Chapter 11

Custom styles and objects

PSTricks allows to define own new styles for assistance and new macros which allow arbitrary area borders for filling and clipping.

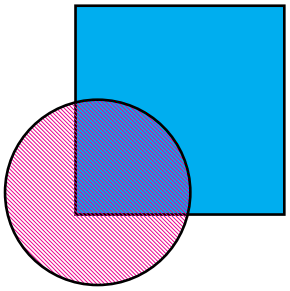
11.1 Custom styles

If certain combinations of parameters (→ 2.2 on p. 21) are often used repeatedly, one may define special styles for them.

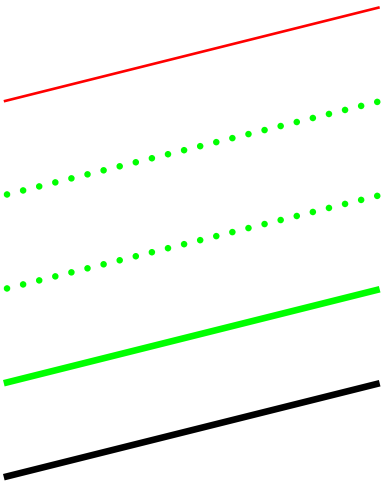
\newpsstyle{*name*}{*list of parameters*}	\addtopsstyle{*name*}{*list of parameters*}

This new style can be passed on to a macro by using the keyword `style` when passing the parameters. Another useful application can be found in chapter 7.2 on p. 87.

Exa 11-1-1



```
\usepackage{pstricks}
\newpsstyle{TransparentMagenta}{%
  fillstyle=vlines,hatchcolor=magenta,
  hatchwidth=0.1\pslinewidth,hatchsep=1\pslinewidth}
\begin{pspicture}(3,3)
  \psframe[fillstyle=solid,fillcolor=cyan](0.75,0.75)(3,3)
  \pscircle[style=TransparentMagenta](1,1){1}
\end{pspicture}
```



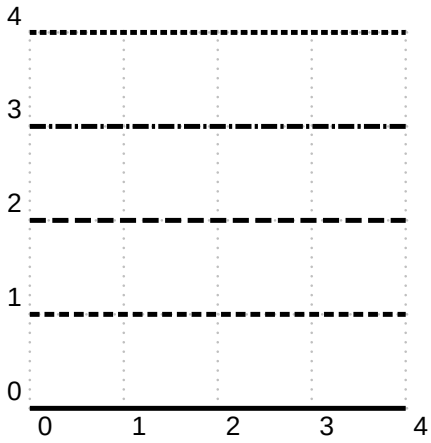
Exa 11-1-2

```
\usepackage{pstricks}
\newpsstyle{Fiber}{linewidth=2pt}
\begin{pspicture}(4,5)
   \psline[style=Fiber](0,0)(4,1)
   \addtopsstyle{Fiber}{linecolor=green}
   \psline[style=Fiber](0,1)(4,2)
   \addtopsstyle{Fiber}{linestyle=dotted}
   \psline[style=Fiber](0,2)(4,3)
   \addtopsstyle{Fiber}{}
   \psline[style=Fiber](0,3)(4,4)
   \addtopsstyle{Fibber}{linecolor=red}
   \psline[style=Fibber](0,4)(4,5)
\end{pspicture}
```

11.2 Custom objects

The styles described in the previous section may also be assigned to a special macro, which makes the application even easier. In the example below the new macro `\dashedV` is assigned a special style; the macro is based on the object `\psline`.

`\newpsobject{`*name*`}{`*name of object*`}{`*list of parameters*`}`



Exa 11-2-1

```
\usepackage{pstricks}
\newpsobject{LineA}{psline}{linewidth=1.5pt}
\newpsobject{LineB}{LineA}{linestyle=dashed,
  dash=3pt 1.5pt}
\newpsobject{LineC}{LineB}{dash=5pt 2pt}
\newpsobject{LineD}{LineB}{dash=5pt 1pt 1pt 1pt}
\newpsobject{LineE}{LineB}{dash=2pt 1pt}
\begin{pspicture}[showgrid=true](4,4)
\LineA(0,0)(4,0)\LineB(0,1)(4,1)\LineC(0,2)(4,2)
\LineD(0,3)(4,3)\LineE(0,4)(4,4)
\end{pspicture}
```

11.3 `\pscustom`

PSTricks offers many ways of creating graphical objects. Nevertheless cases may occur where none of the existing macros satisfies one's needs. In all these cases `\pscustom` can be of great help; the starred version fills the background with the current line colour as usual.

`\pscustom*` `[`*Optionen*`]` `{`*arbitrary code*`}`

\pscustom expects that an arbitrary closed path is created through polylines or polycurves of whatever form. \pscustom starts a new path; the last closing braces terminates it. The closed path may then be filled or tiled arbitrarily with a colour or pattern through the filling function. One does not necessarily have to create line or curve segments which connect to each other. Usually they are automatically amended by \pscustom with lines to result in a closed polycurve.

PSTricks provides special PostScript commands which are used frequently by PSTricks itself as special PSTricks compatible versions. The advantage is that no additional use of the \special command is necessary. These additional macros can **only** be used in connection with \pscustom. All of them are described below.

The macros listed in this section influence the PostScript output virtually without any control by PSTricks and should therefore only be used with at least basic knowledge of PostScript as a programming language. Because of the PostScript-specific code the current scale does not apply for the macros listed here. Only the common PostScript unit bp (big points) or pt is valid.

The macro \pscustom uses \pstverb (→ ?? on p. ??) and \pstunit (→ 2.3 on p. 22) which write \special into the dvi file. The length of the argument is system-specific; problems may arise when putting many small polycurves together within \pscustom because these are all arguments of a single \special command.

