The `textpath.mp` package*

Stephan Hennig†

December 8, 2006

Abstract

The MetaPost package `textpath.mp` provides macros to set text along a free path. It differs from the `txp.mp` package in that is uses LaTeX to typeset the text and therefore 8-bit characters, *e.g.*, accented characters, are supported and text is kerned.

1 A first example

The `textpath.mp` package provides a `textpath` macro, that takes three arguments:

1. a text string that is valid LaTeX input,
2. a path the text should follow, and
3. a numeric value that controls justification of the text on the path.

The macro returns a picture that contains the text transformed to the path. A sample call to `textpath` could be

```
draw textpath("Greetings from MetaPost!", fullcircle scaled 50bp, 0);
```

The resulting output is shown in figure 1.

Interesting, but not what I'd expect. Well, figure 1 looks exactly as what we formulated in our MetaPost programme above. The text is drawn along a circle segment that starts at an angle of zero, runs counter clockwise and the text is left aligned on the path. Left aligned in this context means that the text starts at the beginning of the path.

But the text is inside the circle. Yes, by default, the text is typeset on the left-hand side of the path if you were virtually walking down the path. There are configuration options to change that behaviour (see section 4.6), but those won't help in this situation, since as a result the text would run right-to-left.

Instead, have a look at the following changes to our code and the resulting figure 2:

*This document describes `textpath.mp` v1.5.

†stephanhennig@arcor.de

Figure 1: Our first application.

Figure 2: A refined variant.

```
path p;
p := reverse fullcircle rotated -90 scaled 50bp;
draw textpath("Greetings from MetaPost!", p, 0.5);
```

What a magic path declaration! The path now starts at an angle of −90 degrees and it runs clockwise. Did you notice the last parameter to `textpath` changed to 0.5 in the programme? If it were zero, the text would start at the bottom of the circle and run clockwise along the outside of the circle. With the alignment parameter set to 0.5 the text is centered on the path. That's the reason for rotating the start and end points of the circle to −90 degrees.

By the way, do you notice anything wrong with figure 2? We'll come back to that problem at the end of this manual. The next section discusses the parameters of the `textpath` macro in detail.

1.1 Text strings

1.1.1 Setting up the `textpath.mp` package

Internally, the string you pass to `textpath` as the first argument is processed by LaTeX by means of the `latexmp` package. This is hardcoded, currently, but may change in future versions so that you can use conventional `btex...etex` commands or the `TEX` macro.

The `latexmp` package has to be set up with a proper LaTeX preamble. The easiest way to do so is by providing a preamble file (without `\begin{document}`) and input `latexmp.mp` using the following lines:

```
input latexmp
setupLaTeXMP(preamblefile="mypreamble");
```

The last step in setting up the `textpath.mp` package is adding the following line to the LaTeX preamble in file `mypreamble.tex`:

```
\usepackage{textpathmp}
```

Note the trailing "mp" in the LaTeX package name!

1.1.2 Restrictions on text input and output

Transforming text onto a path, *i.e.*, rotating and shifting the text, should be done character by character, since otherwise there could be characters that move

too far away from the path. Unfortunately, MetaPost doesn't provide guaranteed character-wise access to a text picture generated by LaTeX. Therefore, some provisions have to be taken to enable characer-wise access. Internally, on the LaTeX side the text string passed to `textpath` is pre-processed by the `soul` package.

Invalid input Therefore, input must not contain any material `soul` can't handle. If you're encountering errors from the `soul` package on the console delete the corresponding `ltx-???.tmp` file in your working directory and try with a simpler text string. Explicitely, the following restrictions apply to input:

- Vertical material is not valid.
- Rules are not supported.
- Font selection commands aren't allowed inside a text string. See section 2 for a solution.
- If you want to add arbitrary horizontal space in the copy, you should do that with `\<{\kernXpt}`, where X is the desired space.

In general, try to stick to simple text.

Valid input However, input may contain:

- plain horizontal material,
- accented characters: These are fine in the input as long as you choose the correct input encoding in the LaTeX preamble and you have a font that provides true accented characters, *e.g.*, a font supporting T1 encoding.
- mathematical copy: Section 3 explains how to typeset formulae onto a path.

