pst-text: Manipulate text and characters

Herbert Voß*

November 6, 2006

Abstract

PostScript principally does not know lines in the proper meaning of the word, but only paths and those can have any arbitrary form. Along such paths arbitrary text may be arranged. The package pst-text supports the setting of text along a path and other character manipulations, where several characters naturally result in a text again of course.

It should be noted that the correct result is not guaranteed with every DVI-PS driver. This package was written for Rokicki's dvips programme, which is practically part of every TEX distribution.

Contents

*Thanks to Lars Kotthoff and Geoff Mercer for translating this documentation!

1 Text manipulations

The package `pst-text` principally defines only one macro.

`\pstextpath[<position>](<x,y>){<graphic object>}{<text>}`

<position> specifies the alignment of the text referring to the path.

- **l** text starts at the beginning of the path (default).
- **c** text is aligned symmetrically to the middle of the path.
- **r** text ends at the end of the path.

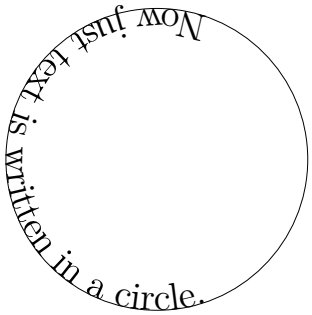
As a basic principle it is to be kept in mind that when the text is longer than the path this option has no effect since the path is filled with text and any overflowing text disappears.

<x,y> is an offset and denotes the values by which the particular characters shall be translated in x and y direction relative to the path. `(<x,y>)` have to be cartesian coordinates as the support for special coordinates allowed by `PSTricks` is not possible here. The dimensions of x and y refer to the current scale. The default is `(0,\TPoffset)`, where `TPoffset` is set to a length of `-0.7ex`.

<graphic object> any arbitrary object which creates a path.

<text> the text to set, which may only consist of alphanumeric characters. No macros are possible within the text, but the text may be put into a `\parbox`.

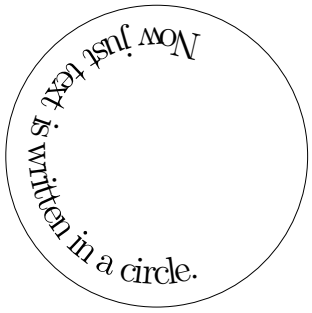
PostScript does not reserve any space for the output, so that the current text is overwritten if corresponding white space has not been provided by TEX. This can be achieved with a vertical feed (`\vspace`) or with a `pspicture` environment.




```
\begin{pspicture}(-2,-2.5)(2,2.5)
\psset{linewidth=0.2pt}
\pstextpath[c](0,0){\pscircle{2}}%
    {\Large Now just text is written in a
      circle.}
\end{pspicture}
```

This first example shows the relatively easy use of the macro. If the path is not required to be drawn the line style can be set to `none`. The following example shows the use of the offset option. It is clear that every single character is translated, because the beginning and the end of the text stayed the same.

Since the text was written in a circle, a positive specification for `TPoffset` causes a translation towards the centre of the circle.



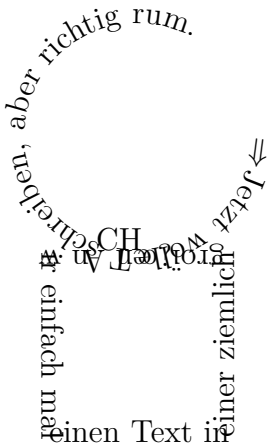

```
\begin{pspicture}(-2,-2.5)(2,2.5)
\psset{linewidth=0.2pt}
\pstextpath[c](0,2ex){\pscircle{2}}%
   {\Large Now just text is written in a
     circle.}
\end{pspicture}
```

1.1 Examples

With `\pscustom` one is offered unlimited possibilities for paths. The following example uses the circle again, but forms an eight, which is composed of four circle parts to get a continuous path. In the second example a square has been appended to a circle. The starting point of the path is always the circle at 0, here marked by ⇒.

