The `textpath.mp` package[*]

Stephan Hennig[†]

February 11, 2007

Abstract

The MetaPost package `textpath.mp` provides macros to typeset 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. Moreover, even mathematical copy can be handled.

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 text starts at the beginning of the path.

But the text is inside the circle. Yes, by default, text is typeset on the left-hand side of a path—if you were virtually walking down the path. There are configuration options to change that behaviour (see section 4.2), but those won't help in this situation, since as a result 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` v1.6.

[†] stephanhennig@arcor.de

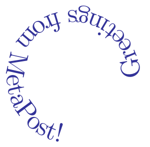

Figure 1: Our first application.

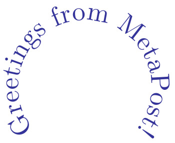

Figure 2: A refined variant.

```
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 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 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 (cf. figure 4). 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.
- 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 formulae: These are supported by a raw interface, see section 3. The standard macros don't support mathematical copy.
- rules: There's quite sophisticated support for rules in mathematical formulae, but only in the raw interface (see section 3).

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 4.2.

When a path is too short to carry all text, characters exceeding path boundaries are drawn at the boundaries, by default. Alternatively, text can be clipped or scaled automatically to fits onto the path. All configuration options of the `textpath.mp` package are described in detail in section 4.3.

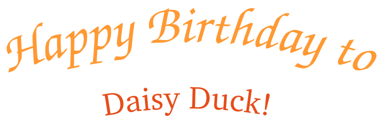

Figure 3: Using different fonts in a MetaPost graphic.

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

2 Handling different fonts

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 provides an additional macro `textpathFont` that accepts LaTeX font selection commands in an additional string argument and applies those before the `soul` package comes into play. Unfortunately, switching the font inside text still isn't possible that way.

The font selection string has to be given as the first argument. The remaining arguments are the same as for `textpath` (see figure 3).

```
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);
```

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.

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

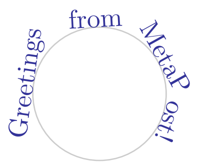

Figure 4: Transforming arbitrary chunks of characters at once using the raw interface.

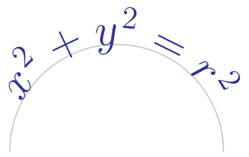

Figure 5: A formula typeset along a path.

a look at figure 4 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 see in figure 4, 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 5. 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 6. Since rules are explicitely supported by `textpathRaw` fractions are no problem.

Awesome! The rules are curved, too! Yes, since formulae can look awful otherwise the `textpath.mp` package puts some efforts into that. Section 4.4 describes an option that lets you choose between curved and straight rules.

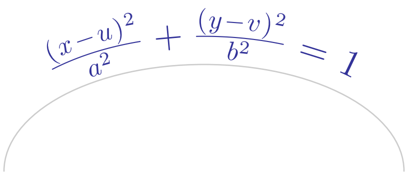

Figure 6: A formula containing fractions.

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 4.4 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 discusses all configuration variables in detail.

Since all of the global variables have been defined as `newinternal`, changing a setting locally to a group is possible using an `interim` declaration. That is, the following code *temporarily* switches clipping on inside `begingroup ... endgroup`:

