The pict2e package*

Hubert Gäßlein† and Rolf Niepraschk‡

2004/02/19

Contents

*This document corresponds to pict2e.sty v0.2j, dated 2004/02/19.

†HubertJG@open.mind.de

‡Rolf.Niepraschk@ptb.de

List of Figures

1 Introduction

Here's a quote from the obsolete original official version of the pict2e package (1994–2003):

> *The package pict2e that is mentioned in the 2nd edition of "LaTeX: A Document Preparation System" has not yet been produced. It is unlikely that the LaTeX3 Project Team will ever produce this package thus we would be very happy if someone else creates it.*

:-) Finally, someone has produced a working implementation of the pict2e package.

This package redefines some of the drawing commands of the LaTeX picture environment. Like the graphics and color packages, it uses driver files.

Currently there are only back-ends for PostScript and PDF. (Other output formats may be added in the future.)

Note/Warning:

- Documentation has been written somewhat "hastily" and may be inaccurate.
- The status of this package is currently somewhere between "beta" and "release" ... Users and package programmers should *not* rely on *any* feature sported by the internal commands. (Especially, the control sequence names may change without notice in future versions of this package.)

2 Usage

To use the pict2e package, you put a `\usepackage[`⟨*optionlist*⟩`]{`⟨*pict2e*⟩`}` instruction in the preamble of your document. Likewise, package writers just say `\RequirePackage[`⟨*optionlist*⟩`]{`⟨*pict2e*⟩`}` in an appropriate place in their package. (Nothing unusual here.)
Like the graphics and color packages, the pict2e package supports a configuration file: see Section 2.2.

2.1 Package options

2.1.1 Driver options

driver	notes	driver	notes
dvips	x	dvipsone	x?
xdvi	x	dviwindo	x?
pdftex	x	dvipdf	x?
vtex	x	textures	x?
dvipdfm	x	pctexps	x?
oztex	(x)	pctex32	x?

x = supported; (x) = supported but untested;
x? = not yet implemented

The driver options are (mostly) implemented by means of definition files (`p2e-*.def`). For details, cf. file `p2e-drivers.dtx`.

You should specify the same driver for pict2e you use with the graphics/x and color packages.

2.1.2 Other options

option	meaning
ltxarrows	draw LaTeX style vectors (default)
pstarrows	draw PSTricks style vectors

2.1.3 Debugging options

option	meaning
original	Suppresses the new definitions (for testing purposes only).
debug	Suppresses the compressing of pdfTeX output; marks the pict2e generated code in the output files (for testing purposes only).
hide	Suppresses all graphics output.

2.2 Configuration file

Similar to the graphics and color packages, in most cases it is not necessary to give a driver option explicitly, if a suitable configuration file `pict2e.cfg` is present on your system (cf. the example file `pict2e-example.cfg`).

2.3 Details: Changes to user-level commands

For details, look up "`pict2e` package" in the index of the LaTeX manual [1].

A collection of quotes from the LaTeX manual [1].

From [1, p. 118]:

> *However, the* `pict2e` *package uses device-driver support to provide enhanced versions of these commands that remove some of their restrictions. The enhanced commands can draw straight lines and arrows of any slope, circles of any size, and lines (straight and curved) of any thickness.*

From [1, p. 179]:

> `pict2e` *Defines enhanced versions of the* `picture` *environment commands that remove restrictions on the line slope, circle radius, and line thickness.*

From [1, pp. 221–223]:

> `\qbezier`
> *(With the* `pict2e` *package, there is no limit to the number of points plotted.)*
>
> `\line` **and** `\vector` *Slopes* $\|x\|, \|y\| \leq 6$ *or* 4*, with no common divisor except* ± 1*:*
> *(These restrictions are eliminated by the* `pict2e` *package.)*
>
> `\line` **and** `\vector` *Smallest extent that can be drawn:*
> *(This does not apply when the* `pict2e` *package is loaded.)*
>
> `\circle` **and** `\circle*` *Largest circles and disks that can be drawn:*
> *(With the* `pict2e` *package, any size circle or disk can be drawn.)*
>
> `\oval` `[`⟨*rad*⟩`]:`
> *An explicit* rad *argument can be used only with the* `pict2e` *package; the default value is the radius of the largest quarter-circle LaTeX can draw without the* `pict2e` *package.*

2.3.1 Line

`\line` `\line(`⟨*X,Y*⟩`){`⟨*LEN*⟩`}`

In the Standard LaTeX implementation the slope arguments (⟨*X,Y*⟩) are restricted to integers in the range $-6 \leq X, Y \leq +6$, with no common divisors except ± 1. Furthermore, only horizontal and vertical lines can assume arbitrary thickness; sloped lines are restricted to the widths given by `\thinlines` and `\thicklines`.

From [1, p. 222]:

> *These restrictions are eliminated by the* `pict2e` *package.*

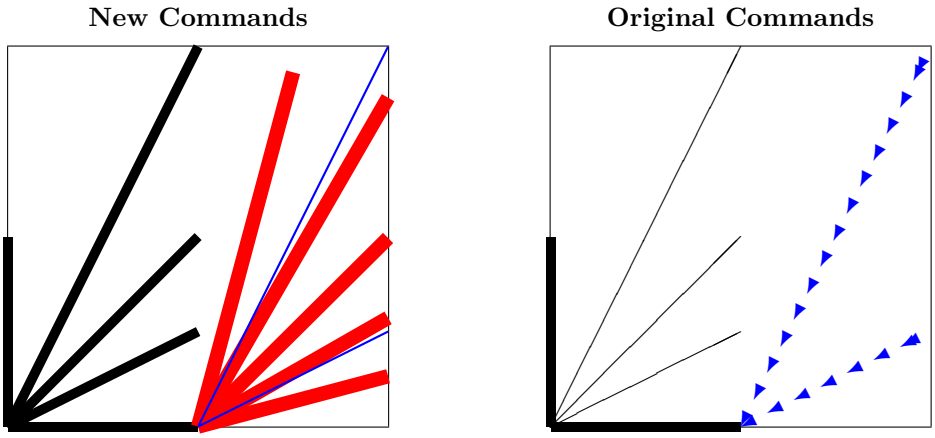


Figure 1: Line

However, the current implementation restricts these arguments to integers in the range $-1000 \le X, Y \le +1000$, which should be enough.

Furthermore, we still have $(0, 0) \mapsto (0, 1)$. I.e., the "impossible" slope is silently converted to a vertical line extending in the upward direction.

In the Standard LaTeX implementation the horizontal extent of sloped lines must be at least 10 pt.

From [1, p. 222]:

> *This does not apply when the* `pict2e` *package is loaded.*

Figure 1 shows the difference between the old and new implementation: The black lines in the left half of each picture have all slopes that conform to the restrictions of Standard LaTeX. However, with the new implementation of pict2e sloped lines may assume any width given by `\linethickness`. The right half demonstrates that now arbitrary slopes are possible. The red lines correspond to angles of $15\,°, 30\,°, 45\,°, 60\,°, 75\,°$, respectively. This was achieved by multiplying the sine and cosine by 1000 and rounding to the nearest integer.

```
\put(50,0){\line(966,259){50}}
\put(50,0){\line(866,500){50}}
\put(50,0){\line(707,707){50}}
\put(50,0){\line(500,866){50}}
\put(50,0){\line(259,966){25}}
```

The blue lines represent "illegal" slopes, with common divisors. Note the funny effect Standard LaTeX produces in such cases. (In LaTeX releases prior to 2003/12/01, some such slopes might even lead to infinite loops! Cf. problem report latex/3570.)

2.3.2 Vector

`\vector` `\vector(`$\langle X, Y \rangle$`){`$\langle LEN \rangle$`}`

In the Standard LaTeX implementation the slope arguments (`(`$\langle X, Y \rangle$`)`) are restricted to integers in the range $-4 \le X, Y \le +4$, with no common divisors except ± 1. Furthermore, arrow heads come only in two shapes, corresponding to `\thinlines` and `\thicklines`. (There's also a flaw: the lines will be printed over the arrow heads.)

From [1, p. 222]:

> *These restrictions are eliminated by the* `pict2e` *package.*

However, the current implementation restricts these arguments to integers in the range $-1000 \le X, Y \le +1000$, which should be enough.

Furthermore, unlike the Standard LaTeX implementation, which silently converts the "impossible" slope to a vertical line extending in the upward direction ($(0, 0) \mapsto (0, 1)$), the pict2e package now treats this as an error.

Figure 2 shows the difference between the old and new implementation: The black arrows have all "legal" slopes; the red ones have "illegal" ones. The new implementation imposes no restriction with respect to line thickness, minimum horizontal extent, and slope.

