TeXdraw

PostScript Drawings from TeX
Edition 2.0
December 1995

Peter Kabal

This is edition 2.0 of the documentation for the TEXdraw macros for the TEX typesetting program.

Peter Kabal
Department of Electrical Engineering
McGill University
3480 University
Montreal, Quebec
Canada H3A 2A7

kabal@TSP.EE.McGill.CA

1 Introduction

TeX is a powerful typesetting program which allows for complex text layouts but by itself lacks a general graphics capability. However, when coupled with an appropriate printer driver program, external graphics files can be inserted into the printed document. In this mode, TeX is instructed to leave space for a drawing. The drawing is inserted by the printer driver program. The TeXdraw macros described here generate the external graphics file from within TeX and generate the instructions to the the print driver program to position the graphics at the appropriate position on the page.

TeXdraw consists of a set of TeX macros that create line drawings and other figures. The drawing primitives include solid lines, patterned lines, Bezier curves, circles and arrows. Other commands allow for the filling of a region with a gray level. The drawing commands generate PostScript code. This limits TeXdraw to systems which use PostScript printers. TeXdraw also provides commands to position TeX text, including mathematics, on the drawing. The final drawing, with text and graphics, can be positioned on the page like any other TeX box.

The basic TeXdraw macros for TeX use the `\special` syntax recognized by the printer driver program `dvips`. However, when invoked as a LaTeX2e package, the TeXdraw macros can be used with any of the PostScript printer driver programs supported by the standard `graphics` package for LaTeX2e.

The basic TeXdraw macros provide only simple drawing commands. However, TeXdraw provides a drawing segment environment which allows parameter changes and coordinate scaling changes to be kept local to the drawing segment. This facility, together with TeX's macro capabilities allows one to modularize drawing units and extend TeXdraw by building more complex graphics entities from simpler elements.

1.1 Distribution information

The TeXdraw routines are provided free of charge without warranty of any kind. Note that the TeXdraw routines are copyrighted. They may be distributed freely provided that the recipients also acquire the right to distribute them freely. The notices to this effect must be preserved when the source files are distributed.

2 Using the TEXdraw Commands

The main TEXdraw macros (commands) are defined in the file '`texdraw.tex`'. These macros may be used directly in TEX. The file '`texdraw.sty`' provides an interface for use with LaTEX2e. The following sections describe the basic commands for TEXdraw.

2.1 Accessing TEXdraw

The form of the user command to run the TEX program depends on which version of TEX is being used, and which other macro packages are preloaded as format files. Typically, installations have at least two versions of TEX — plain TEX which includes basic typesetting macros (usually invoked as '`tex`') and LaTEX2e which includes the LaTEX2e typesetting macros (usually invoked as '`latex`'). An older version of LaTEX, version 2.09, may also be available. The TEXdraw macros can be used with plain TEX and with either version of LaTEX.

For use with plain TEX, the user must read in the TEXdraw macros from the file '`texdraw.tex`'.

```
\input texdraw           % Read in the TeXdraw macros
 ...
\btexdraw
  ...                    % TeXdraw commands to generate a drawing
\etexdraw
```

For use with LaTEX version 2.09, the user reads in the TEXdraw macros from the file '`texdraw.tex`' and optionally defines the `\begin{texdraw}` / `\end{texdraw}` environment.

```
\documentstyle[11pt]{article}  % Article style with the 11pt size options
...
\input texdraw           % Read in the TeXdraw macros
\newenvironment{texdraw}{\leavevmode\btexdraw}{\etexdraw}
 ...
\begin{texdraw}
  ...                    % TeXdraw commands to generate a drawing
\end{texdraw}
...
\end{document}
```

For use with LaTEX2e, the user must load the `texdraw` package (file '`texdraw.sty`'). This package file defines the `\begin{texdraw}` / `\end{texdraw}` environment, brings in the standard `graphics` package and reads in the file '`texdraw.tex`' containing the definitions of the TEXdraw macros.

```
\documentclass[11pt]{article}  % Article class with the 11pt size option
\usepackage{texdraw}           % TeXdraw commands

\begin{document}
 ...
\begin{texdraw}
  ...                     % TeXdraw commands to generate a drawing
\end{texdraw}
 ...
\end{document}
```

As the TEXdraw commands are processed by TEX, an intermediate PostScript file is generated. The intermediate PostScript has a name of the form '*name*`.ps1`'. The name part is derived from the name of the main TEX file being processed. If more than one drawing is produced, the digit in the file name extension is incremented.[1]

The TEXdraw commands to produce a drawing are inserted between `\btexdraw` and `\etexdraw` commands, or for LaTEX, between `\begin{texdraw}` and `\end{texdraw}` commands. This results in a TEX box of appropriate size containing the drawing generated by the TEXdraw commands. The TEXdraw box can be positioned in a document like any other TEX box.

The `\centertexdraw{...}` macro centers the box generated by TEXdraw. The vertical space taken up is equal to the vertical size of the drawing. The `\centertexdraw` macro is normally used in vertical mode (between paragraphs). A `\par` command (a blank line will do also) before a `\centertexdraw` command will terminate horizontal mode and return to vertical mode. For LaTEX, a structured equivalent to the `\centertexdraw{...}` command is shown below.

```
\begin{center}
\begin{texdraw}
  ...
\end{texdraw}
\end{center}
```

The `\everytexdraw` command can be used to define a set of TEXdraw commands that will be executed at the beginning of every TEXdraw drawing. It is invoked as `\everytexdraw{ ...}`, with the desired TEXdraw commands as arguments.

[1] After the ninth PostScript file, the name of the intermediate PostScript file takes the form '*name*`.p10`', with the number increasing from 10 with each file.

`\btexdraw`
: Start a TEXdraw drawing. The drawing is terminated with an `\etexdraw` command.

`\etexdraw`
: End a TEXdraw drawing started with a `\btexdraw` command. The resulting TEXdraw drawing is placed in a box with height equal to the height of the drawing and width equal to the width of the drawing. The depth of the box is zero.

`\begin{texdraw}`
: Start a TEXdraw drawing. The drawing is terminated with an `\end{texdraw}` command. This command is for use with LaTEX.

`\end{texdraw}`
: End a TEXdraw drawing started with a `\begin{texdraw}` command. The resulting TEXdraw drawing is placed in a box with height equal to the height of the drawing and width equal to the width of the drawing. The depth of the box is zero. This command is for use with LaTEX.

`\centertexdraw{ ... }`
: Center a TEXdraw box horizontally. The argument contains TEXdraw commands. The resulting box has the horizontal size `\hsize` and height equal to the height of the drawing.

`\everytexdraw{ ... }`
: Specify TEXdraw commands to be executed at the beginning of every TEXdraw drawing.

2.2 Command syntax

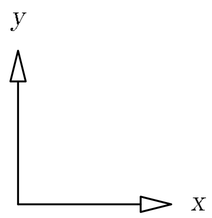
Generally TEXdraw commands that take a single argument need a terminating blank or newline after the argument. Arguments that are self-delimiting, such as coordinates within parentheses and text within braces, do not need the terminating blank. However, even when not needed by the defining syntax of the command, blanks following command arguments are allowed and ignored within the TEXdraw environment.

On entering the TEXdraw environment, TEX is in internal vertical mode (vertical mode inside a `\vbox`). In this mode, spaces can be placed freely between commands. However, any other extraneous input that generates output that is not part of the TEXdraw environment is disallowed.

Blank lines are interpreted as paragraph breaks, equivalent to a `\par` command. The TEXdraw macro `\centertexdraw` is defined with the `\long` attribute to allow `\par` commands and blank lines to be interspersed between TEXdraw commands. The `\btexdraw` and `\etexdraw` commands also allow `\par` command and blank lines to be included.

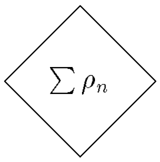
2.3 TEXdraw coordinates

The TEXdraw coordinate system has increasing x to the right and increasing y upward. The coordinates (without the unit) are floating point numbers. Integer values can be written without a decimal point. The size of the drawing is determined by the maximum excursions of the coordinates specified in TEXdraw commands.



Consider the following example of TEXdraw commands to draw a simple figure.

```
\centertexdraw{
  \drawdim cm  \linewd 0.02
  \move(2 2) \lvec(3 3) \lvec(2 4) \lvec(1 3) \lvec(2 2)
  \textref h:C v:C \htext(2 3){$\sum \rho_n$}
}
```



This drawing uses units of centimetres, with a line width of 0.02 cm. The x coordinate ranges between 1 and 3 while the y coordinate ranges between 2 and 4. When included into a document, the size of the drawing is 2 cm by 2 cm. The drawing is placed in a TEX box, with the lower lefthand corner of the box corresponding to TEXdraw coordinate `(1 2)` and the upper righthand corner at `(3 4)`. The `\centertexdraw` command centers the drawing horizontally. The `\textref` command controls the centering of the text. The text in this drawing is centered (both horizontally and vertically) at the coordinate `(2 3)`.

2.4 Coordinate specification

Coordinates are specified within parentheses, with blanks (but no comma) between the values. Leading blanks and trailing blanks are permitted within the parentheses. The coordinates refer to units, which are specified by the `\drawdim` command. The default is inches, but any valid TEX dimension unit can be specified. Symbolic specification of saved coordinate values will be discused later (see Section 3.3 [Saving positions], page 18).

`\drawdim` *dim*
: Set the units to *dim*. The argument *dim* can be any valid TEX dimension unit. The units are used to interpret coordinate values. Examples of valid units: `cm`, `mm`, `in`, `pt`, and `bp`.

Examples of coordinate and scaling specifications:

`\drawdim {cm} \move(2 2)`
: Set the units to centimetres, move to a position 2 cm to the right and 2 cm up from the origin of the drawing coordinate system.

`\drawdim bp`
: Set the units to big points.

`\lvec ( 2.2 +5.5) \lvec(2.3 -2) \lvec(2.2 5.4 )`
: Examples of acceptable coordinate specifications.

2.5 Line vectors

TEXdraw implements moves, line vectors and arrow vectors. There are both absolute and relative motion versions of these vector commands. TEXdraw maintains a current position. Lines are drawn from the current position to a new coordinate, with the new coordinate becoming the new current position. An explicit move can be used to establish a new current position. The position `(0 0)` is used if there is no move to an initial current position.

The `\move` and `\rmove` commands establish a new current position without drawing a line. The `\lvec` and `\rlvec` commands draw a line from the current position to a new position, which then becomes the new current position. The `\avec` and `\ravec` commands draw a line with an arrowhead from the current position to a new coordinate, which then becomes the new current position. The tip of the arrow is at the new current position. The direction of the arrow follows the direction of the line. Since this direction is undefined for zero length vectors, these are not allowed for `\avec` or `\ravec`. Zero length arrow vectors will generate a PostScript print error: `undefinedresult`. For any non-zero length vector, the full size arrowhead is drawn, even if that arrowhead is longer than the line length.

The absolute motion versions of these commands specify the coordinate of the final position.

`\move` (x y)
: Move to coordinate (x y). The new current position is (x y).

`\lvec` (x y)
: Draw a line from the current position to coordinate (x y). The new current position is (x y).