```
draw textpath("A text too long to fit onto p", p, 0);
begingroup
  interim textpathClip := 1;
  draw textpath("A text too long to fit onto p", p, 0);
endgroup;
```

This is convenient since every `beginfig()` statement already opens a group.

4.1 Repeating and Shifting

4.1.1 Variable `textpathRepeat`

If you want to place multiple copies of a piece of text along a path you don't have to manually copy it in the input. Instead, just pass the original text as an argument to `textpath` and set variable `textpathRepeat` to the required number of copies before calling `textpath`. Valid numbers for `textpathRepeat` are integer values greater or equal to one. By default, `textpathRepeat` is set to one.

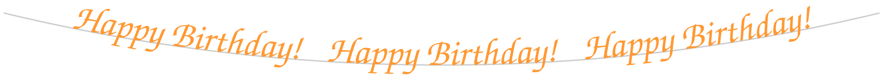

Figure 7: Repeated text manually spaced and centered.

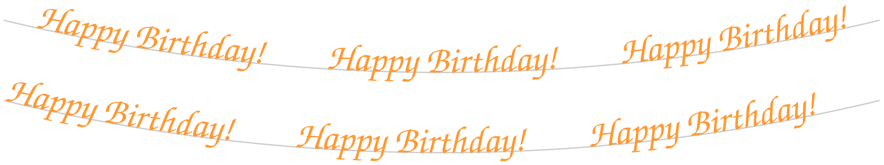

Figure 8: The same text automatically spaced, centered and left aligned.

Note, that the alignment parameter `j` still works when setting `textpathRepeat` to values greater than one. That is, for `j=0` the first text copy still starts at the beginning of a path and for `j=1` the last text copy ends at the end of a path.

4.1.2 Variables `textpathStretch` and `textpathHSpace`

When variable `textpathRepeat` is to a value greater than one, “horizontal” spacing between single copies of the text string is determined manually or automatically, depending on variable `textpathStretch`. If `textpathStretch` is set to one, the amount of space that is inserted between text copies is automatically determined, which is the default. If `textpathStretch` is set to zero variable `textpathHSpace` determines the amount of space between text copies. Note, that `textpathStretch` has only effect when variable `textpathRepeat` is set to a value larger than one.

Figures 7 and 8 show examples of manually and automatically spaced text copies. Note, that when text copies are automatically spaced 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. You shouldn’t be able to spot the start and end points of the path there other than by inspecting the code.

```
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);
```

4.1.3 Variable `textpathShift`

Variable `textpathShift` determines the “vertical” shifting of characters. That is, text can be shifted orthogonally to a path (*i.e.*, the baseline). Positive values

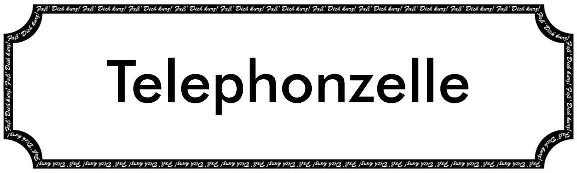

Figure 9: An attempt to resemble the style of classic labels (failing badly). At least this is an example of automatically spacing and shifting text on a path.

of `textpathShift` 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 should be both, the path our text follows and the background of the text. Therefore, we need to "vertically" center the text on the path. A small negative value for `textpathShift` serves our purposes well.

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

4.1.4 Variable `textpathLetterSpace`

When typesetting text along a path with a high curvature the resulting curved text may look squeezed or letter-spaced, depending on path shape. That's because natural inter-letter space refers to unrotated characters and is only proper near the baseline of rotated characters. On a convex shape, therefore, text tends to 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 baseline. The same effect can be seen in figure 10, *e.g.*, between characters "a" and "i".

The `textpath.mp` package provides means to manually adjust inter-letter space. Before transforming a character onto the path the `textpath.mp` package

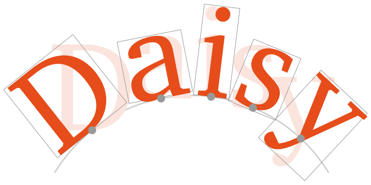

Figure 10: The anchor point of a character.

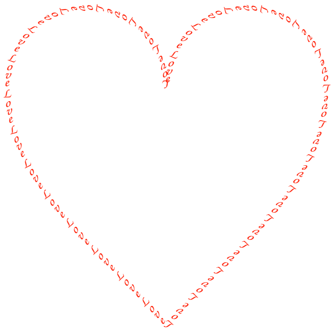

Figure 11: Text with an additional rotation of 90 degrees.

inserts an additional amount of horizontal space of width `textpathLetterSpace` between all characters. This variable is a good friend when shifting or rotating characters. Finding a good value can be a tricky task, but activating debug bit 1 may help here (see section 4.5).

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.

4.2 Rotating

4.2.1 Variable `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 left-to-right reading cultures.

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 anchor point of a character lies on the baseline at the horizontal center of its bounding box (see figure 10). The character is then rotated around its anchor point by an angle corresponding to the slope. 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 all 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;
```

