The pict2e package*

Hubert Gäßlein, Rolf Niepraschk† and Josef Tkadlec‡

2020/06/20

Abstract

This package was described in the 2nd edition of "LaTeX: A Document Preparation System", but the LaTeX project team declined to produce the package. For a long time, LaTeX has included a "pict2e package" that merely produced an apologetic error message.

The new package extends the existing LaTeX `picture` environment, using the familiar technique (cf. the graphics and color packages) of driver files. In the user-level part of this documentation there is a fair number of examples of use, showing where things are improved by comparison with the Standard LaTeX `picture` environment.

Contents

*This document corresponds to pict2e.sty v0.3e, dated 2020/06/20, documentation dated 2020/06/20.

†Rolf.Niepraschk@gmx.de

‡j.tkadlec@email.cz

List of Figures

1 Introduction

Here’s a quote from the obsolete original official version of the pict2e package (1993–2003):

> *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 internal 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, class or package writers just say `\RequirePackage [`⟨*optionlist*⟩`]{pict2e}` in an appropriate place in their class or package file. (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	oztex	(x)
xdvi	x	dvipsone	x?
pdftex	x	dviwindo	x?
vtex	x	dvipdf	x?
dvipdfm	x	textures	x?
dvipdfmx	x	pctexps	x?
xetex	x	pctex32	x?
luatex (> 0.85)	x		

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

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

Note: You should specify the same driver for pict2e you use with the graphics/x and color packages. Otherwise, things may go haywire.

2.1.2 Other options

Currently, there are two options that allow you to choose between variants of the arrows-heads generated by the `\vector` command. See Figure 3 in Section 2.3.2 for the difference.

option	meaning
ltxarrows	Draw LaTeX style vectors (default).
pstarrows	Draw PSTricks style vectors.

2.1.3 Debugging options

These options are (mainly) for development and testing purposes.

option	meaning
original	Suppresses the new definitions.
debug	Suppresses the compressing of pdfTeX output; marks the pict2e generated code in the output files.
hide	Suppresses all graphics output from pict2e.

2.2 Configuration file

Similar to the graphics and color packages, in most cases it is not necessary to give a driver option explicitly with the `\usepackage` (or `\RequirePackage`) command, if a suitable configuration file `pict2e.cfg` is present on your system (see the example file `pict2e-example.cfg`). On many systems it may be sufficient to copy `pict2e-example.cfg` to `pict2e.cfg`; on others you might need to modify your copy to suit your system.

2.3 Details: Changes to user-level commands

This section describes the improvements of the new implementation of (some of) the `picture` commands. For details, look up "`pict2e` package" in the index of the LaTeX manual [1].

Here's a collection of quotes relevant to the pict2e package 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 $|x|, |y| \le 6$ or 4, with no common divisor except ± 1:*
> *(These restrictions are eliminated by the `pict2e` package.)*
>
> **`\line` and `\vector`** *Smallest horizontal extent of sloped lines and vectors 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 $-6 \le X, Y \le +6$, with no common divisors except ± 1. (I.e., X and Y must be relatively prime.) Furthermore, only horizontal and vertical lines can assume arbitrary thickness; sloped lines are restricted to the widths given by the `\thinlines` and `\thicklines` declarations (i.e., 0.4pt and 0.8pt, respectively).

From [1, p. 222]:

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

However, to avoid overflow of TeX's dimens, the slope arguments are real numbers in the range $-16383 \le X, Y \le +16383$. It is usually not a good idea to use slope arguments with the absolute value less then 10^{-4} (the best accuracy is obtained if you use multiples of arguments

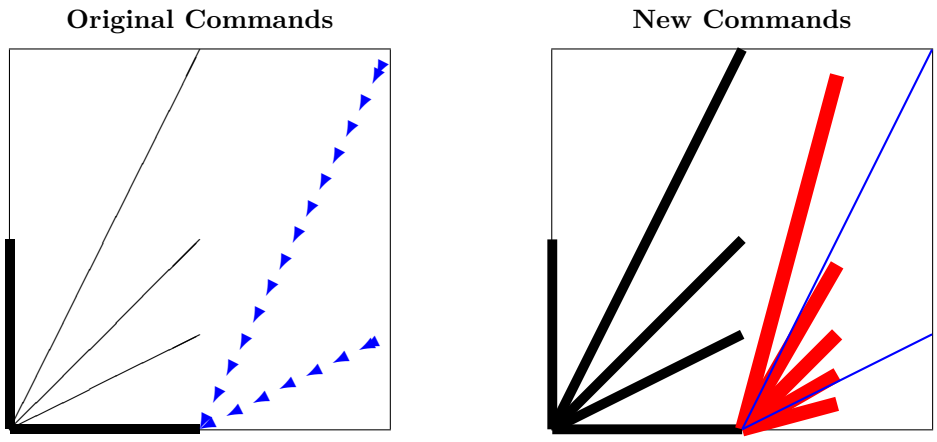


Figure 1: Line

such that you eliminate as much decimal parts as possible). The slope greater then 16384 cannot be obtained.

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.

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 implementations: The black lines in the left half of each picture all have slopes that conform to the restrictions of Standard LaTeX. However, with the new implementation of pict2e sloped lines may assume any arbitrary width given by the `\linethickness` declaration. The right half demonstrates that now arbitrary slopes are possible.

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

The new implementation imposes no restriction with respect to line thickness, minimal horizontal extent, and slope.

The red lines correspond to angles of 15°, 30°, 45°, 60°, and 75°, respectively. This was achieved by multiplying the sine and cosine of each angle by 1000 and rounding to the nearest integer, like this:

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

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. (I.e., X and Y must be relatively prime.) Furthermore, arrow heads come only in two shapes, corresponding to the `\thinlines` and `\thicklines` declarations. (There's also a flaw: the lines will be printed over the arrow heads. See vertical vector in Figure 2.)

From [1, p. 222]:

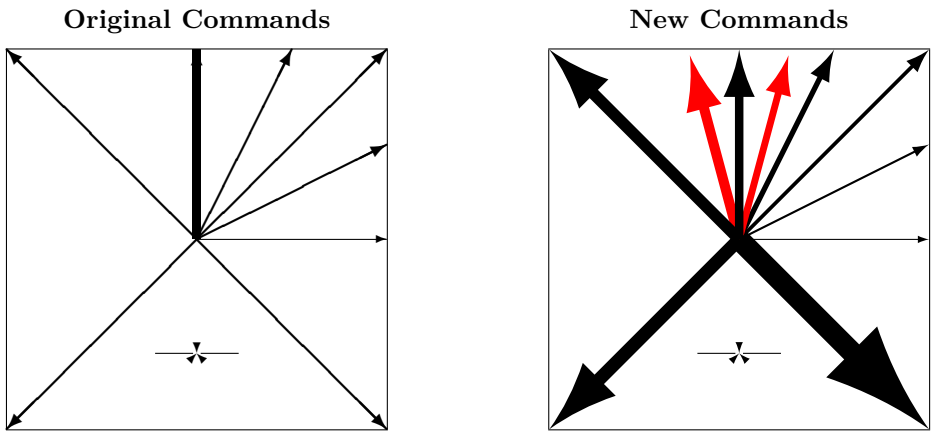


Figure 2: Vector

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

However, to avoid overflow of TEX's dimen arithmetic, the current implementation restricts the slope arguments to real numbers in the range $-1000 \le X, Y \le +1000$, which should be enough. It is usually not a good idea to use slope arguments with the absolute value less then 10^{-4} (the best accuracy is obtained if you use multiples of arguments such that you eliminate as much decimal parts as possible). The slope greater then 16384 cannot be obtained.

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

In the Standard LATEX implementation the horizontal extent of sloped vectors must be at least 10 pt.

From [1, p. 222]:

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

Figure 2 shows the difference between the old and new implementations: The black arrows all have "legal" slopes. The red arrows have slope arguments out of the range permitted by Standard LATEX. Slope arguments that are "illegal" in Standard LATEX produce results similar to those with the `\line` command (this has not been demonstrated here).