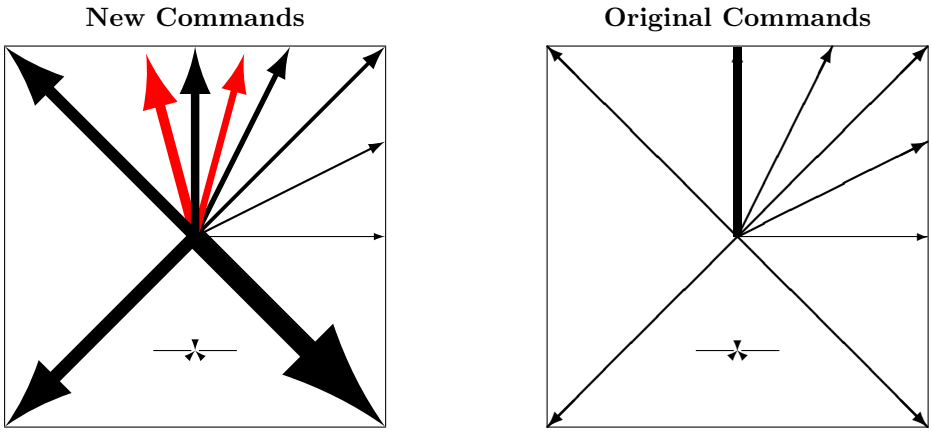


Figure 2: Vector

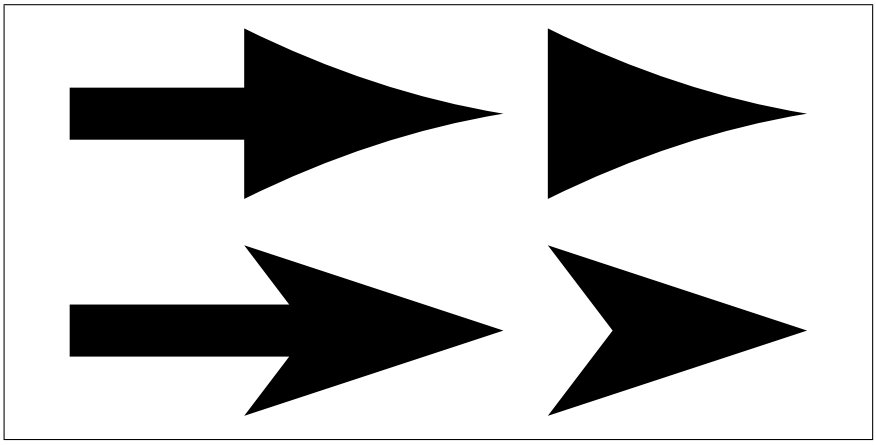


Figure 3: Vector: shape variants

As with Standard LaTeX, the arrow head will always be drawn. In particular, only the arrow head will be drawn, if the total length of the arrow is less than the length of arrow head. See right hand side of Figure 3.

The current version of the pict2e package offers two variants for the shape of the arrow heads, controlled by package options. One variant tries to mimic the fonts used in the Standard LaTeX implementation (package option ltxarrows, the default), though it is difficult to extrapolate from just two design sizes. The other one is implemented like the arrows of the PSTricks package [8] (package option pstarrows). See Figure 3.

2.3.3 Circle and Dot

`\circle` `\circle{`⟨*DIAM*⟩`}`
`\circle*` `\circle*{`⟨*DIAM*⟩`}`

The (hollow) circles and disks (filled circles) from the Standard LaTeX implementation had severe restrictions on the number of different diameters and maximum diameters available.

From [1, p. 222]:

> *With the* `pict2e` *package, any size circle or disk can be drawn.*

With the new implementation there are no more restrictions to the diameter argument. (However, negative or zero diameters are now trapped as an error.)

Furthermore, hollow circles (like sloped lines) can now be drawn with any line thickness. Figure 4 shows the difference.

2.3.4 Oval

`\oval` `\oval[`⟨*rad*⟩`](`⟨*X,Y*⟩`)[`⟨*POS*⟩`]`
`\maxovalrad`

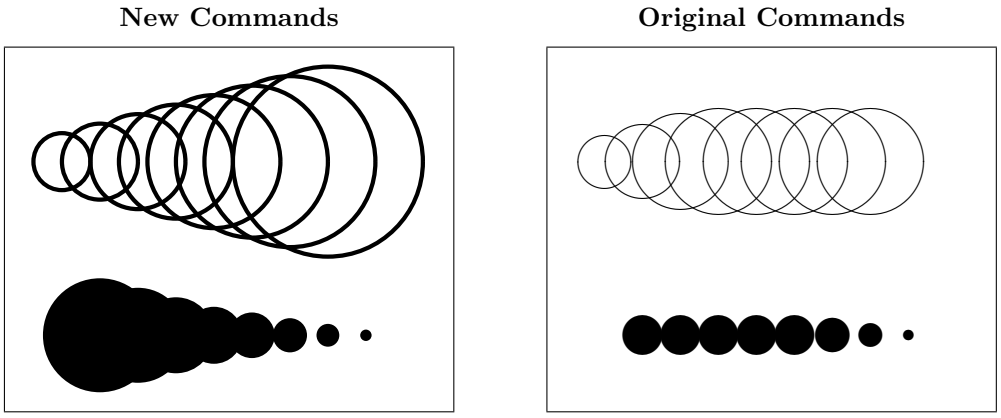


Figure 4: Circle and Dot

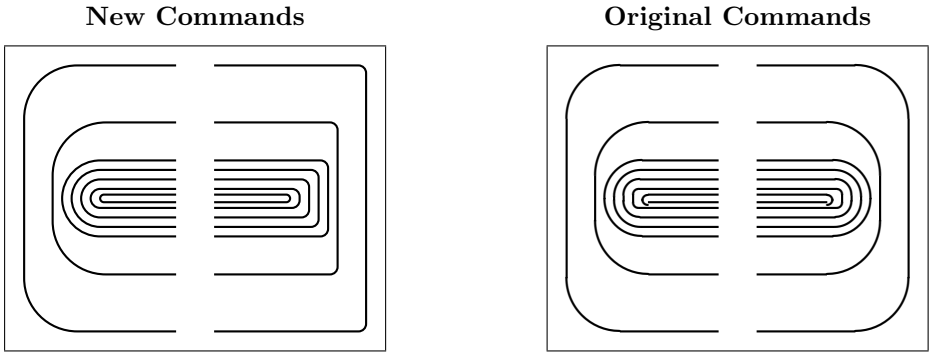


Figure 5: Radius argument for `\oval` vs. `\maxovalrad`

In the Standard LaTeX implementation, the user has no control over the shape of an oval besides its size, since its corners would always consist of the "quarter circles of the largest possible radius less than or equal to *rad*" [1, p. 223].

From [1, p. 223]:

> *An explicit* rad *argument can be used only with the* `pict2e` *package; the default value is the radius of the largest quarter-circle LaTeX can draw without the* `pict2e` *package.*

This default value is 20 pt, a length. However, in an early reimplementation of the picture commands [5], there is such an optional argument too, but it is given as a mere number, to be multiplied by `\unitlength`.

Since both alternatives may make sense, we left the choice to the user. (See Figure 6 for the differences. The number at the centre of each oval gives the relative value of `\unitlength`.) I.e., this implementation of `\oval` will "auto-detect" whether its `[`⟨*rad*⟩`]` argument is a length or a number. Furthermore, the default value is not hard-wired either; the user may access it under the moniker `\maxovalrad`, via `\renewcommand*`. (Names or values of length and counter registers may be given as well, both as an explicit `[`⟨*rad*⟩`]` argument and when redefining `\maxovalrad`.)

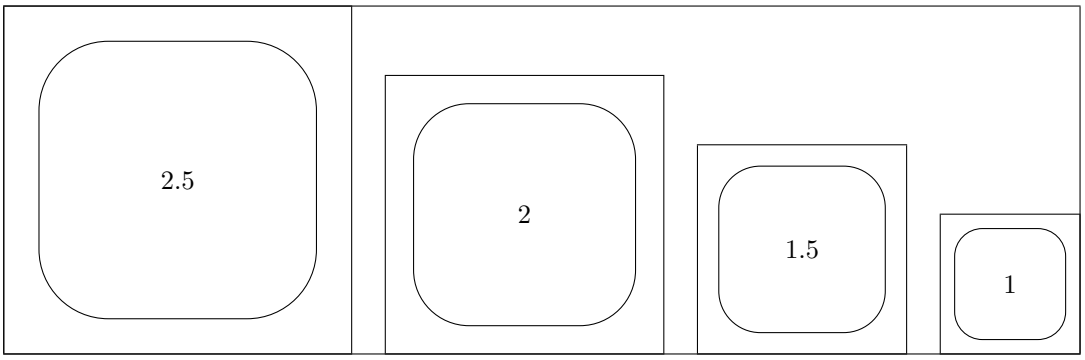
(Both `[`⟨*rad*⟩`]` and default value `\maxovalrad` are ignored in "standard LaTeX mode").

The behaviour of `\oval` in the absence of the `[`⟨*rad*⟩`]` argument is shown in Figure 5, left half of each picture. Note that in the Standard LaTeX implementation there is a minimum radius as well (innermost "salami"). In the right half of each picture, a *rad* argument has been used: it has no effect with the original `\oval` command.

