9. Tricks with Text

In our discussions so far, we’ve been focusing on graphic objects and we’ve treated text only incidentally in Chapter 6, as labels in pictures. We now see how text can be manipulated in various ways using PSTricks.
9.1. Boxing Text

\LaTeX{} has various macros for putting text in boxes (or putting boxes around text) and PSTricks defines its own boxing macros. The advantage of using these is the ease of adorning these boxes using colors, shadows and so on. The simplest of such commands is the \texttt{\psframebox} as in the example below:

\begin{verbatim}
\psframebox[fillstyle=solid,\%
fillcolor=Cyan,\%
linestyle=RoyalBlue]\%
{\color{Red} \LARGE\bfseries Text In A Box}
\end{verbatim}

The distance between the sides of the box and the enclosed text is controlled by the \texttt{framesep} parameter. By default, its value is 3 point, but as with other parameters, can be set to any desired value, as shown in the next example:

\begin{verbatim}
\psframebox[framesep=10pt,\%
fillstyle=solid,\%
fillcolor=Cyan,\%
linecolor=RoyalBlue]\%
{\color{Red} \LARGE\bfseries Text In A Box}
\end{verbatim}

A variant of the \texttt{\psframebox} is the \texttt{\psdbox} which, as the name indicates, doubles each line of the frame.
Recall that the \texttt{doublesep} parameter determines the width of the space between the double lines and the \texttt{doublesep} the color of this space, as mentioned in Chapter 3. The default value of \texttt{doublesep} for the \texttt{psdblframebox} is \texttt{pslinewidth} and the default value of \texttt{doublecolor} is white.

Another variant is the \texttt{psshadowbox} which, obviously enough, draws a (single) frame with a shadow, as shown below:

\begin{verbatim}
\psshadowbox[framesep=10pt,\
fillstyle=solid,\
fillcolor=Cyan,\
linecolor=RoyalBlue,\
doublecolor=Apricot,\
doublesep=3pt]\
{\color{Red}\
\LARGE\bfseries\
Text In A Shadow Box}
\end{verbatim}

Note that the parameters \texttt{shadowsize} and \texttt{shadowcolor} are discussed in Chapter 3.

If you are tired of plain old rectangular boxes, you can try \texttt{psdiabox}, which draws a diamond shaped box:
or \texttt{\pstribox}, which draws a triangular box:

\pstribox[fillstyle=gradient,\
gradbegin=CornflowerBlue,\
gradend=Apricot,\
gradmidpoint=1,\
linecolor=Cyan]\
{\color{Red}\
\large\bfseries\
\renewcommand{\arraystretch}{1.2}\
\begin{tabular}{c}\
Text\\
In A Triangle Box\
\end{tabular}}

Recall that the \texttt{gradient} style of filling requires the \texttt{pst-grad} package, as explained in Chapter 2.

For those who are inclined towards curves than angles, there's a \texttt{\psccirclebox}:
Another parameter for the various boxes is the boxsep whose default value is true. In this case, the box that is produced (in the \TeX\nical sense) is the size of the “frame” around it. If it is set to false, then the box produced is the size of what’s inside, so that the frame is transparent to \TeX. This is apparent only when the boxes are used within some surrounding text as illustrated below:
Thus we find that $x+y=3$ and using this together with $x^2+y^2=3$ found earlier, we see that $x=2$ and $y=1$.

Each of the boxing commands above has a starred version, which draws a solid shape around the enclosed text instead of just a frame. This is similar to the starred versions of graphic objects we’ve seen earlier, but the color of the boxes is determined by `fillcolor` instead of `linecolor` for other graphic objects.
\SaveVerb{box}=\psframebox=
\psframebox*[fillstyle=solid,\
fillcolor=Cyan\]%
{\color{Red}
 \LARGE\bfseries
 Text In A }\UseVerb{box}\}
\vspace{1cm}

\SaveVerb{starbox}=\psframebox*=
\psframebox*[fillstyle=solid,\
fillcolor=Cyan\]%
{\color{Red}
 \LARGE\bfseries
 Text In A }\UseVerb{starbox}\}

(Here, the command pair \SaveVerb and \UseVerb come from the package fancyvrb and are used to get the control sequence strings \psframebox and \psframebox* as arguments of the commands.)