The new implementation imposes no restriction with respect to line thickness, minimal horizontal extent, and slope.

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 the 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; see Figure 3, top row), 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, bottom row).

2.3.3 Circle and Dot

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

The (hollow) circles and disks (filled circles) of 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.*

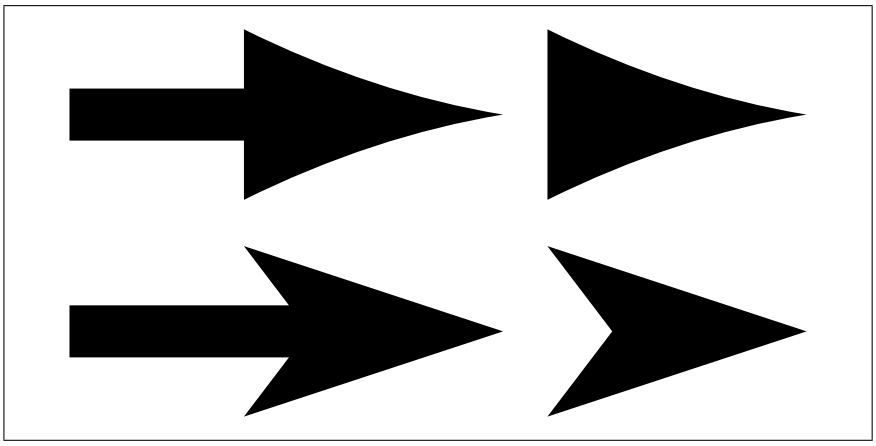


Figure 3: Vector: shape variants of the arrow-heads. Top: LaTeX style vectors. Bottom: PSTricks style vectors.

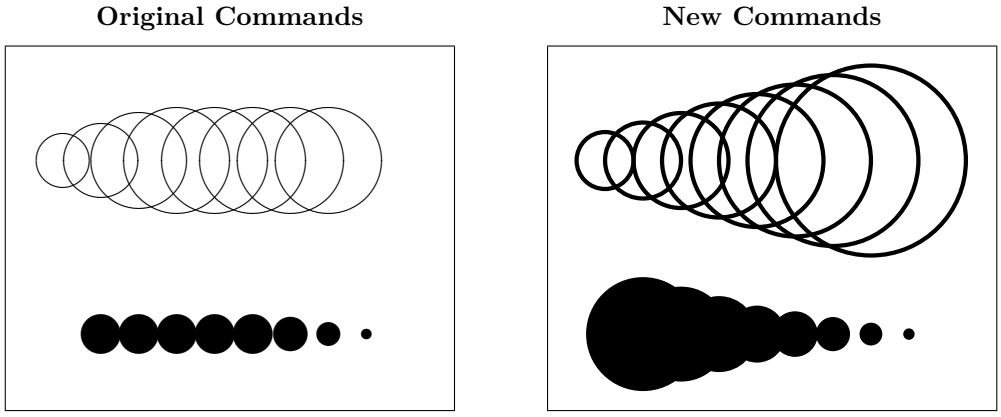


Figure 4: Circle and Dot

With the new implementation there are no more restrictions to the diameter argument. (However, negative 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*⟩`]`

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.) I.e., this implementation of `\oval` will "auto-detect" whether its `[`⟨*rad*⟩`]`

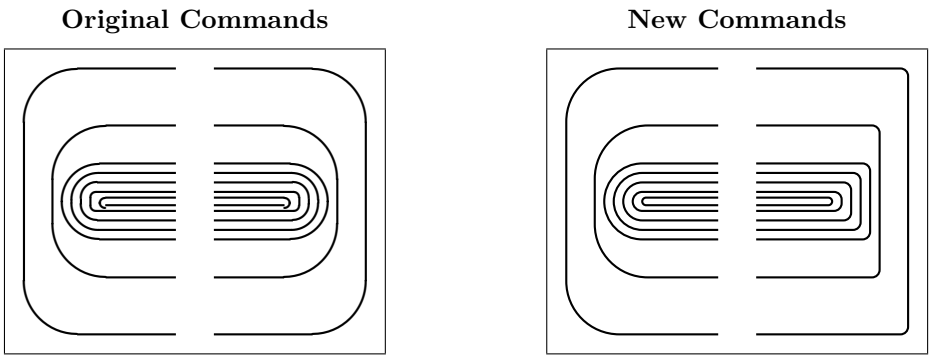


Figure 5: Oval: Radius argument for `\oval` vs. `\maxovalrad`

`\maxovalrad` 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`, by the means of `\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 the 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" is "broken"). 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.)

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 the LaTeX manual [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*⟩`)`
`\qbeziermax` In Standard LaTeX, the *N* argument specifies the number of points to plot: $N+1$ for a positive integer N, appropriate number (at most `\qbeziermax`) for $N = 0$ or if the optional argument is missing. With LaTeX versions prior to 2003/12/01, the quadratic Bezier curves plotted by this package will not match those of the Standard LaTeX implementation exactly, due to a bug in positioning the dots used to produce a curve (cf. latex/3566).

`\bezier` is the obsolescent variant from the old bezier package of vintage LaTeX2.09.

The `\cbezier` command draws a cubic Bezier curve; see [3]. (This is not mentioned in [1] and has been added to the package deliberately.)

From [1, p. 221–223]:

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

More accurately, if the optional argument is absent or is 0, the pict2e package uses primitive operators of the output (back-end) format to draw a full curve.

2.4 Extensions

This section desribe new commands that extend the possibilities of the `picture` environment. It is not our aim to create a powerful collection of macros (like `pstricks` or `tikz`). The main goal of this package is to eliminate the limitations of the standard `picture` commands. But this is done by PostScript and PDF operators that might be easily used for user-level commands and hence significantly improve the drawing possibilities.

2.4.1 Circle arcs

`\arc` `\arc[`⟨*ANGLE1,ANGLE2*⟩`]{`⟨*RAD*⟩`}`
`\arc*` `\arc*[`⟨*ANGLE1,ANGLE2*⟩`]{`⟨*RAD*⟩`}`

These commands are generalizations of `\circle` and `\circle*` commands except that the radius instead of the diameter is given. The optional argument is a comma separated pair of angles given in degrees (implicit value is $[0, 360]$). The arc starts at the point given by $ANGLE1$. If $ANGLE2$ is greater than $ANGLE1$ the arc is drawn in the positive orientation (anticlockwise), if the $ANGLE2$ is smaller than $ANGLE1$ the arc is drawn in the negative orientation (clockwise). The angle of the arc is the absolute value the difference of $ANGLE1$ and $ANGLE2$. Hence the pair $[-10, 80]$ gives the same arc as $[80, -10]$ (a quarter of a circle) while the pairs $[80, 350]$ and $[350, 80]$ give the complementary arc.

In fact, the arc is approximated by cubic Bezier curves with an inaccuracy smaller than 0.0003 (it seems to be sufficiently good).

If `\squarecap` is active then `\arc{`⟨*RAD*⟩`}` produces a circle with a square.

An equivalent `\pIIearc` to `\arc` is defined to solve possible conflicts with other packages.

2.4.2 Line, Vector, polyline, polyvector, and polygon

`\Line` `\Line(`⟨*X1,Y1*⟩`)(`⟨*X2,Y2*⟩`)`
`\polyline` `\polyline(`⟨*X1,Y1*⟩`)(`⟨*X2,Y2*⟩`)...(`⟨*Xn,Yn*⟩`)`
`\Vector` `\Vector(`⟨*X1,Y1*⟩`)(`⟨*X2,Y2*⟩`)`
`\polyvector` `\polyvector(`⟨*X1,Y1*⟩`)(`⟨*X2,Y2*⟩`)...(`⟨*Xn,Yn*⟩`)`
`\polygon` `\polygon(`⟨*X1,Y1*⟩`)(`⟨*X2,Y2*⟩`)...(`⟨*Xn,Yn*⟩`)`
`\polygon*` `\polygon*(`⟨*X1,Y1*⟩`)(`⟨*X2,Y2*⟩`)...(`⟨*Xn,Yn*⟩`)`

A natural way how to describe a line segment is to give the coordinates of the endpoints. The syntax of the `\line`/`\vector` is different because the lines in the standard `picture` environment are made from small line segments of a limited number of slopes given in a font. However, this package changes the `\line` command computing the coordinates of the endpoints and using an internal macro for drawing a line segment with given endpoints. Hence it would be crazy do not use this possibility directly. This is done by the commands `\Line` and `\Vector`. The commands `\polyline` and `\polyvector` draws a stroken line/vector connecting points with given coordinates. The command `\polygon` draws a polygon with given vertices, the star variant gives filled polygon. At least two points should be given.

