

The `pict2e` package*

Hubert Gäßlein, Rolf Niepraschk[†] and Josef Tkadlec[‡]

2016/02/05

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

1	Introduction	2
2	Usage	3
2.1	Package options	3
2.1.1	Driver options	3
2.1.2	Other options	3
2.1.3	Debugging options	3
2.2	Configuration file	3
2.3	Details: Changes to user-level commands	4
2.3.1	Line	4
2.3.2	Vector	5
2.3.3	Circle and Dot	6
2.3.4	Oval	7
2.3.5	Bezier Curves	8
2.4	Extensions	8
2.4.1	Circle arcs	9
2.4.2	Lines, polygons	9
2.4.3	Path commands	9
2.4.4	Ends of paths, joins of subpaths	10
3	Implementation	13
3.1	Initialisation	13
3.2	Preliminaries	13
3.3	Option processing	13
3.4	Output driver check	15
3.5	Mode check	15
3.6	Graphics operators	16
3.7	Low-level operations	17

*This document corresponds to `pict2e.sty` v0.3b, dated 2016/02/05, documentation dated 2016/01/09.

[†]Rolf.Niepraschk@gmx.de

[‡]j.tkadlec@email.cz

3.7.1	Collecting the graphics instructions and handling the output	17
3.7.2	Auxilliary macros	18
3.8	Medium-level operations	18
3.8.1	Transformations	18
3.8.2	Path definitions	19
3.9	“Pythagorean Addition” and Division	20
3.10	High-level operations	23
3.10.1	Line	23
3.10.2	Vector	23
3.10.3	Circle and Dot	27
3.10.4	Oval	28
3.10.5	Circle arcs	31
3.10.6	Lines and polygons	33
3.10.7	Path commands	33
3.10.8	Ends of paths, joins of subpaths	34
3.11	Commands from other packages	34
3.11.1	Package <code>ebezier</code>	34
3.11.2	Other packages	35
3.12	Mode ‘original’	35
3.13	Final clean-up	35

List of Figures

1	Line	5
2	Vector	6
3	Vector: shape variants of the arrow-heads	7
4	Circle and Dot	7
5	Oval: Radius argument for <code>\oval</code> vs. <code>\maxovalrad</code>	8
6	Oval: Radius argument for <code>\oval</code> : length vs. number	11
7	Quadratic Bezier curves	12
8	Cubic Bezier curves	12
9	Quadratic (green) and Cubic Bezier curves	12
10	L ^A T _E X-like implementation of <code>\vector</code>	25
11	PSTricks-like implementation of <code>\vector</code>	26
12	Auxillary macro <code>\pIle@qcircle</code> —draw a quarter circle	27

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 “L^AT_EX: A Document Preparation System” has not yet been produced. It is unlikely that the L^AT_EX3 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 L^AT_EX `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 L ^A T _E X style vectors (default).
pstarrows	Draw P ^S Tricks 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 pdfT _E X output; marks the <code>pict2e</code> generated code in the output files.
hide	Suppresses all graphics output from <code>pict2e</code> .

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 config-

uration 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 $\text{\LaTeX}$ manual [1].

Here’s a collection of quotes relevant to the `pict2e` package from the $\text{\LaTeX}$ 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 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` [$\langle rad \rangle$]:
An explicit `rad` argument can be used only with the `pict2e` package; the default value is the radius of the largest quarter-circle $\text{\LaTeX}$ can draw without the `pict2e` package.

2.3.1 Line

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

In the Standard $\text{\LaTeX}$ implementation the slope arguments $(\langle X,Y \rangle)$ are restricted to integers in the range $-6 \leq X,Y \leq +6$, 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 $\text{\TeX}$ ’s `dimens`, the slope arguments are real numbers in the range $-16383 \leq X,Y \leq +16383$. It is usually not a good idea to use slope arguments with the absolute value less than 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 than 16384 cannot be obtained.

Furthermore, unlike the Standard $\text{\LaTeX}$ 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.

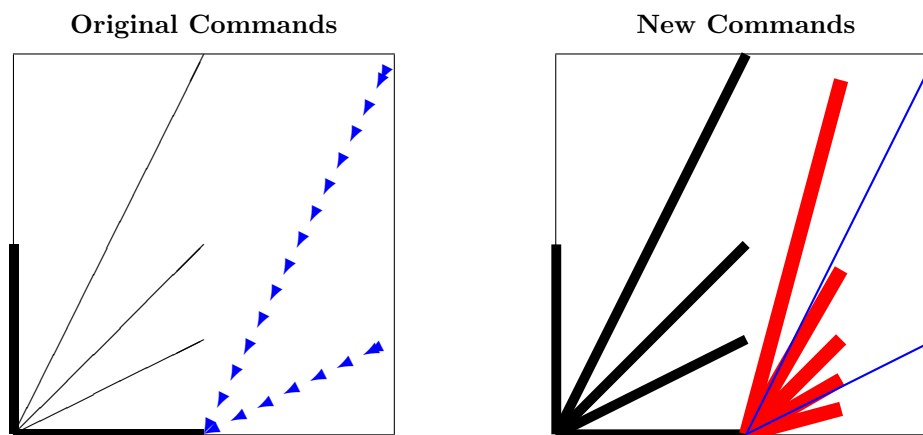


Figure 1: Line

In the Standard $\text{\LaTeX}$ 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 $\text{\LaTeX}$. 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 $\text{\LaTeX}$ produces in such cases. (In $\text{\LaTeX}$ 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 $\text{\LaTeX}$ implementation the slope arguments ($\langle X,Y \rangle$) are restricted to integers in the range $-4 \leq X, Y \leq +4$, 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]:

These restrictions are eliminated by the `pict2e` package.

However, to avoid overflow of $\text{\TeX}$ ’s dimen arithmetic, the current implementation restricts the slope arguments to real numbers in the range $-1000 \leq X, Y \leq +1000$, which should be

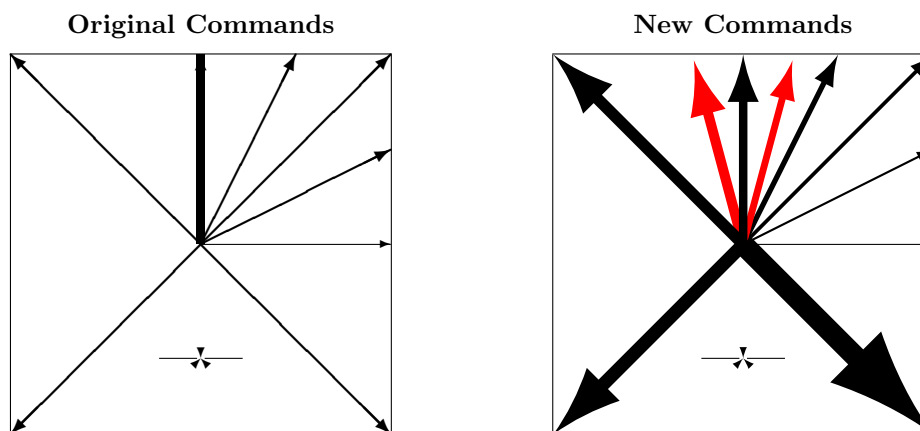


Figure 2: Vector

enough. It is usually not a good idea to use slope arguments with the absolute value less than 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 than 16384 cannot be obtained.

Furthermore, unlike the Standard $\text{\LaTeX}$ 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 $\text{\LaTeX}$ 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 $\text{\LaTeX}$. Slope arguments that are “illegal” in Standard $\text{\LaTeX}$ 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 $\text{\LaTeX}$, 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 $\text{\LaTeX}$ implementation (package option `ltarrows`, 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 $\text{\LaTeX}$ 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 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.

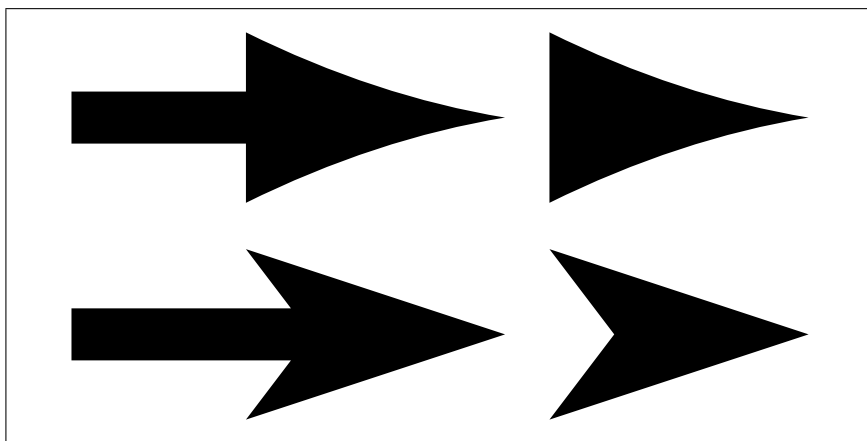


Figure 3: Vector: shape variants of the arrow-heads. Top: L^AT_EX style vectors. Bottom: PSTricks style vectors.

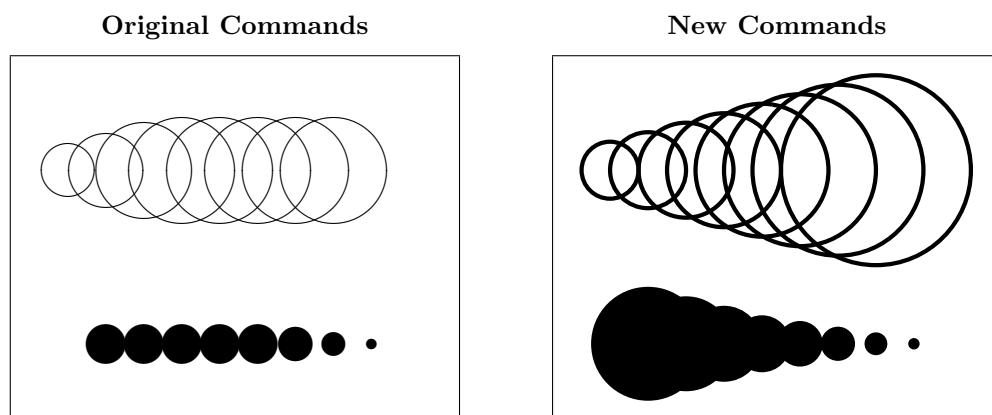


Figure 4: Circle and Dot

2.3.4 Oval

`\oval` `\oval[⟨rad⟩](⟨X,Y⟩)[⟨POS⟩]`

In the Standard L^AT_EX 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 L^AT_EX can draw without the `pict2e` package.

This default value is 20 pt, a length. However, in an early reimplementaion 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⟩]` 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 L^AT_EX mode”).