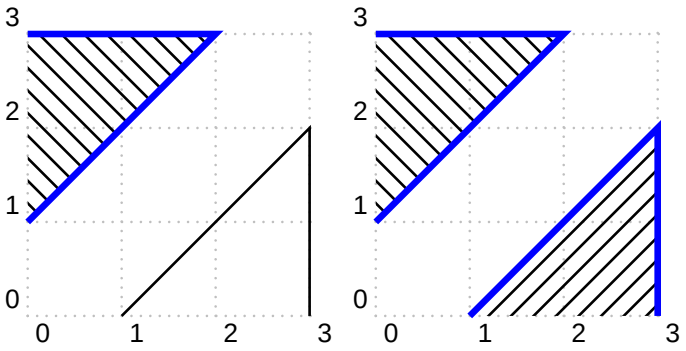
All PSTricks objects which are defined through \begin@SpecialObj...\end@SpecialObj may not appear within the argument of \pscustom. PSTricks will output a corresponding error message in this case.

11.3.1 Parameters

Because \pscustom refers to a **single** closed path there can only be parameters referring to this path. The next example executes \psline outwith and within \pscustom. As can be seen easily from the coordinates, \psline within \pscustom should fill the lower triangle; this does not happen though because the so-called fill parameters linewidth, linecolor, and fillstyle have no effect within \pscustom. There are some exceptions, for example putting arrows. This will be pointed out separately in the examples below.

```
\usepackage{pstricks}

\begin{pspicture}[showgrid=true](3,3)
 \psline[linewidth=2pt,linecolor=blue,fillstyle=vlines](0,1)(2,3)(0,3)
 \pscustom{\psline[linewidth=2pt,linecolor=blue,fillstyle=hlines](1,0)(3,2)(3,0)}
\end{pspicture}\qquad
\begin{pspicture}[showgrid=true](3,3)
 \psline[linewidth=2pt,linecolor=blue,fillstyle=vlines](0,1)(2,3)(0,3)
 \pscustom[linewidth=2pt,linecolor=blue,fillstyle=hlines]{\psline(1,0)(3,2)(3,0)}
\end{pspicture}
```



Exa 11-3-1

As can be seen from the example above, the specification of parameters within \pscustom (left example) does not have any effect on the figure; in contrast to the correct example where the parameters are specified through the optional argument of \pscustom. It is therefore advisable to not use any parameters with the macros within \pscustom as a rule.

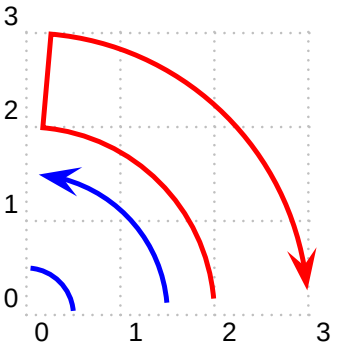
The line styles dashed and dotted can be a problem because they do not know anything about an existing path at the beginning. In these cases \pscustom should be given the line style listed in table 4.2 on p. 52 as a parameter.

The following parameters are **not** available within the \pscustom macro – shadow, border, doubleline, showpoints. The parameters origin and swapaxes only affect \pscustom itself.

11.3.2 Open and closed curves

PSTricks distinguishes between closed curves and open curves. As \pscustom is used itself to create closed polylines and polycurves, it makes little sense to use closed curves within \pscustom.

The actual primary focus of \pscustom is connecting open lines and curves. In principle there will always be a direct line drawn from the end of the line/curve drawn last to the start of the next line or curve if it defines an **end arrow**. This can be seen clearly in the following example where in the first curve a connection from the first arc to the second one because it has an end arrow.



Exa 11-3-2

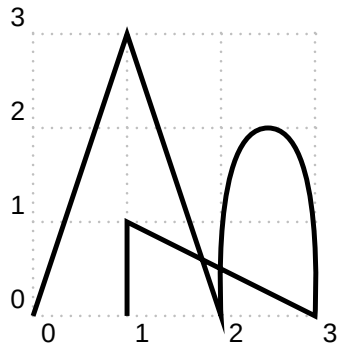
```
\usepackage{pstricks}
\begin{pspicture}[showgrid=true](3,3)
 \psset{linewidth=1.5pt,arrowscale=2}
 \pscustom[linecolor=red]{%
  \psarc(0,0){2}{5}{85}\psarcn{->}(0,0){3}{85}{5}}
 \pscustom[linecolor=blue]{%
  \psarc(0,0){0.5}{5}{85}\psarcn{<-}(0,0){1.5}{85}{5}}
\end{pspicture}
```

The above example assumes that the arrow is defined locally because it is only valid for a single polycurve here.

The macros \psline, \pscurve, and \psbezier start at the current point when their parameters are „incomplete“. It will always be set to the coordinate origin by

PSTricks outwith \pscustom for \psline and \psbezier. In the following example this behaviour results in a polyline whereas without \pscustom two independent lines starting at the coordinate origin are drawn and nothing for \pscurve because if less than three coordinate pairs are specified here, no curve is drawn.

Exa 11-3-3




```
\usepackage{pstricks}
\begin{pspicture}[showgrid=true](3,3)
  \pscustom[linewidth=1.5pt]{%
    \psline(1,3)\psline(2,0)
    \pscurve(2.5,2)(3,0)
    \psline(1,1)(1,0)}
\end{pspicture}
```

The following graphical objects are **not** available within \pscustom – \psgrid, \psdots, \qline, \qdisk. Closed lines or curves should **not** be part of \pscustom; unexpected side effects may occur.

11.3.3 liftpen

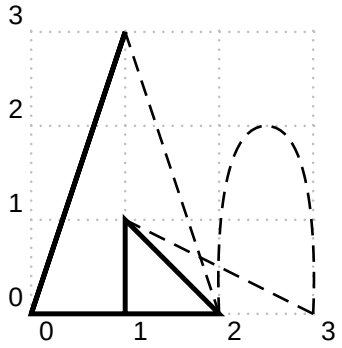
The parameter liftpen is an exceptional tool to control the connection of several partial lines or curves within \pscustom.

Table 11.1: Meaning of the liftpen parameter

value	*meaning*
0	If a new polyline or polycurve does not start at the current point, a line is drawn from the current point to the starting point of the line or curve (default behaviour). If there is a line or curve with incomplete coordinates they are amended by the current point.
1	The current point is not taken into account when the specification of coordinates is incomplete; instead the coordinate origin is taken (only for \psline and \psbezier).
2	Single polylines or polycurves are treated as individual units; they do not use the current point as starting point (with incomplete coordinates) and no line is drawn from the current point to the starting point of the next object.