These command need not be used within a `\put` command (if the coordinates are absolute).

2.4.3 Path commands

`\moveto` `\moveto(`⟨*X,Y*⟩`)`
`\lineto` `\lineto(`⟨*X,Y*⟩`)`
`\curveto` `\curveto(`⟨*X2,Y2*⟩`)(`⟨*X3,Y3*⟩`)(`⟨*X4,Y4*⟩`)`
`\circlearc` `\circlearc[`⟨*N*⟩`]{`⟨*X*⟩`}{`⟨*Y*⟩`}{`⟨*RAD*⟩`}{`⟨*ANGLE1*⟩`}{`⟨*ANGLE2*⟩`}`

These commands directly correspond to the PostScript and PDF path operators. You start defining a path giving its initial point by `\moveto`. Then you can consecutively add a line segment to a given point by `\lineto`, a cubic Bezier curve by `\curveto` (two control points and the endpoint are given) or an arc by `\circlearc` (mandatory parameters are coordinates of the center, radius, initial and final angle).

Drawing arcs is a bit more complicated. There is a special operator only in PostScript (not in PDF) but also in PostScript it is approximated by cubic Bezier curves. Here we use common definition for PostScript and PDF. The arc is drawn such that the initial point given by the initial angle is rotated by $ANGLE2-ANGLE1$ (anticlockwise for positive value and clockwise for negative value) after reducing this difference to the interval $[-720, 720]$. Implicitely (the optional parameter $N = 0$) before drawing an arc a `\lineto` to the initial point of the arc is added. For $N = 1$ `\moveto` instead of `\lineto` is executed—it is useful if you start the path by an arc and do not want to compute and set the initial point. For $N = 2$ the `\lineto` before drawing the arc is omitted—it leads to a bit shorter code for the path but you should be sure that the already defined part of the path ends precisely at the initial point of the arc.

`\closepath` The command `\closepath` is equivalent to `\lineto` to the initial point of the path. After `\strokepath` defining paths you might use either `\strokepath` to draw them or, for closed paths, `\fillpath` `\fillpath` to draw an area bounded by them.

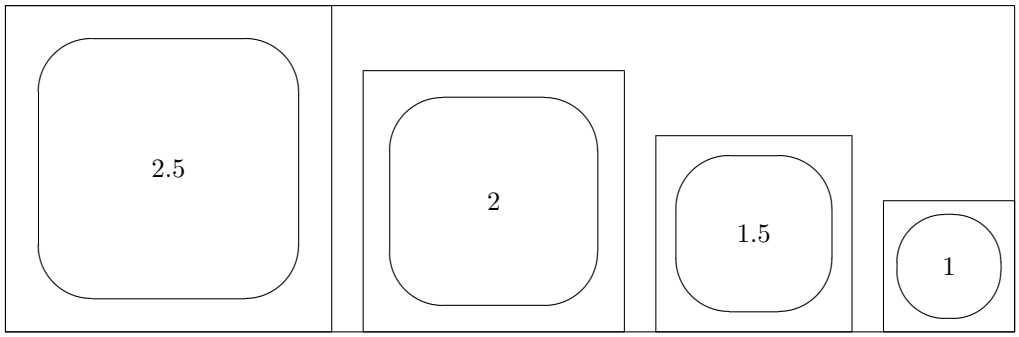
The path construction need not be used within a `\put` command (if the coordinates are absolute).

2.4.4 Ends of paths, joins of subpaths

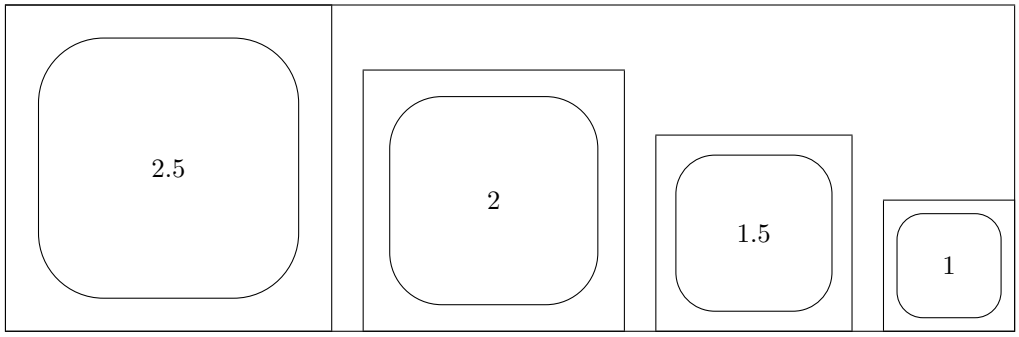
`\buttcap` The shape of ends of paths is controlled by the following commands: `\buttcap` (implicit) `\roundcap` define the end as a line segment, `\roundcap` adds a halfdisc, `\squarecap` adds a halfsquare. `\squarecap` While `\squarecap` is ignored for the path with zero length, `\roundcap` places a disc to the given point. These commands do not apply to `\vector` and to closed paths (`\circle`, full `\oval`, parameter, path constructions ended by `\closepath`).

`\miterjoin` The shape of joins of subpaths is controlled by the following commands: `\miterjoin` `\roundjoin` (implicit) might be defined in such a way that "boundaries" of subpaths are prolonged until `\beveljoin` they intersect (it might be a rather long distance for lines with a small angle between them); `\roundjoin` corresponds to `\roundcap` for both subpaths; `\beveljoin` adds a convex hull of terminal line segments of both subpaths.

Original Commands, [⟨*rad*⟩] or `\maxovalrad` ignored



New Commands, [⟨*rad*⟩] or `\maxovalrad` depends on `\unitlength`



New Commands, [⟨*rad*⟩] or `\maxovalrad` a fixed length

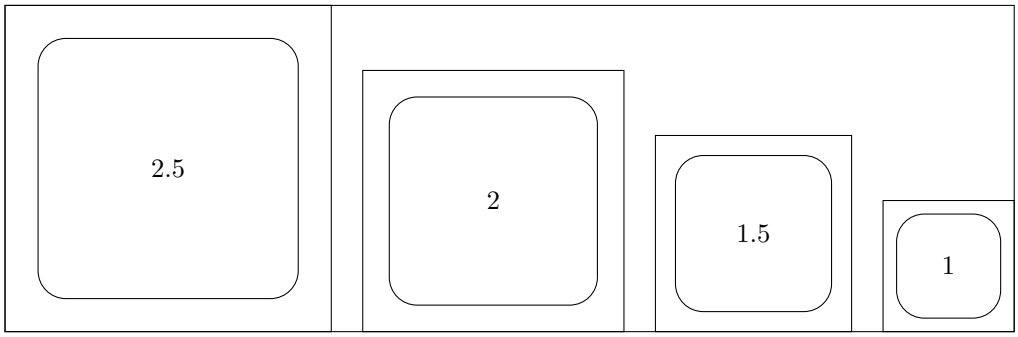


Figure 6: Oval: Radius argument for `\oval`: length vs. number. The number at the centre of each oval gives the relative value of `\unitlength`.