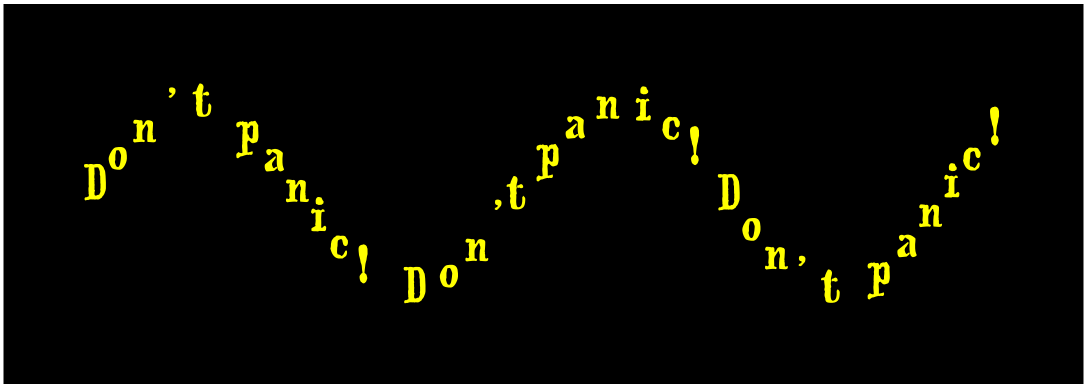

Figure 12: Text with an absolute rotation of zero degrees.

```
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;
```

4.2.2 Variable `textpathAbsRotation`

The interpretation of variable `textpathRotation` changes when variable `textpathAbsRotation` is set to a value of one. Then characters are rotated by a fixed angle of `textpathRotation` degrees, 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;
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;
```

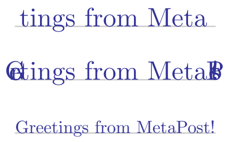

Figure 13: Text on a short horizontal path clipped, not clipped and automatically scaled.

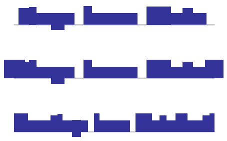

Figure 14: The same text in draft mode.

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 convenient feature is explained.

4.3 Clipping, Scaling and Draft Mode

4.3.1 Variable `textpathClip`

As explained in section 1.2 when total text width is larger than path length characters that don't fit into the path boundaries are clipped. Actually, characters whose anchor point exceeds path boundaries are moved to the corresponding end point of the path, first. Then variable `textpathClip` is asked whether those characters should be drawn or clipped. If variable `textpathClip` is set to one, such characters are clipped (first line in figure 13). If it is set to zero value the characters are drawn at the boundaries, acting as a visual marker that something went wrong (second line in figure 13). By default, `textpathClip` is set to zero, that is, no characters are lost.

Whenever the `textpath.mp` package notices text exceeding path boundaries one of the following warnings is shown on the console depending on variable `textpathClip`:

