The pict2e package*

Hubert Gäßlein†, Rolf Niepraschk‡ and Josef Tkadlec§

2009/08/05

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

List of Figures

*This document corresponds to pict2e.sty v0.2x, dated 2009/08/05, documentation dated 2009/08/05.

†HubertJG@open.mind.de
‡Rolf.Niepraschk@ptb.de
§j.tkadlec@email.cz

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	dvipsone	x?
xdvi	x	dviwindo	x?
pdftex	x	dvipdf	x?
vtex	x	textures	x?
dvipdfm	x	pctexps	x?
xetex	x	pctex32	x?
oztex	(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* $|x|, |y| \leq 6$ *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` *[⟨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 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(⟨X,Y⟩){⟨LEN⟩}`

In the Standard LaTeX implementation the slope arguments (`⟨X,Y⟩`) 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]:

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

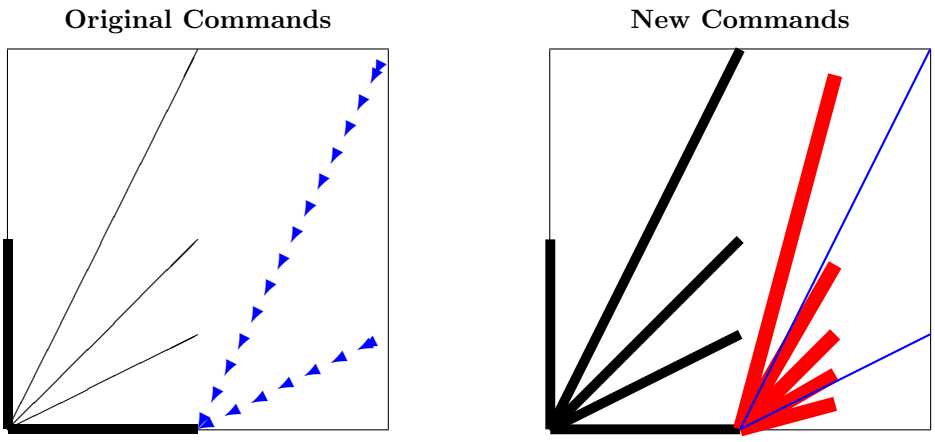


Figure 1: Line

However, to avoid overflow of 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 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 [2] (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.*