The next example is formally equivalent to the previous one; the difference is that the \psline macros now have liftpen=1 as parameter and therefore the second line does not use the end point of the first curve as current point, but the coordinate origin. No \pscurve is drawn in this case because for this curve the incomplete coordinates (at least three pairs) are not amended with liftpen=1. This affects the current point which remains at $(2, 0)$; therefore the connecting line to the new starting point $(1, 1)$ of the last

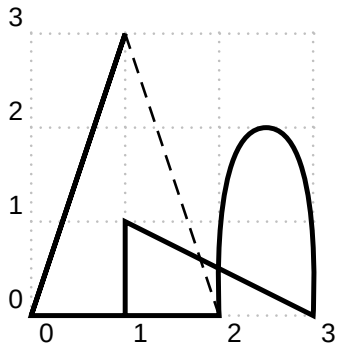
\psline command starts from there.



Exa 11-3-4

```
\usepackage{pstricks}
\begin{pspicture}[showgrid=true](3,3)
  \pscustom[linewidth=1.5pt]{%
    \psline(1,3)\psline[liftpen=1](2,0)
    \pscurve[liftpen=1](2.5,2)(3,0)
    \psline(1,1)(1,0)}
  \psline[linestyle=dashed](1,3)(2,0)
  \pscurve[linestyle=dashed](2,0)(2.5,2)(3,0)
  \psline[linestyle=dashed](3,0)(1,1)
\end{pspicture}
```

The same example with complete coordinates for \pscurve yields:-



Exa 11-3-5

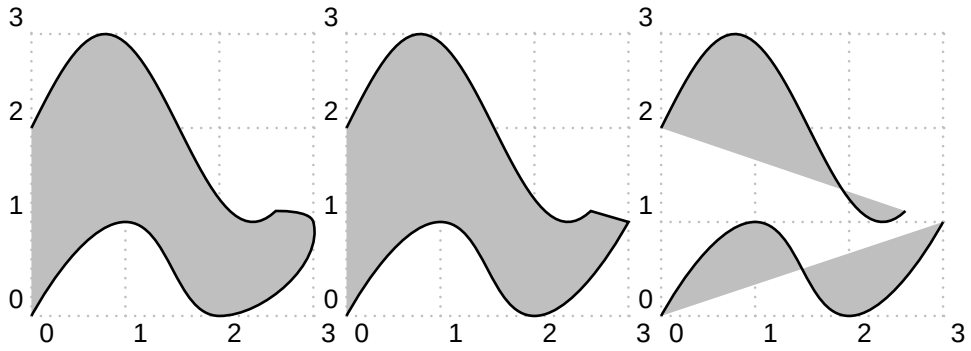
```
\usepackage{pstricks}
\begin{pspicture}[showgrid=true](3,3)
  \pscustom[linewidth=1.5pt]{%
    \psline(1,3)\psline[liftpen=1](2,0)
    \pscurve[liftpen=1](2,0)(2.5,2)(3,0)
    \psline(1,1)(1,0)}
  \psline[linestyle=dashed](1,3)(2,0)
\end{pspicture}
```

To illustrate this slightly complicated matter further, another example will be given which compares the behaviour for the different values of liftpen.

```
\usepackage{pstricks,pst-plot}

\begin{pspicture}[showgrid=true](3,3)
\pscustom[fillcolor=lightgray,fillstyle=solid]{%
  \psplot{0}{2.6}{x RadtoDeg 2 mul sin 2 add}
  \pscurve(3,1)(2,0)(1,1)(0,0)}
\end{pspicture}\quad
\begin{pspicture}[showgrid=true](3,3)
  \pscustom[fillcolor=lightgray,fillstyle=solid]{%
    \psplot{0}{2.6}{x RadtoDeg 2 mul sin 2 add}
    \pscurve[liftpen=1](3,1)(2,0)(1,1)(0,0)}
\end{pspicture}\quad
\begin{pspicture}[showgrid=true](3,3)
  \pscustom[fillcolor=lightgray,fillstyle=solid]{%
    \psplot{0}{2.6}{x RadtoDeg 2 mul sin 2 add}
    \pscurve[liftpen=2](3,1)(2,0)(1,1)(0,0)}
\end{pspicture}
```

Exa 11-3-6



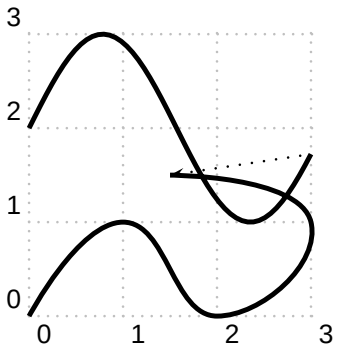
The left example (liftpen=0) uses the end point of the first curve (\psplot) as starting point of the next curve (\pscurve). The example in the middle (liftpen=1) does **not** use the end point of the first curve (\psplot) as the starting point of the next curve (\pscurve); a connecting line from the end point of the first curve to the starting point of the second curve is drawn however. The right example (liftpen=2) neither the end point of the first curve (\psplot) is used as the starting point of the second curve (\pscurve), nor a connecting line is drawn; two individual units are created.

11.3.4 \moveto

Moves the current point to the new coordinates (x, y) without drawing a line.

\moveto(*x*,*y*)

Exa 11-3-7

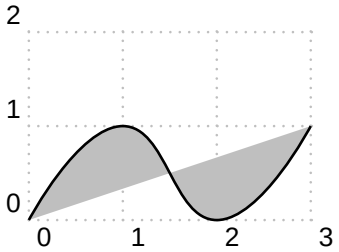


```
\usepackage{pstricks,pst-plot}
\SpecialCoor
\begin{pspicture}[showgrid=true](3,3)
  \pscustom[linewidth=1.5pt]{%
    \psplot{0}{3}{x 180 mul 1.57 div sin 2 add}
    \moveto(1.5,1.5)\pscurve(3,1)(2,0)(1,1)(0,0)}
  \psline[linestyle=dotted]{->}%
      (! 3 dup 180 mul 1.57 div sin 2 add)(1.5,1.5)
\end{pspicture}
```

11.3.5 \newpath

Its use is completely identical to the PostScript command newpath. The current path is deleted and a new one started; all information about the old path is lost. In the following example the first curve is not drawn for this reason.