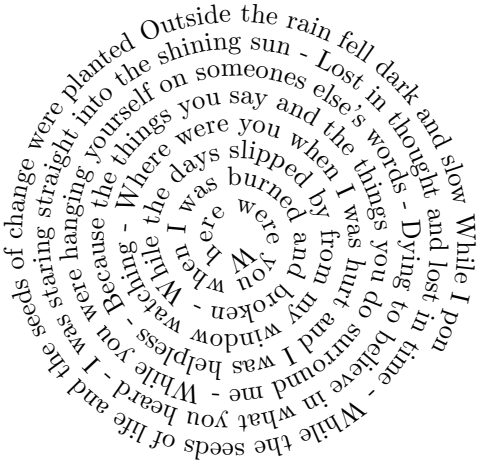

```
\psset{unit=0.75,linestyle=none}
\begin{pspicture}(-2,-4)(2,4)
\pstextpath[l](0,0){%
   \pscustom{
      \psarcn(0,2){2}{0}{-90}
      \psarc(0,-2){2}{90}{0}
      \psarc(0,-2){2}{0}{90}
      \psarcn(0,2){2}{-90}{0}
   }%
}{\large $\Rightarrow$Now we are writing some
   nonsense text which appears in
  a large eight in the correct direction.}
\end{pspicture}
```

It can be easily seen that in the above example the upper circle is larger than the lower. This is because the text is always written on the path, which faces towards the inner on the upper circle and towards the outer on the lower circle (or square) due to the change in direction.




```
\begin{pspicture}(-2,-3.25)(2,3.25)
\psset{linestyle=none}
\pstextpath[l](0,0){%
   \pscustom[unit=0.75]{
      \psarcn(0,2){2}{0}{-90}
      \pspolygon(0,0)(-1.7,0)(-1.7,-3.4)
        (1.7,-3.4)(1.7,0)(0,0)
      \psarcn(0,2){2}{-90}{0}
   }%
}{\large $\Rightarrow$Jetzt wollen wir
  einfach mal einen Text in einer
  ziemlich groïżœn ACHT schreiben,
  aber richtig rum.}
\end{pspicture}
```

The setting of the text along a path is very memory and calculation intensive on the PostScript side, so that with longer texts some seconds may pass until the desired result appears even on faster computers. This is shown in the following example, where the text is the beginning of the song “Into the shining sun” by Pink Floyd. Note how the text is truncated as the path is too short to fit the entire text in.




```
%\usepackage{pst-plot}

\begin{pspicture}(-3,-3)(3,3)
\psset{linestyle=none}
\pstextpath[l](0,0){%
  \parametricplot[plotstyle=curve,%
     plotpoints=500]{0}{3000}{%
       /r {t 1000 div} def t sin r mul t cos r mul }
```

```
}{
Where were you when I was burned and broken -
While the days slipped by from my window watching -
Where were you when I was hurt and I was helpless -
Because the things you say and the things you do surround me -
While you were hanging yourself on someones else's words -
Dying to believe in what you heard -
I was staring straight into the shining sun -

Lost in thought and lost in time -
While the seeds of life and the seeds of change were planted
Outside the rain fell dark and slow
While I pondered on this dangerous but irresistible pastime
}
\end{pspicture}
```

2 Character manipulations

With character manipulations the same issue with the DVI-PS driver applies, namely that the results are only guaranteed for Rokicki's dvips programme.

2.1 \pscharpath

Although this macro has a name similar to \pstextpath, it has a completely different meaning.

```
\pscharpath[<parameters>]{<text>}
\pscharpath*[<parameters>]{<text>}
```

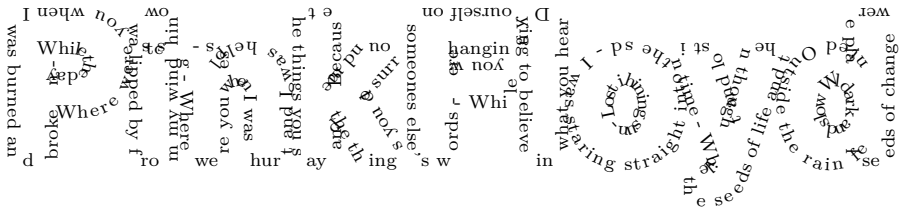
<parameters> All PSTricks parameters, insofar as they make sense, may be specified here.