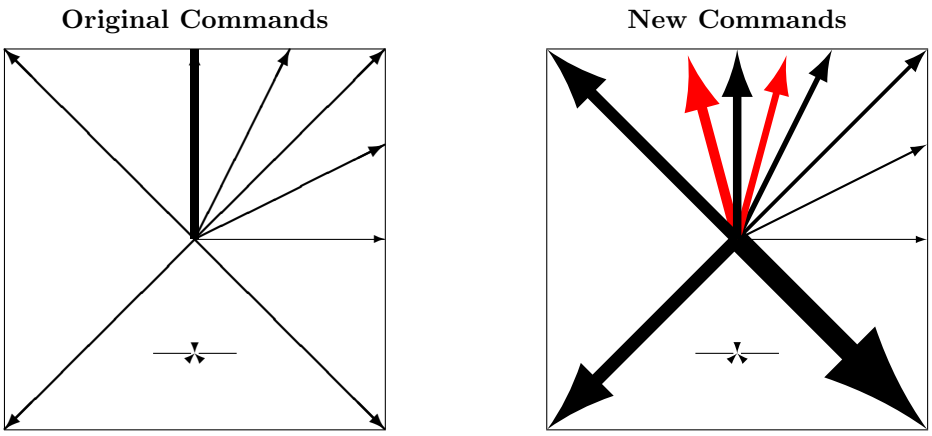


Figure 2: Vector

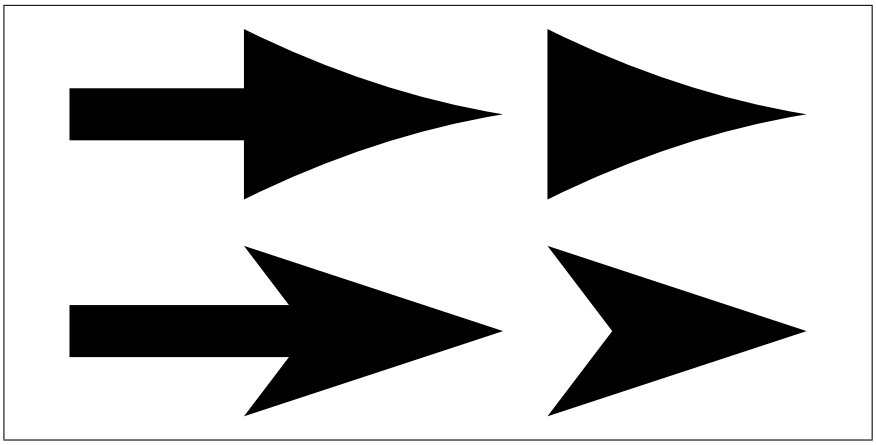


Figure 3: Vector: shape variants of the arrow-heads. Top: LaTeX style vectors. Bottom: PSTricks style vectors.

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 [3], 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*⟩`]`

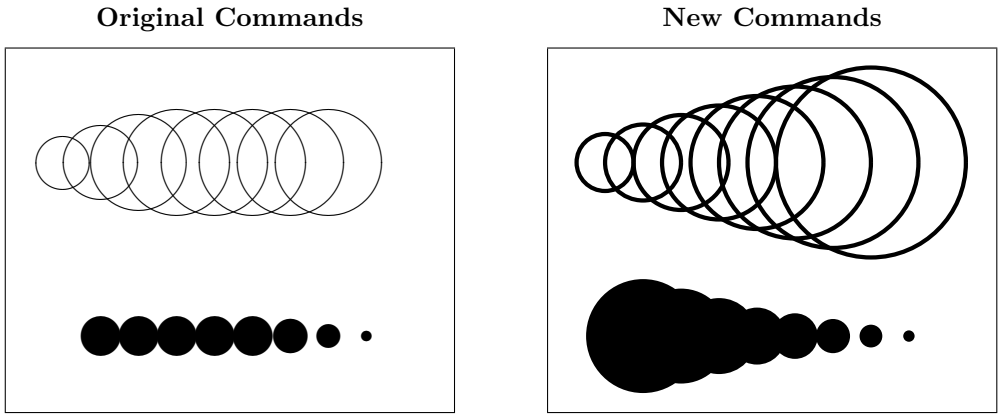


Figure 4: Circle and Dot

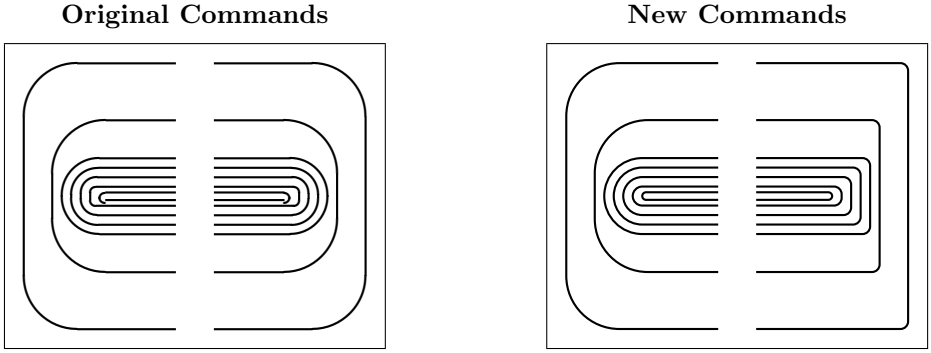


Figure 5: Oval: Radius argument for `\oval` vs. `\maxovalrad`

`\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 [4]. (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 `pgf`). 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.

2.4.2 Lines, polygons

`\Line` `\Line(`⟨*X1,Y1*⟩`)(`⟨*X2,Y2*⟩`)`
`\polyline` `\polyline(`⟨*X1,Y1*⟩`)(`⟨*X2,Y2*⟩`)`$\ldots$`(`⟨*Xn,Yn*⟩`)`
`\polygon` `\polygon(`⟨*X1,Y1*⟩`)(`⟨*X2,Y2*⟩`)`$\ldots$`(`⟨*Xn,Yn*⟩`)`
`\polygon*` `\polygon*(`⟨*X1,Y1*⟩`)(`⟨*X2,Y2*⟩`)`$\ldots$`(`⟨*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 \strokepath \fillpath

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 \roundcap \squarecap

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. 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`, path constructions ended by `\closepath`).

\mitterjoin \roundjoin \beveljoin

The shape of joins of subpaths is controlled by the following commands: `\mitterjoin` (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.