Exa 11-3-8

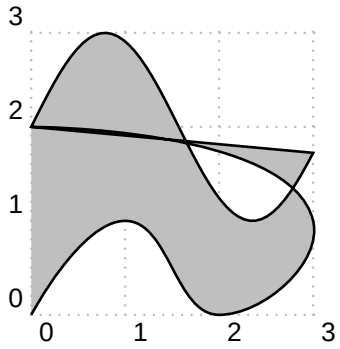


```
\usepackage{pstricks,pst-plot}
\begin{pspicture}[showgrid=true](3,2)
 \pscustom[fillcolor=lightgray,fillstyle=solid]{%
  \psplot{0}{3}{x 180 mul 1.57 div sin 2 add}
  \newpath
  \pscurve(3,1)(2,0)(1,1)(0,0) }
\end{pspicture}
```

11.3.6 \closepath

\closepath is the counterpart to \newpath and its use is completely identical to the PostScript command closepath as well. The current path is closed by connecting the starting point and the end point. When using \moveto many different parts may exist; they are all treated separately. The starting point is made the current point; after that all information about the old path is lost.

The following example determines the starting point $(0, 2)$ as the new current point after the application of \closepath; the next curve looks completely different because it uses this point as starting point.



Exa 11-3-9

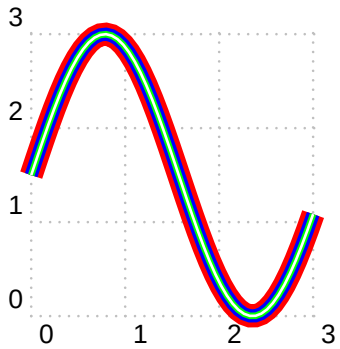
```
\usepackage{pstricks,pst-plot}
\begin{pspicture}[showgrid=true](3,3)
  \pscustom[fillcolor=lightgray,
      fillstyle=solid]{%
    \psplot{0}{3}{x 180 mul 1.57 div sin 2 add}
    \closepath
    \pscurve(3,1)(2,0)(1,1)(0,0)}
\end{pspicture}
```

11.3.7 \stroke

\stroke makes it possible to pass special parameters to single parts within the \pscustom macro and to draw the previously created polyline or polycurve.

\stroke[*Optionen*]

\stroke does not replace the \stroke caused by PSTricks itself in the PostScript code at the end of a macro. This means that one has to choose at least a larger line width to avoid overwriting. The next example illustrates how to create special lines from several colours in an easy manner.



Exa 11-3-10

```
\usepackage{pstricks,pst-plot}
\begin{pspicture}[showgrid=true](3,3)
  \pscustom[linecolor=white]{%
    \psplot{0}{3}{x 180 mul 1.57 div sin
        1.5 mul 1.5 add}
    \stroke[linecolor=red,linewidth=7pt]
    \stroke[linecolor=blue,linewidth=4pt]
    \stroke[linecolor=green,linewidth=2pt]}
\end{pspicture}
```

11.3.8 \fill

\fill, analogous to \stroke, makes it possible to pass special parameters to single parts within the \pscustom macro and to draw the polyline or polycurve created previously.

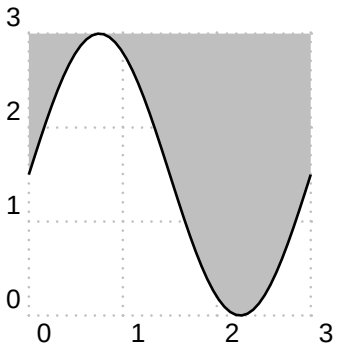
\fill[*Optionen*]

\fill does not replace the \fill caused by PSTricks in the PostScript code at the end of a macro. As \pscustom fills the area with the current fill colour and fill style at the end anyway, possible applications of \fill are limited.

11.3.9 \gsave and \grestore

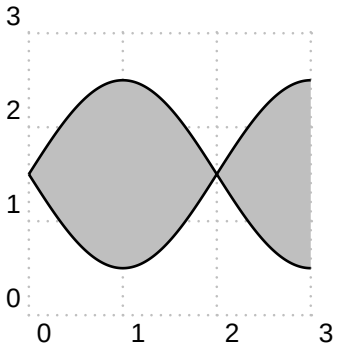
\gsave makes it possible to save the current PostScript stack referring to the graphical output (path specifications, colour, line width, coordinate origin, etc.). \grestore restores all these values. The next example fills an area with \gsave/\grestore without making the borders visible through lines.

Exa 11-3-11




```
\usepackage{pstricks,pst-plot}
\begin{pspicture}[showgrid=true](3,3)
  \pscustom{%
    \psplot{0}{3}{x 180.0 mul 1.5 div
      sin 1.5 mul 1.5 add}
    \gsave
      \psline(3,3)(0,3)% is _not_ drawn
      \fill[fillcolor=lightgray,fillstyle=solid]
    \grestore }
\end{pspicture}
```

Exa 11-3-12



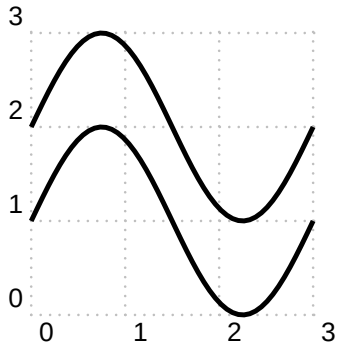

```
\usepackage{pst-plot}\SpecialCoor
\begin{pspicture}[showgrid=true](3,3)
  \pstVerb{/rad {180.0 mul 2 div} def}
  \pscustom[plotpoints=200]{%
    \psplot{0}{3}{x rad sin 1.5 add}
    \gsave
    \psline(! 3 dup rad sin 1.5 add)%
          (!3 dup rad sin neg 1.5 add)
    \psplot{3}{0}{x rad sin neg 1.5 add}
    \fill[fillcolor=lightgray,fillstyle=solid]
    \grestore
    \psplot[liftpen=2]{3}{0}{x rad sin neg 1.5 add}}
\end{pspicture}
```

The macros \gsave and \grestore may only appear **in pairs**, but arbitrarily nested.