`\avec` (x y)
: Draw a line with an arrowhead from the current position to (x y). The new current position is (x y). The arrowhead is aligned with the line, with the tip at (x y).

The relative motion versions of these commands interpret the coordinates as displacements relative to the current position. Given the displacements (dx dy) as a parameter, each of the relative motion commands moves dx units in the x direction and dy units in the y direction.

`\rmove` (dx dy)
: Move from the current position, dx units in the x direction and dy units in the y direction. The final position becomes the new current position.

`\rlvec` (dx dy)
: Draw a line from the current position, dx units in the x direction and dy units in the y direction. The final position becomes the new current position.

`\ravec` (dx dy)
: Draw a line with an arrowhead from the current position, dx units in the x direction and y units in the y direction. The final position becomes the new current position. The arrowhead is aligned with the line, with the tip at the new current position.

Lines can be customized with commands to change the line width, line pattern and line gray level rendition. In addition, commands for changing the type and size of the arrowhead are available.

`\linewd` *width*
: Set the line width to *width* units. Initially *width* is 0.01 inches (corresponding to 3 pixels at 300 pixels to the inch).

`\lpatt` (*pattern*)
: Set lines to have the pattern (*pattern*). A pattern is a sequence of on/off lengths separated by blanks and enclosed in parentheses. The lengths alternately specify the length of a dash and the length of a gap between dashes. Each length is interpreted using the current scaling and drawing units. The pattern is used cyclically. The empty pattern signifies a solid line. The initial line pattern is a solid line, corresponding to the empty pattern `\lpatt ()`.

`\setgray` *level*

Set the gray level of lines. Gray levels are real values from 0 (black) through intermediate values (gray) to 1 (white). The initial gray level is 0 corresponding to black.

`\arrowheadtype t:`*type*

Set the arrowhead type to *type*, where *type* is one of `F`, `T`, `W`, `V`, or `H`. There are two kinds of arrowheads. The first kind is a triangle. There are 3 variants: type `T` is an empty triangle, type `F` is a filled triangle (using the current gray level for lines), type `W` is a triangle filled with white. The second kind of arrowhead is an open ended Vee. There are 2 variants: type `V` has the stem continue to the tip, type `H` has the stem stop at the base of the arrowhead. The initial arrowhead type is `T`.

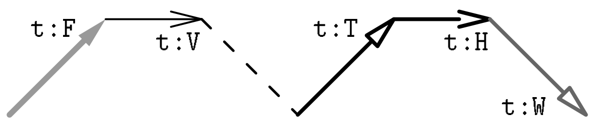
`\arrowheadsize l:`*length* `w:`*width*

Set the arrowhead size to be *length* units long and *width* units wide. The width is measured across the "base" of the arrowhead. The initial arrowhead size has a *length* of 0.16 inches and a *width* of 0.08 inches.

Note that the lines which outline the arrowhead will be drawn with the same line pattern used for the stem. Normally, arrow vectors are drawn with the line pattern set for a solid line. Note that the fill level used for the `F` variant of the arrowhead uses the same gray level as used for lines. The difference between the `T` variant and the `W` variant only shows up if the arrowhead is placed over non-white areas of the drawing. The `W` variant obliterates the area under the arrowhead.

Examples of line parameter and arrowhead settings are shown in the following code.

```
\centertexdraw{
  \drawdim in
  \linewd 0.03 \setgray 0.6 \arrowheadtype t:F \avec(0 0.5)
  \linewd 0.01 \setgray 0   \arrowheadtype t:V \avec(0.5 0.5)
  \linewd 0.015 \lpatt(0.067 0.1) \lvec (1 0)
  \linewd 0.02 \lpatt() \arrowheadtype t:T \avec(1.5 0.5)
  \arrowheadtype t:H \avec(2.0 0.5)
  \setgray 0.4 \arrowheadtype t:W \avec(3.0 0)
}
```



2.6 TEX text

Text may be superimposed on the drawing. The text argument of the `\htext` command is in horizontal mode. This text can be ordinary text, math mode expressions, or even more complicated boxes consisting of tables and the like. The resulting TEX text is placed in a box. The reference point of the box can be chosen to be one of nine locations: horizontally left, center or right; vertically top, center or bottom. The `\htext` command takes one of two forms.

`\htext (`*x y*`){`*text*`}`
`\htext {`*text*`}`

The first form of this command places the TEX text *text* horizontally with the text reference point at the coordinate `(`*x y*`)`. The new current position is `(`*x y*`)`. The second form of this command places the TEX text *text* horizontally with the text reference point at the current position. The text reference point is set with the `\textref` command.

Text can be placed vertically using the `\vtext` command. The text argument is in horizontal mode. The TEX text is placed in a box and then rotated counterclockwise. The reference point is the point in the box, *before* rotation of the text. Not all PostScript printer drivers support vertical text.

`\vtext (x y){`*text*`}`
`\vtext {`*text*`}`

The first form of this command places the TEX text *text* vertically with the text reference point at the coordinate `(`*x y*`)`. The new current position is `(`*x y*`)`. The second form of this command places the TEX text *text* vertically with the text reference point at the current position. In both cases, the TEX text is placed in a box and the box is rotated counterclockwise by 90 degrees about the text reference point. The text reference point is set with the `\textref` command.

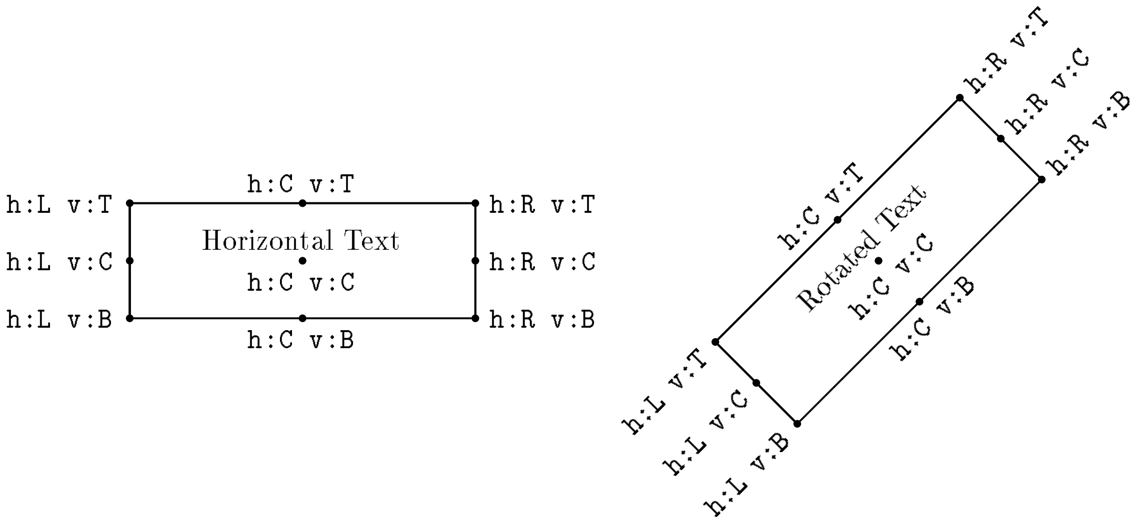
Text can be placed at an arbitrary angle using the `\rtext` command. The text argument is in horizontal mode. The TEX text is placed in a box and then rotated counterclockwise. The reference point is the point in the box, *before* rotation of the text. Not all PostScript printer drivers support rotated text.

`\rtext td:`*angle* `(x y){`*text*`}`
`\rtext td:`*angle* `{`*text*`}`

The first form of this command places the TEX text *text* at an angle with the text reference point at the coordinate `(`*x y*`)`. The new current position is `(`*x y*`)`. The second form of this command places the TEX text *text* at an angle with the text reference point at the current position. In both cases, the TEX text is placed in a box and the box is rotated counterclockwise by *angle* degrees about the text reference point. The text reference point is set with the `\textref` command.

The reference point for subsequent TEX text in a `\htext`, `\vtext` or `\rtext` command is set with the `\textref` command.

`\textref h:`*h-ref* `v:`*v-ref*
Set the text reference point for subsequent text commands. The horizontal reference point *h-ref* is one of `L`, `C` or `R` (left, center or right). The vertical reference point *v-ref* is one of `T`, `C` or `B` (top, center or bottom). For rotated text, the reference point is determined before rotation. The initial text reference point corresponds to `\textref h:L v:B`.

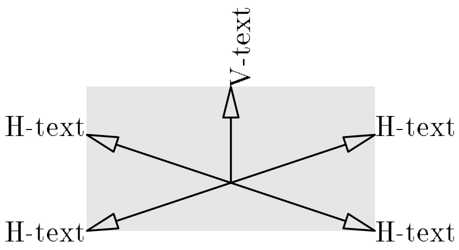


The font used to render the text is determined as for any other TEX text. Normally the font used outside of TEXdraw is in effect. If desired, other fonts can be specified as part of the text. Any font changes within a TEXdraw text command remain local to that command.

Only the coordinate of the text reference point in a `\htext`, `\vtext` or `\rtext` command is used in calculating the size of the drawing. This means that text itself can spill outside of the drawing area determined by TEXdraw. The area of the drawing can be increased to include the text by issuing additional `\move` commands.

```
\centertexdraw{
              \avec(-0.75 -0.25) \textref h:R v:C \htext{H-text}
  \move(0 0) \avec(-0.75 +0.25) \textref h:R v:B \htext{H-text}
  \move(0 0) \avec(0 +0.5)      \textref h:L v:T \vtext{V-text}
  \move(0 0) \avec(+0.75 +0.25) \textref h:L v:B \htext{H-text}
  \move(0 0) \avec(+0.75 -0.25) \textref h:L v:C \htext{H-text}
}
```

Superimposed on this example is a shaded region showing the limits of the TEXdraw box as determined by the coordinates specified.



2.7 Circles, ellipses and arcs

TEXdraw supplies commands to generate circles, ellipses and arcs. There are two forms of the circle command. The `\lcir` command draws a circle of given radius. The `\fcir` command draws a filled circle. In the latter case, the circle is filled by a specified gray level. For the filled circle, the line defining the circumference of the circle is not drawn. Note that the gray level area filled in by the `\fcir` command is opaque, even if the fill is chosen to be white. For either form of the circle command, the drawing size is increased if necessary to contain the circle.

The `\lellip` command generates an ellipse specified by the radius of the ellipse in the x direction and the radius of the ellipse in the y direction. The ellipse is symmetrical about horizontal and vertical lines drawn through the current point. The `\fellip` command draws a filled ellipse. In the latter case, the ellipse is filled by a specified gray level. For the filled ellipse, the line defining the boundary of the ellipse is not drawn. For either form of the ellipse command, the drawing size is increased if necessary to contain the ellipse.

The `\larc` command generates a counterclockwise arc specified by a start angle in degrees and an end angle in degrees. The center of the arc is the current position. Only the arc is drawn, not the line joining the center to the beginning of the arc. Note that the `\larc` command does not affect the size of the drawing.

`\lcir r:`*radius*
: Draw a circle with center at the current position. The radius is specified by *radius*. This command draws a line along the circumference of the circle. The drawing size is increased if necessary to contain the circle.