We’ve mentioned in Chapter 6 that the \rput* command puts the text first in a \psframebox*. But there are occasions when we’ve to use both \rput and \psframebox together instead of a single \rput*, as in the example below:
Note that here, we cannot use \texttt{\rput*} directly for the label $h$, since the default color of \texttt{\psframebox*} is white, (which is OK for the label $r$) but we want the color of the box for $h$ to be cyan, to blend it with its background.

While on the topic of “putting”, we should also mention the command \texttt{\cput} (and of course \texttt{\cput*}) which combines the functions of \texttt{\pscirclebox} and \texttt{\rput} (or \texttt{\rput*}), as shown in the next example:
\Large
\psset{fillstyle=solid,unit=2cm}
\begin{pspicture}(0,-1)(1,1)
  \SpecialCoor
  \pspolygon[fillcolor=Cyan]%
    (0,1)(1;210)(1;330)
  \cput*[fillcolor=Red]%
    (0,1)
    {\color{White} $A$}
  \cput*[fillcolor=Green]%
    (1;210)
    {\color{White} $B$}
  \cput*[fillcolor=Blue]%
    (1;330)
    {\color{White} $C$}
\end{pspicture}
9.2. Clipping

We can clip text, that is, cut off everything outside a specified boundary, using the \psclip....\endpsclip commands. A simple example is given below:

\psclip{\
\psdiamond[/linecolor=Red,\
  /fillstyle=solid,\
  /fillcolor=Yellow]\
  (2,0.25)(2.5,0.5)}\
\color{Blue}\Huge Cut Diamond
\endpsclip

By careful use of coordinates, we can create an overlay effect with suitable clipping, as in the next example:

\begin{pspicture}(0,0)(6,2)\
\rput[bl](0.5,1){\
\color{Blue}\Huge Cut Diamond}\
\psclip{\psdiamond[/linecolor=Yellow]\
  (3,1.25)(2.25,0.5)}\
\rput[bl](0.5,1){\
\color{OliveGreen}\Huge Cut Diamond}\
\endpsclip\
\end{pspicture}

Or special effects like this:
Here, the custom style `bluestyle` and the custom curve \texttt{tearcurve} are defined as follows:

\begin{verbatim}
\newpsstyle{bluestyle}{
    linecolor=Blue,\
    fillstyle=solid,\
    fillcolor=Cyan}
\newcommand{\tearcurve}{
    \pscurve(1,0)(1.1,0.3)(1.2,1)(1.3,1)%
    (1.4,1.2)(1.6,1.6)}
\end{verbatim}
Incidentally, note that the \psclip...\endpsclip commands can be used to clip not only text, but graphic objects also, as shown in the example below:

\begin{pspicture}(0,-7)(6.5,2)
\coloraxes(0,0)(0,-2)(7,2)
\plotsqsin
\plotabssin
\psclip{\ycirc(3.1416,0){0.5}}
\coloraxes(0,0)(0,-2)(7,2)
\plotsqsin
\plotabssin
\endpsclip
\dotline(2.6416,0)(1.1416,-4)
\dotline(3.6416,0)(5.1416,-4)
\psclip{\ycirc(3.1416,-4){2}}
\psset{origin={3.1416,4},unit=1.33cm}
\coloraxes(0,0)(0,-2)(7,2)
\plotsqsin
\plotabssin
\endpsclip
\end{pspicture}

where the various customized commands used are as follows:

\newcommand{\plotsqsin}{%
  \psplot[\plotpoints=500,\plotstyle=curve,\linecolor=Blue]{0}{6.2832}{57.2958 x mul sin 2 exp}}%
\newcommand{\plotabssin}{%
  \psplot[\plotpoints=500,\plotstyle=curve,\linecolor=Green]{0}{6.2832}{57.2958 x mul sin abs}}%
\definecolor{PaleYellow}{cmyk}{0,0,0.2,0}
\newpsobject{ycirc}{pscircle}{\fillstyle=solid,\fillcolor=PaleYellow}
\newpsobject{dotline}{psline}{\linestyle=dotted,\dotsep=1pt}
\newpsobject{coloraxes}{psaxes}{\linestyle=solid,\linecolor=Apricot,\labels=none,\ticks=none}
9.3. Rotating and scaling

There are are also ready to use commands for rotating text left, right or down, leaving the needed amount of spaces.

\begin{code}
\texttt{go straight}
\texttt{\rotateright\{\color{Red} go down\}}
\texttt{\rotatedown\{\color{Green} turn upside down\}}
\texttt{\rotateleft\{\color{Blue} go up\}}
\texttt{go straight again}
\end{code}