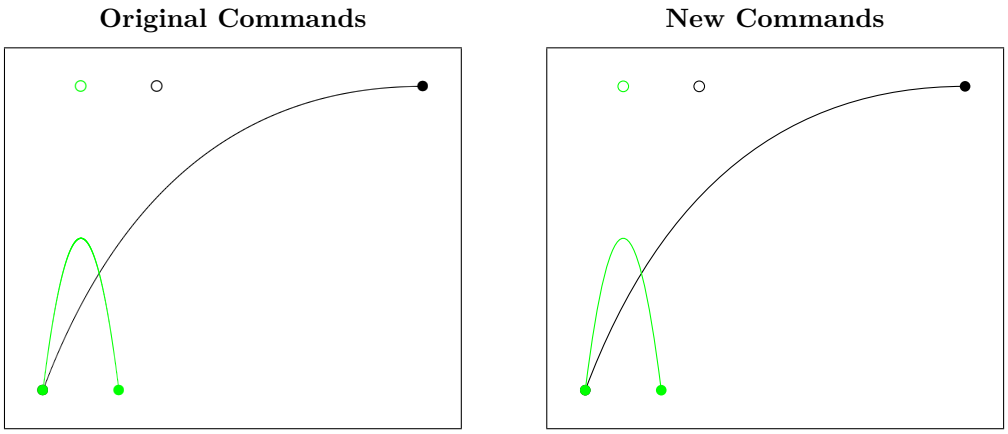


Figure 7: Quadratic Bezier curves

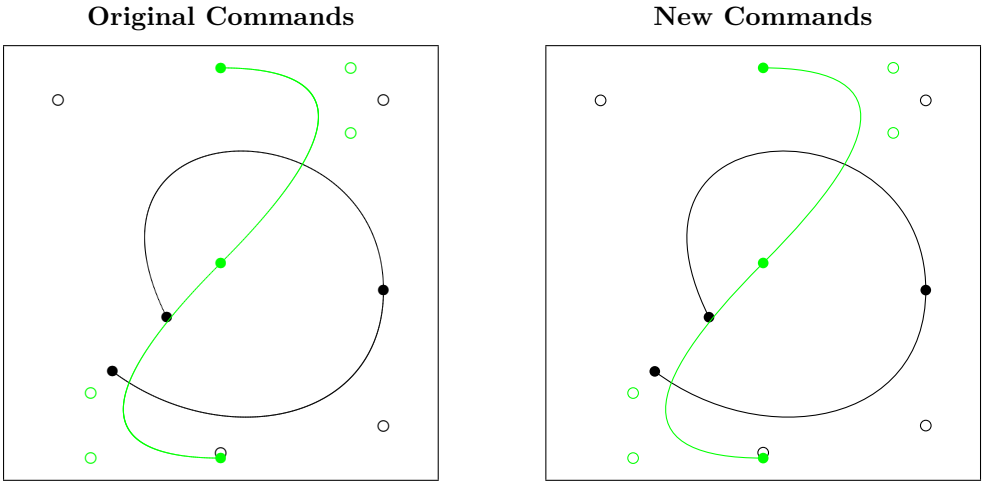


Figure 8: Cubic Bezier curves

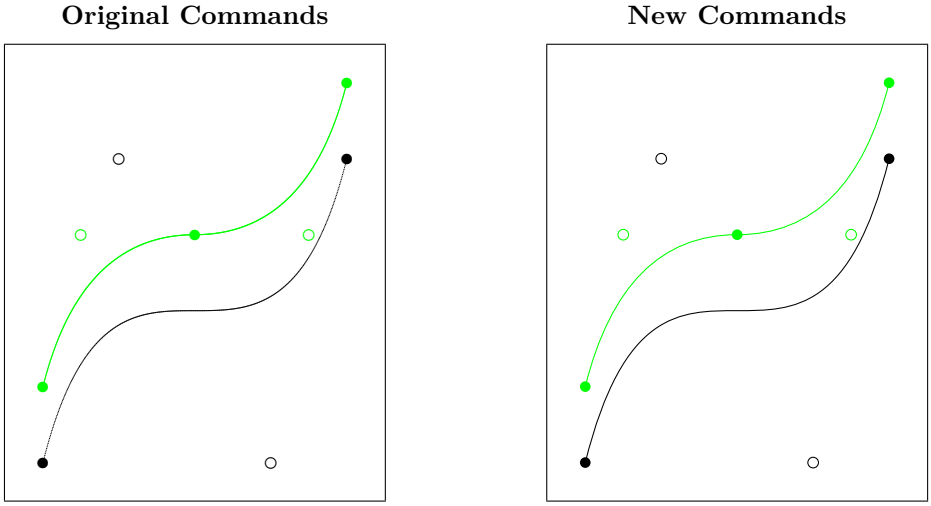


Figure 9: Quadratic (green) and Cubic Bezier curves

3 Implementation

Unlike other packages that have reimplemented or extended some of the commands from Standard LaTeX's `picture` environment, we do not use special fonts, nor draw arbitrary shapes by the means of myriads of small (point) 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.

```
1 ⟨*package⟩
```

3.1 Initialisation

`\Gin@codes` First we save the catcodes of some characters, and set them to fixed values whilst this file is being read. (This is done in almost the same manner as in the graphics and color packages. Alas, we don't need nor want to have `*` as part of control sequence names, so we omit it here.)

```
2 \edef\Gin@codes{%
3  \catcode‘\noexpand\^^A\the\catcode‘\^^A\relax
4  \catcode‘\noexpand\"\the\catcode‘\"\relax
5 % \catcode‘\noexpand\*\the\catcode‘\*\relax
6  \catcode‘\noexpand\!\the\catcode‘\!\relax
7  \catcode‘\noexpand\:\the\catcode‘\:\relax}
8 \catcode‘\^^A=\catcode‘\%
9 \@makeother\"%
10 % \catcode‘\*=11
11 \@makeother\!%
12 \@makeother\:%
```

3.2 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-`⟨*driver*⟩`.def` files. (See file `p2e-drivers.dtx` for details and sample implementations.)

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`. (File `p2e-drivers.dtx` contains an example configuration file suitable for the teTeX and TeXlive distributions; it will be extracted as `pict2e-example.cfg`.)

```
13 \newcommand*\pIIe@mode{-1}
14 \newcommand*\pIIe@code[1]{}
15 \providecommand*\Gin@driver{}
```

`\pIIe@tempa` `\pIIe@tempb` `\pIIe@tempc` At times, we need some temporary 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.

```
16 \newcommand*\pIIe@tempa{}
17 \newcommand*\pIIe@tempb{}
18 \newcommand*\pIIe@tempc{}
```

3.3 Option processing

The driver options are not much of a surprise: they are similar to those of the graphics and color packages.

```
19 \DeclareOption{dvips}{\def\Gin@driver{dvips.def}}
20 \DeclareOption{xdvi}{\ExecuteOptions{dvips}}
```

```
\DeclareOption{dvipdf}{\def\Gin@driver{dvipdf.def}}
\DeclareOption{dvipdfm}{\def\Gin@driver{dvipdfm.def}}
\DeclareOption{dvipdfmx}{\def\Gin@driver{dvipdfmx.def}}
\DeclareOption{pdftex}{\def\Gin@driver{pdftex.def}}
\DeclareOption{luatex}{\def\Gin@driver{luatex.def}}
\DeclareOption{xetex}{\def\Gin@driver{xetex.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}}
```

`\ifpIIe@pdfliteral@ok` `\pIIe@pdfliteral` Check, whether if `\pIIe@pdfliteral` is given in the driver file or `\pdfliteral` available directly.

```
\newif\ifpIIe@pdfliteral@ok
\pIIe@pdfliteral@oktrue
\ifx\pIIe@pdfliteral\@undefined
  \ifx\pdfliteral\@undefined
    \pIIe@pdfliteral@okfalse
    \def\pIIe@pdfliteral#1{%
      \PackageWarning{pict2e}{pdfliteral not supported}%
    }%
  \else
    \let\pIIe@pdfliteral\pdfliteral
  \fi
\fi
```

`\pIIe@buttcap` Do `\buttcap` only if available.

```
\def\pIIe@buttcap{%
  \ifpIIe@pdfliteral@ok
    \buttcap
  \fi
}
```