11.3.10 \translate

\translate moves the coordinate origin to (x, y) for all subsequent graphical operations.

`\translate(`x, y`)`



Exa 11-3-13

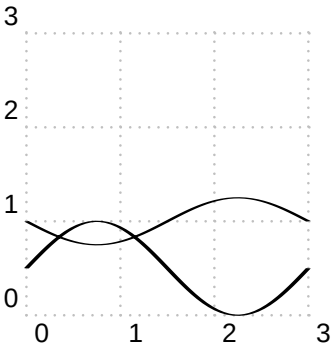
```
\usepackage{pstricks,pst-plot}
\begin{pspicture}[showgrid=true](3,3)
  \pscustom[linewidth=1.5pt]{%
    \translate(0,1)
    \psplot{0}{3}{x 180.0 mul 1.5 div sin}
    \translate(0,1)
    \psplot[liftpen=2]{0}{3}{x 180.0 mul 1.5 div sin}}
\end{pspicture}
```

11.3.11 \scale

In contrast with other length assignments, `\scale` only takes dimensionless values which are internally assumed to have unit `pt`.

`\scale{`*value1* *value2*`}`

Scales the `\pscustom` object with *value1* horizontally and with *value2* vertically; if no second value is specified, the object is scaled proportionally with *value1*. As can be seen from the example, negative values are possible as well; this corresponds to scaling and subsequent mirroring.



Exa 11-3-14

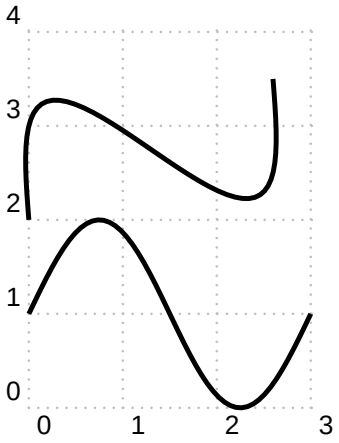
```
\usepackage{pstricks,pst-plot}
\begin{pspicture}[showgrid=true](3,3)
  \pscustom[linewidth=1.5pt]{%
    \scale{1 0.5}
    \translate(0,1)
    \psplot{0}{3}{x 180.0 mul 1.5 div sin}
    \translate(0,1)
    \scale{1 -0.5}
    \psplot[liftpen=2]{0}{3}{x 180.0 mul 1.5 div sin}}
\end{pspicture}
```

11.3.12 \rotate

`\rotate{`*angle in degrees*`}`

Rotates the `\pscustom` object by the specified angle which has to be specified in degrees to be compliant with PostScript.

Exa 11-3-15

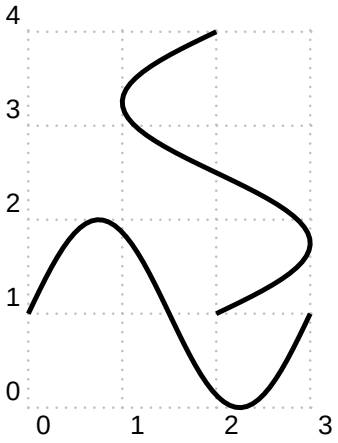



```
\usepackage{pstricks,pst-plot}
\begin{pspicture}[showgrid=true](3,4)
  \pscustom[linewidth=1.5pt]{%
    \translate(0,1)
    \psplot[liftpen=2]{0}{3}{
      x 180.0 mul 1.5 div sin}
    \translate(0,1)
    \rotate{30}
    \psplot[liftpen=2]{0}{3}{
      x 180.0 mul 1.5 div sin}
  }
\end{pspicture}
```

11.3.13 \swapaxes

An example for this has been given already in chapter 3 on p. 31; only it was a parameter there. In any case only the x-y axes are swapped, which is equivalent to `\rotate{-90} \scale{-1 1}`.

Exa 11-3-16

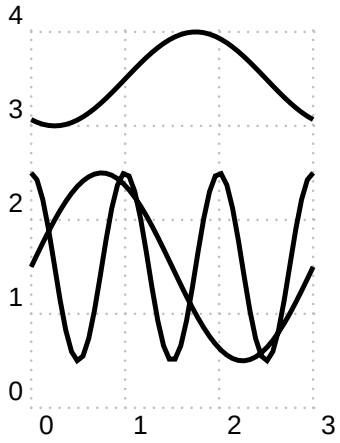



```
\usepackage{pstricks,pst-plot}
\begin{pspicture}[showgrid=true](3,4)
  \pscustom[linewidth=1.5pt]{%
    \translate(0,1)
    \psplot{0}{3}{x 180.0 mul 1.5 div sin}
    \translate(2,0)
    \swapaxes
    \psplot[liftpen=2]{0}{3}%
      {x 180.0 mul 1.5 div sin}}
\end{pspicture}
```

11.3.14 \msave and \mrestore

These commands can be used to save and restore the currently valid coordinate system. In contrast to `\gsave` and `\grestore` all other parameters such as line type, line width, etc. are not affected by this.

The example draws the first sine function with the coordinate origin at `\translate(0,1.5)`; after that the state of the coordinate system is saved, a new origin is set with `\translate(1,2)`, and another sine function is drawn. Based on the current origin `(0,1.5)` a `\translate(1,2)` corresponds to the absolute coordinates $(0, 3.5)$. Afterwards the old state is restored with the macro `\mrestore`; the coordinate origin is at `(0,1.5)` again and the following cosine function refers to it again.