Output The following restrictions apply to the output of `textpath` *et al.*:

- Ligatures are broken in the output.

1.2 Paths

As mentioned before, by default, text is typeset on the left-hand side of a path. That is, the text is rotated to have the natural slope of the path at the desired location. Configuration options to change that behaviour are described in section 4.6.

In case total text width is larger than total path length characters that don't fit on the path are clipped by default (see figure 3). Configuration options to change that behaviour are described in section 4.1.

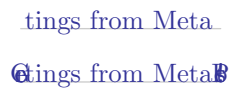

Figure 3: Text on a short horizontal path clipped and unclipped.

Figure 4: Using different fonts in a MetaPost graphic.

1.3 Text justification

Justification of the text on the path can be controlled by the third parameter `j`, which is a numeric value. Setting it to zero makes the text left aligned, *i.e.*, text starts right at be beginning of a path. A value of one makes the text right aligned, *i.e.*, text ends at the end point of a path. A value of 0.5 centers text on a path.

There is a general rule: the point at fraction `j` of the total text width is transformed to the point on the path at fraction `j` of the total path length. Therefore, `j` should be from the intervall $[0, 1]$.

2 Handling different fonts

The package provides a variant of the `textpath` macro called `textpathFont`, that differs from `textpath` in only one respect. It takes an additional fourth argument. That argument is a string containing a valid LaTeX font selection command such as `\itshape`. The font selection string has to be given as the first argument. The remaining arguments are the same as for `textpath` (see figure 4).

```
p := reverse fullcircle rotated -90;
draw textpathFont("\usefont{T1}{pzc}{m}{n}\huge", "Happy Birthday to",
  p scaled 400bp, 0.5) withcolor (1, 0.6, 0.2);
draw textpathFont("\usefont{T1}{bch}{m}{n}\large", "Daisy Duck!",
  p scaled 350bp, 0.5) withcolor (0.9, 0.3, 0.1);
```

Since the `soul` package can't handle font selection commands in the copy, one were stuck to the font that is setup in the LaTeX preamble for the whole MetaPost document. The `textpathFont` macro accounts for that by getting the font selection separate from the text and applying it before the `soul` package comes into play. Unfortunately, switching the font inside text still isn't possible that way.

3 The raw interface

In case you're fine with `textpath` and `textpathFont` and you do not intend to typeset mathematical formulae along a path, you may savely skip this section and read on with section 4 where configuration options are described.

Figure 5: Transforming arbitrary chunks of characters at once using the raw interface.

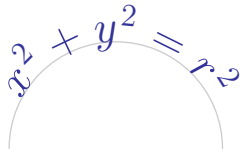

Figure 6: A formula typeset along a path.

The `textpath.mp` package provides a third macro `textpathRaw` to typeset material along a path. This is the so called raw interface. As we've seen in section 1.1.2, input to `textpath` and `textpathFont` is pre-processed by the `soul` package. In contrast, input to the raw interface isn't pre-processed. Have a look at figure 5 to see how MetaPost accesses arbitrary chunks of characters at once without pre-processing on the LaTeX side.

Why is the word "MetaPost" broken between letters "P" and "o"? In fact, MetaPost doesn't process *arbitrary* chunks of characters. It rather collects and processes chunks of characters that are typeset without artificial horizontal space between them. That is, text is broken into chunks by MetaPost at inter word space, kerning, *etc.* Clearly, in our example kerning took place between glyphs "P" and "o" while other glyph pairs weren't kerned.

Arguments to the `textpathRaw` macro are the same as to the `textpath` macro. The differences between `textpath` and `textpathRaw` are:

- Font selection commands are allowed in input to `textpathRaw`.
- Rules are supported.
- Manual horizontal space is allowed in the input.
- Ligatures are preserved.

As we've seen in figure 5, macros `textpath` and `textpathFont` are much better suited to typeset plain text along a path than `textpathRaw`. However, there is one application where `textpathRaw` performs better than `textpath`: mathematical formulae. Have a look at figure 6. Have you ever seen stuff like that with any other text processing or graphics application?