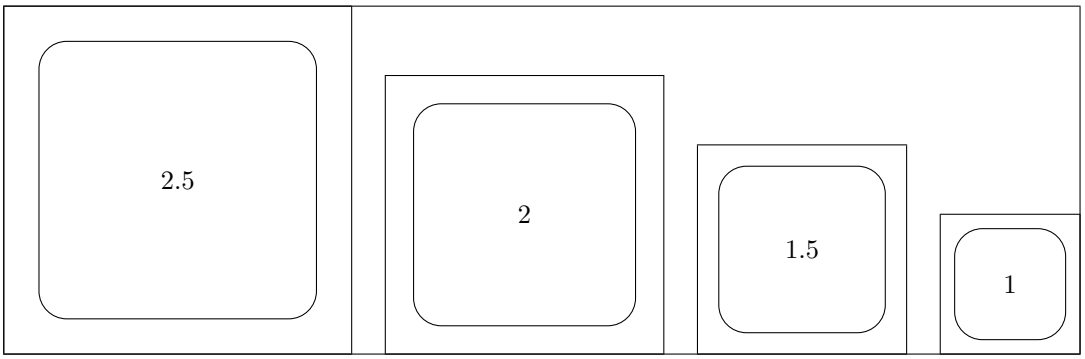
Both `[`⟨*rad*⟩`]` and `\maxovalrad` may be given as an explicit (rigid) length (i.e., with unit) or as a number. In the latter case the value is used as a factor to multiply by `\unitlength`. (A length or counter register will do as well, of course.)

If a number is given, the rounded corners of an oval will scale according to the current value of `\unitlength`. (See Figure 6, first row.)

New Commands, `\maxovalrad` depends on `\unitlength`



New Commands, `\maxovalrad` a fixed length



Original Commands, `\maxovalrad` ignored

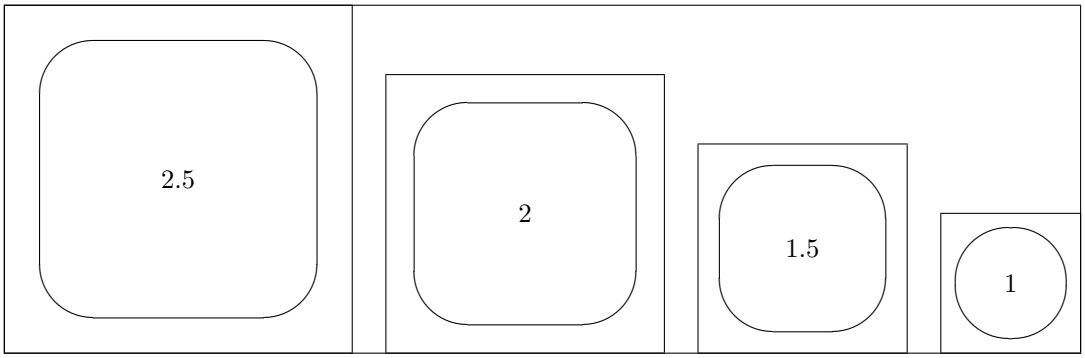


Figure 6: Radius argument for `\oval`: length vs. number

If a length is specified, the rounded corners of an oval will be the same regardless of the current value of `\unitlength`. (See Figure 6, second row.)

The default value is 20 pt as specified for the `[`⟨*rad*⟩`]` argument of `\oval` by [1, p. 223]. (See Figure 6, third row.)

2.3.5 Bezier Curves

`\bezier` `\bezier{`⟨*N*⟩`}(`⟨*AX,AY*⟩`)(`⟨*BX,BY*⟩`)(`⟨*CX,CY*⟩`)`
`\qbezier` `\qbezier[`⟨*N*⟩`](`⟨*AX,AY*⟩`)(`⟨*BX,BY*⟩`)(`⟨*CX,CY*⟩`)`
`\cbezier` `\cbezier[`⟨*N*⟩`](`⟨*AX,AY*⟩`)(`⟨*BX,BY*⟩`)(`⟨*CX,CY*⟩`)(`⟨*DX,DY*⟩`)`

In Standard LaTeX, the *N* argument specifies the number of points to plot, with the maximum defined by `\qbeziermax`. (`\bezier` is the obsolescent variant from the old `bezier` package of vintage LaTeX2.09.) The `\cbezier` command draws a cubic Bezier curve; see [3].

From [1, p. 221–223]:

> *With the* `pict2e` *package, there is no limit to the number of points plotted.*

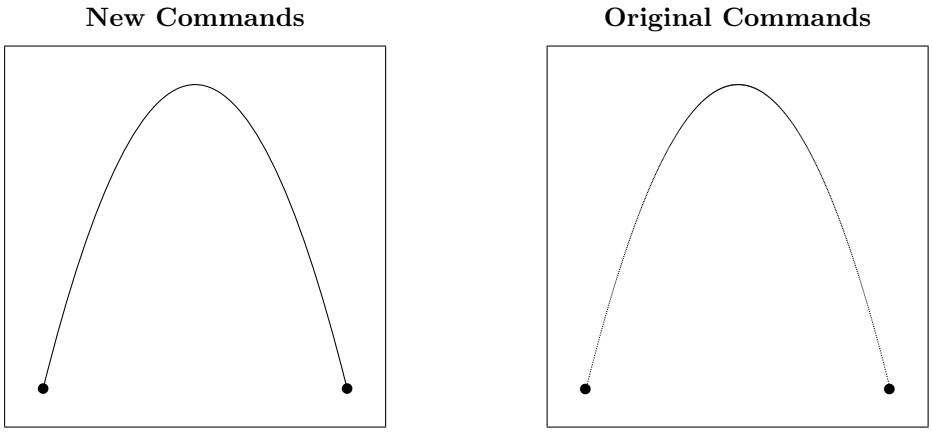


Figure 7: Bezier curves

More accurately, the argument specifying the maximum number of points to plot is simply ignored, as well as `\qbeziermax`.

3 Implementation

Unlike other packages that have reimplemented some of the commands from LaTeX's picture environment, we do not use special fonts, nor draw arbitrary shapes by the means of myriads of small characters, nor do we use sophisticated programming in some back-end programming language.

In its present state, this implementation supports just PostScript and PDF as back-end formats. It just calculates the necessary control points and uses primitive path drawing operators.

3.1 Preliminaries

`\pIIe@mode` `\pIIe@code` `\Gin@driver`

The first two of these commands determine how the pict2e package works internally; they should be defined properly by the `p2e-*.def` driver files. (Cf. file `p2e-drivers.dtx` for details.) The latter command is well known from the graphics and color packages from the Standard LaTeX graphics bundle; it should be set by a package option—most likely in a (system dependent) configuration file `pict2e.cfg`.

```
⟨*package⟩
\newcommand*\pIIe@mode{-1}
\newcommand*\pIIe@code[1]{}
\providecommand*\Gin@driver{}
```

`\pIIe@tempa` `\pIIe@tempb` `\pIIe@tempc`

At times, we need some temporoary storage bins. However, we only use some macros and do not allocate any new registers; the "superfluous" ones from the picture module of the kernel (`ltpictur.dtx`) and the general scratch registers should suffice.

```
\newcommand*\pIIe@tempa{}
\newcommand*\pIIe@tempb{}
\newcommand*\pIIe@tempc{}
```

3.2 Option processing

Driver options.

```
\DeclareOption{dvips}{\def\Gin@driver{dvips.def}}
\DeclareOption{xdvi}{\ExecuteOptions{dvips}}
\DeclareOption{dvipdf}{\def\Gin@driver{dvipdf.def}}
\DeclareOption{dvipdfm}{\def\Gin@driver{dvipdfm.def}}
\DeclareOption{pdftex}{\def\Gin@driver{pdftex.def}}
\DeclareOption{dvipsone}{\def\Gin@driver{dvipsone.def}}
\DeclareOption{dviwindo}{\ExecuteOptions{dvipsone}}
```

```
\DeclareOption{oztex}{\ExecuteOptions{dvips}}
\DeclareOption{textures}{\def\Gin@driver{textures.def}}
\DeclareOption{pctexps}{\def\Gin@driver{pctexps.def}}
\DeclareOption{pctex32}{\def\Gin@driver{pctex32.def}}
\DeclareOption{vtex}{\def\Gin@driver{vtex.def}}
```

Request "original" LaTeX mode.

```
\DeclareOption{original}{\def\pIIe@mode{0}}
```

Arrow options.

```
\DeclareOption{ltxarrows}{\AtEndOfPackage{%
  \let\pIIe@vector=\pIIe@vector@ltx
  \def\pIIe@FAL{1.52}%
  \def\pIIe@FAW{3.2}%
  \def\pIIe@CAW{1.5pt}%
  \def\pIIe@FAI{0.25}%
  }}
\DeclareOption{pstarrows}{\AtEndOfPackage{%
  \let\pIIe@vector=\pIIe@vector@pst
  \def\pIIe@FAL{1.4}%
  \def\pIIe@FAW{2}%
  \def\pIIe@CAW{1.5pt}%
  \def\pIIe@FAI{0.4}%
  }}
```