`\pIIe@FAL` `\pIIe@FAW` `\pIIe@CAW` `\pIIe@FAI` Some macros to parametrize the shape of the vector outline. The following values are "hand optimized" with the aim of emulating LaTeX-style arrows. They also seem suitable for our PSTricks-style arrows. See Figures 10 and 11.

```
\newcommand*\pIIe@FAL{1.52}%
\newcommand*\pIIe@FAW{3.2}%
\newcommand*\pIIe@CAW{1.5pt}%
\newcommand*\pIIe@FAI{0.25}%
```

`\ltxarrows` `\pstarrows` The following user-level macros can be used to change the arrow style (LaTeX-style is the default).

```
\newcommand*\ltxarrows{%
  \let\pIIe@vector=\pIIe@vector@ltx
}
\newcommand*\pstarrows{%
  \let\pIIe@vector=\pIIe@vector@pst
}

\DeclareOption{ltxarrows}{\AtEndOfPackage{\ltxarrows}}
\DeclareOption{pstarrows}{\AtEndOfPackage{\pstarrows}}
```

\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{}%
  \let\pIIe@code\@gobble
}}
```

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
```

3.4 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
```

3.5 Mode check

For PostScript and PDF modes.

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

\pIIe@oldline
\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 definitions.)

```
    \let\pIIe@oldline\line
    \let\pIIe@old@sline\@sline
```

```
    \let\pIIe@oldvector\vector
    \let\pIIe@old@circle\@circle
    \let\pIIe@old@dot\@dot
    \let\pIIe@old@bezier\@bezier
    \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 and demonstration purposes only.

```
    \newcommand*\OriginalPictureCmds{%
      \let\@sline\pIIe@old@sline
      \let\line\pIIe@oldline
      \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 (see page 35).

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

3.6 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
\pIIe@closepath@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}
    \newcommand*\pIIe@closepath@op{closepath}
```

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

(Currently, there are no other modes.)

```
  \fi
```

3.7 Low-level operations

3.7.1 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 we are done with collecting the path of the picture object, 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 3.8.1.

```
              \pIIe@scale@PTtoBP}%
      \if@tempswa
        \edef\y{\pIIe@fill@op}%
      \else
        \edef\x{\x\space
          \strip@pt\@wholewidth\space\pIIe@setlinewidth@op
          \pIIe@linecap\pIIe@linejoin\space}%
        \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}
```

3.7.2 Auxilliary macros

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

Note that since we are using `\@tempdima` and `\@tempdimb` both here and in medium-level macros below, we must be careful not to spoil their values.

`\pIIe@CPx` `\pIIe@CPy` `\pIIe@add@CP` The lengths (coordinates) given as arguments will be stored as "real" numbers using the common trick; i.e., they are put in 'dimen' registers, scaled by 2^{16}. 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}
```

3.8 Medium-level operations

3.8.1 Transformations

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

`\pIIe@PTtoBP` Scaling factor, used below. "pt→bp" ($72/72.27 \approx 0.99626401$). Note the trailing space! (Don't delete it, it saves us some tokens.)

```
\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
```

3.8.2 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}
```

`\pIIe@closepath`

```
\newcommand*\pIIe@closepath{\pIIe@addtoGraph\pIIe@closepath@op}
```

3.9 "Pythagorean Addition" and Division

`\pIIe@pyth` This algorithm is copied from the PiCTeX package [4] by Michael Wichura, with his permission Here is his description:

> Suppose $x > 0$, $y > 0$. Put $s = x + y$. Let $z = (x^2 + y^2)^{1/2}$. Then $z = s \times f$, where
>
> $$f = (t^2 + (1-t)^2)^{1/2} = ((1+\tau^2)/2)^{1/2}$$
>
> and $t = x/s$ and $\tau = 2(t - 1/2)$.

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

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

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

\@tempdimb = abs(y)

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

\@tempdimb = s = abs(x) + abs(y)

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

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

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

\@tempdima = 8 × abs(x)

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

\@tempdimc = $8\,t$ = 8 × abs(x)/s

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

\@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 $\sqrt{(u)}$

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

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

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

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

```
        \global\divide\@tempdimc 8
      \fi
      \edef\x{\endgroup#3=\the\@tempdimc}%
    \x}
```

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

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

\pIIe@divide The following macro for division is a slight modification of the macro from `curve2e` by Claudio Beccari with his permission. Real numbers are represented as dimens in `pt`.

```
  \newcommand*\pIIe@divide[3]{%
```

All definitions inside a group.

```
    \begingroup
    \dimendef\Numer=254\relax \dimendef\Denom=252\relax
    \countdef\Num=254\relax   \countdef\Den=252\relax
    \countdef\I=250\relax     \countdef\Numb=248\relax
    \Numer #1\relax \Denom #2\relax
```

Make numerator and denominator nonnegative, save sign.

```
    \ifdim\Denom<\z@ \Denom -\Denom \Numer=-\Numer \fi
    \ifdim\Numer<\z@ \def\sign{-}\Numer=-\Numer \else \def\sign{}\fi
```

Use \maxdimen for $x/0$ (this should not appear).

```
    \ifdim\Denom=\z@
      \edef\Q{\strip@pt\maxdimen}%
      \PackageWarning{pict2e}%
        {Division by 0, \sign\strip@pt\maxdimen\space used}{}%
    \else
```

Converse to integers and find integer part of the ratio. If it is too large (dimension overflow), use \maxdimen otherwise find the remainder and start the iteration process to find 6 digits of the decimal expression.

```
      \Num=\Numer \Den=\Denom
      \Numb=\Num \divide\Numb\Den
      \ifnum\Numb>16383
        \edef\Q{\strip@pt\maxdimen}%
        \PackageWarning{pict2e}%
          {Division overflow, \sign\strip@pt\maxdimen\space used}{}%
      \else
        \edef\Q{\number\Numb.}%
        \multiply \Numb\Den \advance\Num -\Numb
        \I=6\relax
        \@whilenum \I>\z@ \do{\pIIe@@divide\advance\I\m@ne}%
      \fi
    \fi
```

A useful trick to define #3 outside the group without using \global (if the macro is used inside another group.)

```
    \edef\tempend{\noexpand\endgroup\noexpand#3=\sign\Q\p@}%
    \tempend}
```

\pIIe@@divide Iteration macro for finding decimal expression of the ratio. \Num is the remainder of the previous division, \Den is the denominator (both are integers).

```
  \def\pIIe@@divide{%
```

Reduce both numerator and denominator if necessary to avoid overflow in the next step.

```
    \@whilenum \Num>214748364 \do{\divide\Num\tw@ \divide\Den\tw@}%
```

Find the next digit of the decimal expression.

```
    \multiply \Num 10
    \Numb=\Num \divide\Numb\Den
    \edef\Q{\Q\number\Numb}%
```

Find the remainder.

```
    \multiply \Numb\Den \advance \Num -\Numb
```

Stop the iteration if the remainder is zero.

```
    \ifnum\Num>\z@\else\I=0\fi}
```

3.10 High-level operations

\pIIe@checkslopeargs Common code for \line and \vector.

```
  \newcommand*\pIIe@checkslopeargsline[2]{%
    \pIIe@checkslopeargs{#1}{#2}{16383}}
  \newcommand*\pIIe@checkslopeargsvector[2]{%
    \pIIe@checkslopeargs{#1}{#2}{1000}}
  \newcommand*\pIIe@checkslopeargs[3]{%
    \edef\@tempa{#1}\expandafter\pIIe@checkslopearg\@tempa.:{#3}%
    \edef\@tempa{#2}\expandafter\pIIe@checkslopearg\@tempa.:{#3}%
```