`\fcir f:`*level* `r:`*radius*
: Draw a filled circle with center at the current position. The radius is specified by *radius*. The circle is painted with the gray level specified by *level*. A gray level of 1 corresponds to white, with decreasing values getting darker. The level 0 is full black. This command does not draw a line along the circumference. The drawing size is increased if necessary to contain the circle.

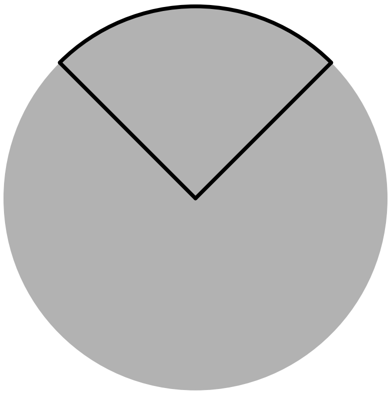
`\lellip rx:`*x-radius* `ry:`*y-radius*
: Draw an ellipse with center at the current position. The radius in the x direction is specified by *x-radius*. The radius in the y direction is specified by *y-radius*. The drawing size is increased if necessary to contain the ellipse.

`\fellip f:`*level* `rx:`*x-radius* `ry:`*y-radius*
: Draw a filled ellipse with center at the current position. The radius in the x direction is specified by *x-radius*. The radius in the y direction is specified by *y-radius*. The ellipse is painted with the gray level specified by *level*. A gray level of 1 corresponds to white, with decreasing values getting darker. The level 0 is full black. This command does not draw a line along the boundary of the ellipse. The drawing size is increased if necessary to contain the ellipse.

`\larc r:`*radius* `sd:`*start-angle* `ed:`*end-angle*
: Draw a counterclockwise arc. The center of the arc is at the current position. The radius is specified by *radius*. The start and end angles (in degrees) are specified by *start-angle* and *end-angle*. This command does not affect the limits (size) of the drawing.

As an example, the following commands draw a filled circle, and superimpose an arc.

```
\centertexdraw{
  \linewd 0.02
  \fcir f:0.7 r:1
  \larc r:1 sd:45 ed:135
  \lvec (+0.707 +0.707) \move (0 0) \lvec (-0.707 +0.707)
}
```



Note that for the arc command, the resulting figure can spill outside of the TEXdraw box as determined by the maximum excursions of the coordinates. Extra moves can be used to compensate for the size of the arc.

2.8 Bezier curves

Bezier curves in TEXdraw use 4 reference coordinates, two as the end points and two others to control the shape of the curve. Let the 4 points be `(`*x0 y0*`)`, `(`*x1 y1*`)`, `(`*x2 y2*`)` and `(`*x3 y3*`)`. The curve starts out tangent to the line joining the first two points and ends up tangent to the line joining the second two points. The control points "pull" at the curve to control the curvature. The amount of pull increases with the distance of the control point from the endpoint.

As the parameter μ varies from 0 to 1, the coordinates of the Bezier curve are given by a pair of parametric cubic equations,

$$\mathrm{x}(\mu) = (1-\mu)^3 x0 + 3\mu(1-\mu)^2 x1 + 3\mu^2(1-\mu)x2 + \mu^3 x3$$
$$y(\mu) = (1-\mu)^3 y0 + 3\mu(1-\mu)^2 y1 + 3\mu^2(1-\mu)y2 + \mu^3 y3 \; .$$

`\clvec (`*x1 y1*`)(`*x2 y2*`)(`*x3 y3*`)`

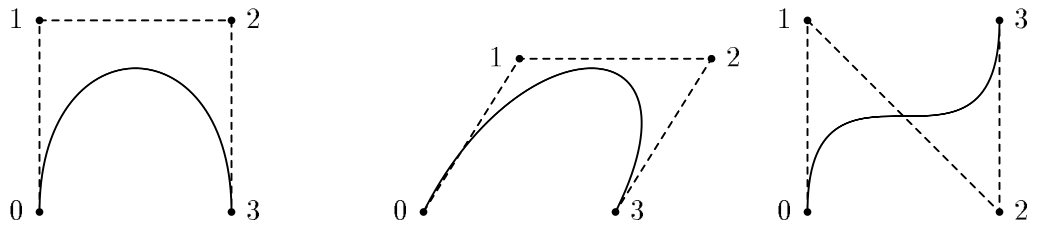
Draw a Bezier curve from the current position to the coordinate `(`*x3 y3*`)` which becomes the new current position. The coordinates `(`*x1 y1*`)` and `(`*x2 y2*`)` serve as control points for the curve. Only the last coordinate given is used to update the size of the drawing.

Note that only 3 coordinate pairs are specified. The other point is the current position before the `\clvec` command is executed. Only the last coordinate specified in the `\clvec` command is used to determine the extent of the drawing. While the Bezier curve passes through the old current position and the new current position, in general the curve will not reach the intermediate control points. The curve is always entirely enclosed by the convex quadrilateral defined by the two end points and the two control points. Note that the curve may pass outside the limits of the drawing as determined by the end point of the curve.

A simple Bezier curve is produced by the following example.

```
\btexdraw
  \move (0 0)
  \clvec (0 1)(1 0)(1 1)
\etexdraw
```

This example is the rightmost of the following Bezier curves. The drawings also show the end points and the control points for each curve.



2.9 Fill commands

PostScript deals with paths consisting of line segments. The paths can be closed and the interior of the closed region filled. From TEXdraw, paths start with a `\move` or `\rmove` command and continue with `\lvec`, `\rlvec` or `\clvec` commands. The TEXdraw fill commands close the path and fill the interior of the closed region. Closing the path means that effectively another `\lvec` line is drawn from the last point specified to the initial point. TEXdraw provides two forms of the fill command. The `\ifill` fills the interior of the region with the given gray level. The lines defining the path are not drawn. The `\lfill` command fills the region defined by the closed path and draws a line along the enclosing path. Note for both forms of the fill command, the gray level used for filling is opaque, even if the gray level is chosen to be white.

`\lfill f:`*level*

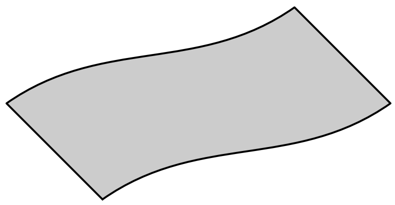
Close the current path, draw the line around the path using the current grey level for lines and paint the interior of the region with specified gray level *level*. Gray levels are real values from 0 (black) through intermediate values (grays) to 1 (white).

`\ifill f:`*level*

Close the current path and paint the interior of the region with gray level *level*. The line around the path is not drawn. Gray levels are real values from 0 (black) through intermediate values (grays) to 1 (white).

The following example draws a "flag" with the interior filled in. The path around the boundary is given in a clockwise order to define a closed path. We could take advantage of the fact that the fill command will close an open path to eliminate one of the `\lvec` commands.

```
\centertexdraw{
\move (0.5 0)
\lvec (0 0.5) \clvec (0.5 0.85)(1 0.65)(1.5 1)
\lvec (2 0.5) \clvec (1.5 0.15)(1 0.35)(0.5 0)
\lfill f:0.8
}
```



In TEXdraw, the `\move` command always terminates any previous paths and starts a new path. Commands that change line parameters (e.g. `\setgray` or `\lpatt`) also terminate paths and start new paths. The circle, ellipse and arc commands do not affect the definition of the current path. The `\avec` command is not appropriate for defining a path to be filled. It ends a subpath at its tail and begins a new subpath at its tip. Filling a region defined by a path with subpaths is more complicated in that each subpath is closed before filling.

3 Drawing Segments and Scaling

TEXdraw provides individually scaled segments which can be used to create relocatable drawing modules.

3.1 Drawing segments

A TEXdraw drawing segment allows for local modifications of parameters and relative positioning. A TEXdraw segment is delimited by a `\bsegment` command and an `\esegment` command. Inside the segment, the initial current position is `(0 0)`. Any changes to parameters such as the gray level and the line width, remain local to the segment. Segments are implemented in TEX using a `\begingroup` and `\endgroup`. Segments can be nested.

`\bsegment`
: Start a drawing segment. The coordinate system is shifted such that the current position corresponds to the coordinate `(0 0)`. Changes to scaling, position and line parameters stay local to the drawing segment.

`\esegment`
: End a drawing segment. The current position in effect before the corresponding `\bsegment` command is restored. The scaling and line parameter values revert to those in effect before the corresponding `\bsegment` command was invoked.

3.2 Drawing paths

Certain subtle interactions occur between drawing segments and fill operations. In PostScript, lines are drawn by first defining a path, then later stroking the path to draw the line. In TEXdraw, this stroking occurs when the line is terminated, say by a `\move` command. PostScript paths are interrupted by, but continue after a drawing segment. This means that a path started before a segment may not be stroked (drawn) until after the segment ends. Consider the following example.

```
\move (0 0)
\lvec (1 1)
\bsegment
  \move (-0.25 -0.25)
  \fcir f:0.8 r:0.5
\esegment
\move (0 0)
```

A PostScript path is started at `(0 0)` and continues with a line to `(1 1)`. This path is interrupted by the segment. The filled circle is drawn next. After the segment, the path continues and is not stroked until the `\move (0 0)` command after the end of the segment. This means that the line appears on top of the filled region.

If the fill operation is to cover the line, the path must be stroked before the fill operation. From TEXdraw, the move commands `\move` and `\rmove`, and the end TEXdraw command `\etexdraw` terminate a path and cause it to be stroked. Within a segment, the end segment command `\esegment` also terminates and strokes a path. In the example above, the line can be stroked by inserting a move command (such as a `\rmove (0 0)` which does not affect the position), before the start of the segment.

3.3 Saving positions

The `\savecurrpos` command saves the current position. The saved position is an absolute position, not one relative to a segment. The position saving mechanism is global; the position can be saved within a nested segment and then used outside of the segment. The *x* and *y* coordinates of the position are saved separately as named coordinates. The names are of the form `*`*name*, with the leading `*` being obligatory. A companion command, `\savepos`, saves a given coordinate (relative to the current segment) as an absolute symbolic position.

`\savecurrpos (*`*px* `*`*py*`)`
: Save the current position as the absolute position referenced by `(*`*px* `*`*py*`)`.

`\savepos (`*x y*`)(*`*px* `*`*py*`)`
: Save the coordinate position `(`*x y*`)` as the absolute position referenced by `(*`*px* `*`*py*`)`. The coordinate `(`*x y*`)` is interpreted in the normal fashion as a coordinate relative to the current segment, using the current scaling factors and drawing unit.

The symbolic names used to specify a saved position can consist of any characters that are not special to TEX, but must start with a `*` character. The symbolic names can be used as the *x* and/or *y* coordinate in any command that needs a coordinate. Symbolic coordinates are not normally used with relative motion commands such as `\rlvec` or `\rmove`. If used with relative motion, the corresponding displacement is equal to the symbolic coordinate value.