`\maxovalrad`

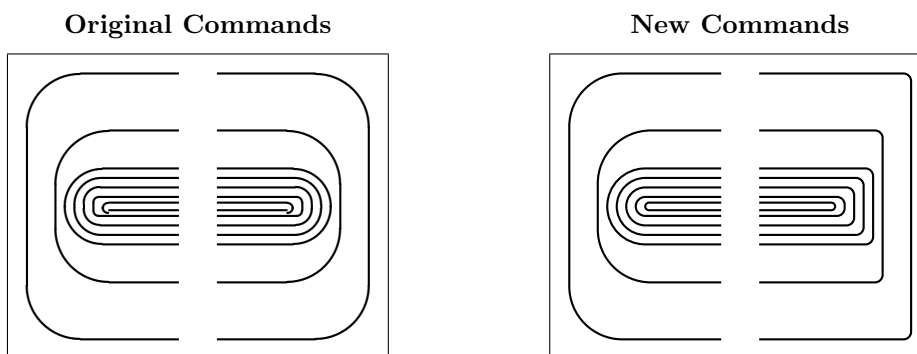


Figure 5: Oval: Radius argument for `\oval` vs. `\maxovalrad`

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 $\text{\LaTeX}$ 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 20pt as specified for the `[<rad>]` argument of `\oval` by the $\text{\LaTeX}$ 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 $\text{\LaTeX}$, 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 $\text{\LaTeX}$ versions prior to 2003/12/01, the quadratic Bezier curves plotted by this package will not match those of the Standard $\text{\LaTeX}$ 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 $\text{\LaTeX}$ 2.09.

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 describe 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 Lines, polygons

```
\Line \Line(⟨X1,Y1⟩)(⟨X2,Y2⟩)
\polyline \polyline(⟨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` 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 command `\Line`. The command `\polyline` draws a stroken line 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 defining paths you might use either `\strokepath` to draw them or, for closed paths, `\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) define the end as a line segment, `\roundcap` adds a halfdisc, `\squarecap` adds a halfsquare. `\roundcap` 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` (implicit) might be defined in such a way that “boundaries” of subpaths are prolonged until 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.

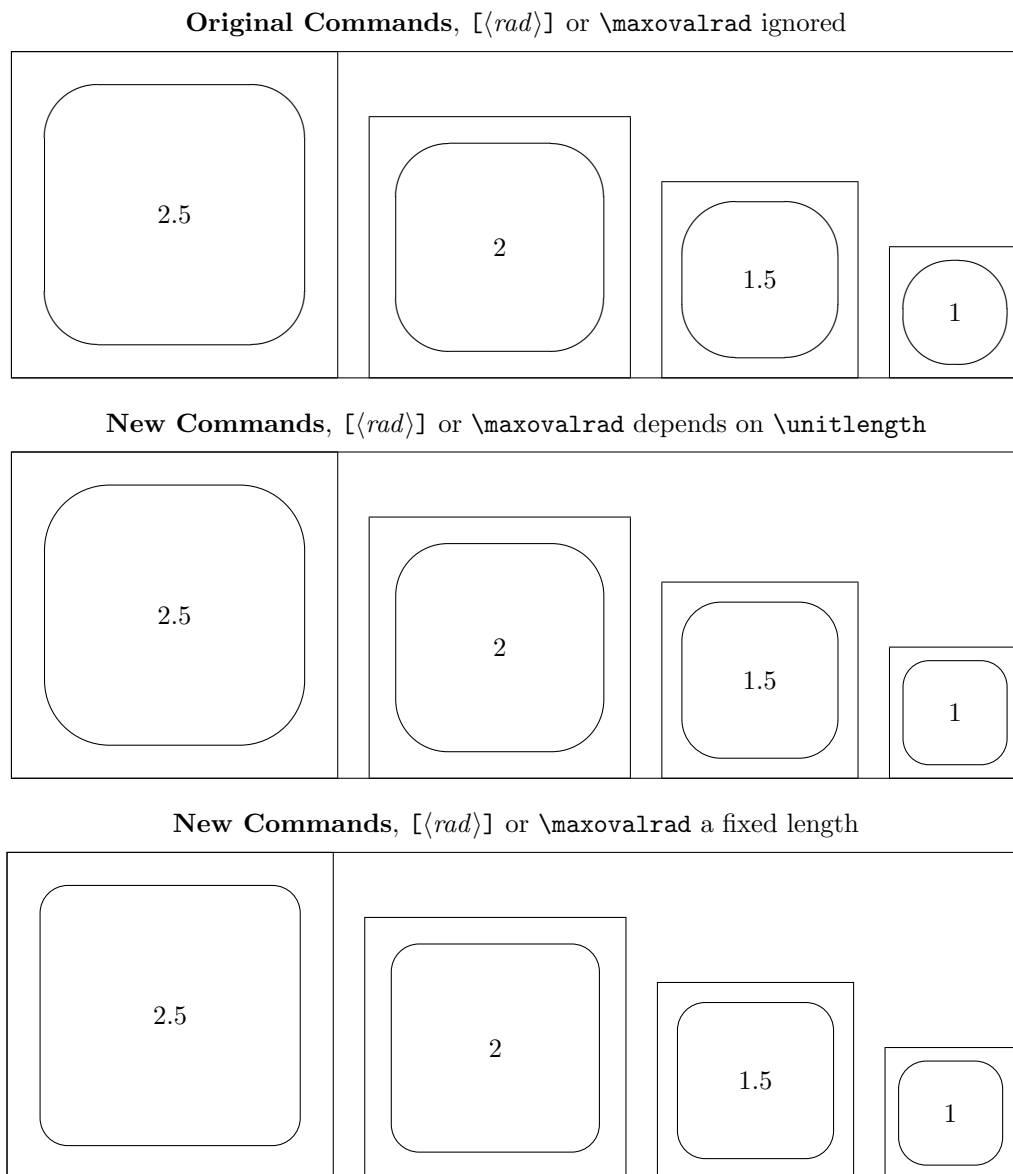


Figure 6: Oval: Radius argument for `\oval`: length vs. number. The number at the centre of each oval gives the relative value of `\unitlength`.

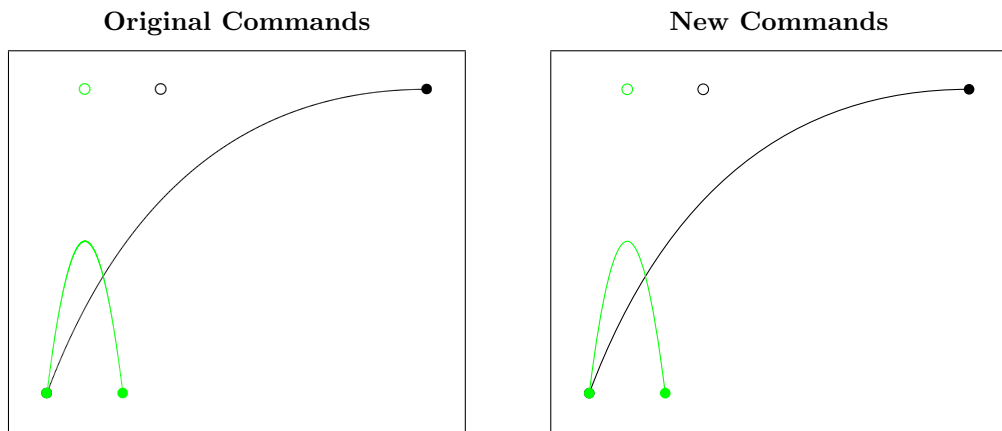


Figure 7: Quadratic Bezier curves

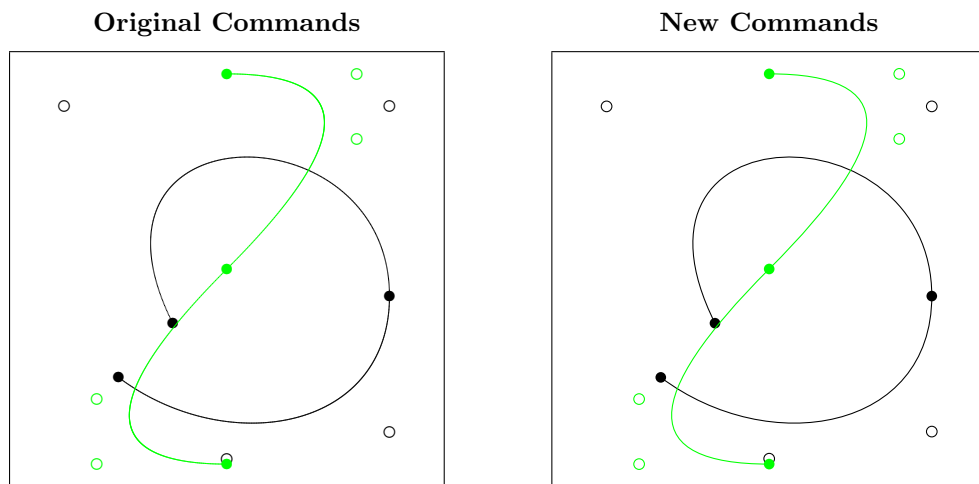


Figure 8: Cubic Bezier curves

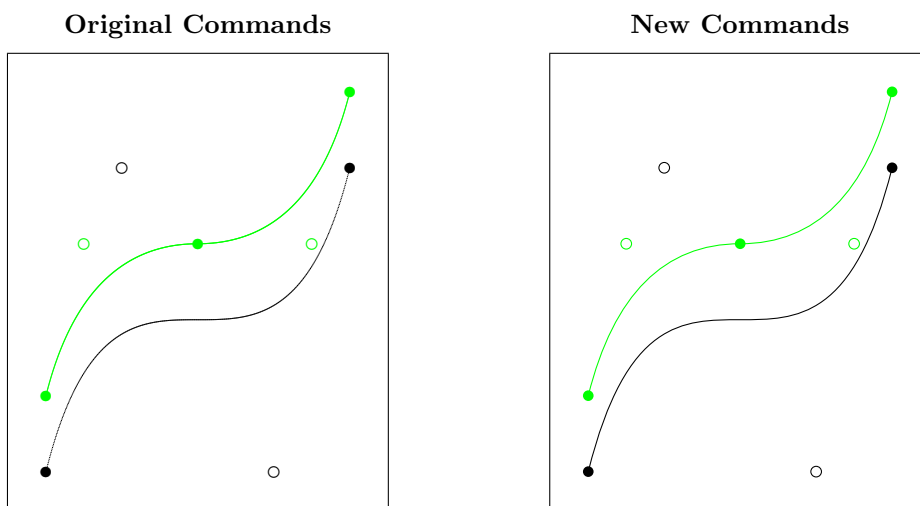


Figure 9: Quadratic (green) and Cubic Bezier curves

3 Implementation

Unlike other packages that have reimplemented or extended some of the commands from Standard L^AT_EX’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` 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.)

`\pIIE@code`
`\Gin@driver`

The latter command is well known from the `graphics` and `color` packages from the Standard L^AT_EX 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 t_EX and T_EXlive 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` 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.

`\pIIE@tempb`
`\pIIE@tempc`

```
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}}
```

```

21 \DeclareOption{dvipdf}{\def\Gin@driver{dvipdf.def}}
22 \DeclareOption{dvipdfm}{\def\Gin@driver{dvipdfm.def}}
23 \DeclareOption{dvipdfmx}{\def\Gin@driver{dvipdfmx.def}}
24 \DeclareOption{pdftex}{\def\Gin@driver{pdftex.def}}
25 \DeclareOption{luatex}{\def\Gin@driver{luatex.def}}
26 \DeclareOption{xetex}{\def\Gin@driver{xetex.def}}
27 \DeclareOption{dvipsone}{\def\Gin@driver{dvipsone.def}}
28 \DeclareOption{dviwindo}{\ExecuteOptions{dvipsone}}
29 \DeclareOption{oztex}{\ExecuteOptions{dvips}}
30 \DeclareOption{textures}{\def\Gin@driver{textures.def}}
31 \DeclareOption{pctexps}{\def\Gin@driver{pctexps.def}}
32 \DeclareOption{pctex32}{\def\Gin@driver{pctex32.def}}
33 \DeclareOption{vtex}{\def\Gin@driver{vtex.def}}

```

Request “original” L^AT_EX mode.