`\pIIe@debug@comment` This makes debugging easier.

```
\newcommand\pIIe@debug@comment{}
\DeclareOption{debug}{%
  \def\pIIe@debug@comment{^^J^^J\@percentchar\space >>> pict2e <<<^^J}%
  \begingroup
    \@ifundefined{pdfcompresslevel}{}{\global\pdfcompresslevel\z@}%
  \endgroup}
```

A special variant of debugging. (Obsolescent? Once used for performance measurements: arctan vs. pyth-add versions of `\vector`.)

```
\DeclareOption{hide}{\AtEndOfPackage{\def\pIIe@code#1{}}}
```

Unknown options default to mode "original."

```
\DeclareOption*{\ExecuteOptions{original}}
```

By default, arrows are in the LaTeX style.

```
\ExecuteOptions{ltxarrows}
```

Like the graphics and color packages, we support a configuration file. (See file `p2e-drivers.dtx` for details and an example.)

```
\InputIfFileExists{pict2e.cfg}{}{}
```

This now should make clear which mode and code we should use.

```
\ProcessOptions\relax
```

3.3 Output driver check

```
\ifnum\pIIe@mode=\z@
  \PackageInfo{pict2e}{Package option 'original' requested}
\else
```

This code fragment is more or less cloned from the graphics and color packages.

```
  \if!\Gin@driver!
    \PackageError{pict2e}
      {No driver specified at all}
      {You should make a default driver option in a file\MessageBreak
       pict2e.cfg\MessageBreak eg: \protect\ExecuteOptions{dvips}}%
  \else
    \PackageInfo{pict2e}{Driver file: \Gin@driver}
    \@ifundefined{ver@\Gin@driver}{\input{\Gin@driver}}{}
    \PackageInfo{pict2e}{Driver file for pict2e: p2e-\Gin@driver}
```

```
    \InputIfFileExists{p2e-\Gin@driver}{}{%
      \PackageError{pict2e}%
        {Driver file ``p2e-\Gin@driver'' not found}%
        {Q: Is the file properly installed? A: No!}}
  \fi
\fi
```

3.4 Mode check

For PostScript and PDF modes.

```
\ifnum\pIIe@mode>\z@
  \ifnum\pIIe@mode<\thr@@
    \RequirePackage{trig}
```

\pIIe@old@sline
\pIIe@oldvector
\pIIe@old@circle
\pIIe@old@dot
\pIIe@old@bezier
\pIIe@old@cbezier
\pIIe@oldoval
\pIIe@old@oval

Saved versions of some macros. (Or dummy definition.)

```
    \let\pIIe@old@sline\@sline
    \let\pIIe@oldvector\vector
    \let\pIIe@old@circle\@circle
    \let\pIIe@old@dot\@dot
    \let\pIIe@old@bezier\@bezier
    \let\pIIe@old@cbezier\@cbezier
    \AtBeginDocument{%
      \@ifundefined{@cbezier}{%
        \def\pIIe@old@cbezier[#1](#2,#3)(#4,#5)(#6,#7)(#8,#9){}%
        }{\let\pIIe@old@cbezier\@cbezier}}
    \let\pIIe@oldoval\oval
    \let\pIIe@old@oval\@oval
```

\OriginalPictureCmds

Switches back to the original definitions; for testing purposes only.

```
    \newcommand*\OriginalPictureCmds{%
      \let\@sline\pIIe@old@sline
      \let\vector\pIIe@oldvector
      \let\@circle\pIIe@old@circle
      \let\@dot\pIIe@old@dot
      \let\@bezier\pIIe@old@bezier
      \let\@cbezier\pIIe@old@cbezier
      \renewcommand*\oval[1][]{\pIIe@oldoval}%
      \let\@oval\pIIe@old@oval
    }
```

Overambitious drivers.

```
  \else
    \PackageError{pict2e}
      {Unsupported mode (\pIIe@mode) specified}
      {The driver you specified requested a mode\MessageBreak
       not supported by this version of this package}%
  \fi
```

Incapable drivers.

```
\else
  \ifnum\pIIe@mode<\z@
    \PackageError{pict2e}
      {No suitable driver specified}
      {You should make a default driver option in a file\MessageBreak
       pict2e.cfg\MessageBreak eg: \protect\ExecuteOptions{dvips}}%
  \fi
\fi
```

Big switch, completed near the end of the package.

```
\ifnum\pIIe@mode>\z@
```

3.5 Graphics operators

The following definitions allow the PostScript and PDF operations below to share some of the code.

```
\ifcase\pIIe@mode\relax
```

```
\pIIe@moveto@op
\pIIe@lineto@op
\pIIe@setlinewidth@op
\pIIe@stroke@op
\pIIe@fill@op
\pIIe@curveto@op
\pIIe@concat@op
```

PostScript

```
\or
  \newcommand*\pIIe@moveto@op{moveto}
  \newcommand*\pIIe@lineto@op{lineto}
  \newcommand*\pIIe@setlinewidth@op{setlinewidth}
  \newcommand*\pIIe@stroke@op{stroke}
  \newcommand*\pIIe@fill@op{fill}
  \newcommand*\pIIe@curveto@op{curveto}
  \newcommand*\pIIe@concat@op{concat}
```

```
\pIIe@moveto@op
\pIIe@lineto@op
\pIIe@setlinewidth@op
\pIIe@stroke@op
\pIIe@fill@op
\pIIe@curveto@op
\pIIe@concat@op
```

PDF

```
\or
  \newcommand*\pIIe@moveto@op{m}
  \newcommand*\pIIe@lineto@op{l}
  \newcommand*\pIIe@setlinewidth@op{w}
  \newcommand*\pIIe@stroke@op{S}
  \newcommand*\pIIe@fill@op{f}
  \newcommand*\pIIe@curveto@op{c}
  \newcommand*\pIIe@concat@op{cm}
```

(Currently, there are no other modes.)

```
\fi
```

3.6 Low-level operations

3.6.1 Collecting the graphics instructions and handling the output

```
\pIIe@GRAPH
\pIIe@addtoGraph
```

We collect all PostScript/PDF output code for a single picture object in a token register.

```
\@ifdefinable\pIIe@GRAPH{\newtoks\pIIe@GRAPH}
\newcommand*\pIIe@addtoGraph[1]{%
  \begingroup
    \edef\x{\the\pIIe@GRAPH\space#1}%
    \global\pIIe@GRAPH\expandafter{\x}%
  \endgroup}
```

```
\pIIe@fillGraph
```

The path will either be filled ...

```
\newcommand*\pIIe@fillGraph{\begingroup \@tempswatrue\pIIe@drawGraph}
```

```
\pIIe@strokeGraph
```

... or stroked.

```
\newcommand*\pIIe@strokeGraph{\begingroup \@tempswafalse\pIIe@drawGraph}
```

```
\pIIe@drawGraph
```

Common code. When the drawing is complete, we output the contents of the token register.

```
\newcommand*\pIIe@drawGraph{%
    \edef\x{\pIIe@debug@comment\space
```

Instead of scaling individual coordinates, we scale the graph as a whole (pt→bp); see Section 3.7.1.

```
            \pIIe@scale@PTtoBP}%
    \if@tempswa
      \edef\y{\pIIe@fill@op}%
    \else
      \edef\x{\x\space\strip@pt\@wholewidth
        \space\pIIe@setlinewidth@op}%
      \edef\y{\pIIe@stroke@op}%
    \fi
    \expandafter\pIIe@code\expandafter{%
      \expandafter\x\the\pIIe@GRAPH\space\y}%
```

Clear the graph and the current point after output.

```
    \global\pIIe@GRAPH{}\xdef\pIIe@CPx{}\xdef\pIIe@CPy{}%
  \endgroup}
```

3.6.2 Auxilliary macros

The following macros save us a plethora of tokens in subsequent code.

\pIIe@CPx
\pIIe@CPy
\pIIe@add@CP

The lengths (coordinates) given as arguments will be stored as real numbers. At the same time, we remember the "current point." (Not strictly necessary for PostScript, but for some operations in PDF, e.g., *rcurveto* emulation.)

```
\newcommand*\pIIe@CPx{} \newcommand*\pIIe@CPy{}
\newcommand*\pIIe@add@CP[2]{%
  \begingroup
    \@tempdima#1\xdef\pIIe@CPx{\the\@tempdima}%
    \@tempdimb#2\xdef\pIIe@CPy{\the\@tempdimb}%
    \pIIe@addtoGraph{\strip@pt\@tempdima\space\strip@pt\@tempdimb}%
  \endgroup}
```

\pIIe@add@nums

Similar, but does not set the "current point." Values need not be coordinates (e.g., may be scaling factors, etc.).

```
\newcommand*\pIIe@add@nums[2]{%
  \begingroup
    \@tempdima#1\relax
    \@tempdimb#2\relax
    \pIIe@addtoGraph{\strip@pt\@tempdima\space\strip@pt\@tempdimb}%
  \endgroup}
```

\pIIe@add@num

Likewise, for a single argument.

```
\newcommand*\pIIe@add@num[1]{%
  \begingroup
    \@tempdima#1\relax
    \pIIe@addtoGraph{\strip@pt\@tempdima}%
  \endgroup}
```

3.7 Medium-level operations

3.7.1 Transformations

Transformation operators; not all are currently used. (Hence, some are untested.)