Exa 11-3-17

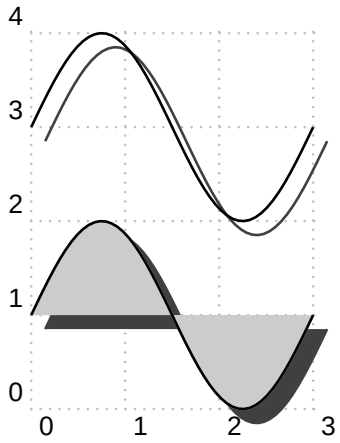
```
\usepackage{pstricks,pst-plot}
\begin{pspicture}[showgrid=true](3,4)
  \pscustom[linewidth=1.5pt]{%
    \translate(0,1.5)
    \psplot{0}{3}{x 180.0 mul 1.5 div sin}%
    \msave
      \translate(1,2) \scale{1 0.5}
      \psplot[liftpen=2]{-1}{2}%
        {x 180.0 mul 1.5 div sin}
    \mrestore
    \psplot[liftpen=2]{0}{3}%
      {x 180.0 mul 0.5 div cos}}
\end{pspicture}
```

The macros \msave and \mrestore may only occur **in pairs**, but can be arbitrarily nested with themselves or with \gsave and \grestore. The nesting has to be balanced pairwise though.

11.3.15 \openshadow

\openshadow creates a copy of the current path using the specified shadow-parameters. In contrast to \closedshadow the original path is not filled with a colour by default; it stays "'open"'. Filling it can be done by the user though.

\openshadow[*Optionen*]



Exa 11-3-18

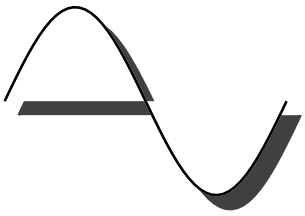
```
\usepackage{pstricks,pst-plot}
\begin{pspicture}[showgrid=true](3,4)
  \pscustom{%
    \translate(0,3)
    \psplot{0}{3}{x 180.0 mul 1.5 div sin}
    \openshadow[shadowsize=6pt]}
  \pscustom[fillcolor=black!20,fillstyle=solid]{%
    \translate(0,1)
    \psplot{0}{3}{x 180.0 mul 1.5 div sin}
    \openshadow[shadowsize=6pt]}
\end{pspicture}
```

11.3.16 \closedshadow

\closedshadow creates a shadow of the ares surrounded by the current path as if it was a non-transparent environment.

\closedshadow[*Optionen*]

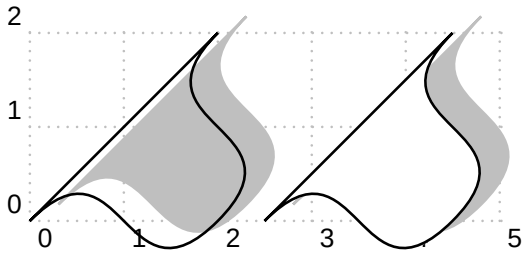
Exa 11-3-19



```
\usepackage{pstricks,pst-plot}
\begin{pspicture}(3,2)
  \pscustom{\translate(0,1)
    \psplot{0}{3}{x 180.0 mul 1.5 div sin}
    \closedshadow[shadowsize=6pt]}
\end{pspicture}
```

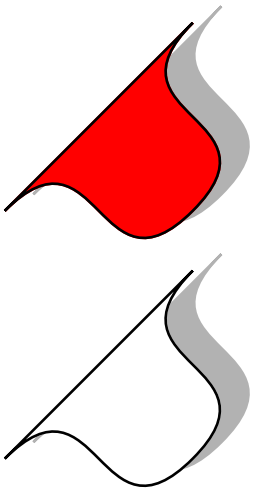
One has to keep in mind how the shadow is created. PSTricks simply copies the closed path, translates it corresponding to the specifications of shadowsize and shadowangle, fills it with shadowcolor, and fills the old path again with fillcolor, which is set to white by default. If the second filling is suppressed (corresponds to \openshadow), it results in the complete shadow copy shown in the following example, in contrast to the correct result next to it on the right.

Exa 11-3-20



When specifying a fillcolor with the options of \pscustom which is different from white, one has to pass the right fill colour to the macro \closedshadow as well.

Exa 11-3-21

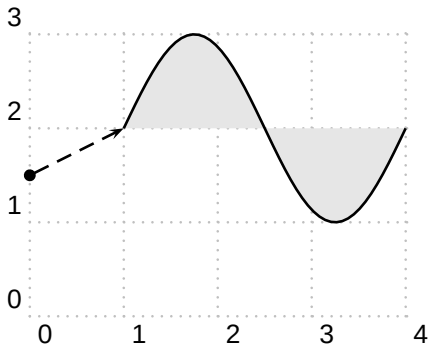


```
\usepackage{pstricks}
\begin{pspicture}(0,-0.25)(2.5,2)
\pscustom[shadowcolor=black!30,fillcolor=red]{%
  \psbezier(0,0)(1,1)(1,-1)(2,0)\psbezier(3,1)(1,1)(2,2)
  \closepath
  \closedshadow[shadowsize=10pt,shadowangle=30]}
\end{pspicture}\\[10pt]
\begin{pspicture}(0,-0.25)(2.5,2)
\pscustom[shadowcolor=black!30,fillcolor=red]{%
  \psbezier(0,0)(1,1)(1,-1)(2,0)\psbezier(3,1)(1,1)(2,2)
  \closepath
  \closedshadow[shadowsize=10pt,shadowangle=30,
    fillcolor=white]}
\end{pspicture}
```

11.3.17 \movepath

Translates the current path by (dx, dy); after that, it is lost. \gsave and \grestore can be used to save and restore the original path however.

`\movepath(`dx, dy`)`



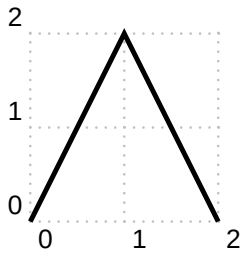
Exa 11-3-22

```
\usepackage{pstricks,pst-plot}
\begin{pspicture}[showgrid=true](4,3)
  \pscustom[fillcolor=black!10,
    fillstyle=solid]{%
     \translate(0,1.5)
     \psplot{0}{3}{x 180.0 mul 1.5 div sin}
     \movepath(1,0.5)}
 \psline[linestyle=dashed]{*->}(0,1.5)(1,2)
\end{pspicture}
```

11.3.18 \lineto

`\lineto` corresponds to `\psline(`x,y`)` in principle, but always draws a line to (x,y) starting from the current point (which therefore has to exist).