A bit incompatible with Standard LaTeX: slope $(0, 0)$ raises an error.

```
    \ifdim #1\p@=\z@ \ifdim #2\p@=\z@ \@badlinearg \fi\fi}
```

```
\def\pIIe@checkslopearg #1.#2:#3{%
  \def\@tempa{#1}%
  \ifx\@tempa\empty\def\@tempa{0}\fi
  \ifx\@tempa\space\def\@tempa{0}\fi
  \ifnum\ifnum\@tempa<\z@-\fi\@tempa>#3 \@badlinearg \fi}
\def\@badlinearg{\PackageError
  {pict2e}{Bad \protect\line\space or \protect\vector\space argument}{}}
```

3.10.1 Line

`\line` `\line(`$\langle x,y\rangle$`){`$\langle l_x\rangle$`}`:

```
\def\line(#1,#2)#3{%
  \begingroup
  \pIIe@checkslopeargsline{#1}{#2}%
  \@tempdima=#1pt\relax \@tempdimb=#2pt\relax
  \@linelen #3\unitlength
  \ifdim\@linelen<\z@ \@badlinearg \else
    \pIIe@sline
    \pIIe@moveto\z@\z@
    \pIIe@lineto\@xdim\@ydim
    \pIIe@strokeGraph
```

Simulated bounding box

```
    \box\@tempboxa
  \fi
  \endgroup}
```

`\pIIe@sline` Common code for `\line` and `\vector`.

```
\newcommand*\pIIe@sline{%
```

Calculation of the endpoints `\@xdim`, `\@ydim` (used for `\line` only).

```
  \ifdim\@tempdima=\z@
    \ifdim\@tempdimb<\z@\@linelen-\@linelen\fi
    \@ydim=\@linelen
    \@xdim=\z@
  \else
    \ifdim\@tempdima<\z@\@linelen-\@linelen\fi
    \ifdim\@tempdimb=\z@
      \@xdim=\@linelen
      \@ydim=\z@
    \else
      \pIIe@divide\@tempdimb\@tempdima\dimen@
      \@ydim=\strip@pt\dimen@\@linelen
      \@xdim=\@linelen
    \fi
  \fi
```

Prepare a box that can be used as a bounding box for `\line` and `\vector` to achieve the same behavior as standard LaTeX outside of a picture environment.

```
  \@ovxx=\ifnum\@xdim=\z@ \z@\else\@linelen\fi
  \@ovyy=\ifnum\@ydim<\z@ \z@\else\@ydim\fi
  \@ovdy=\ifnum\@ydim<\z@ -\@ydim\else\z@\fi
  \setbox\@tempboxa\hbox{%
    \vrule\@height \@ovyy \@depth \@ovdy \@width \z@
    \vrule\@height \z@ \@depth \z@ \@width \@ovxx}}
```

3.10.2 Vector

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

`\vector(`$\langle x,y\rangle$`){`$\langle l_x\rangle$`}`: 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
  \pIIe@checkslopeargsvector{#1}{#2}%
  \@tempdima=#1pt\relax \@tempdimb=#2pt\relax
  \@linelen#3\unitlength
  \ifdim\@linelen<\z@ \@badlinearg \else
    \pIIe@sline
    \@linelen#3\unitlength
    \pIIe@pyth{\@tempdima}{\@tempdimb}\dimen@
    \ifdim\@tempdima=\z@ \else
      \ifdim\@tempdimb=\z@ \else
```

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

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

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

```
    \pIIe@divide{\@tempdimb}\dimen@\@ydim
    \pIIe@divide{\@tempdima}\dimen@\@xdim
```

Rotate the following vector/arrow outlines by angle θ:

$$\cos\theta \quad \sin\theta \quad -\sin\theta \quad \cos\theta \quad 0 \quad 0$$

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

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

```
    \pIIe@vector
    \pIIe@fillGraph
```

Simulated bounding box

```
    \box\@tempboxa
  \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 options, via `\AtEndOfPackage`.

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

LaTeX version The arrows drawn by the variant generated by the ltxarrows package option are modeled after those in the fonts used by the Standard LaTeX version of the picture commands (`ltpictur.dtx`). See Figure 10.

`\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
```

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

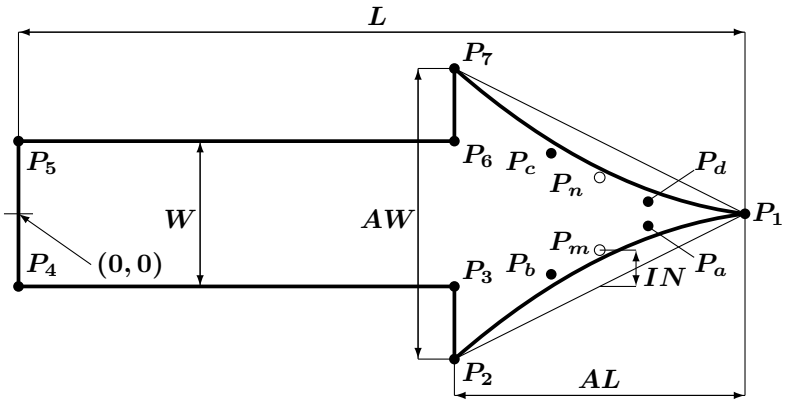


Figure 10: Sketch of the path drawn by the LaTeX-like implementation of \vector. (Note: We are using the redefined macros of pict2e!)

$P_d = P_1 + 1/3(P_n - P_1)$

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

$P_c = P_7 + 1/3(P_n - P_7)$

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

P_1

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

$P_a \quad P_b \quad P_2$

```
\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
```

$P_c \quad P_d \quad P_1$

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

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 11.

\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
```

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

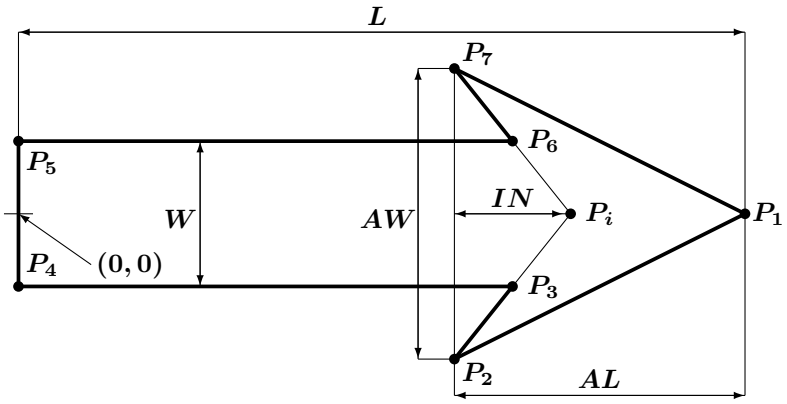


Figure 11: Sketch of the path drawn by the PSTricks-like implementation of \vector. (Note: We are using the redefined macros of pict2e!)

```
    \divide\@ydim\tw@
    \@ovyy\@ydim \advance\@ovyy-\@halfwidth
    \@ovdx\pIIe@FAI\@ovxx
    \pIIe@divide\@ovdx\@ydim\@tempdimc
    \@ovxx\strip@pt\@ovyy\@tempdimc
    \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_i

```
      \pIIe@lineto\@ovdx\z@
    \fi
```

P_7

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

P_1

```
    \pIIe@lineto\@linelen\z@}
```

3.10.3 Circle and Dot

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

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

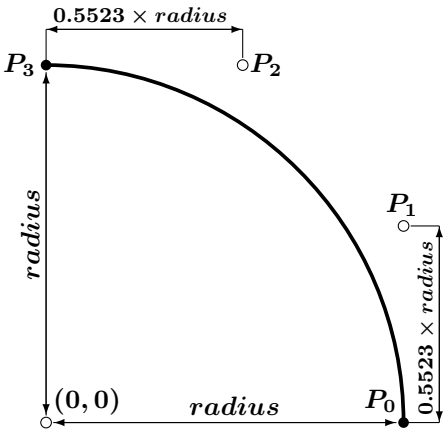


Figure 12: Sketch of the quarter circle path drawn by `\pIIe@qcircle` (NE quarter)

`\@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@ \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. (Note to ourselves: Use of the `@tempswa` switch both here and inside quarter-circle! Hence a group is necessary there.)

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

`\pIIe@circle` Approximate a full circle by four quarter circles, use the standard shape of ends.

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

`\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]).
Sacrifice a save level (otherwise a private "switch" macro were necessary!)

