y-part).

As soon as these time values are known, the desired path is constructed using `subpath`.

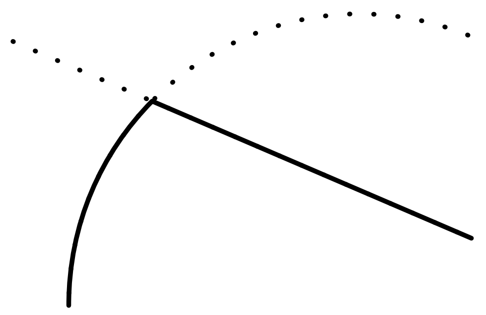


Figure 5: Using time variables and subpaths

8 Text and Labels

METAPOST allows to place labels into a figure. In the simplest form, the text may be included directly, without any typesetting done by TEX:

```
defaultfont := "ptmr8r";
defaultscale := 1.2;
label("this is a label",z0);
```

It will just add the text commands to write the label text in 12pt Times Roman (since a font scaling factor of one refers to 10pt) into the PostScript code. Please note that `:=` is used in the code above, since new values are assigned to the variables, while `=` is used in linear equations.

The action of placing a label at z_0 in the example is rather straightforward. The `label` command centers the label at the given point. In many cases, a suffix is appended to the `label` command to define how the label is placed in relation to the given point, i. e. `top`, `bot` (bottom), `lft` (left), `rt` (right) or `ulft`, `llft`, `urt`, `lrt` (e. g., `ulft` means upper left and `lrt` means lower right). The `label` command may be replaced by `dotlabel`, that draws a dot at the given point in addition to the label. For example, the code

```
dotlabel.urt("this is a label",z0)
```

draws a dot at z_0 and places the given text in upper right direction from z_0.

Only simple text may be used for labels that are included by METAPOST directly. But for real typesetting, we may use one of the best typesetting programs we know, TEX itself. We may include nearly arbitrary TEX stuff into labels. All TEX snippets that occur in the METAPOST source are extracted and typeset with TEX, before the result is included into the figure by METAPOST.

All the TEX stuff has to be embedded into an environment `btex ... etex`, e. g.

```
label(btex $z_0$ etex, z0)
```

will center the label "z_0" at this point. Normally, the plain TEX compiler is used for typesetting the `btex ... etex` fragments. But the name of the program may be passed in command line when calling METAPOST, so to use LATEX we just use the command

```
mpost --tex=latex ex1
```

Let't have a look how we can typeset a label with LATEX, using Euler math fonts to typeset a formula:

Listing 6: Typesetting a label with LATEX

```
filenametemplate "%j-%3c.mps";
verbatimtex
\documentclass{article}
\usepackage{euler}
\begin{document}
etex
beginfig(1);
dotlabel.urt(
  btex $\sqrt{\frac{1}{1+x^2}}$ etex,
  origin);
endfig;
```

The example contains some LATEX code within a `verbatimtex ... etex` environment, that is taken verbatim as LATEX code before typesetting all the labels. Please note that it is used to define a preamble with article as document class and loading the euler package. While the `\begin{document}` is present, the closing `\end{document}` is inserted automatically.

The result of this code, when compiled by METAPOST with using LATEX as typesetter—as explained above—is shown in fig. 6.

Including labels causes some difficulties with fonts. Normally, METAPOST doesn't embed fonts

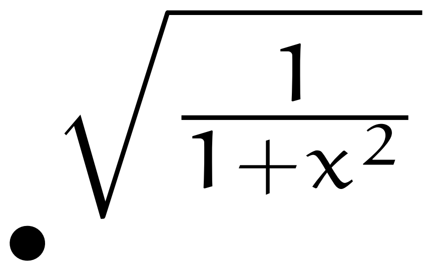


Figure 6: Using LaTeX for typesetting a label

but just adds a reference to the used fonts into the PostScript output. This isn't a problem when METAPOST is included in a TeX document, since TeX will resolve all of these font references. But the figures won't be usable standalone, since PostScript interpreters like GhostScript will complain about unknown fonts.

In recent versions of METAPOST it is possible to run METAPOST in a mode that will produce standalone PostScript output that contains a "real" EPS with all fonts embedded, that can be displayed in any PS interpreter or may be used in other applications besides TeX documents.

A switch named **`prologues`** defines whether METAPOST will embed fonts or not. Just use the definition **`prologues:=3;`** at the start of your METAPOST file to get a standalone EPS figure. The default value of `prologues` is 0, which means that no fonts will be embedded. The meaning of other values of `prologues` may be looked up in the METAPOST manual, they are only relevant for special cases.

9 Conclusion

This article was intended to just give a short introduction to METAPOST. I left out several things, e. g., how to use colors, defining macros, loops etc. Since the article is originally based on a talk that was part of a comparison of tools, it is focussed on what makes METAPOST unique among other drawing tools: solving linear equations and having direct access on intersection points and time variables of paths.

If you are interested in learning METAPOST, please have a look into the METAPOST manual [2] or into the LaTeX Graphics Companion [3], that describes METAPOST (among many other tools).

References

[1] John D. Hobby. A METAFONT-like System with PostScript Output. *TUGboat*, 10(4), 1989.

[2] John D. Hobby. METAPOST – A User's Manual. 2008.

[3] Michel Goossens, Frank Mittelbach, Sebastian Rahtz, Denis Roegel, and Herbert Voss. *LaTeX Graphics Companion, The, 2nd Edition.* Addison-Wesley Professional, 2007.