```

34 \DeclareOption{original}{\def\pIIE@mode{0}}

```

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

```

35 \newif\ifpIIE@pdfliteral@ok
36 \pIIE@pdfliteral@oktrue
37 \ifx\pIIE@pdfliteral\@undefined
38   \ifx\pdfliteral\@undefined
39     \pIIE@pdfliteral@okfalse
40     \def\pIIE@pdfliteral#1{%
41       \PackageWarning{pict2e}{pdfliteral not supported}%
42     }%
43   \else
44     \let\pIIE@pdfliteral\pdfliteral
45   \fi
46 \fi

```

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

```

47 \def\pIIE@buttcap{%
48   \ifpIIE@pdfliteral@ok
49     \buttcap
50   \fi
51 }

```

Arrow shape options. The values for L^AT_EX-style arrows are “hand optimized”; they should be regarded as experimental, i.e., they may change in future versions of this package. The values for PSTricks-style arrows are the default ones used by that bundle. If the `pstricks` package is actually loaded, then `pict2e` will obey the current values of the corresponding internal PSTricks parameters; this feature should be regarded as experimental, i.e., it may change in future versions of this package.

```

52 \DeclareOption{ltxarrows}{\AtEndOfPackage{%
53   \let\pIIE@vector=\pIIE@vector@ltx
54   \def\pIIE@FAL{1.52}%
55   \def\pIIE@FAW{3.2}%
56   \def\pIIE@CAW{1.5pt}%
57   \def\pIIE@FAI{0.25}%
58 }}
59 \DeclareOption{pstarrows}{\AtEndOfPackage{%
60   \let\pIIE@vector=\pIIE@vector@pst
61   \iffalse
62     \def\pIIE@FAL{1.4}%
63     \def\pIIE@FAW{2}%
64     \def\pIIE@CAW{1.5pt}%
65     \def\pIIE@FAI{0.4}%

```

```

66 \else % These are the ltxarrows values, which looks better. (RN)
67 \def\pIIE@FAL{1.52}%
68 \def\pIIE@FAW{3.2}%
69 \def\pIIE@CAW{1.5pt}%
70 \def\pIIE@FAI{0.25}%
71 \fi
72 }}

```

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

```

73 \newcommand*\pIIE@debug@comment{}
74 \DeclareOption{debug}{%
75 \def\pIIE@debug@comment{^^J^^J\@percentchar\space >>> pict2e <<<^^J}%
76 \begingroup
77 \ifundefined{pdfcompresslevel}{\global\pdfcompresslevel\z@}%
78 \endgroup}

```

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

```

79 \DeclareOption{hide}{\AtEndOfPackage{%
80 % \def\pIIE@code#1{%
81 \let\pIIE@code\@gobble
82 }}

```

Unknown options default to mode “original.”

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

By default, arrows are in the L^AT_EX style.

```
84 \ExecuteOptions{ltxarrows}
```

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

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

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

```
86 \ProcessOptions\relax
```

3.4 Output driver check

```

87 \ifnum\pIIE@mode=\z@
88 \PackageInfo{pict2e}{Package option ‘original’ requested}
89 \else

```

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

```

90 \if!\Gin@driver!
91 \PackageError{pict2e}
92 {No driver specified at all}
93 {You should make a default driver option in a file\MessageBreak
94 pict2e.cfg\MessageBreak eg: \protect\ExecuteOptions{dvips}}%
95 \else
96 \PackageInfo{pict2e}{Driver file: \Gin@driver}
97 \ifundefined{ver@\Gin@driver}{\input{\Gin@driver}}{}
98 \PackageInfo{pict2e}{Driver file for pict2e: p2e-\Gin@driver}
99 \InputIfFileExists{p2e-\Gin@driver}{}{%
100 \PackageError{pict2e}%
101 {Driver file ‘p2e-\Gin@driver’ not found}%
102 {Q: Is the file properly installed? A: No!}}
103 \fi
104 \fi

```

3.5 Mode check

For PostScript and PDF modes.

```

105 \ifnum\pIIE@mode>\z@
106   \ifnum\pIIE@mode<\thr@@
107     \RequirePackage{trig}

\pIIE@oldline Saved versions of some macros. (Or dummy definitions.)
\pIIE@oldsline 108   \let\pIIE@oldline\line
\pIIE@oldvector 109   \let\pIIE@oldsline\@sline
\pIIE@old@circle 110   \let\pIIE@oldvector\vector
\pIIE@old@dot 111   \let\pIIE@old@circle\@circle
\pIIE@old@bezier 112   \let\pIIE@old@dot\@dot
\pIIE@old@cbezier 113   \let\pIIE@old@bezier\@bezier
\pIIE@oldoval 114   \AtBeginDocument{%
\pIIE@oldoval 115     \@ifundefined{cbezier}{%
116       \def\pIIE@old@cbezier[#1](#2,#3)(#4,#5)(#6,#7)(#8,#9){}%
117       }\let\pIIE@old@cbezier\@cbezier}}
118   \let\pIIE@oldoval\oval
119   \let\pIIE@oldoval\@oval

\OriginalPictureCmds Switches back to the original definitions; for testing and demonstration purposes only.
120   \newcommand*\OriginalPictureCmds{%
121     \let\@sline\pIIE@oldsline
122     \let\line\pIIE@oldline
123     \let\vector\pIIE@oldvector
124     \let\@circle\pIIE@old@circle
125     \let\@dot\pIIE@old@dot
126     \let\@bezier\pIIE@old@bezier
127     \let\@cbezier\pIIE@old@cbezier
128     \renewcommand*\oval[1][\pIIE@oldoval]{%
129       \let\@oval\pIIE@oldoval
130     }

Overambitious drivers.
131   \else
132     \PackageError{pict2e}
133       {Unsupported mode (\pIIE@mode) specified}
134       {The driver you specified requested a mode\MessageBreak
135        not supported by this version of this package}
136   \fi

Incapable drivers.
137 \else
138   \ifnum\pIIE@mode<\z@
139     \PackageError{pict2e}
140       {No suitable driver specified}
141       {You should make a default driver option in a file\MessageBreak
142        pict2e.cfg\MessageBreak eg: \protect\ExecuteOptions{dvips}}
143   \fi
144 \fi

Big switch, completed near the end of the package (see page 35).
145 \ifnum\pIIE@mode>\z@

```

3.6 Graphics operators

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

```

146 \ifcase\pIIE@mode\relax

```

```

\pIIE@moveto@op PostScript
\pIIE@lineto@op 147 \or
\pIIE@setlinewidth@op 148 \newcommand*\pIIE@moveto@op{moveto}
\pIIE@stroke@op 149 \newcommand*\pIIE@lineto@op{lineto}
\pIIE@fill@op 150 \newcommand*\pIIE@setlinewidth@op{setlinewidth}
\pIIE@curveto@op 151 \newcommand*\pIIE@stroke@op{stroke}
\pIIE@concat@op 152 \newcommand*\pIIE@fill@op{fill}
\pIIE@closepath@op 153 \newcommand*\pIIE@curveto@op{curveto}
154 \newcommand*\pIIE@concat@op{concat}
155 \newcommand*\pIIE@closepath@op{closepath}

```

```

\pIIE@moveto@op PDF
\pIIE@lineto@op 156 \or
\pIIE@setlinewidth@op 157 \newcommand*\pIIE@moveto@op{m}
\pIIE@stroke@op 158 \newcommand*\pIIE@lineto@op{l}
\pIIE@fill@op 159 \newcommand*\pIIE@setlinewidth@op{w}
\pIIE@curveto@op 160 \newcommand*\pIIE@stroke@op{S}
\pIIE@concat@op 161 \newcommand*\pIIE@fill@op{f}
\pIIE@closepath@op 162 \newcommand*\pIIE@curveto@op{c}
163 \newcommand*\pIIE@concat@op{cm}
164 \newcommand*\pIIE@closepath@op{h}

```

(Currently, there are no other modes.)

```
165 \fi
```

3.7 Low-level operations

3.7.1 Collecting the graphics instructions and handling the output

`\pIIE@GRAPH` We collect all PostScript/PDF output code for a single picture object in a token register.

```

\pIIE@addtoGraph 166 \@ifdefinable\pIIE@GRAPH{\newtoks\pIIE@GRAPH}
167 \newcommand*\pIIE@addtoGraph[1]{%
168 \begingroup
169 \edef\x{\the\pIIE@GRAPH\space#1}%
170 \global\pIIE@GRAPH\expandafter{\x}%
171 \endgroup}

```

`\pIIE@fillGraph` The path will either be filled ...

```
172 \newcommand*\pIIE@fillGraph{\begingroup \@tempswatrue\pIIE@drawGraph}
```

`\pIIE@strokeGraph` ... or stroked.

```
173 \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.

```
174 \newcommand*\pIIE@drawGraph{%
175 \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.