```
\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(*), \protect\oval, \protect\arc(*) or
    \protect\circlearc.}%
    {The radius of a circle, dot, arc or oval corner must be greater than zero.}}%
```

3.10.4 Oval

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

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

`\pIIe@defaultUL` `\pIIe@def@UL` The aforementioned behaviour seems necessary, since [1, p. 223] does not specify explicitly whether the `[`⟨*rad*⟩`]` argument should be given in terms of `\unitlength` or as an absolute length. To implement this feature, we borrow from the graphics package: See `\Gin@defaultbp` and `\Gin@def@bp` from `graphics.dtx`.

```
\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!?)

```
\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
    \edef#2{\the\dimen@}}
```

`\oval` `\pIIe@maxovalrad` `\pIIe@oval` 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). Unlike Standard LaTeX, we check for negative or zero radius argument. `\pIIe@maxovalrad` is the internal variant of `\maxovalrad`.

```
\newcommand*\pIIe@maxovalrad{}
\newcommand*\pIIe@oval{}
```

```
\def\pIIe@oval#1(#2,#3){\@ifnextchar[{\@oval(#2,#3)}{\@oval(#2,#3)[]}}
\renewcommand*\oval[1][\maxovalrad]{%
  \begingroup \pIIe@defaultUL\pIIe@maxovalrad{#1}%
    \ifdim\pIIe@maxovalrad<\z@ \pIIe@badcircarg \fi
```

Can't close the group here, since arguments must be parsed.

```
    \pIIe@oval}
```

`\@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}%
```

For the whole oval remove use the standard shape of ends.

```
  \ifnum15=\@tempcnta \pIIe@buttcap \fi
```

"`@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@
```

Now we've finished, draw the oval and finally 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]{%
% \end{macrocode}
% Bit set?
%    \begin{macrocode}
  \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}
```

3.10.5 Quadratic Bezier Curve

`\@bezier` If #1=0 the primitive operators ot the (back-end) format are used. The kernel version of `\@bezier` uses `\put` internally, which features `\@killglue` and `\ignorespaces` commands in turn (at the beginning and end, respectively). Since we don't use `\put`, we have to add the latter commands by hand.

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

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

```
    \@killglue
    \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
    \ignorespaces
  \else
    \pIIe@old@bezier{#1}(#2,#3)(#4,#5)(#6,#7)
  \fi}
```

`\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}
```

3.10.6 Circle arcs

We need some auxiliary dimensions.

```
\ifx\undefined\@arclen \newdimen\@arclen \fi
\ifx\undefined\@arcrad \newdimen\@arcrad \fi
\ifx\undefined\@tempdimd \newdimen\@tempdimd \fi
```

`\pIIe@arc` #1: 0 (implicit) if we connect arc with a current point, 1 if we start drawing by this arc, 2 if we continue drawing. Other parameters: coordinates of the center (dimensions), radius (dimension), initial and final angle. If the final angle is greater then the initial angle, we "draw" in the positive sense (anticlockwise) otherwise in the negative sense (clockwise). First we check whether the radius is not negative and reduce the rotation to the interval $[-720, 720]$.

```
\newcommand*\pIIe@arc[6][0]{%
  \@arcrad #4\relax
  \ifdim \@arcrad<\z@ \pIIe@badcircarg \else
    \@arclen #6\p@ \advance\@arclen -#5\p@
    \ifdim \@arclen<\z@ \def\sign{-}\else\def\sign{}\fi
    \ifdim \sign\@arclen>720\p@
      \PackageWarning {pict2e}{The arc angle is reduced to -720..720}%
      \@whiledim \sign\@arclen>720\p@ \do {\advance\@arclen-\sign360\p@}%
      \@tempdima #5\p@ \advance\@tempdima \@arclen
      \edef\@angleend{\strip@pt\@tempdima}%
      \pIIe@@arc{#1}{#2}{#3}{#4}{#5}{\@angleend}%
    \else
      \pIIe@@arc{#1}{#2}{#3}{#4}{#5}{#6}%
    \fi
  \fi}
```

If the angle (its absolute value) is too large, the arc is recursively divided into 2 parts until the angle is at most 90 degrees.

```
\newcommand*\pIIe@@arc[6]{%
  \begingroup
  \ifdim \sign\@arclen>90\p@
    \divide\@arclen 2
    \@tempdima #5\p@ \advance\@tempdima \@arclen
    \edef\@anglemid{\strip@pt\@tempdima}%
    \def\@temp{\pIIe@@arc{#1}{#2}{#3}{#4}{#5}}%
    \expandafter\@temp\expandafter{\@anglemid}%
    \def\@temp{\pIIe@@arc{2}{#2}{#3}{#4}}%
    \expandafter\@temp\expandafter{\@anglemid}{#6}%
  \else
```

We approximate the arc by a Bezier curve. First we calculate the coordinates of the initial point:

```
    \CalculateSin{#5}\CalculateCos{#5}%
    \@tempdima\UseCos{#5}\@arcrad \advance\@tempdima #2\relax
    \@tempdimb\UseSin{#5}\@arcrad \advance\@tempdimb #3\relax
```

The coordinates are added to the path if and how necessary:

```
    \ifcase #1\relax
        \pIIe@lineto\@tempdima\@tempdimb
    \or \pIIe@moveto\@tempdima\@tempdimb
    \or
    \else \PackageWarning {pict2e}%
          {Illegal obligatory argument in \protect\circlearc.}%
    \fi
```

The distance of control points from the endpoints is $\frac{4}{3}r\tan\frac{\varphi}{4}$ (φ is the angle and r is the radius of the arc).

```
      \@tempdimc\@arclen \divide\@tempdimc\@iv
      \edef\@angle{\strip@pt\@tempdimc}\CalculateTan{\@angle}%
      \@linelen\UseTan{\@angle}\@arcrad \@linelen4\@linelen \divide\@linelen\thr@@
```

Coordinates of the first control point, added to the path:

```
      \advance\@tempdima-\UseSin{#5}\@linelen
      \advance\@tempdimb \UseCos{#5}\@linelen
      \pIIe@add@nums\@tempdima\@tempdimb
```

Coordinates of the endpoint:

```
      \CalculateSin{#6}\CalculateCos{#6}%
      \@tempdima \UseCos{#6}\@arcrad \advance\@tempdima #2\relax
      \@tempdimb \UseSin{#6}\@arcrad \advance\@tempdimb #3\relax
```

Coordinates of the second control point:

```
      \@tempdimc \UseSin{#6}\@linelen \advance\@tempdimc \@tempdima
      \@tempdimd-\UseCos{#6}\@linelen \advance\@tempdimd \@tempdimb
```

Adding the second control point and the endpoint to the path

```
      \pIIe@add@nums\@tempdimc\@tempdimd
      \pIIe@add@CP\@tempdima\@tempdimb
      \pIIe@addtoGraph\pIIe@curveto@op
    \fi
    \endgroup}
```

`\arc` The `\arc` command generalizes (except that the radius instead of the diameter is used) the standard `\circle` adding as an obligatory first parameter comma separated pair of angles (initial and final). We start with `\pIIearc` to avoid conflicts with other packages.

```
\newcommand*\pIIearc
  {\@ifstar{\@tempswatrue\pIIe@arc@}{\@tempswafalse\pIIe@arc@}}
\newcommand*\pIIe@arc@[2][0,360]{\pIIe@arc@@(#1){#2}}
\def\pIIe@arc@@(#1,#2)#3{%
  \if@tempswa
    \pIIe@moveto\z@\z@
    \pIIe@arc{\z@}{\z@}{#3\unitlength}{#1}{#2}%
    \pIIe@closepath\pIIe@fillGraph
  \else
    \pIIe@arc[1]{\z@}{\z@}{#3\unitlength}{#1}{#2}%
    \pIIe@strokeGraph
  \fi}
\ifx\undefined\arc
\else
    \PackageWarning{pict2e}{\protect\arc\space redefined}%
\fi
\let\arc\pIIearc
```