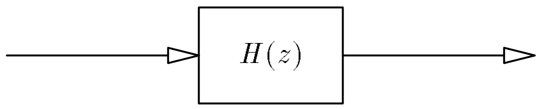
On exit from a segment, the position and graphics state on entry is restored. Any changes to line types, scaling and position are discarded. However, it is sometimes useful alter the position on exit from a segment. The `\savepos` command allows for the saving of a position within the segment. This position can be restored after the `\esegment` with a `\move` command using the saved symbolic position. This approach can be used to build modules which operate in a manner analogous to the basic relative motion line vector commands.

The following example defines a macro which draws a box 0.75 inches wide by 0.5 inches high containing centered text. On leaving the macro the position will be set at a point on the righthand side of the box.

```
\def\tbox #1{\bsegment
              \lvec (0 +0.25)    \lvec (0.75 +0.25)
              \lvec (0.75 -0.25) \lvec (0 -0.25) \lvec (0 0)
              \textref h:C v:C \htext (0.375 0){#1}
              \savepos (0.75 0)(*ex *ey)
            \esegment
            \move (*ex *ey)}
```

With this definition, we can treat `\tbox` in the same way as the basic vector commands, stringing them together to form a block diagram as in this example.

```
\centertexdraw{
  \ravec (1 0) \tbox{$H(z)$} \ravec (1 0)
}
```



3.4 Scaling coordinates

There are two scale factors available, the unit scale factor and the segment scale factor. The overall scale factor is the product of these two. There are absolute and relative versions of commands to change these scale factors.

The unit scale factor is normally used to affect global scale changes. Changes to the unit scale factor remains local to a segment, but propagate to inferior segments. The default value is unity.

The segment scale factor is used for local scale changes. It remains local to a segment. The segment scale factor is reset to unity on entry into each segment. This means that changes to the segment scale factor do not propagate to inferior segments.

`\setunitscale` *scale*
: Set the unit scaling to *scale*. The argument *scale* is a real number which is used to scale coordinate values. The overall scaling factor is the product of the unit scale factor and the segment scale factor.

`\relunitscale` *value*
: Adjust the unit scale factor by multiplying by *value*. This has the effect of multiplying the overall scale factor by the same factor. The overall scaling factor is the product of the unit scale factor and the segment scale factor.

`\setsegscale` *scale*
: Set the segment scale factor. The argument *scale* is a real number which is used to scale coordinate values. The overall scale factor is the product of the unit scale factor and the segment scale factor.

`\relsegscale` *value*
: Adjust the segment scale factor by multiplying by *value*. This has the effect of multiplying the current overall scale factor by the same factor. The overall scaling factor is the product of the unit scale factor and the segment scale factor.

In addition to the unit scale factor and the segment scale factor, the scaling can be controlled by the choice of drawing units with the command `\drawdim` (see Section 2.4 [Coordinate specification], page 6).

`\drawdim cm \setunitscale 2.54`
: Set the units to centimetres scaled by 2.54. Together these commands are effectively the same as `\drawdim in`.

The segment scale can be used to allow scale changes in segments so that values are in more convenient units. For example suppose dimensions in a segment are multiples of one third of an inch. The segment scale can be set once to make 1 drawing unit equal 0.3333 inches. From that point on, coordinates can be specified with integer values.

The following example defines a macro to draw a rectangular box which is twice as wide as it is high. The width is specified as an argument.

```
\def\mybox #1{\bsegment
               \setsegscale #1
               \lvec (0 +0.25) \lvec (1 +0.25) \lvec (1 -0.25)
               \lvec (0 -0.25) \lvec (0 0)
             \esegment}
```

3.5 Drawing size

The effective size of the drawing is determined by the maximum excursions of the coordinates supplied to TEXdraw commands. The minimum and maximum scaled x and y coordinates are tallied. Note that `\move` commands contribute to the determination of the calculated size of the drawing, even though they do not generate visible lines. The circle and ellipse commands add a compensation for the radii of circles and ellipses. The final TEXdraw drawing is placed in a TEX box with lower lefthand corner corresponding to `(`x-min y-min`)` and upper righthand corner at `(`x-max y-max`)`.

Text generated by `\htext`, `\vtext` or `\rtext` can spill outside the box as determined above. Only the text reference point is guaranteed to be in the drawing box. Arcs can also spill outside the drawing box. Note also that the widths of lines, and the sizes of arrowheads do not affect the size of the drawing. The calculated size of the drawing will never be larger than the actual size of the drawing. In extreme cases in which text or lines extend far outside the drawing, extra `\move` commands should be used to establish the size of the drawing so that the TEXdraw box includes all of the drawing.

TEXdraw provides the `\drawbb` command to draw a box which indicates the effective size of the drawing. Whenever `\drawbb` is invoked, a ruled box is drawn around the drawing as it has been sized up to that point. Normally `\drawbb` is invoked just before the end of a drawing to indicate the effective size of the final drawing.

`\drawbb` Draw a ruled box around the effective size of a drawing produced by TEXdraw commands.

3.6 Initial current position

The first operation in a drawing should be a move to establish the current position. The current position can be established explicitly through a `\move` command or a text positioning command such as `\htext` with a coordinate. However, if an attempt is made to use a drawing command which needs a current position and none has been established, TEXdraw implicitly sets the initial current position to `(0 0)`. The size of the TEXdraw figure is normally determined from the sequence of coordinates specified, but will include the implicit initial position in case another initial position has not been explicitly specified.

4 Using TEXdraw with LaTEX

The LaTEX typesetting system uses a structured approach to declaring typesetting environments. For LaTEX2e, the `texdraw` package defines the `texdraw` environment. The TEXdraw environment is started with a `\begin{texdraw}` command and terminated with an `\end{texdraw}` command. All of the basic TEXdraw commands can be used within the `texdraw` environment.

As an example, a LaTEX2e variant of an earlier example can be constructed as follows.

```
\documentclass{article}
\usepackage{texdraw}
 ...
\begin{document}
 ...
\newcommand{\tbox}[1]{%
   \bsegment
     \lvec (0 +0.25)    \lvec (0.75 +0.25)
     \lvec (0.75 -0.25) \lvec (0 -0.25) \lvec (0 0)
     \textref h:C v:C \htext (0.375 0){#1}
     \savepos (0.75 0)(*ex *ey)
   \esegment
   \move (*ex *ey)}
\begin{center}
\begin{texdraw}
  \ravec (1 0) \tbox{$H(z)$} \ravec (1 0)
\end{texdraw}
\end{center}
 ...
\end{document}
```

This example illustrates the use of the LaTEX command `\newcommand` as an alternative to the plain TEX command `\def`. Instead of the basic TEXdraw command `\centertexdraw`, a nested combination of the LaTEX centering environment and the TEXdraw environment is used.

4.1 PostScript printer drivers

The `texdraw` package uses the printer driver interface provided by the standard LaTEX2e `graphics` package. Any options to the `texdraw` package are passed to the `graphics` package. Specifically, the name of the PostScript driver to be used can be specified as an option to the `texdraw` package. With no explicit printer driver option, the default printer driver associated with the `graphics` package is used.

The `texdraw` package can be used with any of the printer drivers supported by the `graphics` package that allow for the importation of PostScript graphics files, viz., `dvips`, `xdvi`, `dvi2ps`, `dvialw`, `dvilaser`, `dvipsone`, `dviwindo`, `dvitops`, `oztex`, `psprint`, `textures`, `pctexps`, and `pctexwin`. Not all of these drivers support the text rotation needed for the TEXdraw commands `\vtext` and `\rtext`. Of the drivers listed above, only the following support support text rotation: `dvips`, `xdvi`, `dvi2ps`, `dvitops`, `textures`, and `pctexps`.

5 More Details

The first part of this chapter offers some suggestions for strategies to isolate errors in TEX and TEXdraw input. The second part of this chapter discusses implementational issues. An awareness of these issues is useful if TEXdraw is to be extended.

5.1 Errors while using TEXdraw

TEX input is notoriously difficult to debug. If TEX reports errors, so much the better. If the cause is not immediately obvious, consider using a binary search strategy, removing sections of code with the premature insertion of the `\bye` (or `\end{document}` for LaTEX) command (with the appropriate closing of any open groups and the like). Other strategies include the insertion of `\message{I am here}` at appropriate places. Try using `\tracingmacros=1`. Many problems turn out to be due to an incorrect number of macro arguments or incorrectly delimited macro arguments. The `\tracingmacros=1` option writes the macro arguments and macro expansions to the TEX log file.

Certain errors may not manifest themselves until well after the offending command. For instance, if a closing parenthesis is missing from a TEXdraw coordinate, TEX continues searching for the parenthesis. If one is found, perhaps many lines later, the TEXdraw error message `invalid coordinate` will be printed at this later point.

All input in the TEXdraw environment should be intended for interpretation by TEXdraw commands. TEXdraw places text inside a zero size box (the text itself extends outside the box). Extraneous input manifests itself as a non-zero size TEXdraw text box. This causes the TEXdraw text and the PostScript graphics to be displaced from one another. An error message is issued if a non-zero width TEXdraw text box is detected. If this error message appears, look for unintended character sequences amongst the commands to TEXdraw.

Several TEXdraw commands pass their arguments "raw" to the PostScript file. That means that invalid arguments can generate PostScript errors when the document is printed. For instance the argument of the `\setgray` command is passed straight through to the PostScript file. If this argument is non-numeric, a PostScript error results. Not all PostScript printers report errors back to the user. The print may just stop prematurely. One approach to debugging is to use a PostScript previewer on a workstation. That way, one can determine at which point in the drawing the PostScript error occurs.

5.2 Extending TEXdraw

TEXdraw is implemented using a combination of TEX commands and PostScript code. This section discusses some of the implementational issues as they relate to extending TEXdraw.

TEXdraw as implemented, offers a basic set of drawing features. These are adequate for certain tasks such as producing block diagrams. There are different approaches to extending TEXdraw to include other functions. In some cases, the desired functionality can be achieved by writing a TEX macro which builds on top of the existing TEXdraw commands. As these extensions become more complex, the limitations of TEX for computations become increasingly evident. In other cases, access to different features of PostScript is desired. The appropriate approach would be to write new PostScript procedures which can be accessed by TEX macros.

Included with TEXdraw is a set of macros for directly accessing PostScript functions. These are described in an appendix (see Appendix A [PostScript Commands], page 31).

TEXdraw also comes with a toolbox of routines for handling much of the user interface, converting between different coordinate representations and the like. The macros for coordinate decoding and for computations involving coordinates are described in an appendix (see Appendix B [TEXdraw Toolbox], page 35).

5.2.1 Scaling

The scaling commands provided in TEXdraw are designed to affect only the coordinate values specified in commands. For instance, changing the `\setunitscale` value changes the interpretation of the coordinate in an `\avec (`*x y*`)` command, but does not change the line width or arrowhead sizes in effect. None of the TEXdraw scaling commands affect the size of TEX text produced by, for instance, the `\htext` command. Scale changes will however affect the positioning of text for subsequent commands.

The line parameters are changed only if the corresponding commands to change them are issued. If the `\linewd` command is given, the current coordinate scaling is used to determine the line width. To achieve a behaviour more like a global scaling, whenever the scale factor is changed, the line parameters should be set again.