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

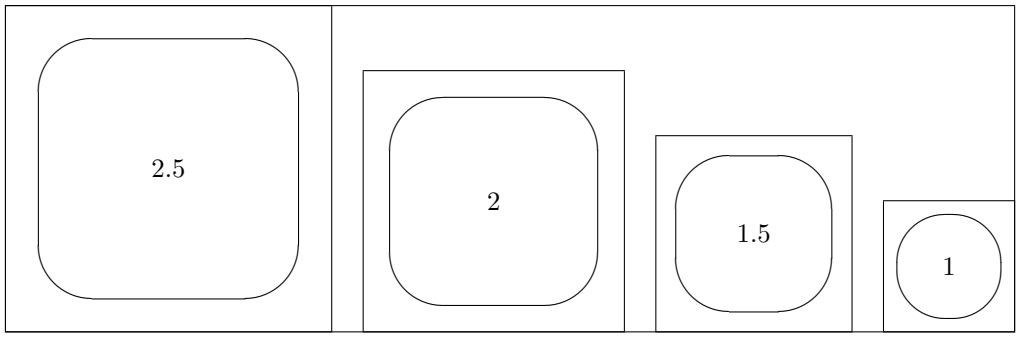
[1] Leslie Lamport: *LaTeX – A Document Preparation System*, 2nd ed., 1994

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

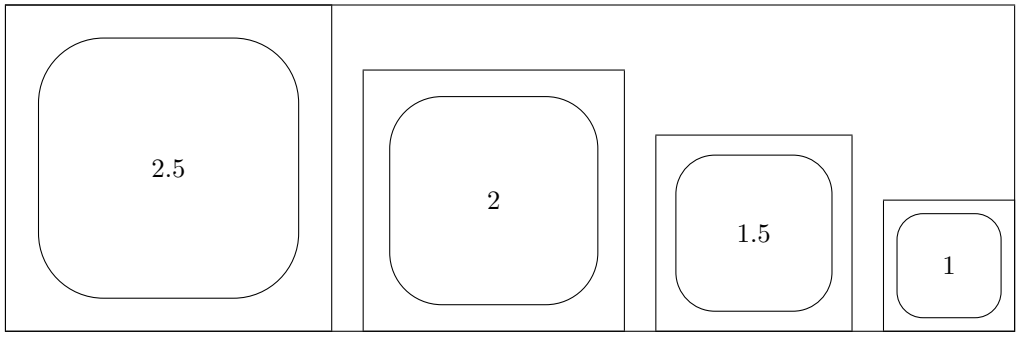
[3] David Carlisle: *The pspicture package.* `CTAN: macros/latex/contrib/carlisle/`, 1992

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

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



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



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

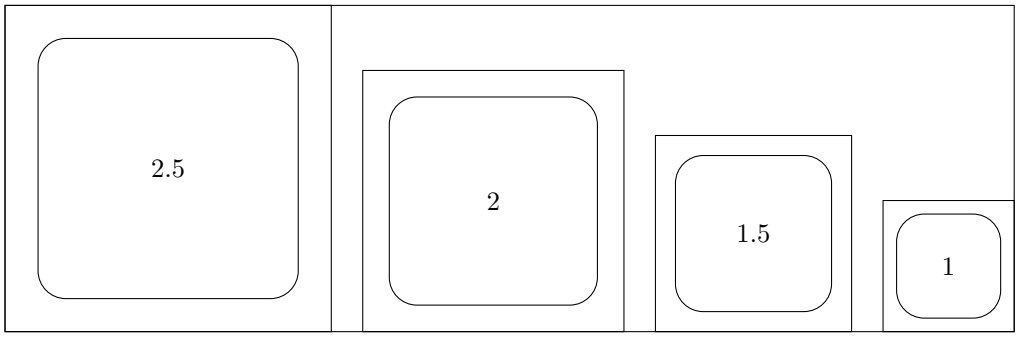


Figure 6: Oval: Radius argument for `\oval`: length vs. number. The number at the centre of each oval gives the relative value of `\unitlength`.