<text> The text to set, which may only consist of alphanumeric characters, therefore no macros are possible within the text.

Normally, one will define ones own font size, which is best done with \DeclareFixedFont, since this macro is very fast because it simply sets the size without having to look up any font tables.


```
\DeclareFixedFont{\RM}{T1}{ptm}{b}{n}{2cm}
\pscharpath{\RM TeXnik}\\
\pscharpath[linecolor=lightgray]{\RM TeXnik}\\
\psset{fillstyle=gradient,gradbegin=red,gradend=cyan}
\pscharpath[gradangle=90]{\RM TeXnik}\\
\pscharpath[linestyle=none,gradangle=-90]{\RM TeXnik}
```

Normally the path, here the outer line of the characters, is deleted after the macro **\pscharpath** has ended. With the asterisk version it is preserved and can be used for other "baubles", for instance for **\pstextpath**, where the saved path can be used as input for the text to set.




```
\DeclareFixedFont{\SF}{T1}{phv}{b}{n}{2.5cm}
\pstextpath(0,-1ex){%
  \pscharpath*[linestyle=none]{\SF Pink Floyd}}{
    \scriptsize < ... Text ... >
  }
```

2.2 \pscharclip

\pscharclip is practically identical to **\pscharpath** with the only difference being that it sets the clipping path to the current path.

```
\pscharclip[<parameters>]{<text>} %  TeX example
 ...
\endpscharclip
\pscharclip*[<parameters>]{<text>}%  TeX example
 ...
\endpscharclip
\begin{pscharclip}[<parameters>]{<text>} %  LaTeX example
 ...
\end{pscharclip}
```

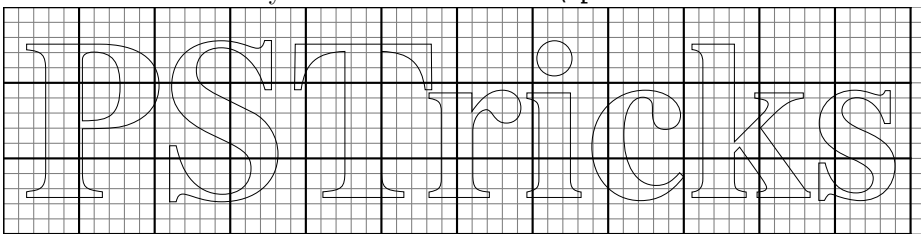
```
\begin{pscharclip*}[<parameters>]{<text>}%  LaTeX example
 ...
\end{pscharclip}
```

Using this one can "write" *within* a font, whereas it is not really easy to get the "base" congruent. How to deal with this best shall is shown in the following worked example.

The base is best formed as a `minipage`, thus enabling it to be moved to arbitrary spots. To have clear coordinates on one hand and only the interesting area shown on the other hand, one uses a `pspicture*` environment. Let us presume that we use a font size of 3cm and want to use the width of the whole page.

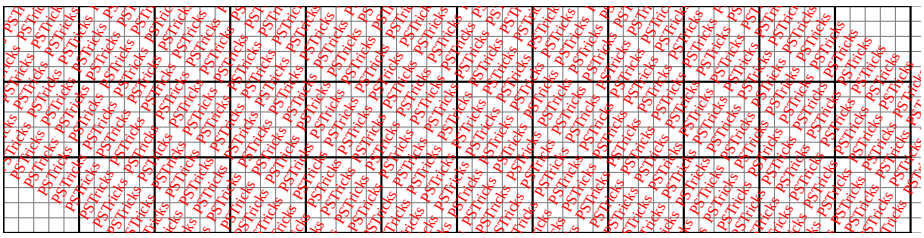
```
\begin{pspicture*}(\linewidth,3cm)
...
\end{pspicture}
```