```
Package textpath warning:  Overfull path!  Clipped.
```

or

```
Package textpath warning:  Overfull path!  Not Clipped.
```

Clipping works well with `textpathRepeat` set to values larger than one.

4.3.2 Variable `textpathAutoScale`

When automatically generating documents clipping might not be an option due to potential loss of information. However, finding proper settings for all kinds of input may be a difficult—or impossible—task. Actually, when input is too long to fit onto a path the clean solution were to reflow text or rearrange the

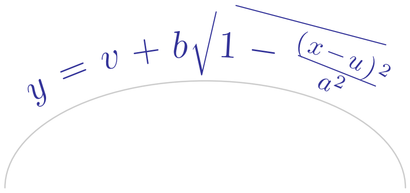

Figure 15: A formula with straight rules.

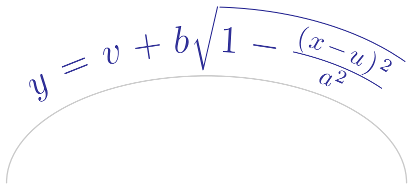

Figure 16: A formula with curved rules.

layout. If you can't afford that manual work, but still want to be sure that all text appears in the output you can instruct the `textpath.mp` package to automatically scale input so that it fits onto a path (last line in figure 13). Clearly, legibility will suffer from scaling, too. But at least all text remains in order this way.

Automatic scaling can be activated by setting variable `textpathAutoScale` to one. Variable `textpathClip` is ignored in that case. If `textpathAutoScale` is set to zero, which is the default, scaling is never applied and `textpathClip` may take effect.

Whenever text is scaled the following warning is output on the console (the scaling value might be different):