The reason `textpathRaw` doesn't fail as badly on mathematical input as it does on plain text is that in formulae characters are rarely typeset without extra space between each other. That is, MetaPost quite often breaks formulae into chunks and hence naturally processes them character-wise. The exception to this are operator names such as log, sin, arctan *etc.* and sometimes products of variables. The latter can be avoided by explicitely inserting `\cdot` between factors.

What about more complicated formulae with fractions etc.*?* Have a look at figure 7. Since rules are explicitely supported by `textpathRaw` fractions are no problem.

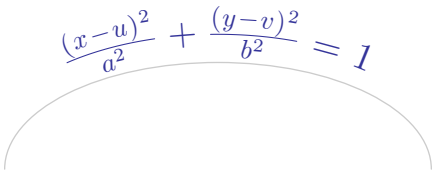

Figure 7: A formula containing fractions.

Awesome! The rules are curved, too! Yes, since formulae can look awful otherwise the `textpath.mp` package puts some efforts into that. Section 4.8 describes an option that lets you choose between curved and straight rules.

Any restrictions? Well, on concave shapes formulae look even more ugly than plain text. For a technical restriction, formulae shouldn't be of too large height and depth. Section 4.10 shows a solution to another problem with root operators not discussed so far.

That's it—almost. The remaining chapter describes global variables to configure the `textpath.mp` package, *i.e.*, the way macros `textpath`, `textpathFont` and `textpathRaw` work. Quite some interesting effects can be achieved with certain settings. So don't miss the last chapter!

4 Configuration variables

The `textpath.mp` offers several configuration options through global variables. Some of them have already been mentioned in the other sections. This chapter summarizes all configuration variables.

4.1 `textpathClip`

As explained in section 1.2 when total text width is larger than path length text that doesn't fit into the path boundaries is clipped. Actually, the location of characters that exeed path boundaries is clipped to the boundaries first. Then variable `textpathClip` is read and if it is set to zero value those characters are drawn at the boundary, acting as a visual marker that something went wrong, or, if `textpathClip` is set to one, those characters are omitted (see figure 3). By default, `textpathClip` is set to one, that is, characters may be clipped.

4.2 `textpathRepeat`

Variable `textpathRepeat` determines the number of copys of the text string that should be typeset along a path. Valid are integer values greater or equal to one. By default, `textpathRepeat` is set to one.

Note, that the alignment parameter `j` still works when setting `textpathRepeat` to arbitrary values. That is, for a value of zero text starts at the beginning of a path and for a value of one text ends at the end of a path.

Figure 8: Repeated text manually spaced and centered.

4.3 `textpathStretch` and `textpathHSpace`

When variable `textpathRepeat` is greater than one (see section 4.2), horizontal spacing between single copies of the text string is determined manually or automatically, depending on variable `textpathStretch`. If `textpathStretch` is set to zero value variable `textpathHSpace` determines the spacing between text string copies. If `textpathStretch` is set to one, spacing is automatically calculated, which is the default.

Figure 8 shows an example of manually spaced text copies. Note, that the space on the left and right of the total text is quite unrelated to `textpathHSpace` and automatically results from path length, text width, the number of text copies, `textpathHSpace`, and the justification parameter `j`.

```
f := "\usefont{T1}{pzc}{m}{n}\Large";
p := subpath (5.7,6.3) of fullcircle scaled 1400bp;
draw p withcolor .8white;
textpathRepeat := 3;
textpathStretch := 0;
textpathHSpace := 10pt;
draw textpathFont(f, "Happy Birthday", p, 0.5) withcolor (1, 0.6, 0.2);
```

In contrast, when text copies are automatically spaced, *i.e.*, `textpathStretch` is set to one, the sum of space before and after all text copies equals the space between text copies. This is quite handy when typesetting text along a cyclic path as in figure 9.

4.4 `textpathShift`

Variable `textpathShift` determines the amount all characters are shifted orthogonally to a path (the base line). Positive values result in characters shifted to the left of a path—looking into the natural direction of a path—, while negative values shift characters to the right of a path.