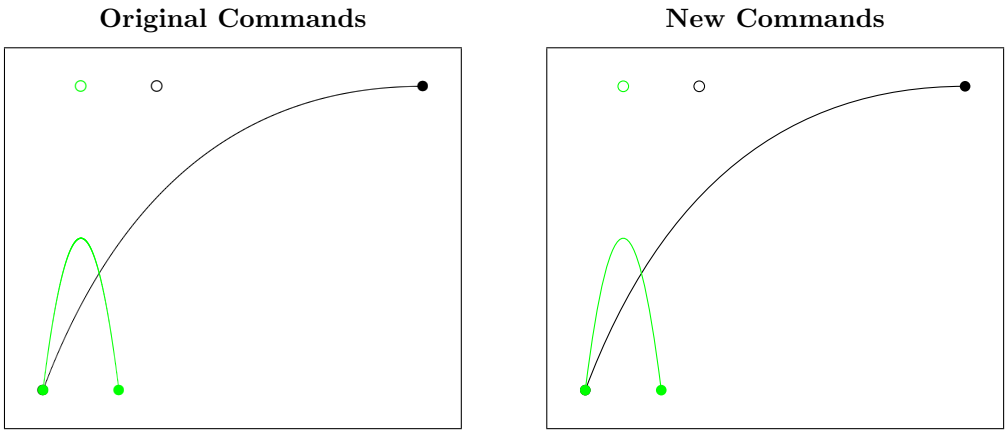


Figure 7: Quadratic Bezier curves

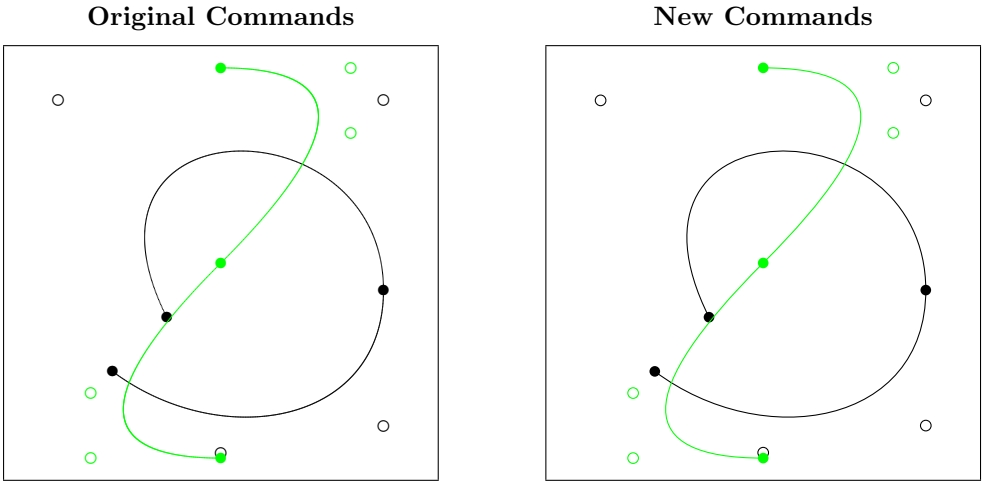


Figure 8: Cubic Bezier curves

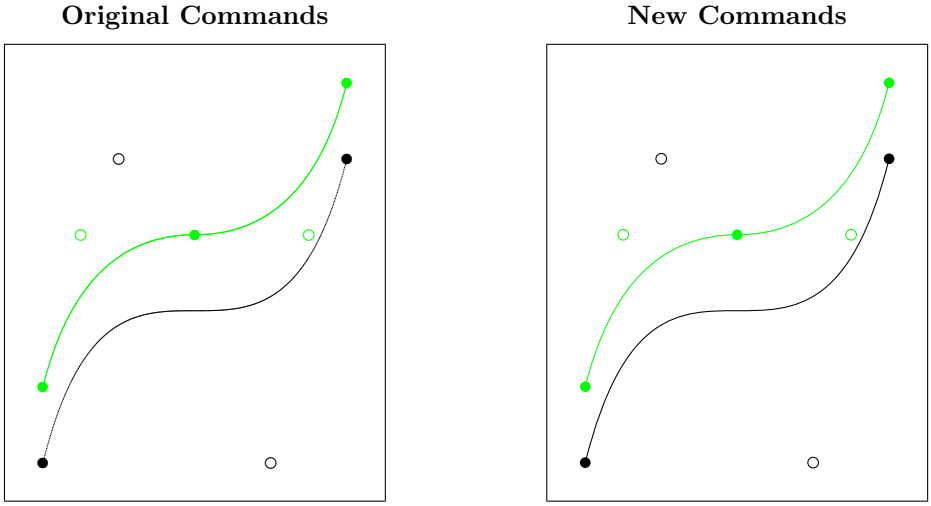


Figure 9: Quadratic (green) and Cubic Bezier curves