`\lineto(`x,y`)`



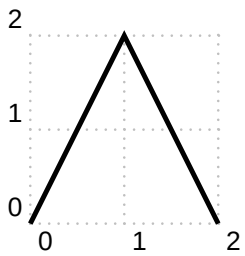
Exa 11-3-23

```
\usepackage{pstricks}
\begin{pspicture}[showgrid=true](2,2)
  \pscustom[linewidth=1.5pt]{%
    \psline(0,0)(1,2)\lineto(2,0)}
\end{pspicture}
```

11.3.19 \rlineto

`\rlineto` corresponds to `\lineto(`x,y`)` in principle; only it is a relative translation from the current point.

`\rlineto(`dx, dy`)`



Exa 11-3-24

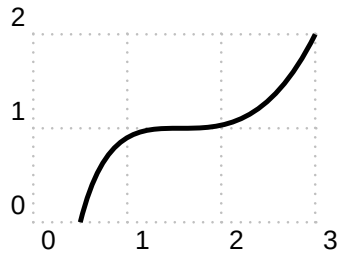
```
\usepackage{pstricks}
\begin{pspicture}[showgrid=true](2,2)
  \pscustom[linewidth=1.5pt]{%
    \psline(0,0)(1,2)\rlineto(1,-2)}
\end{pspicture}
```

11.3.20 \curveto

`\curveto` corresponds to `\psbezier(`x_1,y_1`)(`x_2,y_2`)(`x_3,y_3`)`in principle; the current point is assumed to be the first one.

\curveto(x_1,y_1)(x_2,y_2)(x_3,y_3)

Exa 11-3-25



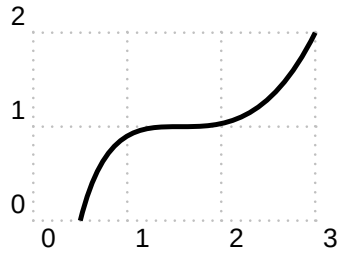

```
\usepackage{pstricks}
\begin{pspicture}[showgrid=true](3,2)
  \pscustom[linewidth=1.5pt]{%
      \moveto(0.5,0)\curveto(1,2)(2,0)(3,2)}
\end{pspicture}
```

11.3.21 \rcurveto

\rcurveto corresponds to \curveto(x_1,y_1)(x_2,y_2)(x_3,y_3)in principle, only **all** are assumed to be coordinates relative to the current point.

\rcurveto(dx_1, dy_1)(dx_2, dy_2)(dx_3, dy_3)

Exa 11-3-26



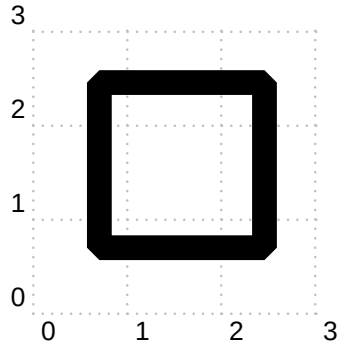

```
\usepackage{pstricks}
\begin{pspicture}[showgrid=true](3,2)
  \pscustom[linewidth=1.5pt]{%
      \moveto(0.5,0)
      \rcurveto(0.5,2)(1.5,0)(2.5,2)}
\end{pspicture}
```

11.3.22 \code

\code inserts the PostScript code given as parameter directly into the PostScript output. This macro is identical to \addto@pscode and should be preferred over using \special in **all** cases. Another example for \code can be found in section **??** on p. ??.

\code{*PostScript code*}

Exa 11-3-27

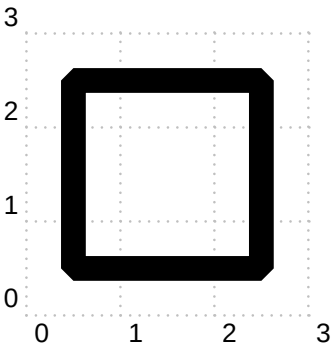



```
\usepackage{pstricks}
\begin{pspicture}[showgrid=true](3,3)
  \pscustom{%
    \code{%
      newpath
      20 20 moveto 0 50 rlineto
      50 0 rlineto 0 -50 rlineto
     -50 0 rlineto closepath
      2 setlinejoin 7.5 setlinewidth stroke}}
\end{pspicture}
```

11.3.23 \dim

\dim converts the current PSTricks unit to pt, such that one can do calculations in the units specified by the pspicture environment, and puts the result on the PostScript stack.

\dim{*value unit*}



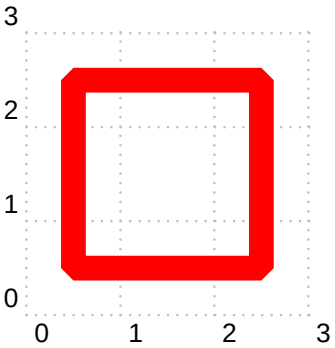
Exa 11-3-28

```
\usepackage{pstricks}
\begin{pspicture}[showgrid=true](3,3)
  \pscustom{%
    \code{newpath}
    \dim{0cm}\dim{-2cm} \dim{2cm}\dim{0cm}
    \dim{0cm}\dim{2cm}  \dim{0.5cm}\dim{0.5cm}
    \code{
      moveto rlineto rlineto rlineto closepath
      2 setlinejoin 7.5 setlinewidth
      stroke}}
\end{pspicture}
```

11.3.24 \coor

\coor converts the specified coordinates from the PSTricks unit to pt, such that one can do calculations in the units specified by the pspicture environment, and puts the result on the PostScript stack. Using \coor with several coordinates has a clear advantage over using \dim. \coor internally uses the macro \pst@@getcoor{*#1*} which is called recursively when more than one coordinate pair is supplied.