In figure 9 the thick black border is both, the path our text should follow and the background of the text. Therefore, we need to "vertically" center the text on the path. A small negative value of `textpathShift` serves our purposes well.

```
% Font Brush Script Italic is available on CTAN.
f := "\usefont{T1}{pbsi}{xl}{n}\fontsize{2.1pt}{2.1pt}\selectfont";
w := 210bp;
h := .276w;
r := .19h;
```

Figure 9: An attempt to resemble the style of classic labels. (Failed badly.)

```
p := (-.5w,0)--(-.5w,.5h-r)--quartercircle rotated -90 scaled (2r)
  shifted (-.5w,.5h)--(0,.5h);
p := p--reverse p reflectedabout ((0,-1),(0,1));
p := p--reverse p reflectedabout ((-1,0),(1,0))--cycle;
draw p withpen pensquare scaled 3.5pt;
textpathRepeat := 30;
textpathShift := -.6pt;
draw textpathFont(f, "Fa\ss' Dich kurz!", p, 0.5) withcolor white;
label(textext
  ("\usefont{T1}{bfu}{mb}{n}\fontsize{22pt}{22pt}\selectfont Telephonzelle"),
  origin);% Bitstream Futura
```

4.5 textpathLetterSpace

If you're typesetting text along a path with a high curvature the text may look squeezed or letter-spaced, depending on path shape, because natural inter letter space refers to unrotated characters and is only proper near the base line of rotated characters. On a convex shape, therefore, text can look letter-spaced where, in fact, it isn't. In contrast, on a concave shape glyphs may touch each other. Comparing figures 1 and 2 the latter looks lighter, since inter letter space raises with the distance from the base line. The same effect can be seen in figure 10, *e.g.*, between characters "a" and "i".

The `textpath.mp` package provides means to manually tweak inter letter space. Before transforming a character onto the path the `textpath.mp` package inserts an additional amount of horizontal space of width `textpathLetterSpace` between all characters. This variable is a good friend when shifting or rotating characters. By default, `textpathLetterSpace` is set to zero, that is, text is typeset at its natural width.

Variable `textpathLetterSpace` has only effect when typesetting text through macros `textpath` and `textpathFont`. In macro `textpathRaw` this variable is ignored.

4.6 textpathRotation

In section 1.2 it was already mentioned that text turns out on the left-hand side of a path, by default. This is an appropriate setting for cultures where text is read left to right.

Figure 10: The anchor point of a character.

The `textpath.mp` package, internally, calculates for each character in the text string the position of its anchor point on the path and the slope of the path at that location. The character is then rotated around its anchor point by an angle corresponding to the slope. The anchor point of a character lies at its horizontal center on the base line (see figure 10). Finally, the anchor point is translated to the required coordinates on the path. As a result, characters appear on the left-hand side of a path.

You can manually apply an additional rotation to every character by setting variable `textpathRotation` to a non-zero value. A value of, *e.g.*, 180 degrees turns each character upside down, so that it turns out on the right-hand side of a path. A value of, *e.g.*, 90 degrees results in characters "floating" with the path. By default, `textpathRotation` is set to zero.

In figure 11, note that the heart shape is derived from half circles and the text would have been running on the inside of the path by default, cf. figure 1.

```
string f, t;
path heart;
f := "\usefont{T1}{pzc}{m}{it}\tiny";
t := "Love";
heart := halfcircle shifted (-0.5bp,0bp)..{dir-50}(0bp,-1.5bp);
heart := heart--reverse heart reflectedabout ((0,0),(0,1))--cycle;
heart := heart scaled 60bp;
textpathRotation := 90;
textpathLetterSpace := 1pt;
textpathRepeat := 30;
draw textpathFont(f, t, heart, 0) withcolor red+0.1green;
```