```

176 \pIIE@scale@PTtoBP}%
177 \if@tempswa
178 \edef\y{\pIIE@fill@op}%
179 \else
180 \edef\x{\x\space
181 \strip@pt\@wholewidth\space\pIIE@setlinewidth@op
182 \pIIE@linecap\pIIE@linejoin\space}%
183 \edef\y{\pIIE@stroke@op}%

```

```

184      \fi
185      \expandafter\pIIE@code\expandafter{%
186      \expandafter\x\the\pIIE@GRAPH\space\y}%
Clear the graph and the current point after output.
187      \global\pIIE@GRAPH{}\xdef\pIIE@CPx{}\xdef\pIIE@CPy{%
188      \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` 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
`\pIIE@CPy` remember the “current point.” (Not strictly necessary for PostScript, but for some operations
`\pIIE@add@CP` in PDF, e.g., *rcurveto* emulation.)

```

189 \newcommand*\pIIE@CPx{} \newcommand*\pIIE@CPy{}
190 \newcommand*\pIIE@add@CP[2]{%
191   \begingroup
192   \@tempdima#1\xdef\pIIE@CPx{\the\@tempdima}%
193   \@tempdimb#2\xdef\pIIE@CPy{\the\@tempdimb}%
194   \pIIE@addtoGraph{\strip@pt\@tempdima\space\strip@pt\@tempdimb}%
195   \endgroup}

```

`\pIIE@add@nums` Similar, but does not set the “current point.” Values need not be coordinates (e.g., may be scaling factors, etc.).

```

196 \newcommand*\pIIE@add@nums[2]{%
197   \begingroup
198   \@tempdima#1\relax
199   \@tempdimb#2\relax
200   \pIIE@addtoGraph{\strip@pt\@tempdima\space\strip@pt\@tempdimb}%
201   \endgroup}

```

`\pIIE@add@num` Likewise, for a single argument.

```

202 \newcommand*\pIIE@add@num[1]{%
203   \begingroup
204   \@tempdima#1\relax
205   \pIIE@addtoGraph{\strip@pt\@tempdima}%
206   \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.)

```

207 \newcommand*\pIIE@PTtoBP{0.99626401 }
208 \ifcase\pIIE@mode\relax

```

`\pIIE@concat` PostScript: Use some operators directly.

```

\pIIE@translate 209 \or
\pIIE@rotate 210 \newcommand*\pIIE@concat[6]{%
\pIIE@scale 211 \begingroup
\pIIE@scale@PTtoBP

```

```

212     \pIle@addtoGraph{[]}%
213     \@tempdima#1\relax \@tempdimb#2\relax
214     \pIle@add@nums\@tempdima\@tempdimb
215     \@tempdima#3\relax \@tempdimb#4\relax
216     \pIle@add@nums\@tempdima\@tempdimb
217     \@tempdima#5\relax \@tempdimb#6\relax
218     \pIle@add@nums\@tempdima\@tempdimb
219     \pIle@addtoGraph[] \pIle@concat@op}%
220   \endgroup}
221   \newcommand*\pIle@translate[2]{\pIle@add@nums{#1}{#2}\pIle@addtoGraph{translate}}
222   \newcommand*\pIle@rotate[1]{\pIle@add@num{#1}\pIle@addtoGraph{rotate}}
223   \newcommand*\pIle@scale[2]{\pIle@add@nums{#1}{#2}\pIle@addtoGraph{scale}}
224   \newcommand*\pIle@scale@PTtoBP{\pIle@PTtoBP \pIle@PTtoBP scale}

\pIle@concat PDF: Emulate. :-(
\pIle@translate 225 \or
\pIle@rotate 226 \newcommand*\pIle@concat[6]{%
\pIle@scale 227 \begingroup
\pIle@scale@PTtoBP 228 \@tempdima#1\relax \@tempdimb#2\relax
229 \pIle@add@nums\@tempdima\@tempdimb
230 \@tempdima#3\relax \@tempdimb#4\relax
231 \pIle@add@nums\@tempdima\@tempdimb
232 \@tempdima#5\relax \@tempdimb#6\relax
233 \pIle@add@nums\@tempdima\@tempdimb
234 \pIle@addtoGraph\pIle@concat@op
235 \endgroup}
236 \newcommand*\pIle@translate[2]{\pIle@concat\p@z@z@p@{#1}{#2}}
237 \newcommand*\pIle@rotate[1]{%
238 \begingroup
239 \@tempdima#1\relax
240 \edef\pIle@tempa{\strip@pt\@tempdima}%
241 \CalculateSin\pIle@tempa
242 \CalculateCos\pIle@tempa
243 \edef\pIle@tempb{\UseSin\pIle@tempa}%
244 \edef\pIle@tempc{\UseCos\pIle@tempa}%
245 \pIle@concat{\pIle@tempc\p@}{\pIle@tempb\p@}%
246 {-\pIle@tempb\p@}{\pIle@tempc\p@}\z@z@
247 \endgroup}
248 \newcommand*\pIle@scale[2]{\pIle@concat{#1}\z@z@{#2}\z@z@}
249 \newcommand*\pIle@scale@PTtoBP{\pIle@PTtoBP 0 0 \pIle@PTtoBP 0 0 \pIle@concat@op}

(Currently, there are no other modes.)
250 \fi

```

3.8.2 Path definitions

\pIle@moveto Simple things ...

```

251 \newcommand*\pIle@moveto[2]{%
252 \pIle@add@CP{#1}{#2}\pIle@addtoGraph\pIle@moveto@op}

```

\pIle@lineto ... have to be defined, too.

```

253 \newcommand*\pIle@lineto[2]{%
254 \pIle@add@CP{#1}{#2}\pIle@addtoGraph\pIle@lineto@op}

```

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

```

255 \ifcase\pIle@mode\relax

```

`\pIe@rcurveto` PostScript: Use the “rcurveto” operator directly.

```

256 \or
257 \newcommand*\pIe@rcurveto[6]{%
258 \begingroup
259 \@tempdima#1\relax \@tempdimb#2\relax
260 \pIe@add@nums\@tempdima\@tempdimb
261 \@tempdima#3\relax \@tempdimb#4\relax
262 \pIe@add@nums\@tempdima\@tempdimb
263 \@tempdima#5\relax \@tempdimb#6\relax
264 \pIe@add@CP\@tempdima\@tempdimb
265 \pIe@addtoGraph{rcurveto}%
266 \endgroup}

```

`\pIe@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 `\pIe@CPx` and `\pIe@CPy`.

```

267 \or
268 \newcommand*\pIe@rcurveto[6]{%
269 \begingroup
270 \@tempdima#1\advance\@tempdima\pIe@CPx\relax
271 \@tempdimb#2\advance\@tempdimb\pIe@CPy\relax
272 \pIe@add@nums\@tempdima\@tempdimb
273 \@tempdima#3\advance\@tempdima\pIe@CPx\relax
274 \@tempdimb#4\advance\@tempdimb\pIe@CPy\relax
275 \pIe@add@nums\@tempdima\@tempdimb
276 \@tempdima#5\advance\@tempdima\pIe@CPx\relax
277 \@tempdimb#6\advance\@tempdimb\pIe@CPy\relax
278 \pIe@add@CP\@tempdima\@tempdimb
279 \pIe@addtoGraph\pIe@rcurveto@op
280 \endgroup}

```

(Currently, there are no other modes.)

```

281 \fi

```

`\pIe@curveto` This is currently only used for Bezier curves and for drawing the heads of L^AT_EX-like arrows. Note: It’s the same for PostScript and PDF.

```

282 \newcommand*\pIe@curveto[6]{%
283 \begingroup
284 \@tempdima#1\relax \@tempdimb#2\relax
285 \pIe@add@nums\@tempdima\@tempdimb
286 \@tempdima#3\relax \@tempdimb#4\relax
287 \pIe@add@nums\@tempdima\@tempdimb
288 \@tempdima#5\relax \@tempdimb#6\relax
289 \pIe@add@CP\@tempdima\@tempdimb
290 \pIe@addtoGraph\pIe@curveto@op
291 \endgroup}

```

`\pIe@closepath`

```

292 \newcommand*\pIe@closepath{\pIe@addtoGraph\pIe@closepath@op}

```

3.9 “Pythagorean Addition” and Division

`\pIe@pyth` This algorithm is copied from the P_JCT_EX 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 $\tau = 2(t - 1/2)$.

```

293 \newcommand*\pIe@pyth[3]{%
294 \begingroup
295 \@tempdima=#1\relax
\@tempdima = abs(x)
296 \ifnum\@tempdima<\z@\@tempdima=-\@tempdima\fi
297 \@tempdimb=#2\relax
\@tempdimb = abs(y)
298 \ifnum\@tempdimb<\z@\@tempdimb=-\@tempdimb\fi
\@tempdimb = s = abs(x) + abs(y)
299 \advance\@tempdimb\@tempdima
300 \ifnum\@tempdimb=\z@
\@tempdimc = z =  $\sqrt{x^2 + y^2}$ 
301 \@tempdimc=\z@
302 \else
\@tempdima =  $8 \times \text{abs}(x)$ 
303 \multiply\@tempdima 8\relax
\@tempdimc =  $8t = 8 \times \text{abs}(x)/s$ 
304 \pIe@divide\@tempdima\@tempdimb\@tempdimc
\@tempdimc =  $4\tau = (8t - 4)$ 
305 \advance\@tempdimc -4pt
306 \multiply\@tempdimc 2
307 \edef\pIe@tempa{\strip@pt\@tempdimc}%
\@tempdima =  $(8\tau)^2$ 
308 \@tempdima=\pIe@tempa\@tempdimc
\@tempdima =  $[64 + (8\tau)^2]/2 = (8f)^2$ 
309 \advance\@tempdima 64pt
310 \divide\@tempdima 2\relax
initial guess at  $\sqrt{u}$ 
311 \@dashdim=7pt
\@dashdim =  $\sqrt{u}$ 
312 \pIe@@pyth\pIe@@pyth\pIe@@pyth
313 \edef\pIe@tempa{\strip@pt\@dashdim}%
314 \@tempdimc=\pIe@tempa\@tempdimb
\@tempdimc =  $z = (8f) \times s/8$ 
315 \global\divide\@tempdimc 8
316 \fi
317 \edef\x{\endgroup#3=\the\@tempdimc}%
318 \x}

```

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

```

319 \newcommand*\pIe@@pyth{%
320 \pIe@divide\@tempdima\@dashdim\@tempdimc
321 \advance\@dashdim\@tempdimc
322 \divide\@dashdim\tw@}

```

\pIe@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.

```

323 \newcommand*\pIe@divide[3]{%

```

All definitions inside a group.

```

324 \begingroup
325 \dimendef\Numer=254\relax \dimendef\Denom=252\relax
326 \countdef\Num=254\relax \countdef\Den=252\relax
327 \countdef\I=250\relax \countdef\Numb=248\relax
328 \Numer #1\relax \Denom #2\relax

```

Make numerator and denominator nonnegative, save sign.