\coor$(x_1,y_1)(x_2,y_2)\ldots(x_n,y_n)$



Exa 11-3-29

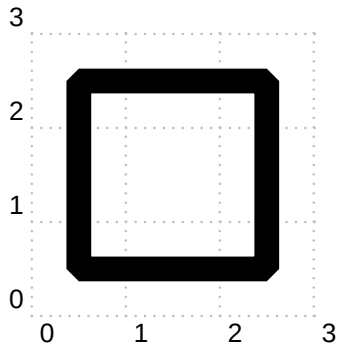
```
\usepackage{pstricks}
\begin{pspicture}[showgrid=true](3,3)
  \pscustom[linecolor=red]{%
    \code{newpath}\coor(0,-2)(2,0)(0,2)(0.5,0.5)
    \code{ moveto rlineto rlineto rlineto
      closepath
      2 setlinejoin 7.5 setlinewidth stroke}}
\end{pspicture}
```

11.3.25 \rcoor

\rcoor is in principle identical to \coor except that the coordinates are put on the stack in reverse order (reverse coor).

\rcoor$(x_1,y_1)(x_2,y_2)\ldots(x_n,y_n)$

Exa 11-3-30




```
\usepackage{pstricks}
\begin{pspicture}[showgrid=true](3,3)
  \pscustom{%
    \code{newpath}\rcoor(0.5,0.5)(0,2)(2,0)(0,-2)
    \code{ moveto rlineto rlineto rlineto
      closepath
      2 setlinejoin 7.5 setlinewidth stroke}}
\end{pspicture}
```

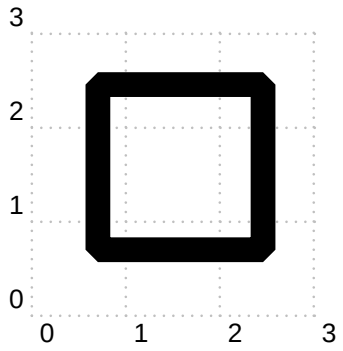
11.3.26 \file

\file inserts the contents of a file without any expansion as PostScript code. Only comment lines starting with "'%"' are ignored.

\file{*filename*}

The following example reads the contents of the file file.ps previously written with filecontents and executes the corresponding commands.

Exa 11-3-31




```
\usepackage{pstricks}
\begin{filecontents*}{file.ps}
% file "file.ps",  demo for \file  hv 2007-01-01
newpath 20  20 moveto   0 50 rlineto 50 0 rlineto
 0 -50 rlineto -50 0 rlineto closepath
2 setlinejoin 7.5 setlinewidth stroke
\end{filecontents*}
\begin{pspicture}[showgrid=true](3,3)
  \pscustom{\file{data/file.ps}}
\end{pspicture}
```

11.3.27 \arrows

\arrows defines the type of the beginning and the end of the line or curve to insert.

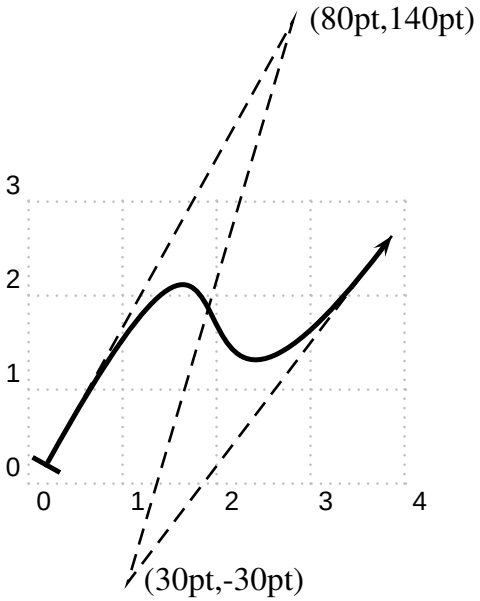
\arrows{*arrow type*}

Internally the two PostScript procedures ArrowA and are used; parameters are to be given as follows:-

```
x2 y2 x1 y1 ArrowA
x2 y2 x1 y1 ArrowB
```

Both draw an arrow from (x_2,y_2) to (x_1,y_1). ArrowA sets the current point to the end of the arrow and leaves (x_2,y_2) on the stack. ArrowB on the other hand does not change

the current point, but leaves the four values x2 y2 x1' z1' on the stack, where (x_1', y_1') is the point where a line or curve connects.



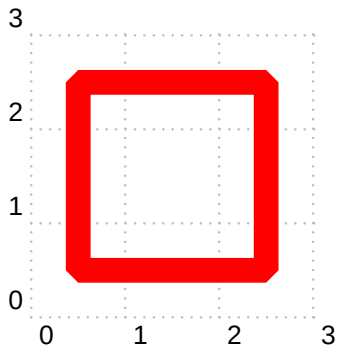
Exa 11-3-32

```
\usepackage{pstricks}
\SpecialCoor
\begin{pspicture}[showgrid=true](4,3)
 \pscustom[linewidth=1.5pt]{%
  \arrows{|->}
  \code{%
     80 140 5 5 ArrowA    % 80 140 on the stack
     30 -30 110 75 ArrowB % 30 -30 105.41 68.986
     curveto}}            % curve for three points
   \psline[linestyle=dashed]%
    (5pt,5pt)(80pt,140pt)(30pt,-30pt)(110pt,75pt)
 \uput[0](80pt,140pt){(80pt,140pt)}
 \uput[0](30pt,-30pt){(30pt,-30pt)}
\end{pspicture}
```

11.3.28 \setcolor

\setcolor sets the current colour and uses \pst@usecolor internally.

\setcolor{*colour name*}



Exa 11-3-33

```
\usepackage{pstricks}
\begin{pspicture}[showgrid=true](3,3)
  \pscustom{%
    \code{newpath}
    \rcoor(0.5,0.5)(0,2)(2,0)(0,-2)
    \setcolor{red}
    \code{ moveto rlineto rlineto rlineto
      closepath
      2 setlinejoin 7.5 setlinewidth stroke}}
\end{pspicture}
```