\pIIe@PTtoBP

Scaling factor, used below. "pt→bp" (72/72.27 = 0.99626401). Note the trailing space!

```
\newcommand*\pIIe@PTtoBP{0.99626401 }

\ifcase\pIIe@mode\relax
```

\pIIe@concat
\pIIe@translate
\pIIe@rotate
\pIIe@scale
\pIIe@scale@PTtoBP

PostScript: Use some operators directly.

```
\or
  \newcommand*\pIIe@concat[6]{%
    \begingroup
      \pIIe@addtoGraph{[}%
      \@tempdima#1\relax \@tempdimb#2\relax
      \pIIe@add@nums\@tempdima\@tempdimb
      \@tempdima#3\relax \@tempdimb#4\relax
      \pIIe@add@nums\@tempdima\@tempdimb
      \@tempdima#5\relax \@tempdimb#6\relax
      \pIIe@add@nums\@tempdima\@tempdimb
      \pIIe@addtoGraph{] \pIIe@concat@op}%
    \endgroup}
  \newcommand*\pIIe@translate[2]{\pIIe@add@nums{#1}{#2}\pIIe@addtoGraph{translate}}
  \newcommand*\pIIe@rotate[1]{\pIIe@add@num{#1}\pIIe@addtoGraph{rotate}}
  \newcommand*\pIIe@scale[2]{\pIIe@add@nums{#1}{#2}\pIIe@addtoGraph{scale}}
  \newcommand*\pIIe@scale@PTtoBP{\pIIe@PTtoBP \pIIe@PTtoBP scale}
```

\pIIe@concat
\pIIe@translate
\pIIe@rotate
\pIIe@scale
\pIIe@scale@PTtoBP

PDF: Emulate. :-(

```
\or
  \newcommand*\pIIe@concat[6]{%
```

```
    \begingroup
      \@tempdima#1\relax \@tempdimb#2\relax
      \pIIe@add@nums\@tempdima\@tempdimb
      \@tempdima#3\relax \@tempdimb#4\relax
      \pIIe@add@nums\@tempdima\@tempdimb
      \@tempdima#5\relax \@tempdimb#6\relax
      \pIIe@add@nums\@tempdima\@tempdimb
      \pIIe@addtoGraph\pIIe@concat@op
    \endgroup}
  \newcommand*\pIIe@translate[2]{\pIIe@concat\p@\z@\z@\p@{#1}{#2}}
  \newcommand*\pIIe@rotate[1]{%
    \begingroup
      \@tempdima#1\relax
      \edef\pIIe@tempa{\strip@pt\@tempdima}%
      \CalculateSin\pIIe@tempa
      \CalculateCos\pIIe@tempa
      \edef\pIIe@tempb{\UseSin\pIIe@tempa}%
      \edef\pIIe@tempc{\UseCos\pIIe@tempa}%
      \pIIe@concat{\pIIe@tempc\p@}{\pIIe@tempb\p@}%
        {-\pIIe@tempb\p@}{\pIIe@tempc\p@}\z@\z@
    \endgroup}
  \newcommand*\pIIe@scale[2]{\pIIe@concat{#1}\z@\z@{#2}\z@\z@}
  \newcommand*\pIIe@scale@PTtoBP{\pIIe@PTtoBP 0 0 \pIIe@PTtoBP 0 0 \pIIe@concat@op}
```

(Currently, there are no other modes.)

```
\fi
```

3.7.2 Path definitions

`\pIIe@moveto` Simple things ...

```
  \newcommand*\pIIe@moveto[2]{%
      \pIIe@add@CP{#1}{#2}\pIIe@addtoGraph\pIIe@moveto@op}
```

`\pIIe@lineto` ... have to be defined, too.

```
  \newcommand*\pIIe@lineto[2]{%
      \pIIe@add@CP{#1}{#2}\pIIe@addtoGraph\pIIe@lineto@op}
```

We'll use `\pIIe@rcurveto` to draw quarter circles. (`\circle` and `\oval`).

```
\ifcase\pIIe@mode\relax
```

`\pIIe@rcurveto` PostScript: Use the "rcurveto" operator directly.

```
\or
  \newcommand*\pIIe@rcurveto[6]{%
    \begingroup
      \@tempdima#1\relax \@tempdimb#2\relax
      \pIIe@add@nums\@tempdima\@tempdimb
      \@tempdima#3\relax \@tempdimb#4\relax
      \pIIe@add@nums\@tempdima\@tempdimb
      \@tempdima#5\relax \@tempdimb#6\relax
      \pIIe@add@CP\@tempdima\@tempdimb
      \pIIe@addtoGraph{rcurveto}%
    \endgroup}
```

`\pIIe@rcurveto` PDF: It's necessary to emulate the PostScript operator "rcurveto". For this, the "current point" must be known, i.e., all macros which change the "current point" must set `\pIIe@CPx` and `\pIIe@CPy`.

```
\or
  \newcommand*\pIIe@rcurveto[6]{%
    \begingroup
      \@tempdima#1\advance\@tempdima\pIIe@CPx\relax
      \@tempdimb#2\advance\@tempdimb\pIIe@CPy\relax
      \pIIe@add@nums\@tempdima\@tempdimb
```

```
      \@tempdima#3\advance\@tempdima\pIIe@CPx\relax
      \@tempdimb#4\advance\@tempdimb\pIIe@CPy\relax
      \pIIe@add@nums\@tempdima\@tempdimb
      \@tempdima#5\advance\@tempdima\pIIe@CPx\relax
      \@tempdimb#6\advance\@tempdimb\pIIe@CPy\relax
      \pIIe@add@CP\@tempdima\@tempdimb
      \pIIe@addtoGraph\pIIe@curveto@op
    \endgroup}
```

(Currently, there are no other modes.)

```
\fi
```

`\pIIe@curveto` This is currently only used for Bezier curves and for drawing the heads of LaTeX-like arrows. Note: It's the same for PostScript and PDF.

```
\newcommand*\pIIe@curveto[6]{%
  \begingroup
    \@tempdima#1\relax \@tempdimb#2\relax
    \pIIe@add@nums\@tempdima\@tempdimb
    \@tempdima#3\relax \@tempdimb#4\relax
    \pIIe@add@nums\@tempdima\@tempdimb
    \@tempdima#5\relax \@tempdimb#6\relax
    \pIIe@add@CP\@tempdima\@tempdimb
    \pIIe@addtoGraph\pIIe@curveto@op
  \endgroup}
```

3.8 Pythagorean Addition, and Division

`\pIIe@pyth` This algorithm is copied from the PICTEX package (see [4]) by Michael Wichura, with his permission (cite Email!?).

```
\newcommand*\pIIe@pyth[3]{%
  \begingroup
    \@tempdima=#1\relax
```

`\@tempdima` $= \mathrm{abs}(x)$

```
    \ifnum\@tempdima<\z@\@tempdima=-\@tempdima\fi
    \@tempdimb=#2\relax
```

`\@tempdimb` $= \mathrm{abs}(y)$

```
    \ifnum\@tempdimb<\z@\@tempdimb=-\@tempdimb\fi
```

`\@tempdimb` $= s = \mathrm{abs}(x) + \mathrm{abs}(y)$

```
    \advance\@tempdimb\@tempdima
    \ifnum\@tempdimb=\z@
```

`\@tempdimc` $= z = \sqrt{(x^2 + y^2)}$

```
      \global\@tempdimc=\z@
    \else
```

`\@tempdima` $= 8 \times \mathrm{abs}(x)$

```
      \multiply\@tempdima 8\relax
```

`\@tempdimc` $= 8\,t = 8 \times \mathrm{abs}(x)/s$

```
      \pIIe@@divide\@tempdima\@tempdimb
```

`\@tempdimc` $= 4\tau = (8\,t - 4)$

```
      \advance\@tempdimc -4pt
      \multiply\@tempdimc 2
      \edef\pIIe@tempa{\strip@pt\@tempdimc}%
```

`\@tempdima` $= (8\,\tau)^2$

```
      \@tempdima=\pIIe@tempa\@tempdimc
```

`\@tempdima` $= [64 + (8\,\tau)^2]/2 = (8\,f)^2$

```
      \advance\@tempdima 64pt
      \divide\@tempdima 2\relax
```

initial guess at $\sqrt{(u)}$

```
264       \@dashdim=7pt
```

\@dashdim $= \sqrt{(u)}$

```
265       \pIIe@@pyth\pIIe@@pyth\pIIe@@pyth
266       \edef\pIIe@tempa{\strip@pt\@dashdim}%
267       \@tempdimc=\pIIe@tempa\@tempdimb
```

\@tempdimc $= z = (8\,f) \times s/8$

```
268       \global\divide\@tempdimc 8
269     \fi
270   \endgroup
271   #3=\@tempdimc}
```

\pIIe@@pyth \@dashdim $= g \leftarrow (g + u/g)/2$

```
272 \newcommand*\pIIe@@pyth{%
273   \pIIe@@divide\@tempdima\@dashdim
274   \advance\@dashdim\@tempdimc
275   \divide\@dashdim\tw@}
```