```

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

```

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

```

331 \ifdim\Denom=\z@
332 \edef\Q{\strip@pt\maxdimen}%
333 \PackageWarning{pict2e}%
334 {Division by 0, \sign\strip@pt\maxdimen\space used}{}%
335 \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.

```

336 \Num=\Numer \Den=\Denom
337 \Numb=\Num \divide\Numb\Den
338 \ifnum\Numb>16383
339 \edef\Q{\strip@pt\maxdimen}%
340 \PackageWarning{pict2e}%
341 {Division overflow, \sign\strip@pt\maxdimen\space used}{}%
342 \else
343 \edef\Q{\number\Numb}%
344 \multiply \Numb\Den \advance\Num -\Numb
345 \I=6\relax
346 \@whilenum \I>\z@ \do{\pIe@@divide\advance\I\m@ne}%
347 \fi
348 \fi

```

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

```

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

```

`\pIe@@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).

```

351 \def\pIe@@divide{%

```

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

```

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

```

Find the next digit of the decimal expression.

```

353 \multiply \Num 10
354 \Numb=\Num \divide\Numb\Den
355 \edef\Q{\Q\number\Numb}%

```

Find the remainder.

```

356 \multiply \Numb\Den \advance \Num -\Numb

```

Stop the iteration if the remainder is zero.

```

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

```

3.10 High-level operations

`\pIe@checkslopeargs` Common code for `\line` and `\vector`.

```

358 \newcommand*\pIe@checkslopeargsline[2]{%
359 \pIe@checkslopeargs{#1}{#2}{16383}}
360 \newcommand*\pIe@checkslopeargsvector[2]{%
361 \pIe@checkslopeargs{#1}{#2}{1000}}
362 \newcommand*\pIe@checkslopeargs[3]{%
363 \def\@tempa{#1}\expandafter\pIe@checkslopearg\@tempa.:{#3}%
364 \def\@tempa{#2}\expandafter\pIe@checkslopearg\@tempa.:{#3}%

```

A bit incompatible with Standard L^AT_EX: slope (0,0) raises an error.

```

365 \ifdim #1\p@=\z@ \ifdim #2\p@=\z@ \@badlinearg \fi\fi}
366 \def\pIe@checkslopearg #1.#2:#3{%
367 \def\@tempa{#1}%
368 \ifx\@tempa\empty\def\@tempa{0}\fi
369 \ifx\@tempa\space\def\@tempa{0}\fi
370 \ifnum\ifnum\@tempa<\z@-\fi\@tempa>#3\@badlinearg \fi}
371 \def\@badlinearg{\PackageError
372 {pict2e}{Bad \protect\line\space or \protect\vector\space argument}{}}

```

3.10.1 Line

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

```

373 \def\line{#1,#2}#3{%
374 \pIe@checkslopeargsline{#1}{#2}%
375 \@tempdima=#1pt\relax \@tempdimb=#2pt\relax
376 \@linelen #3\unitlength
377 \ifdim \@linelen<\z@ \@badlinearg \else \@sline \fi}

```

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

```

378 \def\@sline{%
379 \begingroup
380 \ifdim \@tempdima=\z@
381 \ifdim \@tempdimb<\z@\@linelen-\@linelen\fi
382 \@ydim=\@linelen
383 \@xdim=\z@
384 \else
385 \ifdim \@tempdimb=\z@
386 \ifdim \@tempdima<\z@\@linelen-\@linelen\fi
387 \@xdim=\@linelen
388 \@ydim=\z@
389 \else
390 \ifnum \@tempdima<\z@\@linelen-\@linelen\fi
391 \pIe@divide\@tempdimb\@tempdima\dimen@
392 \@ydim=\strip@pt\dimen@\@linelen
393 \@xdim=\@linelen
394 \fi
395 \fi
396 \pIe@moveto\z@\z@
397 \pIe@lineto\@xdim\@ydim
398 \pIe@strokeGraph
399 \endgroup}

```

3.10.2 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.

```

400 \def\vector(#1,#2)#3{%
401   \begingroup
402   \pIIE@checkslopeargsvector{#1}{#2}%
403   \@tempdima=#1pt\relax \@tempdimb=#2pt\relax
404   \@linelen#3\unitlength
405   \ifdim\@linelen<\z@ \badlinearg \else
406     \pIIE@pyth{\@tempdima}{\@tempdimb}\dimen@
407     \ifdim\@tempdima=\z@
408       \else\ifdim\@tempdimb=\z@
409         \else

```

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

```

410       \pIIE@divide\dimen@{\@tempdima}\@xdim
411       \@linelen\strip@pt\@xdim\@linelen
412       \ifdim\@linelen<\z@\@linelen-\@linelen\fi
413     \fi
414   \fi

```

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

```

415   \pIIE@divide{\@tempdimb}\dimen@\@ydim
416   \pIIE@divide{\@tempdima}\dimen@\@xdim

```

Rotate the following vector/arrow outlines by angle θ .

```

417   \pIIE@concat\@xdim\@ydim{-\@ydim}\@xdim\z@\z@

```

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

```

418   \pIIE@vector
419   \pIIE@fillGraph
420   \fi
421   \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`.

```

422   \newcommand*\pIIE@vector{}

```

`\pIIE@FAL` Some macros to parametrize the shape of the vector outline. See Figures 10 and 11.

```

\pIIE@FAW 423 \newcommand*\pIIE@FAL{}\newcommand*\pIIE@FAW{}\newcommand*\pIIE@CAW{}
\pIIE@CAW 424 \newcommand*\pIIE@FAI{}
\pIIE@FAI 425 \newcommand*\pIIE@@firstnum{}\newcommand*\pIIE@@secondnum{}
\pIIE@@firstnum 426 \iffalse% the pstricks values gives too small arrows. (RN)
\pIIE@@secondnum 427 \AtBeginDocument{%
428   \ifpackageloaded{pstricks}{%
429     \def\pIIE@FAL{\psk@arrowlength}%
430     \def\pIIE@FAW{\expandafter\pIIE@@secondnum\psk@arrowsize}%
431     \def\pIIE@CAW{\expandafter\pIIE@@firstnum\psk@arrowsize}%
432     \def\pIIE@FAI{\psk@arrowinset}%
433     \def\pIIE@@firstnum#1 #2 {\#1\p@}%
434     \def\pIIE@@secondnum#1 #2 {\#2}%
435   }{}%
436 }
437 \fi

```

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

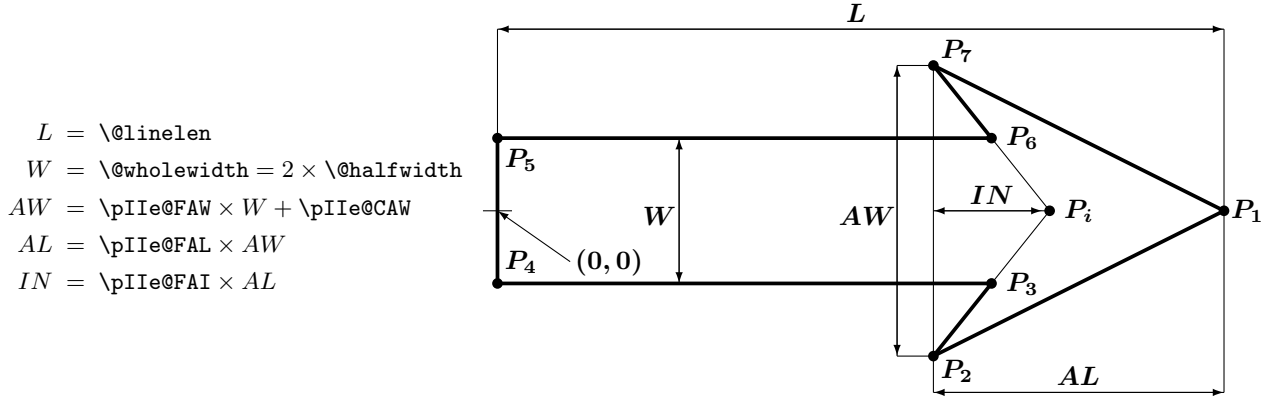


Figure 11: Sketch of the path drawn by the PSTricks-like implementation of `\vector`. (Note: We are using the redefined macros of `pict2e!`)

$P_c \quad P_d \quad P_1$
459 `\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.

```

460 \newcommand*\pIIE@vector@pst{%
461   \@ydim\pIIE@FAW\@wholewidth \advance\@ydim\pIIE@CAW\relax
462   \@ovxx\pIIE@FAL\@ydim
463   \@xdim\@linelen \advance\@xdim-\@ovxx
464   \divide\@ydim\tw@
465   \@ovyy\@ydim \advance\@ovyy-\@halfwidth
466   \@ovdx\pIIE@FAI\@ovxx
467   \pIIE@divide\@ovdx\@ydim\@tempdimc
468   \@ovxx\strip@pt\@ovyy\@tempdimc
469   \advance\@ovxx\@xdim
470   \advance\@ovdx\@xdim
471   \pIIE@moveto\@linelen\z@
472   \pIIE@lineto\@xdim{-\@ydim}%
473   \ifdim\@xdim>\z@
474     \pIIE@lineto\@ovxx{-\@halfwidth}%
475     \pIIE@lineto\z@{-\@halfwidth}%
476     \pIIE@lineto\z@\@halfwidth}%
477   \pIIE@lineto\@ovxx\@halfwidth}%
478   \else
479     \pIIE@lineto\@ovdx\z@
480   \fi

```

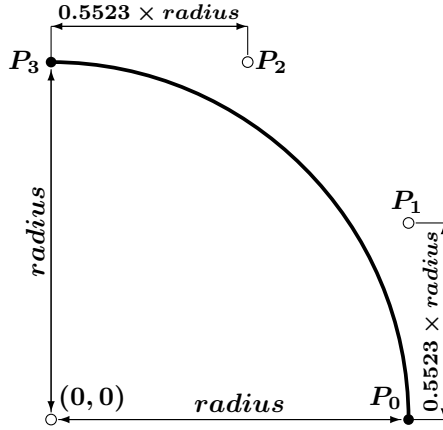


Figure 12: Sketch of the quarter circle path drawn by `\pIe@qcircle` (NE quarter)

```

P7
481 \pIe@lineto\xdim\ydim
P1
482 \pIe@lineto\@linelen\z@}

```

3.10.3 Circle and Dot

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

```
483 \def\@circle#1{\begingroup \@tempwafalse\pIe@circ{#1}}
```

`\@dot` ... or filled.

```
484 \def\@dot#1{\begingroup \@tempwattrue\pIe@circ{#1}}
```

`\pIe@circ` Common code.

```
485 \newcommand*\pIe@circ[1]{%
```

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

```

486 \@tempdima#1\unitlength
487 \ifdim\@tempdima<\z@ \pIe@badcircarg \fi
488 \divide\@tempdima\tw@
489 \pIe@circle\@tempdima

```

With the current state of affairs, we could use `\pIe@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.)

```

490 \if@tempswa \pIe@fillGraph \else \buttcap \pIe@strokeGraph \fi
491 \endgroup}

```

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

```

492 \newcommand*\pIe@circle[1]{%
493 \pIe@qcircle[1]\z@{#1}\pIe@qcircle \@ne{#1}%
494 \pIe@qcircle \tw@{#1}\pIe@qcircle\thr@@{#1}\pIe@closepath}

```

`\pIe@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!)