4.7 `textpathAbsRotation`

The interpretation of variable `textpathRotation` changes when variable `textpathAbsRotation` is set to a value of one. Then characters are rotated by `textpathRotation` degrees only, ignoring path slope (see figure 12). By default, `textpathAbsRotation` is set to zero.

```
picture pic;
f := "\usefont{T1}{fwb}{m}{n}\Large";% From the emerald package
t := "Don't panic! Don't panic! Don't panic!";
p := origin
for i:=1 upto 20: ..(i, sind(i*45)) endfor;
```

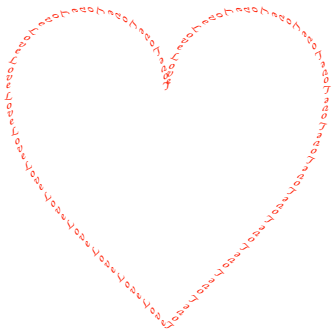

Figure 11: Automatically spaced text with an additional rotation of 90 degrees.

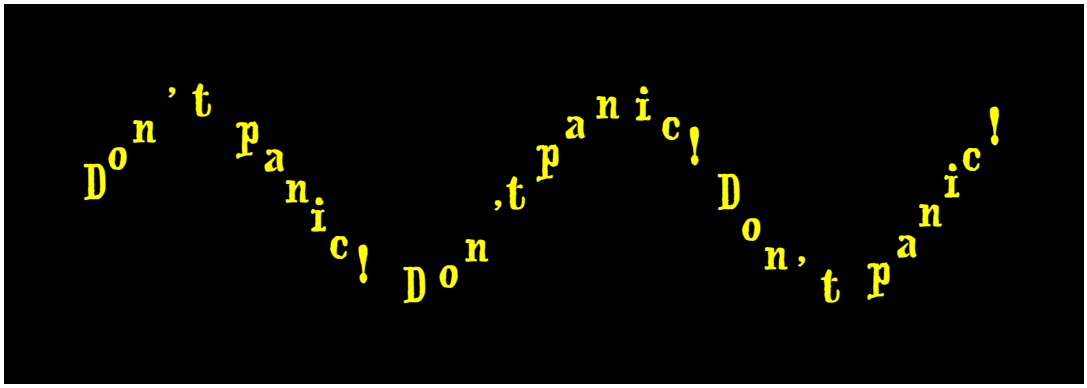

Figure 12: Text without rotation.

```
p := p xscaled 20 yscaled 35;
textpathRotation := 0;
textpathAbsRotation := 1;
textpathLetterSpace := 6pt;
pic := textpathFont(f, t, p, 0);
background := black;
bboxmargin := 30bp;
unfill bbox pic;
draw pic withcolor red+green;
```

We could have used `textpathRepeat` instead of repeating the text manually in the last example as well. But since that required some more letters to type I chose not to do so. In case the path definition inside the for loop—or rather the for loop inside the path definition—in the last listing looks uncommon to you, have a look at chapter 10 of the MetaPost manual, where this usage is explained.

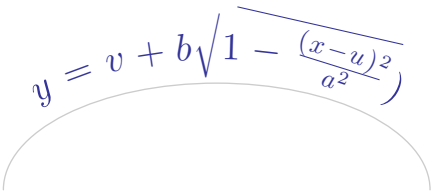

Figure 13: Straight rules in a formula.

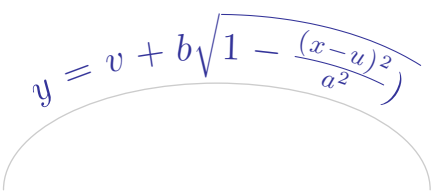

Figure 14: Curved rules in a formula.

4.8 `textpathFancyStrokes`

The remaining three variables apply only to macro `textpathRaw`. In section 3 figure 7 showed a formula containing fractions, where the rules were automatically bent to the shape of the path. Variable `textpathFancyStrokes` determines if rules in the input to `textpathRaw` should appear curved or straight in the output. A value of zero results in straight rules. If `textpathFancyStrokes` is set to one, which is the default, the `textpath.mp` package tries to adjust the shape of rules to that of the path.

Figures 13 and 14 show a similar formula as in figure 7. Clearly, both, the square root and the fraction look more decent with curved strokes.