```
Package textpath warning:  Overfull path!  Scaled to 0.66368.
```

Scaling works well with `textpathRepeat` set to values larger than one.

4.3.3 Variable `textpathDraft`

So far, two ways to recognize overfull paths have been mentioned. When switching off clipping and scaling text that doesn't fit onto a path acts as a visual marker at path boundaries. Additionally, a warning is output on the console.

To further help spotting overfull paths the `textpath.mp` package provides a draft mode that can be activated by setting variable `textpathDraft` to one. In draft mode, if a path is overfull, instead of glyphs the filled bounding boxes of all glyphs are ouput. Clipping and scaling still takes place in draft mode (see figure 14). However, when scaling is activated in draft mode, input is only scaled in width so that it remains large enough to be easy to spot.

By default, `textpathDraft` is set to zero, that is, draft mode is deactivated.

4.4 Options for the raw interface

4.4.1 Variable `textpathFancyStrokes`

The remaining three variables apply only to macro `textpathRaw`. Figure 6 in section 3 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

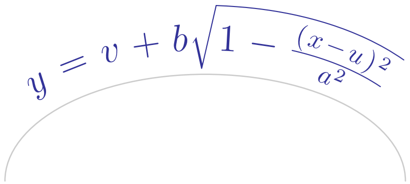

Figure 17: A curved square root without a gap.

Figure 18: Another logo that didn't fit as an example in the other sections.

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 15 and 16 show a formula similar to that in figure 6. Clearly, both, the square root and the fraction look more decent with curved strokes.

4.4.2 Variable `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.

4.4.3 Variable `textpathCureSqrt`

Did you notice the small gap that is still present in the square root operator in figure 16? 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 not a glyph, but in most cases a plain 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 a glyph (cf. figure 10). The left-most point of the rule, however, is rotated according to the slope of the path at the *right-most point* of the V part glyph. Both angles are slightly different in general and therefore, V and rule part of a square root miss each other and a gap might become visible.

For some mathematical fonts 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 16 this variable was manually set to zero to

font name	slot
`cmex10`	`112 to 116`
`cmsy5`	`112`
`cmsy6`	`112`
`cmsy7`	`112`
`cmsy8`	`112`
`cmsy9`	`112`
`cmsy10`	`112`
`cmbsy10`	`112`
`fourier-mex`	`114`
`fourier-ms`	`112`
`MnSymbolE5`	`186`
`MnSymbolE5`	`189`
`MnSymbolE10`	`186`

Table 1: Fonts supported by option `textpathCureSqrt`.

bit	output
0	item
1	item's bounding box
2	item's anchor point

Table 2: The bits of variable `textpathDebug`.

provoke the gap. In figure 17 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. Multiple root operators in *one* formula are ok, however.

Table 1 lists the font names and slots that are currently supported (known to the author to contain a root sign glyph). If you happen to use a mathematical character set that isn't supported, don't hesitate to contact me. I'd be happy to add them.

4.5 Debugging

4.5.1 Variable `textpathDebug`

There is one variable left to be described. For debugging purposes variable `textpathDebug` lets you control if and how all items, *i.e.*, glyphs and rules, are output. For every item in the input it can be controlled if the item, the item's bounding box, and the item's anchor point shall be drawn. Bits 0 to 2 of variable `textpathDebug` determine what to output. Table 2 shows the correspondence between bits in `textpathDebug` and the details to output.

Bits can be set independently. That is, for a value of, *e.g.*, $3 = 1+2 = 2^0+2^1$ all text (and rules) are drawn together with the glyph's bounding box (see figure 19), whereas for a value of $6 = 2+4 = 2^1+2^2$ bounding boxes and anchor points are drawn (see figure 20).

Bounding boxes are drawn with the `currentpen`. Anchor points are represented by filled circles with a diameter of 3bp. An exception are anchor points

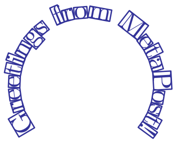

Figure 19: Debug mode showing glyphs and bounding boxes.

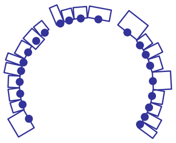

Figure 20: Debug mode showing bounding boxes and anchor points.

of fancy rules, that can't be shown, currently.

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. When drawing the bounding boxes of the characters one realizes that the left-most and right-most points of the glyph's bounding boxes on the path do line up (see, *e.g.*, figure 19).

Is there a fix? There's no automatic solution. The problem is that "G" and "!" have quite different glyph shapes. 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?

4.6 Summary

Table 3 lists all configuration variables, their default values and in what macros they have effect.

5 Numeric considerations

Internally, current MetaPost versions use 16-bit fixed point calculations. However, the largest value that can be stored in a variable is $4096 = 2^{12}$ (big points). At one point, the `textpath.mp` package needs to calculate and store the length of a path `p`. The largest radius of a full circle passed to `textpath` is therefore approx. 651bp = 9in = 22cm, a bit more than the width of paper formats letter or DIN A4. MetaPost is able to handle full circles with a larger radius, however, it cannot compute and store their path length.

To typeset text along circular paths with a larger radius, the solution is just not to pass to `textpath` a `fullcircle scaled 2r`, but to build paths with `halfcircle`, `quartercircle` or, even better, just select a minimal sub-path of, *e.g.*, a `fullcircle` with the `subpath` command.

6 Epilogue

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.

variable name	default value	valid in macros `textpath`, `textpathFont`	`textpathRaw`
`textpathRepeat`	1	✓	✓
`textpathStretch`	1	✓	✓
`textpathHSpace`	0pt	✓	✓
`textpathShift`	0pt	✓	✓
`textpathLetterSpace`	0pt	✓	
`textpathRotation`	0	✓	✓
`textpathAbsRotation`	0	✓	✓
`textpathClip`	1	✓	✓
`textpathAutoScale`	0	✓	✓
`textpathDraft`	0	✓	✓
`textpathFancyStrokes`	1		✓
`textpathStrokePrecision`	10		✓
`textpathCureSqrt`	1		✓
`textpathDebug`	1	✓	✓

Table 3: Summary of configuration variables.

What is the nice calligraphic font you're using throughout this manual? Since we're mostly dealing with decorative graphics here, URW Zapf Chancery Medium Italic is used in many exampless. This font is part of the PSNFSS package and should already be installed in your `texmf` tree. Other decorative fonts with ready-made LaTeX support are Brush ScriptX Italic or the fonts from the Emerald package. All 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 baseline legibility decreases fastly. A decorative, but elegant look can be achieved best with large fonts and just slightly indicating a curved baseline, similar to figure 3.

Happy TEXing!
Stephan Hennig