```

495 \newcommand*\pIIe@qcircle[3][0]{%
496   \begingroup
497     \@ovro#3\relax \@ovri0.55228474983\@ovro
498     \@tempdimc\@ovri \advance\@tempdimc-\@ovro
499     \ifnum#1>\z@ \@tempswatrue \else \@tempswafalse \fi
500     \ifcase#2\relax
        NE
501     \pIIe@qcircle\@ovro\z@\z@\@ovri\@tempdimc\@ovro{-\@ovro}\@ovro
502     \or
        NW
503     \pIIe@qcircle\z@\@ovro{-\@ovri}\z@{-\@ovro}\@tempdimc{-\@ovro}\@ovro%
504     \or
        SW
505     \pIIe@qcircle{-\@ovro}\z@\z@{-\@ovri}\@tempdimc{-\@ovro}\@ovro{-\@ovro}%
506     \or
        SE
507     \pIIe@qcircle\z@{-\@ovro}\@ovri\z@\@ovro{-\@tempdimc}\@ovro\@ovro
508     \fi
509   \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.

```

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

```

`\pIIe@badcircarg` Obvious cousin to `\@badlinearg` from the L^AT_EX kernel.

```

512 \newcommand*\pIIe@badcircarg{%
513   \PackageError{pict2e}%
514     {Illegal argument in \protect\circle(*), \protect\oval, \protect\arc(*) or
515     \protect\circlearc.}%
516     {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 L^AT_EX manual [1, p. 223].

```

517 \newcommand*\maxovalrad{20pt}

```

`\pIIe@defaultUL` 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`.

```

518 \newcommand*\pIIe@defaultUL[2]{%
519   \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!?)

```

520 \newcommand*\pIIe@def@UL{}
521 \def\pIIe@def@UL#1\relax#2#3{%
522 %   \if!#1!%

```

```

523 %      \def#2{#3}% \edef ?
524 %      \else
525 %      \edef#2{\strip@pt\dimen@}%
526 %      \fi
527      \edef#2{\the\dimen@}}

```

`\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 L^AT_EX, we check for negative or zero radius argument. `\pIe@maxovalrad` is the internal variant of `\maxovalrad`.

```

528 \newcommand*\pIe@maxovalrad{}
529 \renewcommand*\oval[1][\maxovalrad]{%
530   \begingroup \pIe@defaultUL\pIe@maxovalrad{#1}%
531   \ifdim\pIe@maxovalrad<\z@ \pIe@badcircarg \fi

```

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

```

532   \pIe@oldoval}

```

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

```

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

```

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

```

534 \@ovxx#1\unitlength \divide\@ovxx\tw@
535 \@ovyy#2\unitlength \divide\@ovyy\tw@
536 \@tempdimc \ifdim\@ovyy>\@ovxx \@ovxx \else \@ovyy \fi
537 \ifdim\pIe@maxovalrad<\@tempdimc \@tempdimc\pIe@maxovalrad\relax \fi

```

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

```

538 \@xdim\@ovxx \advance\@xdim-\@tempdimc
539 \@ydim\@ovyy \advance\@ydim-\@tempdimc

```

Determine which parts of the oval we have to draw.

```

540 \pIe@get@quadrants{#3}%

```

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

```

541 \ifnum15=\@tempcnta \pIe@buttcap \fi

```

“@tempswa = false” means, that we have to suppress the ‘moveto’ in the following quadrant.

```

542 \@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)

```

543 \ifnum9=\@tempcnta
544 \pIe@qoval\z@{-\@ovyy}{\@xdim}{-\@ovyy}\thr@@\@tempdimc\@ovxx\z@

```

Bit 0 set! (NE)

```

545 \@tempcnta\@ne
546 \fi

```

Bit 0 set? (NE)

```

547 \pIe@qoval\@ovxx\z@\@ovxx\@ydim\z@\@tempdimc\z@\@ovyy

```

Bit 1 set? (NW)

```

548 \pIe@qoval\z@\@ovyy{-\@xdim}\@ovyy\@ne\@tempdimc{-\@ovxx}\z@

```

Bit 2 set? (SW)

```

549 \pIe@qoval{-\@ovxx}\z@{-\@ovxx}{-\@ydim}\tw@\@tempdimc\z@{-\@ovyy}%

```

Bit 3 set? (SE)

```

550 \pIe@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.

```
551 \pIIE@strokeGraph
552 \endgroup}
```

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

```
553 \newcommand*\pIIE@qoval[8]{%
554 % \end{macrocode}
555 % Bit set?
556 % \begin{macrocode}
557 \ifodd\@tempcnta
558 \if@tempswa\pIIE@moveto{#1}{#2}\fi
559 \pIIE@lineto{#3}{#4}\pIIE@qcircle{#5}{#6}\pIIE@lineto{#7}{#8}%
560 \@tempswafalse
561 \else
562 \@tempswatrue
563 \fi
```

Shift by one bit.

```
564 \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 L^AT_EX kernel.) We abuse \@setfpsbit from the float processing modules of the kernel.

```
565 \newcommand*\pIIE@get@quadrants[1]{%
566 \@ovttrue \@ovbtrue \@ovltrue \@ovrtrue \@tempcnta\z@
567 \tfor\reserved@a:=#1\do{\csname @ov\reserved@a false\endcsname}%
568 \if@ovr \if@ovb \@setfpsbit2\fi \if@ovt \@setfpsbit4\fi \fi
569 \if@ovl \if@ovb \@setfpsbit1\fi \if@ovt \@setfpsbit8\fi \fi}
570 % \end{macrocode}
571 % \end{macro}
572 %
573 % \subsubsection{Quadratic Bezier Curve}
574 % \label{sec:implementation:bezier-curves}
575 %
576 % \begin{macro}{\@bezier}
577 % \changes{v0.1u}{2003/11/21}{Change calculation of cubic bezier parameters
578 % to use less tokens (HjG)}
579 % \changes{v0.2o}{2004/06/25}
580 % {Supply \cmd{\ignorespaces} to match kernel version (HjG)}
581 % \changes{v0.2p}{2004/07/27}{\cmd{\@killglue} added. (RN)}
582 %
583 % If \#1=0 the primitive operators of the (back-end) format are used.
584 % The kernel version of \cmd{\@bezier} uses \cmd{\put} internally,
585 % which features \cmd{\@killglue} and \cmd{\ignorespaces} commands
586 % in turn (at the beginning and end, respectively).
587 % Since we don't use \cmd{\put}, we have to add the latter commands
588 % by hand.
589 % \begin{macrocode}
590 \def\@bezier#1(#2,#3)(#4,#5)(#6,#7){%
591 \ifnum #1=\z@

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

592 \@killglue
593 \begingroup
594 \ovxx#2\unitlength \ovyy#3\unitlength
595 \ovdx#4\unitlength \ovdy#5\unitlength
```

```

596      \@xdim#6\unitlength \@ydim#7\unitlength
          
$$P_1 = P_m + 1/3(P_0 - P_m)$$

597      \pIe@bezier@QtoC\@ovxx\@ovdx\@ovro
598      \pIe@bezier@QtoC\@ovyy\@ovdy\@ovri
          
$$P_2 = P_m + 1/3(P_3 - P_m)$$

599      \pIe@bezier@QtoC\@xdim\@ovdx\@clnwd
600      \pIe@bezier@QtoC\@ydim\@ovdy\@clnht
          
$$(P_{0x}, P_{0y})$$

601      \pIe@moveto\@ovxx\@ovyy
          
$$(P_{1x}, P_{1y}) \quad (P_{2x}, P_{2y}) \quad (P_{3x}, P_{3y})$$

602      \pIe@curveto\@ovro\@ovri\@clnwd\@clnht\@xdim\@ydim
603      \pIe@strokeGraph
604      \endgroup
605      \ignorespaces
606      \else
607      \pIe@old@bezier{#1}{#2,#3}{#4,#5}{#6,#7}
608      \fi}

```

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

```

609      \newcommand*\pIe@bezier@QtoC[3]{%
610      \@tempdimc#1\relax      \advance\@tempdimc-#2\relax
611      \divide\@tempdimc\thr@@ \advance\@tempdimc #2\relax
612      #3\@tempdimc}

```

3.10.5 Circle arcs

We need some auxiliary dimensions.

```

613      \ifx\undefined\@arclen \newdimen\@arclen \fi
614      \ifx\undefined\@arcrad \newdimen\@arcrad \fi
615      \ifx\undefined\@tempdimd \newdimen\@tempdimd \fi

```

`\pIe@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 $[-720, 720]$.

```

616      \newcommand*\pIe@arc[6][0]{%
617      \@arcrad #4\relax
618      \ifdim \@arcrad<\z@ \pIe@badcircarg \else
619      \@arclen #6\p@ \advance\@arclen -#5\p@
620      \ifdim \@arclen<\z@ \def\sign{-}\else\def\sign{}\fi
621      \ifdim \sign\@arclen>720\p@
622      \PackageWarning {pict2e}{The arc angle is reduced to -720..720}%
623      \@whiledim \sign\@arclen>720\p@ \do {\advance\@arclen-\sign360\p@}%
624      \@tempdima #5\p@ \advance\@tempdima \@arclen
625      \edef\@angleend{\strip@pt\@tempdima}%
626      \pIe@arc{#1}{#2}{#3}{#4}{#5}{\@angleend}%
627      \else
628      \pIe@arc{#1}{#2}{#3}{#4}{#5}{#6}%
629      \fi
630      \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.

```

631 \newcommand*\pIIE@@arc[6]{%
632 \begingroup
633 \ifdim \sign\@arclen>90\p@
634 \divide\@arclen 2
635 \@tempdima #5\p@ \advance\@tempdima \@arclen
636 \edef\@anglemid{\strip@pt\@tempdima}%
637 \def\@temp{\pIIE@@arc{#1}{#2}{#3}{#4}{#5}}%
638 \expandafter\@temp\expandafter{\@anglemid}%
639 \def\@temp{\pIIE@@arc{2}{#2}{#3}{#4}}%
640 \expandafter\@temp\expandafter{\@anglemid}{#6}%
641 \else

```

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

```

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

```

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

```

645 \ifcase #1\relax
646 \pIIE@lineto\@tempdima\@tempdimb
647 \or \pIIE@moveto\@tempdima\@tempdimb
648 \or
649 \else \PackageWarning {pict2e}%
650 {Illegal obligatory argument in \protect\circlearc.}%
651 \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).