5.2.2 Resolution

TEXdraw scales coordinates before passing them to PostScript. Keeping track of the coordinate scaling is necessary, in any event, to allow TEXdraw to compute the maximum excursions of the coordinates. TEXdraw uses pixel units in its PostScript code. One pixel unit is equal to 1/300 of an inch. TEXdraw issues PostScript commands with integer valued pixel coordinates. This sets the positioning resolution for TEXdraw. The passing of integer valued coordinates which correspond to the device resolution keeps lines aligned with the device grid; parallel lines of the same width will be rendered with the same width.

The position saving mechanism in TEXdraw (see Section 3.3 [Saving positions], page 18) associates the pixel coordinates of a position with the specified name.

TEXdraw uses the limited real number representation provided by TEX. These operations are based on the representation of dimensions as real-valued numbers of points. Internally in TEX, dimensions are stored 32-bit values, normalized so that 1 pt corresponds to the scaled point (sp) value of 65536. Dimensions with magnitudes between 0.000015 pt and 32767 pt can be represented. This is also the dynamic range of the TEXdraw pixel coordinates passed to PostScript. TEXdraw must convert from user supplied coordinates using the scaling factor (which itself consists of two components, the unit scale and the segment scale) and a pixel conversion factor. The use of limited precision real numbers in these computations can cause accumulation of error when relative scaling is used repeatedly.

5.2.3 Text placement

While in the TEXdraw environment, TEX text is placed in a TEX box while PostScript code is written to the intermediate file. At the end of the TEXdraw environment, the size of the drawing is determined. A TEX box of this size is created. The TEX `\special` mechanism is used to instruct the PostScript driver program to position the PostScript drawing from the intermediate file in this area. Next, the text generated by TEXdraw is positioned and placed in the box. Note that when the document is printed, the PostScript drawing is placed on the page before the TEX text; TEX text will appear on top of graphics.

The rotation of text is carried out with in-line PostScript code which does not appear in the intermediate PostScript file. This code is sent to the PostScript driver with a `\special` command. This PostScript code is embedded in the dvi (device independent) file that TEX produces.

5.2.4 The intermediate PostScript file

The intermediate PostScript file consists of a header, a body and a trailer following Encapsulated PostScript File (EPSF) standards. The header sets up PostScript definitions and default parameter values. The trailer includes the `BoundingBox` information which gives the coordinates in default PostScript units (72 per inch) for the lower lefthand corner and the upper righthand corner of the drawing. The body of the intermediate PostScript file contains the PostScript commands generated by TEXdraw.

Many moves in TEXdraw serve only to position text or to reset saved positions. TEXdraw buffers move commands in order to be able to collapse runs of moves. Only the last move of a run of moves is actually written to the PostScript file. However the intermediate moves still affect the size of the drawing. The expunging of moves means that the PostScript file `BoundingBox` information may indicate a drawing size larger than the PostScript commands themselves would warrant.

Drawing segments in TEXdraw show up in the PostScript file as saves and restores of the PostScript graphics state. Segment starts are buffered and only written out if necessary. This way "empty" segments do not generate output to the PostScript file. These empty segments arise if a segment contains only moves and text commands. The moves inside the segment are not needed since they are local to the segment, and the text commands do not generate output to the PostScript file.

If TEXdraw is used only for moves and text, no intermediate PostScript file will be created.

5.3 How TEXdraw merges graphics and text

TEXdraw creates a box which is the same size as the graphic. The printer driver will place the PostScript graphic into this space. Any TEX text generated by the TEXdraw commands will be superimposed on this graphic.

The LaTEX2e front-end for TEXdraw is enabled by including the `texdraw` package. The `texdraw` package automatically invokes the standard `graphics` package distributed with LaTEX2e. The `graphics` package has support for a number of different printer drivers, including a number for PostScript printers. Any options to the `texdraw` package are passed on to the `graphics` package. Such an option can be used to select a driver other than the default one.

Within the `graphics` package, the driver option is used to select definitions for the low-level macros which generate the `\special` commands needed to request insertion of a graphics file and to rotate text.[1] TEXdraw uses the user-level macros defined by the `graphics` package (see Section 4.1 [PostScript printer drivers], page 23). When not used with the LaTEX2e front-end, TEXdraw defines versions of these macros that are suitable for use with the `dvips` printer driver.

[1] Not all PostScript drivers support text rotation.

Appendix A PostScript Commands

This appendix describes a set of macros for accessing some of the PostScript builtin functions. Each of these macros issues a single PostScript command. The extra services provided by TEXdraw are the interpretation of coordinates in user units relative to the current drawing segment and the writing of a pending TEXdraw move to the PostScript file. This last operation establishes the current point in PostScript. The user of these commands should be familiar with the concepts of path construction and filling in PostScript. Further details on the PostScript functions used can found in the *PostScript Language Reference Manual, Second Edition*, Adobe Systems, Addison-Wesley, 1990.

These macros are distributed in file '`txdps.tex`'.

The `\PSsetlinecap` and `\PSsetlinejoin` commands control the way line ends and line joins are rendered. The default values set by TEXdraw (round caps and round join) are appropriate for most drawings. Changes to these parameters apply to the current and subsequent paths.

`\PSsetlinecap` *type*
: Set the line cap parameter. The value `0` gives a butt cap; `1` gives a round cap; and `2` gives a projecting square cap. The initial value is corresponds to a round cap.

`\PSsetlinejoin` *type*
: Set the line join parameter. The value `0` gives a miter join; `1` gives a round join; and `2` gives a bevel join. The initial value corresponds to a round join.

PostScript paths and fill operations can be controlled by a number of functions. By design, TEXdraw always maintains a defined PostScript current point. Some of the following macros cause the PostScript current point to become undefined. The PostScript current point must be set again (say with a `\PSmoveto` command) before invoking basic TEXdraw commands.

`\PSstroke`
: Stroke a PostScript path. The current path is stroked with the current gray level (set with `\setgray`) and the current line pattern (set with `\lpatt`). The PostScript current point becomes undefined.

`\PSnewpath`
: Establish a new path. The PostScript current point becomes undefined.

`\PSclosepath`
: Close a subpath. A new subpath is started.

`\PSfill` Fill a region defined by a path. Each subpath is closed and the enclosed regions painted with the current gray level. The PostScript current point becomes undefined. The gray level can be set with the TEXdraw command `\setgray`.

The following line commands interpret coordinates relative to the current TEXdraw scaling and drawing segment. The specified coordinate affects the drawing size as determined by TEXdraw.

`\PSlineto (`*x y*`)`
Add a line segment to the current path. This command is identical to the TEXdraw command `\lvec`. The PostScript current point must be defined before this command is issued.

`\PSmoveto (`*x y*`)`
Move to the coordinate specified by `(`*x y*`)`. The PostScript current point becomes defined.

The following macros provide access to the general arc commands in PostScript. The coordinates are interpreted relative to the current TEXdraw scaling and drawing segment. The specified coordinate affects the drawing size as determined by TEXdraw.

`\PSarc r:`*radius* `sd:`*start-angle* `ed:`*end-angle* `(`*x y*`)`
Draw a counterclockwise arc. The center of the arc is at the given position. The radius is specified by *radius*. The start and end angles (in degrees) are specified by *start-angle* and *end-angle*. If the PostScript current point is defined, this command also draws the line from the current point to the beginning of the arc. The line and arc become part of the current path. The current point becomes defined.

`\PSarcn r:`*radius* `sd:`*start-angle* `ed:`*end-angle* `(`*x y*`)`
Draw a clockwise arc. The center of the arc is at the given position. The radius is specified by *radius*. The start and end angles (in degrees) are specified by *start-angle* and *end-angle*. If the PostScript current point is defined, this command also draws the line from the current point to the beginning of the arc. The line and arc become part of the current path. The current point becomes defined.

The macro `\writeps` provides the general facility to write arbitrary PostScript commands to the PostScript file. This macro is used by the preceding commands and by the TEXdraw commands themselves. This facility has to be used with care since changes in position or scaling resulting from the PostScript commands are not known to TEXdraw.

`\writeps` {<*ps-commands*>}

Write PostScript commands to the intermediate PostScript file. Before the commands are inserted, any pending TEXdraw move is written to the PostScript file. The PostScript scaling gives 300 units/inch.

Appendix B TEXdraw Toolbox

This appendix describes some of the macros supplied with TEXdraw which can be used to define additional commands for creating drawings. The macros described here work in the user specified coordinate system. Some of these toolbox macros are used by the TEXdraw commands themselves, others are supplied in an auxiliary file '`txdtools.tex`'.

B.1 Coordinate parsing

The coordinate parsing macro `\getpos` is useful for creating new commands. This macro takes care of stripping leading and trailing blanks from coordinates specified between parentheses. In addition, symbolic coordinates are translated to the corresponding relative coordinate using the segment offset and scaling in effect.

The macro `\currentpos` returns the relative coordinates of the current position. The returned values are relative to the current segment and the current scaling. The macro `\cossin` returns the real-valued cosine and sine of the direction of the line joining two points. The macro `\vectlen` returns the length of a vector. The results appear as the value of user supplied macro names.

`\getpos (`*x y*`)\`*mx*`\`*my*
: Decode coordinate values. The coordinates specified by `(`*x y*`)` are decoded. Symbolic coordinates are translated to the corresponding relative coordinate using the current segment offset and scaling. The resulting character strings representing the real-valued coordinates are assigned to the macros specified by `\`*mx* and `\`*my*.

`\currentpos \`*mx*`\`*my*
: Return the coordinates of the current position. The coordinates are relative to the current segment offset and scaling. The resulting character strings representing the real-valued coordinates are assigned to the macros specified by `\`*mx* and `\`*my*.

`\cossin (`*x1 y1*`)(`*x2 y2*`)\`*cosa*`\`*sina*
: Return the cosine and sine of the direction of a vector joining two points. The cosine and sine of the angle of the vector which goes from `(`*x1 y1*`)` to `(`*x2 y2*`)`. The character strings representing these real-valued quantities are assigned to the macros specified by `\`*cosa* and `\`*sina*.

`\vectlen (`*x1 y1*`)(`*x2 y2*`)\`*len*
: Return the length of a vector joining two points. The length of the vector is relative to the current scaling. The character string representing the real-valued length is assigned to the macro specified by `\`*len*.

B.2 Real arithmetic

The TEXdraw toolbox supplies macros to perform real arithmetic on coordinate values. The result appears as the value of a user supplied macro name.

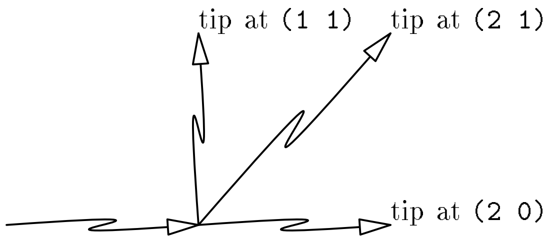
`\realadd` {*value1*} {*value2*} *sum*
: Add two real quantities, assigning the resultant character string representing the sum to the macro *sum*.

`\realmult` {*value1*} {*value2*} *prod*
: Multiply two real quantities, assigning the resultant character string representing the product to the macro *prod*.