A better effect can be obtained by computing the heights of various upright boxes and raising and lowering them by the appropriate lengths:

\begin{code}
\newlength{\dlen}
\settoheight{\dlen}{\rotateright\{\color{Red} go down\}}
\newlength{\ulen}
\settoheight{\ulen}{\rotateleft\{\color{Blue} go up\}}
\begin{code}
\begin{code}
\texttt{go straight}
\texttt{\raisebox{-\dlen}{\rotateright\{\color{Red} go down\}}}
\texttt{\rotatedown{\color{Green} turn upside down}}
\texttt{\rotateleft\{\color{Blue} go up\}}
\texttt{\raisebox{\ulen-\dlen}{go straight again}}
\end{code}
\end{code}
\end{code}

For such manipulation of long pieces of text, these commands also have the “environmental” forms \texttt{\begin{Rotateleft}...\end{Rotateleft}} and others.
\textbf{Question}: Why did the tachyon cross the street?

\begin{Rotatedown}
\parbox{\linewidth}{
\textbf{Answer}: Because it's already on the other side}
\end{Rotatedown}

Text can also be \textit{scaled}, using the command \texttt{\scalebox}. The general form of this command is

\texttt{\scalebox{number1 number2}{text}}

where \texttt{number1} is the horizontal scaling and \texttt{number2} is the vertical scaling. If only one number is specified, it is used for scaling in both directions. This is illustrated in the examples below:

\scalebox{0.8 4}{% 
\color{Red}tall and lean\}

\scalebox{4 0.8}{% 
\color{Green}short and fat\}

\scalebox{2}{% 
\color{Blue}large but proportional\}

Using negative numbers for scaling, we can flip text around either axis:
We also have the \scaleboxto command with the general form

\scaleboxto(number1,number2){text}

With this command text is scaled to have width number1 units and height plus depth equal to number2 units. If one of the numbers is set to 0, then the box is scaled to have width and height (plus depth) equal to the other number. (Of course, we cannot set both numbers equal to 0).

\scaleboxto(1.5,1){\color{Red} tall and lean}
\bigskip
\scaleboxto(7,0.2){\color{Green} short and fat}
\bigskip
\scaleboxto(3,0){\color{Blue} small but proportional}
9.4. Text along a path

One of the interesting features of the PostScript language is that it treats text as graphical object. This allows various manipulations of text. The package \texttt{pst-text} provides the command \texttt{\pstextpath} to set text along a specified path. Look at this example:

\begin{verbatim}
\begin{pspicture}(0,0)(3,2)
  \pstextpath{%
    \psframe[framearc=0.3,linewidth=Blue](0,0)(3,1.4)}{%
    \color{Red}\Large Now we have text going around a box}
\end{pspicture}
\end{verbatim}

Now we have text going around a box

Note that the general form of the command \texttt{\pstextpath} is

\begin{verbatim}
\pstextpath{graphic}{text}
\end{verbatim}

where, \textit{graphic} specifies the path along which the specified \textit{text} is to be set.

By default, \texttt{\pstextpath} draws also the graphic specified, but this can be suppressed by setting \texttt{linestyle=none}, as shown below:

\begin{verbatim}
\begin{pspicture}(-3,-1)(3,5)
  \colaxes[labels=none](0,0)(-3,-1)(3,5)
  \pstextpath{%
    \psplot[linestyle=none]{-2}{2}{4 x 2 exp sub}%%%%
    \color{Red} This is the graph of the equation $y=4-x^2$ for $-2 \leq x \leq 2$}
\end{pspicture}
\end{verbatim}

This is the graph of the equation $y=4-x^2$ for $-2 \leq x \leq 2$
(Note that \colaxes used in the above example is a custom command, discussed in Chapter 7).