4.9 `textpathStrokePrecision`

When you're typesetting formulae along paths with a high curvature it may be difficult for rules to follow the path. Variable `textpathStrokePrecision` determines the number of auxillary points that are calculated along a stroke. For higher values of `textpathStrokePrecision` strokes follow a path more precisely. By default, `textpathStrokePrecision` is set to a value of ten, which should be enough for most cases.

This variable applies only if `textpathFancyStrokes` is set to one. Otherwise, it is ignored.

4.10 `textpathCureSqrt`

Did you notice the small gap that is still present in the square root operator in figure 14? A root sign actually consists of two parts, a leading V part and a trailing horizontal rule spanning the arguments. The V part, in fact, is a single glyph, while the rule is, well, a rule. If typeset horizontally, both parts overlap so that they visually melt into one glyph.

When typeset onto a path, the V part is rotated according to the slope of the path at its anchor point, which lies at the horizontal center of the glyph. The resulting rotating angle is slightly different from the rotating angle that should be applied to the top right corner of the glyph. Therefore, V and rule part of a square root are missing each other and a gap becomes visible.

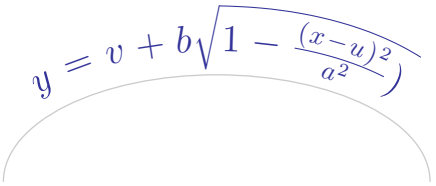

Figure 15: A curved square root without a gap.

Figure 16: Another logo that didn't fit as an example in the other sections.

The `textpath.mp` package tries to fill this gap automatically when variable `textpathCureSqrt` is set to a value of one, which is the default. In fact, in figure 14 this variable was manually set to zero to provoke the gap. In figure 15 the gap has been filled. Don't expect too fancy results from this option in difficult cases.

This variable applies only if `textpathFancyStrokes` is set to one. Otherwise, it is ignored. Moreover, it only works when `textpathRepeat` is set to a value of one, *i.e.*, there are no multiple copies of formulae.

4.11 Summary

Table 1 lists all configuration variables, their default values and in what macros they have effect.

5 Epilogue

Do you remember when we asked about a problem with figure 2 in section 1? *Yes, the first and last characters do not visually line up properly!* Well spotted! Actually, the code is correct. The problem is that "G" and "!" have quite different glyph shapes. When virtually drawing the bounding boxes of the rotated characters one realizes that the left-most and right-most points of the text on the path do line up.

Is there a fix? There's no automatic solution. If the text started, *e.g.*, with letters "B", "E" or even "W" *etc.* those characters would visually line up with an exclamation mark. As a work around for letters "C", "G" *etc.* the alignment parameter could be reduced to a value slightly less that 0.5. But honestly, did you really notice the difference?

Where's the source code of the last examples? The code of all figures shown in this manual can be found in file `textpathfigs.mp`, which is part of this package.

What is the nice calligraphic font you're using throughout this manual? Since we're dealing with decorative graphics here, URW Zapf Chancery Medium Italic

variable name	default value	valid in macros		
		`textpath`	`textpathFont`	`textpathRaw`
`textpathClip`	1	+	+	+
`textpathRepeat`	1	+	+	+
`textpathStretch`	1	+	+	+
`textpathHSpace`	0pt	+	+	+
`textpathShift`	0pt	+	+	+
`textpathLetterSpace`	0pt	+	+	−
`textpathRotation`	0	+	+	+
`textpathAbsRotation`	0	+	+	+
`textpathFancyStrokes`	1	−	−	+
`textpathStrokePrecision`	10	−	−	+
`textpathCureSqrt`	1	−	−	+

Table 1: Summary of configuration variables.

available through the PSNFSS package is used in many examples. Other decorative fonts with ready-made LaTeX support are Brush Script Italic or the fonts from the EMERALD package. Both packages are available on CTAN.

Anything else? While you're asking, don't try to do too fancy stuff with this package. Text—and mathematical copy—is most legible when typeset horizontally. With increasing curvature of the base line legibility decreases fastly. A decorative, but elegant look can be achieved best by just slightly indicating a curved base line, similar to figure 4.

Happy TeXing!
Stephan Hennig