3.10.7 Line, Vector, polyline, polyvector, and polygon

`\Line` `\polyline` `\Vector` `\polyvector` `\polygon` We use recursive macros for `\polyline`, `\polyvector`, and `\polygon`.

```
\let\lp@r( \let\rp@r)
\def\Line(#1,#2)(#3,#4){\polyline(#1,#2)(#3,#4)}
\def\polyline(#1,#2){%
  \@killglue
  \pIIe@moveto{#1\unitlength}{#2\unitlength}%
  \@ifnextchar\lp@r{\@polyline}{\PackageWarning{pict2e}%
    {Polygonal lines require at least two vertices!}%
  \ignorespaces}}
\def\@polyline(#1,#2){%
```

```
    \pIIe@lineto{#1\unitlength}{#2\unitlength}%
    \@ifnextchar\lp@r{\@polyline}{\pIIe@strokeGraph\ignorespaces}}
  \def\Vector(#1,#2)(#3,#4){\polyvector(#1,#2)(#3,#4)}
  \def\polyvector(#1,#2){%
    \@killglue
    \@ifnextchar\lp@r{\begingroup\@polyvector(#1,#2)}{%
      \PackageWarning{pict2e}%
      {Polygonal vectors require at least two vertices!}\ignorespaces}}
  \def\@polyvector(#1,#2)(#3,#4){%
```

See the similar definition for `\vector` (3.10.2)

```
    \@xdim#1\unitlength \@tempdima=#3\unitlength
    \@ydim#2\unitlength \@tempdimb=#4\unitlength
    \advance\@tempdima-\@xdim \advance\@tempdimb-\@ydim
    \ifdim\@tempdima=\z@ \@linelen\@tempdimb \else
      \ifdim\@tempdimb=\z@ \@linelen\@tempdima \else
        \pIIe@pyth\@tempdima\@tempdimb\@linelen
      \fi
    \fi
    \ifdim\@linelen<\z@ \@linelen-\@linelen\fi
    \pIIe@divide{\@tempdima}\@linelen\@ovxx
    \pIIe@divide{\@tempdimb}\@linelen\@ovyy
```

Note the shift to the previous point in addition to the rotation.

```
    \pIIe@concat\@ovxx\@ovyy{-\@ovyy}\@ovxx\@xdim\@ydim
    \pIIe@vector \pIIe@fillGraph
    \@ifnextchar\lp@r{\@polyvector(#3,#4)}{\endgroup\ignorespaces}}
  \def\polygon{%
    \@killglue
    \@ifstar{\begingroup\@tempswatrue\@polygon}%
      {\begingroup\@tempswafalse\@polygon}}
  \def\@polygon(#1,#2){%
    \pIIe@moveto{#1\unitlength}{#2\unitlength}%
    \@ifnextchar\lp@r{\@@polygon}{\PackageWarning{pict2e}%
      {Polygons require at least two vertices!}%
    \ignorespaces}}
  \def\@@polygon(#1,#2){\pIIe@lineto{#1\unitlength}{#2\unitlength}%
    \@ifnextchar\lp@r{\@@polygon}{\pIIe@closepath
      \if@tempswa\pIIe@fillGraph\else\pIIe@strokeGraph\fi
      \endgroup
      \ignorespaces}}
```

3.10.8 Path commands

`\moveto` `\lineto` `\curveto` `\circlearc` `\closepath` `\strokepath` `\fillpath`

Direct access to path constructions in PostScript and PDF.

```
  \def\moveto(#1,#2){%
    \@killglue
    \pIIe@moveto{#1\unitlength}{#2\unitlength}%
    \ignorespaces}
  \def\lineto(#1,#2){%
    \@killglue
    \pIIe@lineto{#1\unitlength}{#2\unitlength}%
    \ignorespaces}
  \def\curveto(#1,#2)(#3,#4)(#5,#6){%
    \@killglue
    \pIIe@curveto{#1\unitlength}{#2\unitlength}{#3\unitlength}{#4\unitlength}%
      {#5\unitlength}{#6\unitlength}%
    \ignorespaces}
  \newcommand*\circlearc[6][0]{%
    \@killglue
```

```
    \pIIe@arc[#1]{#2\unitlength}{#3\unitlength}{#4\unitlength}{#5}{#6}%
    \ignorespaces}
\def\closepath{\pIIe@closepath}
\def\strokepath{\pIIe@strokeGraph}
\def\fillpath{\pIIe@fillGraph}
```

3.10.9 Ends of paths, joins of subpaths

\buttcap
\roundcap
\squarecap
\miterjoin
\roundjoin
\beveljoin

Ends of paths and joins of subpaths in PostScript and PDF.

```
\ifcase\pIIe@mode\relax
\or
  \newcommand*\pIIe@linecap@op{setlinecap}
  \newcommand*\pIIe@linejoin@op{setlinejoin}
\or
  \newcommand*\pIIe@linecap@op{J}
  \newcommand*\pIIe@linejoin@op{j}
\fi
\def\pIIe@linecap{}
\def\pIIe@linejoin{}
\def\buttcap{\edef\pIIe@linecap{ 0 \pIIe@linecap@op}}
\def\roundcap{\edef\pIIe@linecap{ 1 \pIIe@linecap@op}}
\def\squarecap{\edef\pIIe@linecap{ 2 \pIIe@linecap@op}}
\def\miterjoin{\edef\pIIe@linejoin{ 0 \pIIe@linejoin@op}}
\def\roundjoin{\edef\pIIe@linejoin{ 1 \pIIe@linejoin@op}}
\def\beveljoin{\edef\pIIe@linejoin{ 2 \pIIe@linejoin@op}}
```

3.11 Commands from other packages

3.11.1 Package ebezier

One feature from [3].

\cbezier
\@cbezier
\pIIe@@cbezier

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

Like the kernel version of `\@bezier`, the original version of `\@cbezier` uses `\put` internally, which features `\@killglue` and `\ignorespaces` commands in turn (at the beginning and end, respectively). Since we don't use `\put`, we have to add the latter commands by hand.
Original head of the macro:

```
\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){%
    \@killglue
    \pIIe@moveto{#2\unitlength}{#3\unitlength}%
    \pIIe@curveto{#4\unitlength}{#5\unitlength}%
      {#6\unitlength}{#7\unitlength}{#8\unitlength}{#9\unitlength}%
    \pIIe@strokeGraph
    \ignorespaces}%
}
```

3.11.2 Other packages

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

3.12 Mode 'original'

Other branch of the big switch, started near the beginning of the code (see page 16).

```
\else
```

`\oval` `\maxovalrad` `\OriginalPictureCmds`

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
```

3.13 Final clean-up

Restore Catcodes.

```
\Gin@codes
\let\Gin@codes\relax
```

⟨/package⟩

Acknowledgements

We would like to thank Michael Wichura for granting us permission to use his implementation of the algorithm for "pythagorean addition" from his PiCTeX package. Thanks go to Michael Vulis (MicroPress) for hints regarding a driver for the VTeX system. Walter Schmidt has reviewed the documentation and code, and has tested the VTeX driver. The members of the "TeX-Stammtisch" in Berlin, Germany, have been involved in the development of this package as our guinea pigs, i.e., alpha-testers; Jens-Uwe Morawski and Herbert Voss have also been helpful with many suggestions and discussions. Thanks to Claudio Beccari (`curve2e`) for some macros and testing. Thanks to Petr Olšák for some macros.

Finally we thank the members of The LaTeX Team for taking the time to evaluate our new implementation of the picture mode commands, and eventually accepting it as the "official" pict2e package, as well as providing the `README` file.

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

Change History

Index

Numbers written in italic refer to the page where the corresponding entry is described; numbers underlined refer to the code line of the definition; numbers in roman refer to the code lines where the entry is used.

Q

R

S

T

U

V