```

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

```

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

```

655 \advance\@tempdima-\UseSin{#5}\@linelen
656 \advance\@tempdimb \UseCos{#5}\@linelen
657 \pIIE@add@nums\@tempdima\@tempdimb

```

Coordinates of the endpoint:

```

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

```

Coordinates of the second control point:

```

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

```

Adding the second control point and the endpoint to the path

```

663 \pIIE@add@nums\@tempdimc\@tempdimd
664 \pIIE@add@CP\@tempdima\@tempdimb
665 \pIIE@addtoGraph\pIIE@curveto@op
666 \fi
667 \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.

```

668 \newcommand*\pIIEarc
669 {\ifstar{\@tempswatrue\pIIE@arc@}{\@tempswafalse\pIIE@arc@}}

```

```

670 \newcommand*\pIIE@arc@[2][0,360]{\pIIE@arc@@(#1){#2}}
671 \def\pIIE@arc@@(#1,#2)#3{%
672   \if@tempswa
673     \pIIE@moveto\z@\z@
674     \pIIE@arc{\z@}{\z@}{#3\unitlength}{#1}{#2}%
675     \pIIE@closepath\pIIE@fillGraph
676   \else
677     \pIIE@arc[1]{\z@}{\z@}{#3\unitlength}{#1}{#2}%
678     \pIIE@strokeGraph
679   \fi}
680 \ifx\undefined\arc
681 \else
682   \PackageWarning{pict2e}{\protect\arc\space redefined}%
683 \fi
684 \let\arc\pIIEarc

```

3.10.6 Lines and polygons

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

```

\polyline 685 \let\lp@r( \let\rp@r)
\polygon 686 \def\Line(#1,#2)(#3,#4){\polyline(#1,#2)(#3,#4)}
687 \def\polyline(#1,#2){%
688   \@killglue
689   \pIIE@moveto{#1\unitlength}{#2\unitlength}%
690   \@ifnextchar\lp@r{\@polyline}{\PackageWarning{pict2e}%
691     {Polygonal lines require at least two vertices!}%
692   \ignorespaces}}
693 \def\@polyline(#1,#2){%
694   \pIIE@lineto{#1\unitlength}{#2\unitlength}%
695   \@ifnextchar\lp@r{\@polyline}{\pIIE@strokeGraph\ignorespaces}}
696 \def\polygon{%
697   \@killglue
698   \@ifstar{\begingroup\@tempswatrue\@polygon}%
699   {\begingroup\@tempswafalse\@polygon}}
700 \def\@polygon(#1,#2){%
701   \pIIE@moveto{#1\unitlength}{#2\unitlength}%
702   \@ifnextchar\lp@r{\@@polygon}{\PackageWarning{pict2e}%
703     {Polygons require at least two vertices!}%
704   \ignorespaces}}
705 \def\@@polygon(#1,#2){\pIIE@lineto{#1\unitlength}{#2\unitlength}%
706   \@ifnextchar\lp@r{\@@polygon}{\pIIE@closepath
707     \if@tempswa\pIIE@fillGraph\else\pIIE@strokeGraph\fi
708   \endgroup
709   \ignorespaces}}

```

3.10.7 Path commands

`\moveto` Direct access to path constructions in PostScript and PDF.

```

\lineto 710 \def\moveto(#1,#2){%
\curveto 711   \@killglue
\circlearc 712   \pIIE@moveto{#1\unitlength}{#2\unitlength}%
\closepath 713   \ignorespaces}
\strokepath 714 \def\lineto(#1,#2){%
\fillpath 715   \@killglue
716   \pIIE@lineto{#1\unitlength}{#2\unitlength}%
717   \ignorespaces}
718 \def\curveto(#1,#2)(#3,#4)(#5,#6){%
719   \@killglue

```

```

720 \pIIE@curveto{#1\unitlength}{#2\unitlength}{#3\unitlength}{#4\unitlength}%
721 {#5\unitlength}{#6\unitlength}%
722 \ignorespaces}
723 \newcommand*\circlearc[6][0]{%
724 \@killglue
725 \pIIE@arc[#1]{#2\unitlength}{#3\unitlength}{#4\unitlength}{#5}{#6}%
726 \ignorespaces}
727 \def\closepath{\pIIE@closepath}
728 \def\strokepath{\pIIE@strokeGraph}
729 \def\fillpath{\pIIE@fillGraph}

```

3.10.8 Ends of paths, joins of subpaths

`\buttcap` Ends of paths and joins of subpaths in PostScript and PDF.

```

\roundcap 730 \ifcase\pIIE@mode\relax
\squarecap 731 \or
\miterjoin 732 \newcommand*\pIIE@linecap@op{setlinecap}
\roundjoin 733 \newcommand*\pIIE@linejoin@op{setlinejoin}
\beveljoin 734 \or
735 \newcommand*\pIIE@linecap@op{J}
736 \newcommand*\pIIE@linejoin@op{j}
737 \fi
738 \def\pIIE@linecap{}
739 \def\pIIE@linejoin{}
740 \def\buttcap{\edef\pIIE@linecap{ 0 \pIIE@linecap@op}}
741 \def\roundcap{\edef\pIIE@linecap{ 1 \pIIE@linecap@op}}
742 \def\squarecap{\edef\pIIE@linecap{ 2 \pIIE@linecap@op}}
743 \def\miterjoin{\edef\pIIE@linejoin{ 0 \pIIE@linejoin@op}}
744 \def\roundjoin{\edef\pIIE@linejoin{ 1 \pIIE@linejoin@op}}
745 \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` #1, the maximum number of points to use, is simply ignored, as well as `\qbeziermax`.
`\@cbezier` Like the kernel version of `\@bezier`, the original version of `\@cbezier` uses `\put` internally,
`\pIIE@@cbezier` 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 L^AT_EX kernel's `\qbezier` and `\bezier`:

```

746 \AtBeginDocument{\@ifundefined{cbezier}{\newcommand}{\renewcommand}*%
747 \cbezier[2][0]{\pIIE@@cbezier[#1]#2}%
748 \@ifdefinable\pIIE@@cbezier{}%
749 \def\pIIE@@cbezier#1) #2(#3)#4(#5)#6({\@cbezier#1)(#3)(#5){}%
750 \def\@cbezier[#1](#2,#3)(#4,#5)(#6,#7)(#8,#9){%
751 \@killglue
752 \pIIE@moveto{#2\unitlength}{#3\unitlength}%
753 \pIIE@curveto{#4\unitlength}{#5\unitlength}%
754 {#6\unitlength}{#7\unitlength}{#8\unitlength}{#9\unitlength}%
755 \pIIE@strokeGraph
756 \ignorespaces}%
757 }

```

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).

```
758 \else

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

759 \renewcommand*\oval[1][]{\pIIfoldoval}
760 \newcommand*\maxovalrad{20pt}
761 \newcommand*\OriginalPictureCmds{}
762 \fi
```

3.13 Final clean-up

Restore Catcodes.

```
763 \Gin@codes
764 \let\Gin@codes\relax
765 \</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 $\text{P}\text{T}\text{E}\text{X}$ package. Thanks go to Michael Vulis (MicroPress) for hints regarding a driver for the $\text{V}\text{T}\text{E}\text{X}$ system. Walter Schmidt has reviewed the documentation and code, and has tested the $\text{V}\text{T}\text{E}\text{X}$ 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 $\text{L}\text{A}\text{T}\text{E}\text{X}$ 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: *L^AT_EX – A Document Preparation System*, 2nd ed., 1994
- [2] Michel Goossens, Frank Mittelbach, Alexander Samarin: *The L^AT_EX Companion*, 1993
- [3] Gerhard A. Bachmaier: *The ebezier package*. CTAN: macros/latex/contrib/ebezier/, 2002
- [4] Michael Wichura: *The PiCT_EX 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

v0.1a	General: First version. (RN)	1	v0.1y	<code>\pIe@vector@ltx</code> : First implementation. (RN,HjG)	24
v0.1d	<code>\pIe@drawGraph</code> : “gsave/grestore” added. (RN)	16	v0.2h	<code>\pIe@badcircarg</code> : New error message. (RN,HjG)	27
v0.1g	<code>\pIe@circle</code> : Changed code (using <code>\pIe@add@qcircle</code>). (HjG,RN) . . .	26	<code>\pIe@circ</code> : Check for negative or zero di- ameter argument (RN,HjG)	26	
	<code>\pIe@drawGraph</code> : “gsave/grestore” re- moved. (RN)	16	<code>\pIe@def@UL</code> : Check for negative or zero radius argument (RN,HjG)	27	
v0.1h	<code>\pIe@addtoGraph</code> : Added newline code (to be improved eventually). (RN,HjG)	16	v0.2j	General: First release to CTAN (2004/02/19 v0.2j). (LaTeX Team) . .	1
v0.1i	<code>\pIe@drawGraph</code> : “gsave/grestore” re- stored for PDF (see ‘p2e-drivers.dtx’). (RN)	16	v0.2k	General: Better control for indexing temporary registers while debugging (HjG)	1
				Better control over funny pagestyle while debugging (HjG)	1
v0.1t	<code>\pIe@get@quadrants</code> : Rename <code>\pIe@get@ovalquadrants</code> to <code>\pIe@get@quadrants</code> (RN)	29	v0.2l	General: Even better control over funny pagestyle while debugging (RN)	1
v0.1u	<code>\pIe@qcircle</code> : New ancillary macro (HjG)	27	<code>\line</code> : Macro added (RNH/HjG)	22	
	<code>\pIe@add@CP</code> : Rename <code>\pIe@add@XY</code> to <code>\pIe@add@CP</code> (HjG)	17	v0.2n	General: Second release to CTAN (2004/04/22 v0.2n). (RN/HjG)	1
	<code>\pIe@bezier@QtoC</code> : New ancillary macro (HjG)	30	<code>\pIe@circ</code> : Allow zero diameter (RN/HjG)	26	
	<code>\pIe@drawGraph</code> : Clear current point af- ter output (HjG)	17	<code>\pIe@def@UL</code> : Moved radius test to <code>\oval</code> , where it belongs (RN/HjG) . .	27	
	<code>\pIe@qcircle</code> : Change coding of quad- rant number to match bit number in <code>\pIe@get@quadrants</code> (HjG)	26	<code>\pIe@maxovalrad</code> : Allow zero diameter (RN/HjG)	28	
	<code>\pIe@qoval</code> : New ancillary macro (HjG)	29		Moved radius test from <code>\pIe@def@UL</code> (RN/HjG)	28
	<code>\pIe@vector</code> : New ancillary macro (HjG)	23	v0.2o	<code>\cbezier</code> : Supply <code>\ignorespaces</code> to match kernel version (HjG)	33
	<code>\pIe@vector@ltx</code> : New ancillary macro (HjG)	24	General: Third release to CTAN (2004/06/25 v0.2o). (RN/HjG)	1	
	<code>\pIe@vector@pst</code> : New ancillary macro (HjG)	25	<code>\Gin@codes</code> : Save and restore catcodes (HjG)	12	
v0.1v	<code>\pIe@qcircle</code> : Exchange <code>\@xdim</code> and <code>\@ydim</code> to <code>\@ovri</code> and <code>\@ovro</code> (HjG)	26	<code>\line</code> : Use <code>\pIe@checkslopeargs</code> (HjG)	22	
v0.1w	General: Index use of temporary registers while debugging (HjG)	1	<code>\vector</code> : Use <code>\pIe@checkslopeargs</code> (HjG)	23	
	<code>\pIe@qoval</code> : Rename <code>\pIe@oval</code> to <code>\pIe@qoval</code> (HjG)	29	v0.2p	<code>\cbezier</code> : <code>\@killglue</code> added. (RN) . .	33
v0.1x	<code>\pIe@firstnum</code> : New ancillary macro (RN,HjG)	23	General: Fourth release to CTAN (2004/07/28 v0.2p). (RN)	1	
	<code>\pIe@secondnum</code> : New ancillary macro (RN,HjG)	23	v0.2q	General: Fourth release to CTAN (2004/08/06 v0.2q). (RN/HjG)	1
	<code>\pIe@FAI</code> : Introduce “inset”. (RN,HjG)	23	v0.2r	<code>\pIe@pyth</code> : Two wrong global assign- ment changed. (RN)	20
			v0.2t	<code>\line</code> : All lines by <code>\@sline</code> (JT)	22