`\realdiv` {*value1*} {*value2*} *result*
: Divide two real quantities, assigning the resultant character string representing the result of *value1*/*value2* to the macro *result*.

B.3 Arrow curve

This example illustrates the use of the TEXdraw toolbox routines to do computations with the coordinates. The problem will be tackled in two parts. First, we will produce a macro to place an arrowhead on a Bezier curve. Then given this macro, we will produce a macro which can draw a "wiggly" line from the current position to a given coordinate.



The first macro, `\cavec`, uses the `\cossin` command to determine the the cosine and sine of the angle of the line joining the second control point to the end point of the Bezier curve. Recall that the Bezier curve is tangent to this line at the end point. After drawing the Bezier curve, the scaling is set locally to absolute units of 0.05 inches. We go back down the line from the end point by 0.05 inches and draw an arrow vector to the end point from there. This arrow vector is mostly arrowhead, with little or no tail.

```
\def\cavec (#1 #2)(#3 #4)(#5 #6){
  \clvec (#1 #2)(#3 #4)(#5 #6)
  \cossin (#3 #4)(#5 #6)\cosa\sina
  \rmove (0 0)
  \bsegment
    \drawdim in \setsegscale 0.05
    \move ({-\cosa} -\sina)  \avec (0 0)
  \esegment}
```

Note the use of macros as arguments to a `\move` command. Minus signs are put in front of the macros. However, the value of the macro `\cosa` or `\sina` could be negative. Fortunately, TeX accepts two minus signs in a row and interprets the result as positive. Note that the `\rmove (0 0)` command before the beginning of the segment ensures that the Bezier curve is stroked before the arrowhead is drawn.

The second macro `\caw` builds on `\cavec`. The goal is to produce a wiggly vector that can be used as a pointer in a drawing. Consider the following symmetrical normalized Bezier curve.

```
\centertexdraw{ \move (0 0) \cavec (1.4 0.1)(-0.4 -0.1)(1 0) }
```

This curve has the appropriate wiggle. Now we want to be able to draw this curve, appropriately scaled and rotated. The macro `\caw` needs to do computations on the coordinates. First, `\caw` uses the macros `\getpos` and `\currentpos` to get the positions of the end and start of the curve. Next, the length of the vector is calculated using the macro `\vectlen`. A local macro `\rotatecoord` is used to rotate a coordinate pair about the origin, using the cosine and sine of the rotation angle. The vector length is used to scale the normalized curve. The remaining code draws the rotated, normalized curve.

```
\def\caw (#1 #2){
  \currentpos \xa\ya
  \cossin ({\xa} \ya)(#1 #2)\cosa\sina

% The nominal wiggly curve is (0 0) (1+dx dy) (-dx -dy) (1 0)
% Find the rotated offset (dx dy) -> (du dv)
  \rotatecoord (0.4 0.1)\cosa\sina \du\dv

% calculate the length of the vector
  \vectlen ({\xa} \ya)(#1 #2)\len

% draw the curve in normalized units
  \bsegment
    \setsegscale {\len}
    \realadd \cosa \du \tmpa  \realadd \sina \dv \tmpb
    \cavec ({\tmpa} \tmpb)({-\du} -\dv)({\cosa} \sina)
```

```
  \esegment
  \move (#1 #2)}

% rotate a coordinate (x y)
% arguments: (x y) cosa sina x' y'
%  x' = cosa * x - sina * y;  y' = sina * x + cosa * y
\def\rotatecoord (#1 #2)#3#4#5#6{
  \getpos (#1 #2)\xarg\yarg
  \realmult \xarg {#3} \tmpa  \realmult \yarg {#4} \tmpb
  \realadd \tmpa {-\tmpb} #5
  \realmult \xarg {#4} \tmpa  \realmult \yarg {#3} \tmpb
  \realadd \tmpa \tmpb #6}
```

Finally, the new macro can be used as follows.

```
\centertexdraw{
  \arrowheadtype t:W
  \move (0 0)
  \cavec (1.4 0.1)(-0.4 -0.1)(1 0)
  \move (1 0) \caw (1 1) \htext{tip at \tt (1 1)}
  \move (1 0) \caw (2 1) \htext{tip at \tt (2 1)}
  \move (1 0) \caw (2 0) \htext{tip at \tt (2 0)}

}
```

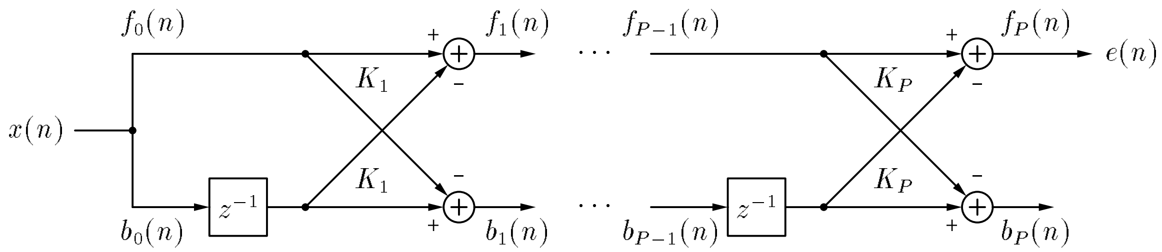
Note that the Bezier curve in the macro `\cavec` lies below the arrowhead. The example then draws an arrowhead of type `W` to erase the part of the line below the arrowhead.

Appendix C Examples

This appendix shows examples of the use of TEXdraw.

C.1 Block diagram of a lattice filter

The block diagram of a lattice filter uses a library of extended commands built from the basic TEXdraw commands.



The block diagram uses a "delay" block. This is defined as a segment which leaves the current position at the end of this block. A second macro, `\bdot`, draws a "big" dot which is used to mark junctions of lines. The `\Ttext` command centers text above a given point. The offset to position the text is local to a segment, resulting in no change to the current point. Similar macros to position text below a point (`\Btext`), to the left of a point (`\Ltext`) and to the right of a point (`\Rtext`) are used in the final drawing.

```
\def\delay {\bsegment
              \setsegscale 0.3
              \lvec (0 +0.5) \lvec (1 +0.5) \lvec (1 -0.5)
              \lvec (0 -0.5) \lvec (0 0)
              \textref h:C v:C  \htext (0.5 0){$z^{-1}$}
              \savepos (1 0)(*ex *ey)
            \esegment
            \move (*ex *ey)}
\def\bdot {\fcir f:0 r:0.02 }
\def\Ttext #1{\bsegment
               \textref h:C v:B  \htext (0 +0.06){#1}
              \esegment}
```

Several of the block diagram elements scale with the size of the summing nodes. The radius of the circles for the summing nodes is defined as the macro `\cradius`. The summing nodes will have enclosed plus signs, appropriately scaled. The plus sign is drawn by the macro **\pluss**. The macro **\pcir** draws both the circle and the plus sign. The incoming lines to a summing node will be labelled with plus or minus signs (characters this time), placed at the appropriate position with respect to the center of the summing node. These positions are given in terms of compass directions. The macro **\putwnw** places text west by north-west relative to the center of the summing node.

```
\def\cradius {0.08}
\def\pluss {\bsegment
              \setsegscale {\cradius}
              \move (-0.5 0) \lvec (+0.5 0)
              \move (0 -0.5) \lvec (0 +0.5)
            \esegment}
\def\pcir {\lcir r:{\cradius} \pluss}
\def\puttext (#1 #2)#3{\bsegment
                         \setsegscale {\cradius}
                         \textref h:C v:C \htext (#1 #2){#3}
                       \esegment}
\def\putwnw #1{\puttext (-1.7 +1.2){#1}}
```

The block diagram has vectors arriving and departing from the summing nodes (circles). One could calculate the points of intersection of the lines with the circles, and then enter the values into the TEXdraw code. However, in this example, we implement an automated procedure. Two macros are needed, an arrow vector to a circle (\avectoc) and an arrow vector leaving from a circle (\avecfrc). The macros will calculate the point of intersection with the circle and start or end the vector at the intersection point.

The arrow macros use scaling and relative positioning inside of a drawing segment. In the case of the macro **\avectoc**, a move is made to the final point (center of the circle), then within a drawing segment, a scaled move is made back towards the initial point to determine the intersection point with the circle.

```
\def\avectoc (#1 #2){\currentpos \xa\ya
                     \cossin ({\xa} \ya)(#1 #2)\cosa\sina
                     \savepos (#1 #2)(*tx *ty)
                     \bsegment
                       \move (*tx *ty)
                       \setsegscale {\cradius}
                       \rmove ({-\cosa} -\sina)
                       \savecurrpos (*ex *ey)
                     \esegment
                     \avec (*ex *ey)
                     \move (#1 #2)}
\def\avecfrc (#1 #2){\currentpos \xa\ya
                     \cossin ({\xa} \ya)(#1 #2)\cosa\sina
```

```
\bsegment
  \setsegscale {\cradius}
  \move ({\cosa} \sina)
  \savecurrpos (*ex *ey)
\esegment
\move (*ex *ey)
\avec (#1 #2)}
```

Having defined these macros, we are ready to draw the block diagram. The first and last sections of the lattice filter are very similar, differing mainly in the text labels. With more effort, code could be shared between the commands used to draw these blocks.

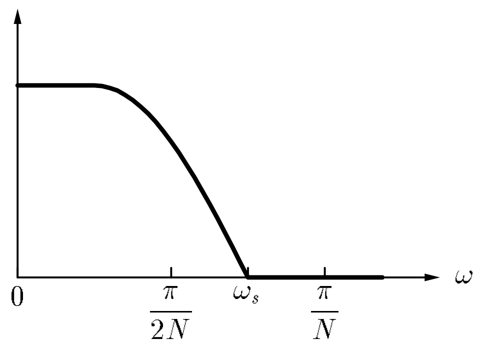
```
\centertexdraw{
\drawdim in
\arrowheadtype t:F  \arrowheadsize l:0.08 w:0.04
\def\pl {$\scriptscriptstyle +$} \def\mn {$\scriptscriptstyle -$}

\move (0 +0.63) \move (0 -0.60) \move (0 0) % compensate for the text size

% Input to the first stage
\bsegment
  \Ltext{$x(n)$}
  \lvec (0.3 0) \bdot \lvec (0.3 +0.4) \move (0.3 0) \lvec (0.3 -0.4)
  \savepos (0.3 0)(*ex *ey)
\esegment
\move (*ex *ey)

% first lattice stage
\bsegment
  \move (0 +0.4)  \avectoc (1.7 +0.4)
  \pcir \putwnw{\pl} \puts{\mn}
  \avecfrc (2.1 +0.4)
  \move (0 -0.4)  \avec (0.4 -0.4) \delay \avectoc (1.7 -0.4)
  \pcir \putwsw{\pl} \putn{\mn}
  \avecfrc (2.1 -0.4)
  \move (0.9 +0.4)  \bdot  \avectoc (1.7 -0.4)
  \move (0.9 -0.4)  \bdot  \avectoc (1.7 +0.4)
  \move (0.1 +0.42) \Ttext {$f_0(n)$}
  \move (2.0 +0.42) \Ttext {$f_1(n)$}
  \move (0.1 -0.4)  \Btext {$b_0(n)$}
  \move (2.0 -0.4)  \Btext {$b_1(n)$}
  \textref h:L v:B  \htext (1.15 +0.2){$K_1$}
  \textref h:L v:T  \htext (1.15 -0.2){$K_1$}
  \savepos (2.1 0)(*ex *ey)
\esegment
\move (*ex *ey)
```

```
% center section
\bsegment
  \textref h:C v:C \htext (0.3 +0.4){$\cdots$}
  \htext (0.3 -0.4){$\cdots$}
  \savepos (0.6 0)(*ex *ey)
\esegment
\move (*ex *ey)

% last lattice stage
\bsegment
  \move (0 +0.4)  \avectoc (1.7 +0.4)
  \pcir \putwnw{\pl} \puts{\mn}
  \avecfrc (2.3 +0.4) \Rtext{$e(n)$}
  \move (0 -0.4)  \avec (0.4 -0.4) \delay \avectoc (1.7 -0.4)
  \pcir \putwsw{\pl} \putn{\mn}
  \avecfrc (2.1 -0.4)
  \move (0.9 +0.4)  \bdot  \avectoc (1.7 -0.4)
  \move (0.9 -0.4)  \bdot  \avectoc (1.7 +0.4)
  \move (0.1 +0.42) \Ttext {$f_{P-1}(n)$}
  \move (2.0 +0.42) \Ttext {$f_P(n)$}
  \move (0.1 -0.4)  \Btext {$b_{P-1}(n)$}
  \move (2.0 -0.4)  \Btext {$b_P(n)$}
  \textref h:L v:B  \htext (1.15 +0.2){$K_P$}
  \textref h:L v:T  \htext (1.15 -0.2){$K_P$}
\esegment
}
```