\pIIe@@divide This division algorithm is similar to the one used for the tangent in the trig package [6]. It does not allow large arguments. A very small denominator argument leads to division by zero.

```
276 \newcommand*\pIIe@@divide[2]{%
```

2^{16} (sp)

```
277   \@tempcnta\p@
```

$2^{30} >$ \maxdimen !

```
278   \multiply\@tempcnta\two@fourteen
279   \@clnwd#2\relax
280   \divide\@clnwd\@iv
281   \divide\@tempcnta\@clnwd
282   \@clnht\@tempcnta sp\relax
283   \@clnwd#1\relax
284   \@tempdimc\strip@pt\@clnwd\@clnht}
```

\pIIe@divide Improve this documentation!

$$\#3 = \frac{2^{30} \cdot \#1}{2^{-2} \cdot \#2 \cdot 2^{16}}$$

(The factor 2^{16} is implied by TEX's division.)

```
285 \newcommand*\pIIe@divide[3]{%
286   \pIIe@@divide{#1}{#2}%
287   #3=\@tempdimc}
```

3.9 High-level operations

3.9.1 Line

\@sline (The implementation here is different from \vector!)

```
288   \def\@sline{%
289     \begingroup
290       \ifnum\@xarg<\z@\@linelen-\@linelen\fi
291       \pIIe@divide{\@yarg\p@}{\@xarg\p@}\dimen@
292       \@ydim=\strip@pt\dimen@\@linelen
293       \pIIe@moveto\z@\z@
294       \pIIe@lineto\@linelen\@ydim
295       \pIIe@strokeGraph
296     \endgroup}
```

3.9.2 Vector

\vector Unlike \line, \vector must be redefined, because the kernel version checks for illegal slope arguments.

\vector(⟨x,y⟩){⟨l_x⟩}: Instead of calculating $\theta = \arctan\frac{y}{x}$, we use "pythagorean addition" [4] to determine $s = \sqrt{x^2+y^2}$ and to obtain the length of the vector $l = l_x \cdot \frac{s}{x}$ and the values of $\sin\theta = \frac{y}{s}$ and $\cos\theta = \frac{x}{s}$ for the rotation of the coordinate system.

```
\def\vector(#1,#2)#3{%
  \begingroup
    \@xarg#1\relax \@yarg#2\relax
    \@linelen#3\unitlength
```

Missing macro: \@abs{⟨*val*⟩} – should be in the kernel! It would save a plethora of tokens here and in other packages and even in the kernel itself (e.g., cf. \vector). Should we define it in this package?

```
    \ifnum\ifnum\@xarg<\z@-\fi\@xarg>\@m \@badlinearg \fi
    \ifnum\ifnum\@yarg<\z@-\fi\@yarg>\@m \@badlinearg \fi
```

A bit incompatible with standard LaTeX, and with \line: slope $(0, 0)$ raises an error.

```
    \ifnum\@xarg=\z@ \ifnum\@yarg=\z@ \@badlinearg \fi\fi
    \ifdim\@linelen<\z@\@badlinearg\else
      \pIIe@pyth{\@xarg\p@}{\@yarg\p@}\dimen@%
      \ifnum\@xarg=\z@
      \else\ifnum\@yarg=\z@
        \else
```

This calculation is only necessary, if the vector is actually sloped.

```
          \pIIe@divide\dimen@{\@xarg\p@}\@xdim
          \@linelen\strip@pt\@xdim\@linelen
          \@linelen\ifdim\@linelen<\z@-\fi\@linelen
        \fi
      \fi
```

$\sin\theta$ and $\cos\theta$

```
      \pIIe@divide{\@yarg\p@}\dimen@\@ydim
      \pIIe@divide{\@xarg\p@}\dimen@\@xdim
```

Rotate the following vector/arrow outlines by angle θ.

```
      \pIIe@concat\@xdim\@ydim{-\@ydim}\@xdim\z@\z@
```

Internal command to draw the outline of the vector/arrow.

```
      \pIIe@vector
      \pIIe@fillGraph
    \fi
  \endgroup}
```

\pIIe@vector This command should be \def'ed or \let to a macro that generates the vector's outline path. Now initialized by package option.

```
\newcommand*\pIIe@vector{}
```

\pIIe@FAL Some macros to parametrize the shape of the vector outline. Should be user-level macros or
\pIIe@FAW changeable via key-value interface. See Figures 8 and 9.
\pIIe@CAW
\pIIe@FAI
\pIIe@@firstnum
\pIIe@@secondnum

```
\newcommand*\pIIe@FAL{}\newcommand*\pIIe@FAW{}\newcommand*\pIIe@CAW{}
\newcommand*\pIIe@FAI{}
\newcommand*\pIIe@@firstnum{}\newcommand*\pIIe@@secondnum{}
\AtBeginDocument{%
  \@ifpackageloaded{pstricks}{%
    \def\pIIe@FAL{\psk@arrowlength}%
    \def\pIIe@FAW{\expandafter\pIIe@@secondnum\psk@arrowsize}%
    \def\pIIe@CAW{\expandafter\pIIe@@firstnum\psk@arrowsize}%
    \def\pIIe@FAI{\psk@arrowinset}%
    \def\pIIe@@firstnum#1 #2 {#1\p@}%
    \def\pIIe@@secondnum#1 #2 {#2}%
  }{}%
}
```

L = \@linelen
W = \@wholewidth $= 2 \times$ \@halfwidth
AW = \pIIe@FAW $\times W +$ \pIIe@CAW
AL = \pIIe@FAL $\times AW$
IN = \pIIe@FAI $\times AW/2$

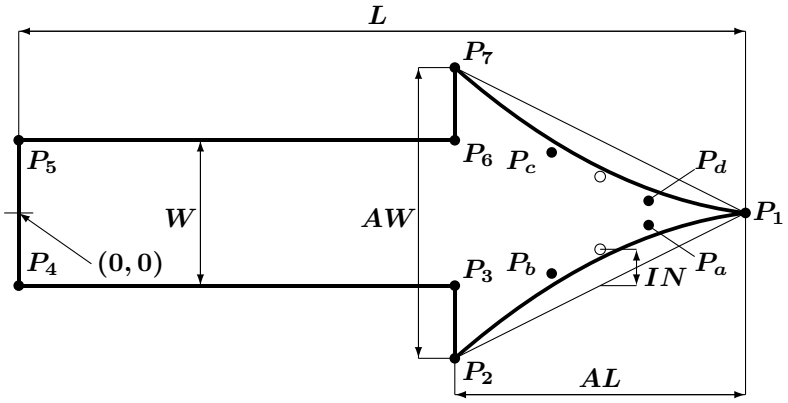


Figure 8: Sketch of the path drawn by the LaTeX-like implementation of \vector. (Note: We are using the redefined macros of pict2e!)

LaTeX version The arrows drawn by the variant generated by the ltxarrows package option are modeled after those in the fonts used by standard LaTeX. See Figure 8.

\pIIe@vector@ltx The arrow outline. (Not yet quite the same as with LaTeX's fonts.)

Problem: Extrapolation. There are only two design sizes (thicknesses) for LaTeX's line drawing fonts. Where can we go from there?

Note that only the arrow head will be drawn, if the length argument of the \vector command is smaller than the calculated length of the arrow head.

```
\newcommand*\pIIe@vector@ltx{%
  \@ydim\pIIe@FAW\@wholewidth \advance\@ydim\pIIe@CAW\relax
  \@ovxx\pIIe@FAL\@ydim
  \@xdim\@linelen \advance\@xdim-\@ovxx
  \divide\@ydim\tw@
  \divide\@ovxx\tw@ \advance\@ovxx\@xdim
  \@ovyy\@ydim
  \divide\@ovyy\tw@ \advance\@ovyy-\pIIe@FAI\@ydim
```

$P'_{7a} = P_m + 1/3(P_0 - P_m)$

```
  \pIIe@bezier@QtoC\@linelen\@ovxx\@ovro
  \pIIe@bezier@QtoC\z@\@ovyy\@ovri
```

$P''_{7a} = P_m + 1/3(P_3 - P_m)$

```
  \pIIe@bezier@QtoC\@xdim\@ovxx\@clnwd
  \pIIe@bezier@QtoC\@ydim\@ovyy\@clnht
```

P_1

```
  \pIIe@moveto\@linelen\z@
```

P_{2a}/P_2

```
  \pIIe@curveto\@ovro{-\@ovri}\@clnwd{-\@clnht}\@xdim{-\@ydim}%
  \ifdim\@xdim>\z@
```

P_3

```
    \pIIe@lineto\@xdim{-\@halfwidth}%
```

P_4

```
    \pIIe@lineto\z@{-\@halfwidth}%
```

P_5

```
    \pIIe@lineto\z@{\@halfwidth}%
```

P_6

```
    \pIIe@lineto\@xdim{\@halfwidth}%
  \fi
```

P_7

```
  \pIIe@lineto\@xdim\@ydim
```

P_{7a}/P_1

```
  \pIIe@curveto\@clnwd\@clnht\@ovro\@ovri\@linelen\z@}
```

L = \@linelen
W = \@wholewidth $= 2 \times$ \@halfwidth
AW = \pIIe@FAW $\times W +$ \pIIe@CAW
AL = \pIIe@FAL $\times AW$
IN = \pIIe@FAI $\times AL$