v0.2u	(2011/04/05 v0.2y). (JT) 1
General: Fifth release to CTAN	v0.2z
(2008/06/29 v0.2u). (JT) 1	General: 10th release to CTAN
v0.2v	(2011/04/05 v0.2z). (JT) 1
General: Sixth release to CTAN	v0.3a
(2008/07/19 v0.2v). (JT) 1	General: 11th release to CTAN
v0.2w	(2016/01/09 v0.3a). (JT) 1
General: Seventh release to CTAN	\pIle@circle: Changed code, closepath
(2008/07/19 v0.2w). (JT) 1	seems to be necessary. 26
v0.2x	v0.3b
General: Eighth release to CTAN	General: 12th release to CTAN
(2009/08/08 v0.2x). (JT) 1	(2016/02/05 v0.3b). (RN) 1
v0.2y	New option ‘luatex’ (RN) 12
General: Ninth release to CTAN	

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.

Symbols		
\! 6, 11	\@tempa . . . 363, 364, 367–370	E
\" 4, 9	\@tempdimd . . . 615, 662, 663	\empty 368
\# 583	\@undefined 37, 38	\end 554, 570, 571
\% 8	\@whiledim 623	F
* 5, 10	\@whilenum 346, 352	\fillpath 9, <u>710</u>
\: 7, 12	\^ 3, 8	G
\@@polygon . . . 702, 705, 706	A	\Gin@codes 2, 763, 764
\@angle 653, 654	\arc 8, 514, <u>668</u>	\Gin@driver . . <u>13</u> , 19, 21–
\@angleend 625, 626	\arc* 8	27, 30–33, 90, 96–99, 101
\@anglemid . . . 636, 638, 640	B	I
\@arclen . . . 613, 619–621,	\begin 556, 576, 589	\I 327, 345, 346, 357
623, 624, 633–635, 652	\beveljoin 9, <u>730</u>	\iffalse 61, 426
\@arcrad . . . 614, 617, 618,	\bezier 7	\ifpIle@pdfliteral@ok .
643, 644, 654, 659, 660	\buttcap 9, 49, 490, <u>730</u>	 <u>35</u> , 48
\@badlinearg	C	\ifx 37, 38,
365, 370, 371, 377, 405	\CalculateCos . . 242, 642, 658	368, 369, 613–615, 680
\@bezier	\CalculateSin . . 241, 642, 658	\ignorespaces
113, 126, 576, 584, 590	\CalculateTan 653	 580, 585, 605,
\@cbezier 117, 127, <u>746</u>	\catcode 3–8, 10	692, 695, 704, 709,
\@circle 111, 124, <u>483</u>	\cbezier 7, <u>746</u>	713, 717, 722, 726, 756
\@dot 112, 125, <u>484</u>	\changes 577, 579, 581	L
\@gobble 81	\circle 6, 514	\label 574
\@ifnextchar	\circle* 6	\Line 8, <u>685</u>
. 690, 695, 702, 706	\circlearc . . . 8, 515, 650, <u>710</u>	\line 3, 108, 122, 372, <u>373</u>
\@ifpackageloaded 428	\closepath 9, <u>710</u>	\lineto 8, <u>710</u>
\@ifstar 669, 698	\cmd 580, 581, 584, 585, 587	\lp@r 685, 690, 695, 702, 706
\@killglue 581,	\countdef 326, 327	M
585, 592, 688, 697,	\curveto 8, <u>710</u>	\m@ne 346
711, 715, 719, 724, 751	D	\maxdimen 332, 334, 339, 341
\@makeother 9, 11, 12	\Den 326, 336,	\maxovalrad . . 7, <u>517</u> , 529, <u>759</u>
\@oval 119, 129, <u>533</u>	337, 344, 352, 354, 356	\miterjoin 9, <u>730</u>
\@polygon 698–700	\Denom 325, 328, 329, 331, 336	\moveto 8, <u>710</u>
\@polyline 690, 693, 695	\dimendef 325	
\@setfpsbit 568, 569		
\@sline 109, 121, 377, <u>378</u>		
\@temp 637–640		

N		
\newdimen	613–615	
\newif	35	
\Num	326, 336, 337, 344, 352–354, 356, 357	
\Numb	327, 337, 338, 343, 344, 354–356	
\number	343, 355	
\Numer	325, 328–330, 336	
O		
\OriginalPictureCmds	120, 759	
\oval	6, 118, 128, 514, 528, 759	
P		
\pdfliteral	38, 44	
\pIe@arc	626, 628, 631, 637, 639	
\pIe@cbezier	746	
\pIe@divide	346, 351	
\pIe@firstnum	423	
\pIe@pyth	312, 319	
\pIe@qcircle	501, 503, 505, 507, 510	
\pIe@secondnum	423	
\pIe@add@CP	189, 252, 254, 264, 278, 289, 664	
\pIe@add@num	202, 222	
\pIe@add@nums	196, 214, 216, 218, 221, 223, 229, 231, 233, 260, 262, 272, 275, 285, 287, 657, 663	
\pIe@addtoGraph	166, 194, 200, 205, 212, 219, 221– 223, 234, 252, 254, 265, 279, 290, 292, 665	
\pIe@arc	616, 674, 677, 725	
\pIe@arc@	669, 670	
\pIe@arc@@	670, 671	
\pIe@badcircarg	487, 512, 531, 618	
\pIe@bezier@QtoC	446–449, 597–600, 609	
\pIe@buttcap	47, 541	
\pIe@CAW	56, 64, 69, 423, 439, 461	
\pIe@checkslopearg	363, 364, 366	
\pIe@checkslopeargs	358	
\pIe@checkslopeargsline	358, 374	
\pIe@checkslopeargsvector	360, 402	
\pIe@circ	483, 484, 485	
\pIe@circle	489, 492	
\pIe@closepath	292, 494, 675, 706, 727	
\pIe@closepath@op	147, 156, 292	
\pIe@code	13, 80, 81, 185	
\pIe@concat	209, 225, 417	
\pIe@concat@op	147, 156, 219, 234, 249	
\pIe@CPx	187, 189, 270, 273, 276	
\pIe@CPy	187, 189, 271, 274, 277	
\pIe@curveto	451, 459, 602, 720, 753	
\pIe@curveto@op	147, 156, 279, 290, 665	
\pIe@debug@comment	73, 175	
\pIe@def@UL	518	
\pIe@defaultUL	518, 530	
\pIe@divide	304, 320, 323, 391, 410, 415, 416, 467	
\pIe@drawGraph	172, 173, 174	
\pIe@FAI	57, 65, 70, 423, 445, 466	
\pIe@FAL	54, 62, 67, 423, 440, 462	
\pIe@FAW	55, 63, 68, 423, 439, 461	
\pIe@fill@op	147, 156, 178	
\pIe@fillGraph	419, 490, 675, 707, 729	
\pIe@get@quadrants	540, 565	
\pIe@GRAPH	166, 186, 187	
\pIe@linecap	182, 738, 740–742	
\pIe@linecap@op	732, 735, 740–742	
\pIe@linejoin	182, 739, 743–745	
\pIe@linejoin@op	733, 736, 743–745	
\pIe@lineto	253, 397, 453– 456, 458, 472, 474– 477, 479, 481, 482, 559, 646, 694, 705, 716	
\pIe@lineto@op	147, 156, 254	
\pIe@maxovalrad	528, 537	
\pIe@mode	13, 34, 87, 105, 106, 133, 138, 145, 146, 208, 255, 730	
\pIe@moveto	251, 396, 450, 471, 511, 558, 601, 647, 673, 689, 701, 712, 752	
\pIe@moveto@op	147, 156, 252	
\pIe@old@cbezier	108, 126, 607	
\pIe@old@cbezier	108, 127	
\pIe@old@circle	108, 124	
\pIe@old@dot	108, 125	
\pIe@old@oval	108, 129	
\pIe@old@sline	108, 121	
\pIe@old@line	108, 122	
\pIe@oldoval	108, 128, 532, 759	
\pIe@oldvector	108, 123	
\pIe@pdfliteral	35	
\pIe@pdfliteral@okfalse	39	
\pIe@pdfliteral@oktrue	36	
\pIe@PTtoBP	207, 224, 249	
\pIe@pyth	293, 406	
\pIe@qcircle	493, 494, 495, 559	
\pIe@qoval	544, 547–550, 553	
\pIe@rcurveto	256, 267, 511	
\pIe@rotate	209, 225	
\pIe@scale	209, 225	
\pIe@scale@PTtoBP	176, 209, 225	
\pIe@setlinewidth@op	147, 156, 181	
\pIe@stroke@op	147, 156, 183	
\pIe@strokeGraph	398, 490, 551, 603, 678, 695, 707, 728, 755	
\pIe@tempa	16, 240– 244, 307, 308, 313, 314	
\pIe@tempb	16, 243, 245, 246	
\pIe@tempc	16, 244–246	
\pIe@translate	209, 225	
\pIe@vector	53, 60, 418, 422	
\pIe@vector@ltx	53, 438	
\pIe@vector@pst	60, 460	
\pIearc	668, 684	
\polygon	8, 685	
\polygon*	8	
\polyline	8, 685	
\psk@arrowinset	432	
\psk@arrowlength	429	
\psk@arrowsize	430, 431	
\put	584, 587	
Q		
\Q	332, 339, 343, 349, 355	
\qbezier	7	
\qbeziermax	7	
R		
\roundcap	9, 730	
\roundjoin	9, 730	
\rp@r	685	
S		
\sign	330, 334, 341, 349, 620, 621, 623, 633	

<code>\squarecap</code>	9, <u>730</u>	U	V
<code>\strokepath</code>	9, <u>710</u>	<code>\undefined</code> . . .	613–615, 680
<code>\subsubsection</code>	573	<code>\vector</code>	4, 110, 123, 372, <u>400</u>
		<code>\UseCos</code>	244, 643, 656, 659, 662
T		<code>\UseSin</code>	243, 644, 655, 660, 661
<code>\tempend</code>	349, 350	<code>\UseTan</code>	654