The macros used in this example are similar to the block diagram macros defined in the file '`blockdiagram.tex`'.

C.2 Filter response graph

This example shows the response of a canonical filter. TEXdraw is not well suited for general purpose graphing — it has no coordinate translation facility nor does it have separate x and y scaling. Nonetheless, for certain simple graphs, TEXdraw is adequate.



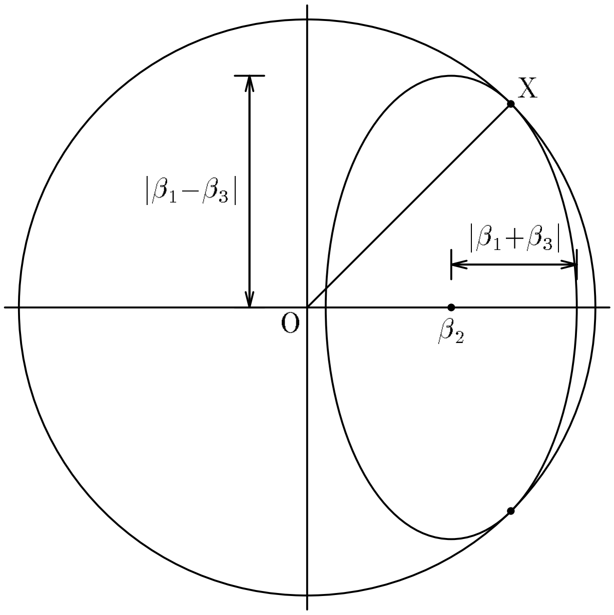
In this example, macro `\ticklab` places a labelled axis tick at a given position. The data is specified in a straightforward manner, having been scaled beforehand to give the desired aspect ratio for the graph.

```
\centertexdraw{
\arrowheadtype t:F  \arrowheadsize l:0.08 w:0.04
\def\ds {\displaystyle}
\def\ticklab (#1 #2)#3{\move(#1 #2)
                       \bsegment
                         \lvec (0 0.05)
                         \textref h:C v:T \htext (0 -0.05){#3}
                       \esegment}
\def\Rtext #1{\bsegment
                \textref h:L v:C \htext (+0.08 0){#1}
              \esegment}

\move (2.4 -0.3)    % move to set the size

\move (0 0)
% Axes
\avec (0 +1.4)
\move (0 0) \avec (2.2 0) \Rtext{$\omega$}
\ticklab (0 0)   {0}
\ticklab (0.8 0) {$\ds {\pi \over 2N} $}
\ticklab (1.2 0) {$\omega_s$}
\ticklab (1.6 0) {$\ds {\pi \over N} $}

\linewd 0.025
\move (0 1)
\lvec (0.4 1)
\lvec (0.44 0.998)
\lvec (0.48 0.988)
\lvec (0.52 0.973)
\lvec (0.56 0.951)
 ...
\lvec (1.08 0.233)
\lvec (1.12 0.156)
\lvec (1.16 0.078)
```

```
\lvec (1.20 0)
\lvec (1.9 0)
}
```

C.3 Geometric construction

This example shows a geometric construction which places an ellipse tangent to an enclosing circle. The size of the ellipse is determined from geometric considerations. Macros are used to modularize the code. The example alters the unit scale factor. This allows the drawing to be carried out in units normalized to the radius of the circle.




```
\centertexdraw{
\arrowheadtype t:V \arrowheadsize l:0.08 w:0.04
\linewd 0.01
\setunitscale 1.5           % circle will have radius 1.5 inches

\def\Btext #1{\bsegment
               \textref h:C v:T \htext (0 -0.04){#1}
             \esegment}
\def\Ttext #1{\bsegment
               \textref h:C v:B \htext (0 +0.04){#1}
             \esegment}
\def\Ltext #1{\bsegment
               \textref h:R v:C \htext (-0.04 0){#1}
             \esegment}
\def\bdot {\fcir f:0 r:0.0133 }
```

```
\def\vtick {\bsegment
              \move (0 -0.05) \lvec (0 +0.05)
            \esegment}
\def\htick {\bsegment
              \move (-0.05 0) \lvec (+0.05 0)
            \esegment}
\def\Hlen #1#2{\bsegment
                 \vtick \avec ({#1} 0) \vtick \avec (0 0)
                 \relsegscale 0.5
                 \move ({#1} 0) \Ttext {#2}
               \esegment}
\def\Vlen #1#2{\bsegment
                 \htick \avec (0 {#1}) \htick \avec (0 0)
                 \relsegscale 0.5
                 \move (0 {#1}) \Ltext {#2}
               \esegment}

\lcir r:1                        % circle
\move (-1.05 0) \lvec ( 1.05 0)  % axes
\move (0 -1.05) \lvec (0  1.05)

\move (0 0) \lvec (0.707 0.707) \bdot
\rmove (0.02 0.02) \textref h:L v:B \htext {X}
\move (0.707 -0.707) \bdot
\textref h:R v:T \htext(-0.02 -0.02){O}

\move (0.5 0)                    % center of ellipse
\bsegment
  \lellip rx:0.435 ry:0.804
  \bdot \Btext {$\beta_2$}
  \move (0 0.15) \Hlen {0.435}{$|\beta_1{+}\beta_3|$}
  \move (-0.7 0) \Vlen {0.804}{$|\beta_1{-}\beta_3|$}
\esegment
}
```

Appendix D Alphabetic listing of commands

`\arrowheadsize l:`*length* `w:`*width*

Set the arrowhead size to be *length* units long and *width* units wide. The width is measured across the "base" of the arrowhead. The initial arrowhead size has a *length* of 0.16 inches and a *width* of 0.08 inches.

`\arrowheadtype t:`*type*

Set the arrowhead type to *type*, where *type* is one of `F`, `T`, `W`, `V`, or `H`. There are two kinds of arrowheads. The first kind is a triangle. There are 3 variants: type `T` is an empty triangle, type `F` is a filled triangle (using the current gray level for lines), type `W` is a triangle filled with white. The second kind of arrowhead is an open ended Vee. There are 2 variants: type `V` has the stem continue to the tip, type `H` has the stem stop at the base of the arrowhead. The initial arrowhead type is `T`.

`\avec (`*x y*`)`

Draw a line with an arrowhead from the current position to `(`*x y*`)`. The new current position is `(`*x y*`)`. The arrowhead is aligned with the line, with the tip at `(`*x y*`)`.

`\begin{texdraw}`

Start a TEXdraw drawing. The drawing is terminated with an `\end{texdraw}` command. This command is for use with LaTEX.

`\bsegment`

Start a drawing segment. The coordinate system is shifted such that the current position corresponds to the coordinate `(0 0)`. Changes to scaling, position and line parameters stay local to the drawing segment.

`\btexdraw`

Start a TEXdraw drawing. The drawing is terminated with an `\etexdraw` command.

`\centertexdraw { ... }`

Center a TEXdraw box. The argument contains TEXdraw commands. The resulting box has the horizontal size `\hsize` and height equal to the height of the drawing.

`\clvec (`*x1 y1*`)(`*x2 y2*`)(`*x3 y3*`)`

Draw a Bezier curve from the current position to the coordinate `(`*x3 y3*`)` which becomes the new current position. The coordinates `(`*x1 y1*`)` and `(`*x2 y2*`)` serve as control points for the curve. Only the last coordinate given is used to update the size of the drawing.

`\drawbb` Draw a ruled box around the effective size of a drawing produced by TEXdraw commands.

`\drawdim` *dim*

Set the units to *dim*. The argument *dim* can be any valid TEX dimension unit. The units are used to interpret coordinate values. Examples of valid units: `cm`, `mm`, `in`, `pt`, and `bp`.

`\end{texdraw}`

End a TEXdraw drawing started with a `\begin{texdraw}` command. The resulting TEXdraw drawing is placed in a box with height equal to the height of the drawing and width equal to the width of the drawing. The depth of the box is zero. This command is for use with LaTEX.

`\esegment`

End a drawing segment. The current position in effect before the corresponding `\bsegment` command is restored. The scaling and line parameter values revert to those in effect before the corresponding `\bsegment` was invoked.

`\etexdraw`

End a TEXdraw drawing started with a `\btexdraw` command. The resulting TEXdraw drawing is placed in a box with height equal to the height of the drawing and width equal to the width of the drawing. The depth of the box is zero.

`\everytexdraw { ... }`

Specify TEXdraw commands to be executed at the beginning of every TEXdraw drawing.

`\fcir f:`*level* `r:`*radius*

Draw a filled circle with center at the current position. The radius is specified by *radius*. The circle is painted with the gray level specified by *level*. A gray level of 1 corresponds to white, with decreasing values getting darker. The level 0 is full black. This command does not draw a line along the circumference. The drawing size is increased if necessary to contain the circle.

`\fellip f:`*level* `rx:`*x-radius* `ry:`*y-radius*

Draw a filled ellipse with center at the current position. The radius in the x direction is specified by *x-radius*. The radius in the y direction is specified by *y-radius*. The ellipse is painted with the gray level specified by *level*. A gray level of 1 corresponds to white, with decreasing values getting darker. The level 0 is full black. This command does not draw a line along the boundary of the ellipse. The drawing size is increased if necessary to contain the ellipse.