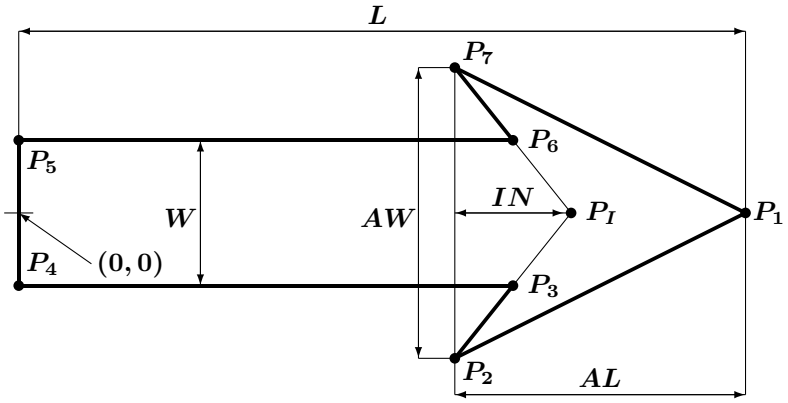


Figure 9: Sketch of the path drawn by the PSTricks-like implementation of \vector. (Note: We are using the redefined macros of pict2e!)

PSTricks version The arrows drawn by the variant generated by the pstarrows package option are modeled after those in the pstricks package [8]. See Figure 9.

\pIIe@vector@pst The arrow outline. Note that only the arrowhead will be drawn, if the length argument of the \vector command is smaller than the calculated length of the arrow head.

```
\newcommand*\pIIe@vector@pst{%
  \@ydim\pIIe@FAW\@wholewidth \advance\@ydim\pIIe@CAW\relax
  \@ovxx\pIIe@FAL\@ydim
  \@xdim\@linelen \advance\@xdim-\@ovxx
  \divide\@ydim\tw@
  \@ovyy\@ydim \advance\@ovyy-\@halfwidth %a
  \@ovdx\pIIe@FAI\@ovxx %d
  \pIIe@@divide\@ovdx\@ydim % d/c
  \@ovxx\strip@pt\@ovyy\@tempdimc %b
  \advance\@ovxx\@xdim
  \advance\@ovdx\@xdim
                    P_1
  \pIIe@moveto\@linelen\z@
                    P_2
  \pIIe@lineto\@xdim{-\@ydim}%
  \ifdim\@xdim>\z@
                    P_3
    \pIIe@lineto\@ovxx{-\@halfwidth}%
                    P_4
    \pIIe@lineto\z@{-\@halfwidth}%
                    P_5
    \pIIe@lineto\z@{\@halfwidth}%
                    P_6
    \pIIe@lineto\@ovxx{\@halfwidth}%
  \else
                    P_1a
    \pIIe@lineto\@ovdx\z@
  \fi
                    P_7
  \pIIe@lineto\@xdim\@ydim
                    P_1
  \pIIe@lineto\@linelen\z@}
```

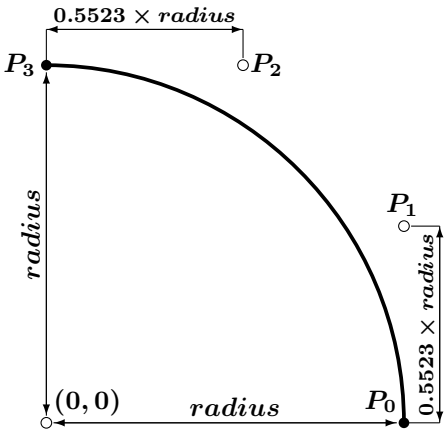


Figure 10: Sketch of the path drawn by `\pIIe@qcircle` (NE quarter)

3.9.3 Circle and Dot

`\@circle` The circle will either be stroked ...

```
\def\@circle#1{\begingroup \@tempswafalse\pIIe@circ{#1}}
```

`\@dot` ... or filled.

```
\def\@dot#1{\begingroup \@tempswatrue\pIIe@circ{#1}}
```

`\pIIe@circ` Common code.

```
\newcommand*\pIIe@circ[1]{%
```

We need the radius instead of the diameter. Unlike Standard LaTeX, we check for negative or zero diameter argument.

```
    \@tempdima#1\unitlength
    \ifdim\@tempdima>\z@\else \pIIe@badcircarg \fi
    \divide\@tempdima\tw@
    \pIIe@circle\@tempdima
```

With the current state of affairs, we could use `\pIIe@drawGraph` directly; but that would possibly be a case of premature optimisation. (Use of `@tempswa` both here and inside quarter-circle!)

```
    \if@tempswa \pIIe@fillGraph \else \pIIe@strokeGraph \fi
  \endgroup}
```

`\pIIe@circle` Approximate a full circle by four quarter circles.

```
\newcommand*\pIIe@circle[1]{%
  \pIIe@qcircle[1]\z@{#1}\pIIe@qcircle  \@ne{#1}%
  \pIIe@qcircle  \tw@{#1}\pIIe@qcircle\thr@@{#1}}
```

`\pIIe@qcircle` Approximate a quarter circle, using cubic Bezier splines.
#1=Switch (0=no 'moveto', 1='moveto'), #2=Quadrant No., #3=Radius.
0 = 1st Quadrant (NE) 1 = 2nd Quadrant (NW)
2 = 3rd Quadrant (SW) 3 = 4th Quadrant (SE)
(PostScript: We could use the `arc` operator!)
0.55228474983 ="magic number" (see [3]).

```
\newcommand*\pIIe@qcircle[3][0]{%
  \begingroup
    \@ovro#3\relax \@ovri0.55228474983\@ovro
    \@tempdimc\@ovri \advance\@tempdimc-\@ovro
    \ifnum#1>\z@ \@tempswatrue \else \@tempswafalse \fi
    \ifcase#2\relax
            NE
      \pIIe@@qcircle\@ovro\z@\z@\@ovri\@tempdimc\@ovro{-\@ovro}\@ovro
    \or
```

```
                NW
      \pIIe@@qcircle\z@\@ovro{-\@ovri}\z@{-\@ovro}\@tempdimc{-\@ovro}{-\@ovro}%
    \or
                SW
      \pIIe@@qcircle{-\@ovro}\z@\z@{-\@ovri}{-\@tempdimc}{-\@ovro}\@ovro{-\@ovro}%
    \or
                SE
      \pIIe@@qcircle\z@{-\@ovro}\@ovri\z@\@ovro{-\@tempdimc}\@ovro\@ovro
    \fi
  \endgroup}
```

`\pIIe@@qcircle` Ancillary macro; saves us some tokens above.

Note: Use of `rcurveto` instead of `curveto` makes it possible (or at least much easier) to re-use this macro for the rounded corners of ovals.

```
\newcommand*\pIIe@@qcircle[8]{%
  \if@tempswa\pIIe@moveto{#1}{#2}\fi \pIIe@rcurveto{#3}{#4}{#5}{#6}{#7}{#8}}
```

`\pIIe@badcircarg` Obvious cousin to `\@badlinearg` from the LaTeX kernel.

```
\newcommand*\pIIe@badcircarg{%
  \PackageError{pict2e}%
    {Illegal argument in \protect\circle(*) or \protect\oval}%
    {The radius of a circle, dot or oval corner must be greater than zero.}}%
```

3.9.4 Oval

`\maxovalrad` User level command, may be redefined by `\renewcommand*`. It may be given as an explicit (rigid) length (with unit) or as a number. (dimen and count registers should work, too.) In the latter case it is used as a factor to multiply by `\unitlength`. The default value is 20 pt as specified for the `[`⟨*rad*⟩`]` argument of `\oval` by [1, p. 223].

```
\newcommand*\maxovalrad{20pt}
```

`\pIIe@defaultUL` `\pIIe@def@UL` See `\Gin@defaultbp` and `\Gin@def@bp` from `graphics.dtx`. This seems necessary, since [1, p. 223] does not specify whether the `[`⟨*rad*⟩`]` argument should be given in terms of `\unitlength` or as an absolute length.

```
\newcommand*\pIIe@defaultUL[2]{%
  \afterassignment\pIIe@def@UL\dimen@#2\unitlength\relax{#1}{#2}}
```

However, things are simpler in our case, since we always need the value stored in `\dimen@`. Hence, we could/should omit the unnecessary argument!?) Unlike Standard LaTeX, we check for negative or zero radius argument.

```
\newcommand*\pIIe@def@UL{}
\def\pIIe@def@UL#1\relax#2#3{%
%   \if!#1!%
%     \def#2{#3}% \edef ?
%    \else
%      \edef#2{\strip@pt\dimen@}%
%    \fi
  \ifdim\dimen@>\z@\else \pIIe@badcircarg \fi
  \edef#2{\the\dimen@}}
```