The text can be set exactly into the centre with a `\rput` instruction.




```
\DeclareFixedFont{\RM}{T1}{ptm}{b}{n}{3cm}
\begin{pspicture*}(\linewidth,3cm)
  \psgrid%
  \begin{pscharclip}[linewidth=0.1pt]{%
    \rput(0.5\linewidth,1.5){\RM PSTricks}}%
  \end{pscharclip}
\end{pspicture*}
```

The text "lying below" the font is put into a `minipage` of the width `\linewidth`. Since this text underlies the clipping path, it does not matter how long it really is, the essential thing is that the the whole area is covered. This is especially important when the text is further manipulated such as rotated. So one may view the following example with `\begin{minipage}{\linewidth}`.



```
\DeclareFixedFont{\Rm}{T1}{ptm}{m}{n}{2mm}
\begin{pspicture*}(\linewidth,3cm)
  \psgrid%
  \rput{60}(0.5\linewidth,1.5){%
    \begin{minipage}{0.6\linewidth}
       \setstretch{0.5}
       \color{red}
       \multido{\i=1+1}{500}{\Rm PSTricks }
    \end{minipage}%
  }
\end{pspicture*}
```

Both of these can be overlaid where, because of the clipping path, only the inner of the large letters seems transparent. In the second example the `minipage` has been additionally rotated, the line colour was ignored and the line spacing within the `minipage` was halved (package `setspace`).

```
\DeclareFixedFont{\RM}{T1}{ptm}{b}{n}{3cm}
\DeclareFixedFont{\Rm}{T1}{ptm}{m}{n}{2mm}
\begin{pspicture*}(\linewidth,3cm)
  \begin{pscharclip}[linewidth=0.1pt]{%
    \rput(0.5\linewidth,1.5){\RM PSTricks}}%
  \rput{60}(0.5\linewidth,1.5){%
    \begin{minipage}{0.6\linewidth}
       \setstretch{0.5}
       \color{red}
       \multido{\i=1+1}{500}{\Rm PSTricks }
    \end{minipage}%
  }
  \end{pscharclip}
\end{pspicture*}

```

```
\begin{pspicture*}(\linewidth,3cm)
  \begin{pscharclip}[linewidth=0.1pt,linestyle=none]{%
    \rput(0.5\linewidth,1.5){\RM PSTricks}}%
    \rput{-60}(0.5\linewidth,1.5){%
      \begin{minipage}{0.6\linewidth}
         \setstretch{0.5}
         \multido{\i=1+1}{500}{\Rm  PSTricks }
      \end{minipage}%
  }
  \end{pscharclip}
\end{pspicture*}
```

Generally it is not of interest what one bases `\pscharclip` on. Using things sucha as a graphic can lead to some interesting possibilities. It should be kept in mind that alternatively `pscharpath` may be used in conjunction with `psboxfill`.

References

[1] Denis Girou. Présentation de PSTricks. *Cahier GUTenberg*, 16:21–70, April 1994.

[2] Michel Goosens, Frank Mittelbach, and Alexander Samarin. *The LaTeX Graphics Companion*. Addison-Wesley Publishing Company, Reading, Mass., 1997.

[3] Nikolai G. Kollock. *PostScript richtig eingesetzt: vom Konzept zum praktischen Einsatz*. IWT, Vaterstetten, 1989.

[4] Herbert Voß. *PSTricks – Grafik für TeX und LaTeX*. DANTE – Lehmanns, Heidelberg/Hamburg, third edition, 2006.

[5] Timothy Van Zandt. *PSTricks - PostScript macros for generic TeX*. http://www.tug.org/application/PSTricks, 1993.

[6] Timothy Van Zandt. *multido.tex - a loop macro, that supports fixed-point addition*. CTAN:/graphics/pstricks/generic/multido.tex, 1997.

[7] Timothy Van Zandt. *pst-coil: Coils and zigzags*. CTAN:graphics/pstricks/generic/, 1999.

[8] Timothy Van Zandt and Denis Girou. Inside PSTricks. *TUGboat*, 15:239–246, September 1994.

Index