What if we need something like this?

\begin{pspicture}(-3,-1)(3,5)
\colaxes[labels=none](0,0)(-3,-1)(3,5)
\psset{linecolor=Blue}
\psplot{-2}{2}{4 x 2 exp sub}
\psset{linestyle=none,unit=1.12cm}
\pstextpath{\psplot{-2}{2}{4 x 2 exp sub}}{This is the graph of the equation \( y = 4 - x^2 \) for \(-2 \leq x \leq 2\)}
\end{pspicture}

The trick is to first draw the curve and then use \pstextpath to set the text along a slightly scaled up version of the curve, without actually drawing the second curve:
But this is not exactly what we want. The trouble is that the command \texttt{\textbackslash pstextpath}, by default, places the beginning of the text at the beginning of the path; however, it has an optional parameter which can be used to shift the position of the text:

\begin{pspicture}(-3,-1)(3,5)
\colaxes[labels=none](0,0)(-3,-1)(3,5)
\psset{linecolor=Blue}
\psplot{-2}{2}{4 x 2 exp sub}
\psset{linestyle=none,xunit=1.13cm,yunit=1.05cm}
\psplot{-2}{2}{4 x 2 exp sub}{
\color{Red} This is the graph of the equation $y=4-x^2$ for $-2 \leq x \leq 2$}
\end{pspicture}

This is the graph of the equation $y=4-x^2$ for $-2 \leq x \leq 2$

Note how we used the optional value \texttt{c} to center the text relative to the curve. (Note also the slight difference in \texttt{xunit} and \texttt{yunit} to get the text at the top just right). Other optional values are \texttt{l} (the default) for left justification and \texttt{r} for right justification. These are illustrated in the next example:
\begin{pspicture}(-3,-1)(3,5)
  \colaxes[labels=none](0,0)(-3,-1)(3,5)
  \psset{linecolor=Blue}
  \psplot{-2.2}{2.2}{4 x 2 exp sub}
  \psset{unit=1.15cm,linestyle=none}
  \pstextpath[l]{\psplot{-2.1}{2.1}{4 x 2 exp sub}}{\color{Red}\textit{increasing}}
  \pstextpath[r]{\psplot{-2.1}{2.1}{4 x 2 exp sub}}{\color{Red}\textit{decreasing}}
  \psset{unit=1.07cm}
  \pstextpath[c]{\psplot{-2.1}{2.1}{4 x 2 exp sub}}{\color{Red}\textit{turning}}
\end{pspicture}
9.5. Text as graphic

The package \texttt{pst-char} provides the command \texttt{\textbf{pscharpath}} which can be used to embellish text with colors and the like just as it were a graphic object. We give a couple of examples to illustrate this:

\begin{pspicture}(0,-1)(8,2)
\DeclareFixedFont{\bigrm}{T1}{ptm}{m}{n}{1.5cm}
\pscharpath[fillstyle=solid,\linecolor=Red\fillcolor=SkyBlue,]
{\bigrm PSTricks}
\end{pspicture}

Here, the command \texttt{\textbf{DeclareFixedFont}} is the \LaTeX{} way of specifying the font to be used.

\begin{pspicture}(0,-1)(8,2)
\DeclareFixedFont{\bigsf}{T1}{phv}{b}{n}{1.5cm}
\pscharpath[linecolor=Yellow,\linecolor=Red\fillcolor=SkyBlue,gradbegin=Yellow,gradend=Red,gradmidpoint=1,gradangle=5]{\bigsf PSTricks}
\end{pspicture}

This package also contains the command \texttt{\textbf{pscharclip}}...\texttt{endpscharclip}, which like \texttt{\textbf{psclip}}...\texttt{endpsclip} pair, clips any object within them, but this time to the shape of the specified text:
(Here, the text to be clipped is “PostScript” written 500 times, in small font, specified by \texttt{smallrm}, which is generated by the code starting with \texttt{newcounter}, put in a box 8 centimeters wide and turned through ninety degrees.)

As in the case of \texttt{psclip} this can also be used to produce an overlay effect.
Here, the command `\firstpara` is defined by

\newcommand{\firstpara}{%
   \scriptsize
   \LaTeX\ has only limited drawing capabilities, while
   PostScript is a page description language which has a rich set of
   drawing commands; and there are programs (such as `\textsf{dvips}`)
   which translate the `\texttt{dvi}` output to PostScript. So, the
   natural question is whether one can include PostScript code in a
   \LaTeX\ source file itself for programs such as `\textsf{dvips}` to
   process after the `\LaTeX` compilation? This is the idea behind the
   `\textsf{PSTricks}` package of Timothy van Zandt. The beauty of it is
   one need not know PostScript to use it---the necessary PostScript
   code can be generated by \LaTeX\ macros defined in the package}

just typesets the opening paragraph of our tutorial in `\tiny` font in an
8 centimeter wide box.