`\oval` `\pIIe@maxovalrad` The variant of `\oval` defined here takes an additional optional argument, which specifies the maximum radius of the rounded corners (default = 20 pt, as given above). `\pIIe@maxovalrad` is the internal variant of `\maxovalrad`.

```
\newcommand*\pIIe@maxovalrad{}
\renewcommand*\oval[1][\maxovalrad]{%
  \begingroup \pIIe@defaultUL\pIIe@maxovalrad{#1}%
```

Can't close the group here, since arguments must be parsed. (This is done by calling the saved original.)

```
    \pIIe@oldoval}
```

`\@oval` (This is called in turn by the saved original.)

```
\def\@oval(#1,#2)[#3]{%
```

In analogy to circles, we need only half of the size value.

```
  \@ovxx#1\unitlength \divide\@ovxx\tw@
  \@ovyy#2\unitlength \divide\@ovyy\tw@
  \@tempdimc \ifdim\@ovyy>\@ovxx \@ovxx \else \@ovyy \fi
  \ifdim\pIIe@maxovalrad<\@tempdimc \@tempdimc\pIIe@maxovalrad\relax \fi
```

Subtract the radius of the corners to get coordinates for the straight line segments.

```
  \@xdim\@ovxx \advance\@xdim-\@tempdimc
  \@ydim\@ovyy \advance\@ydim-\@tempdimc
```

Determine which parts of the oval we have to draw.

```
  \pIIe@get@quadrants{#3}%
```

"`@tempswa` = false" means, that we have to suppress the 'moveto' in the following quadrant.

```
  \@tempswatrue
```

The following isn't strictly necessary, but yields a single (unfragmented) path even for `[r]` (right half of oval only). Useful for future extensions.
Bits 3 and 0 set? (SE/NE)

```
  \ifnum9=\@tempcnta
    \pIIe@qoval\z@{-\@ovyy}{\@xdim}{-\@ovyy}\thr@@\@tempdimc\@ovxx\z@
```

Bit 0 set! (NE)

```
    \@tempcnta\@ne
  \fi
```

Bit 0 set? (NE)

```
  \pIIe@qoval\@ovxx\z@\@ovxx\@ydim\z@\@tempdimc\z@\@ovyy
```

Bit 1 set? (NW)

```
  \pIIe@qoval\z@\@ovyy{-\@xdim}\@ovyy\@ne\@tempdimc{-\@ovxx}\z@
```

Bit 2 set? (SW)

```
  \pIIe@qoval{-\@ovxx}\z@{-\@ovxx}{-\@ydim}\tw@\@tempdimc\z@{-\@ovyy}%
```

Bit 3 set? (SE)

```
  \pIIe@qoval\z@{-\@ovyy}{\@xdim}{-\@ovyy}\thr@@\@tempdimc\@ovxx\z@
```

Close the group opened by `\oval` above.

```
  \pIIe@strokeGraph
  \endgroup}
```

`\pIIe@qoval` Ancillary macro; saves us some tokens above.
(PostScript: We could use the `arc` or `arcto` operator!)

```
  \newcommand*\pIIe@qoval[8]{%
```

Bit set?

```
    \ifodd\@tempcnta
      \if@tempswa\pIIe@moveto{#1}{#2}\fi
      \pIIe@lineto{#3}{#4}\pIIe@qcircle{#5}{#6}\pIIe@lineto{#7}{#8}%
      \@tempswafalse
    \else
      \@tempswatrue
    \fi
```

Shift by one bit.

```
    \divide\@tempcnta\tw@}
```

`\pIIe@get@quadrants` According to the parameter (`tlbr`) bits are set in `\@tempcnta`:

0 = 1st Quadrant (NE) 1 = 2nd Quadrant (NW)
2 = 3rd Quadrant (SW) 3 = 4th Quadrant (SE)

(Cf. `\@oval` and `\@ovvert` in the LaTeX kernel.) We abuse `\@setfpsbit` from the float processing modules of the kernel.

```
  \newcommand*\pIIe@get@quadrants[1]{%
```

```
    \@ovttrue \@ovbtrue \@ovltrue \@ovrtrue \@tempcnta\z@
    \@tfor\reserved@a:=#1\do{\csname @ov\reserved@a false\endcsname}%
    \if@ovr \if@ovb\@setfpsbit2\fi \if@ovt\@setfpsbit4\fi \fi
    \if@ovl \if@ovb\@setfpsbit1\fi \if@ovt\@setfpsbit8\fi \fi}
```

3.9.5 Quadratic Bezier Curve

`\@bezier` #1, the maximum number of points to use, is simply ignored, as well as `\qbeziermax`.

```
\def\@bezier#1(#2,#3)(#4,#5)(#6,#7){%
```

$$P_0 = (\#2, \#3) \quad P_m = (\#4, \#5) \quad P_3 = (\#6, \#7)$$

```
  \begingroup
    \@ovxx#2\unitlength \@ovyy#3\unitlength
    \@ovdx#4\unitlength \@ovdy#5\unitlength
    \@xdim#6\unitlength \@ydim#7\unitlength
```

$$P_1 = P_m + 1/3(P_0 - P_m)$$

```
    \pIIe@bezier@QtoC\@ovxx\@ovdx\@ovro
    \pIIe@bezier@QtoC\@ovyy\@ovdy\@ovri
```

$$P_2 = P_m + 1/3(P_3 - P_m)$$

```
    \pIIe@bezier@QtoC\@xdim\@ovdx\@clnwd
    \pIIe@bezier@QtoC\@ydim\@ovdy\@clnht
```

$$(P_{0x}, P_{0y})$$

```
    \pIIe@moveto\@ovxx\@ovyy
```

$$(P_{1x}, P_{1y}) \quad (P_{2x}, P_{2y}) \quad (P_{3x}, P_{3y})$$

```
    \pIIe@curveto\@ovro\@ovri\@clnwd\@clnht\@xdim\@ydim
    \pIIe@strokeGraph
  \endgroup}
```

`\pIIe@bezier@QtoC` Ancillary macro; saves us some tokens above.
Transformation: quadratic bezier parameters → cubic bezier parameters.
(Missing: Reference for mathematical formula. Or is this trivial?)

```
\newcommand*\pIIe@bezier@QtoC[3]{%
  \@tempdimc#1\relax      \advance\@tempdimc-#2\relax
  \divide\@tempdimc\thr@@ \advance\@tempdimc #2\relax
  #3\@tempdimc}
```

3.10 Commands from other packages

3.10.1 Package ebezier

One feature from [3].

`\cbezier` #1, the maximum number of points to use, is simply ignored, as well as `\qbeziermax`.
`\@cbezier` Original head of the macro:
`\pIIe@@cbezier` `\def\cbezier{\@ifnextchar [{\@cbezier}{\@cbezier[0]}}`
Changed analogous to the LaTeX kernel's `\qbezier` and `\bezier`:

```
\AtBeginDocument{\@ifundefined{cbezier}{\newcommand}{\renewcommand}*%
    \cbezier[2][0]{\pIIe@@cbezier[#1]#2}%
  \@ifdefinable\pIIe@@cbezier{}%
  \def\pIIe@@cbezier#1)#2(#3)#4(#5)#6({\@cbezier#1)(#3)(#5)(}%
  \def\@cbezier[#1](#2,#3)(#4,#5)(#6,#7)(#8,#9){%
    \pIIe@moveto{#2\unitlength}{#3\unitlength}%
    \pIIe@curveto{#4\unitlength}{#5\unitlength}%
      {#6\unitlength}{#7\unitlength}{#8\unitlength}{#9\unitlength}%
    \pIIe@strokeGraph}%
}
```

3.10.2 Other packages

Other macros from various packages may be included in future versions of this package.

3.11 Mode ‘original’

Other branch of the big switch near the beginning of the code (see page 10).

```
\else
```

`\oval` `\maxovalrad` Gobble the new optional argument and continue with saved version. `\maxovalrad` is there to avoid error messages in case the user's document redefines it with `\renewcommand*`. Likewise, `\OriginalPictureCmds` is only needed for test documents.

```
  \renewcommand*\oval[1][]{\pIIe@oldoval}
  \newcommand*\maxovalrad{20pt}
  \newcommand*\OriginalPictureCmds{}
\fi
```

⟨/package⟩

References

[1] Leslie Lamport: *LaTeX – A Document Preparation System*, 2nd ed., 1994

[2] Michel Goossens, Frank Mittelbach, Alexander Samarin: *The LaTeX Companion*, 1993

[3] Gerhard A. Bachmaier: *The ebezier package.* `CTAN: macros/latex/contrib/ebezier/`, 2002

[4] Michael Wichura: *The PiCTeX package.* `CTAN: graphics/pictex`, 1987

[5] David Carlisle: *The pspicture package.* `CTAN: macros/latex/contrib/carlisle/`, 1992

[6] David Carlisle: *The trig package.* `CTAN: macros/latex/required/graphics/`, 1999

[7] Kresten Krab Thorup: *The pspic package.* `CTAN: macros/latex209/contrib/misc/`, 1991

[8] Timothy Van Zandt: *The pstricks bundle.* `CTAN: graphics/pstricks/`, 1993, 1994, 2000