`\htext (`*x y*`){`*text*`}`

`\htext {`*text*`}`

The first form of this command places the TEX text *text* horizontally with the text reference point at the coordinate `(`*x y*`)`. The new current position is `(`*x y*`)`. The second form of this command places the TEX text *text* horizontally with the text reference point at the current position. The text reference point is set with the `\textref` command.

`\ifill f:`*level*

Close the current path and paint the interior of the region with gray level *level*. The line around the path is not drawn. Gray levels are real values from 0 (black) through intermediate values (grays) to 1 (white).

`\larc r:`*radius* `sd:`*start-angle* `ed:`*end-angle*

Draw a counterclockwise arc. The center of the arc is at the current position. The radius is specified by *radius*. The start and end angles (in degrees) are specified by *start-angle* and *end-angle*. This command does not affect the limits (size) of the drawing.

`\lcir r:`*radius*

Draw a circle with center at the current position. The radius is specified by *radius*. This command draws a line along the circumference of the circle. The drawing size is increased if necessary to contain the circle.

`\lellip rx:`*x-radius* `ry:`*y-radius*

Draw an ellipse with center at the current position. The radius in the x direction is specified by *x-radius*. The radius in the y direction is specified by *y-radius*. The drawing size is increased if necessary to contain the ellipse.

`\lfill f:`*level*

Close the current path, draw the line around the path using the current grey level for lines and paint the interior of the region with specified gray level *level*. Gray levels are real values from 0 (black) through intermediate values (grays) to 1 (white).

`\linewd` *width*

Set the line width to *width* units. Initially *width* is 0.01 inches (corresponding to 3 pixels at 300 pixels to the inch).

`\lpatt (`*pattern*`)`

Set lines to have the pattern (*pattern*). A pattern is a sequence of on/off lengths separated by blanks and enclosed in parentheses. The lengths alternately specify the length of a dash and the length of a gap between dashes. Each length is interpreted using the current scaling and drawing units. The pattern is used cyclically. The empty pattern signifies a solid line. The initial line pattern is a solid line, corresponding to the empty pattern `\lpatt ()`.

`\lvec (`*x y*`)`

Draw a line from the current position to coordinate (*x y*). The new current position is (*x y*).

`\move (`*x y*`)`

Move to coordinate (*x y*). The new current position is (*x y*).

`\ravec` (*dx dy*)

Draw a line with an arrowhead from the current position, *dx* units in the *x* direction and *y* units in the *y* direction. The final position becomes the new current position. The arrowhead is aligned with the line, with the tip at the new current position.

`\relsegscale` *value*

Adjust the segment scale factor by multiplying by *value*. This has the effect of multiplying the current overall scale factor by the same factor. The overall scaling factor is the product of the unit scale factor and the segment scale factor.

`\relunitscale` *value*

Adjust the unit scale factor by multiplying by *value*. This has the effect of multiplying the overall scale factor by the same factor. The overall scaling factor is the product of the unit scale factor and the segment scale factor.

`\rlvec` (*dx dy*)

Draw a line from the current position, *dx* units in the *x* direction and *dy* units in the *y* direction. The final position becomes the new current position.

`\rmove` (*dx dy*)

Move from the current position, *dx* units in the *x* direction and *dy* units in the *y* direction. The final position becomes the new current position.

`\rtext td:`*angle* `(x y){`*text*`}`
`\rtext td:`*angle* `{`*text*`}`

The first form of this command places the TEX text *text* at an angle with the text reference point at the coordinate (*x y*). The new current position is (*x y*). The second form of this command places the TEX text *text* at an angle with the text reference point at the current position. In both cases, the TEX text is placed in a box and the box is rotated counterclockwise by *angle* degrees about the text reference point. The text reference point is set with the `\textref` command.

`\savecurrpos` (**px *py*)

Save the current position as the absolute position referenced by (**px *py*).

`\savepos` (*x y*)(**px *py*)

Save the coordinate position (*x y*) as the absolute position referenced by (**px *py*). The coordinate (*x y*) is interpreted in the normal fashion as a coordinate relative to the current segment, using the current scaling factors and drawing unit.

`\setgray` *level*

Set the gray level of lines. Gray levels are real values from 0 (black) through intermediate values (gray) to 1 (white). The initial gray level is 0 corresponding to black.

`\setsegscale` *scale*

Set the segment scale factor. The argument *scale* is a real number which is used to scale coordinate values. The overall scale factor is the product of the unit scale factor and the segment scale factor.

`\setunitscale` *scale*

Set the unit scaling to *scale*. The argument *scale* is a real number which is used to scale coordinate values. The overall scaling factor is the product of the unit scale factor and the segment scale factor.

`\texdrawbox { ... }`

Create a TEXdraw box. The argument contains TEXdraw commands. This macro returns a TEX box with height equal to the height of the drawing and width equal to the width of the drawing. The depth of the box is zero.

`\textref h:`*h-ref* `v:`*v-ref*

Set the text reference point for subsequent text commands. The horizontal reference point *h-ref* is one of `L`, `C` or `R` (left, center or right). The vertical reference point *v-ref* is one of `T`, `C` or `B` (top, center or bottom). For rotated text, the reference point is determined before rotation. The initial text reference point corresponds to `\textref h:L v:B`.

`\vtext (x y){`*text*`}`
`\vtext {`*text*`}`

The first form of this command places the TEX text *text* vertically with the text reference point at the coordinate `(`*x y*`)`. The new current position is `(`*x y*`)`. The second form of this command places the TEX text *text* vertically with the text reference point at the current position. In both cases, the TEX text is placed in a box and the box is rotated counterclockwise by 90 degrees about the text reference point. The text reference point is set with the `\textref` command.

Command Index

\

\arc ... 13
\arrowheadsize ... 9
\arrowheadtype ... 9
\avec ... 8
\begin{texdraw} ... 5
\bsegment ... 17
\btexdraw ... 5
\centertexdraw ... 5
\clvec ... 14
\cossin ... 35
\currentpos ... 35
\drawbb ... 21
\drawdim ... 7
\end{texdraw} ... 5
\esegment ... 17
\etexdraw ... 5
\everytexdraw ... 5
\fcir ... 13
\fellip ... 13
\getpos ... 35
\htext ... 10
\ifill ... 15
\lcir ... 12
\lellip ... 13
\lfill ... 15
\linewd ... 8
\lvec ... 8
\move ... 8
\PSarc ... 32
\PSarcn ... 32
\PSclosepath ... 31
\PSfill ... 31
\PSlineto ... 32
\PSmoveto ... 32
\PSnewpath ... 31
\PSsetlinecap ... 31
\PSsetlinejoin ... 31
\PSstroke ... 31
\ravec ... 8
\realadd ... 36
\realdiv ... 36
\realmult ... 36
\relsegscale ... 20
\relunitscale ... 20
\rlvec ... 8
\rmove ... 8
\rtext ... 10
\savecurrpos ... 18
\savepos ... 18
\setgray ... 8
\setsegscale ... 20
\setunitscale ... 19
\textref ... 11
\vectlen ... 35
\vtext ... 10
\writeps ... 32

A

arc ... 32
arcn ... 32

C

closepath ... 31

F

fill ... 31

L

lineto ... 32

M

moveto ... 32

N

newpath ... 31

S

setlinecap.. 31
setlinejoin....................................... 31
stroke 31

Concept Index

A

accessing TEXdraw ... 3, 23
angle of a vector ... 35
arcs ... 12, 32
arrowhead parameters ... 8
arrows ... 7

B

Bezier curves ... 14

C

circles ... 12
command syntax ... 5
coordinate parsing ... 35
coordinate specification ... 6
coordinate, symbolic ... 18
coordinates ... 6
cosine of a vector direction ... 35
current position ... 7, 21, 35
current position in PostScript ... 31
curves ... 14

D

dashed lines ... 8
direction of a line ... 35
distribution ... 1
dotted lines ... 8
drawing segments ... 17
`dvi2ps` printer driver ... 23
`dvialw` printer driver ... 23
`dvilaser` printer driver ... 23
`dvips` printer driver ... 1, 23, 28
`dvipsone` printer driver ... 23
`dvitops` printer driver ... 23
`dviwindo` printer driver ... 23

E

ellipses ... 12
Encapsulated PostScript File ... 28
errors while using TEXdraw ... 25
example, arrow curve ... 36
example, block diagram ... 39
example, circle and ellipse ... 44
example, graph ... 42

F

fill operations, interaction with drawing segments ... 17
filled circles ... 12
filling regions ... 15, 31

G

`graphics` package ... 1, 3, 23, 28
graphics placement ... 28
gray levels for lines ... 8

I

implementation ... 26
initial current position ... 21
invoking TEXdraw ... 3, 23

L

LaTEX ... 1, 3, 23
length of a vector ... 35
line cap ... 31
line join ... 31
line width ... 8
lines ... 7, 32
listing of commands ... 47

M

moves ... 7, 32

O

`oztex` printer driver ... 23

P

painting regions ... 15
paths ... 15, 17, 31

`pctexps` printer driver 23
`pctexwin` printer driver 23
placement of graphics and text 28
plain TEX 3
position specification 6
positions, saving 18
PostScript commands 31
PostScript printer drivers 23, 28
printer drivers 23, 28
problems while using TEXdraw 25
`psprint` driver 23

R

relative positioning 8
relative scaling 19
resolution 27
rotated text 10, 23, 27, 28

S

saving positions 18
scaling 26
scaling coordinates 19
segment scale 19
segments 17
sine of a vector direction 35
size of the drawing 21
stroking lines 17, 31
symbolic coordinate 18
syntax of commands 5

T

`texdraw` package 3, 23, 28
text commands 10
text placement 28
text rotation 10, 23, 27, 28
`textures` printer driver 23

U

unit scale 19

V

vectors 7
vertical text 10

W

width of lines 8

X

`xdvi` driver 23

Table of Contents

1 Introduction 1
1.1 Distribution information 1
2 Using the TEXdraw Commands 3
2.1 Accessing TEXdraw 3
2.2 Command syntax 5
2.3 TEXdraw coordinates 6
2.4 Coordinate specification 6
2.5 Line vectors 7
2.6 TEX text 10
2.7 Circles, ellipses and arcs 12
2.8 Bezier curves 14
2.9 Fill commands 15
3 Drawing Segments and Scaling 17
3.1 Drawing segments 17
3.2 Drawing paths 17
3.3 Saving positions 18
3.4 Scaling coordinates 19
3.5 Drawing size 21
3.6 Initial current position 21
4 Using TEXdraw with LaTEX 23
4.1 PostScript printer drivers 23
5 More Details 25
5.1 Errors while using TEXdraw 25
5.2 Extending TEXdraw 26
5.2.1 Scaling 26
5.2.2 Resolution 27
5.2.3 Text placement 27
5.2.4 The intermediate PostScript file 28
5.3 How TEXdraw merges graphics and text 28
Appendix A PostScript Commands 31

Appendix B TEXdraw Toolbox **35**

B.1 Coordinate parsing 35
B.2 Real arithmetic 36
B.3 Arrow curve 36

Appendix C Examples **39**

C.1 Block diagram of a lattice filter 39
C.2 Filter response graph 42
C.3 Geometric construction 44

Appendix D Alphabetic listing of commands **47**

Command Index **53**

Concept Index **55**