Using ebezier

Gerhard A. Bachmaier

March 1, 2005

Abstract

The package ebezier is an extension of the (old) bezier.sty which is now part of LaTeX 2_ε. It defines linear and cubic Bernsteĭn polynomials together with some plotting macros for arcs.

With the aid of the calc package also the calculation of square roots and henceforward lengths is supplied.

Preamble

If you want to draw complicated and/or lots of pictures, you should use PostScript for generating your plots and dvips to include them in TeX documents. PostScript can plot lines with arbitrary slope and unlimited length and circles with arbitrary radius just by using one command. See also the LaTeX Graphics Companion[4] for further possibilities. There is also a new package pict2e[8] available which is preferrable for PDF and PostScript.

This package will support also lines with arbitrary slopes and unlimited length, but each line has to be generated as a sample of points. Each point reduces TeX's memory and you will very likely have to overcome some `TeX capacity exceeded...` messages.

Exact circles would involve trigonometric functions or square roots to be evaluated by TeX. Even with some tricks for reducing the effort of the calculation algorithm there would be hundreds of calculations for each point.* But they may be quite well approximated by cubic bezier curves, also supplied in this package (The quality of interpolation is discussed in some detail in the Section *Fitting Arcs*.) In fact, the small circles in the LaTeX-`lcircle` fonts are also generated by the same method.

For draft papers use all kind of bezier curves with small number of points, just for the final run increase the numbers. TeX memory can be set free again

*To use TeX for complex computations is as satisfactory as using your desk calculator for writing tasks. But if you really want to do it e.g. the digits 7353 can be read (rotating by 180^0) as `ESEL`, the german word for "donkey".

with `\clearpage` at the end of complicated pictures. It's also a good idea to have them at an extra page (option `[p]` for `figure` environments).

For optical constructions the software LaTeXPiX[9] may be a starting point. This software supports cubic bezier curves defined in this package or from `bez123`[5].

1 Mathematical Definitions

A Bernsteĭn polynomial of degree $n-1 (n \geq 2)$ is defined by n points $z_1, z_2, \ldots, z_n$

$$\mathcal{B}_{n-1}[t] = \sum_{i=0}^{n-1} \binom{n-1}{i} (1-t)^{n-1-i} t^i z_{i+1} \quad t \in [0,1].^\dagger$$

The points z_i, $i \in \{1, \ldots, n\}$, may be considered as real numbers, then $\mathcal{B}$ is really a polynomial in t. Or they denote points in a plane, which notation we will use further. In this case both *components* are polynominials and the graph for $\mathcal{B}$ is—part of—an algebraic curve.

All these graphs have in common:

- The graph is contained in the convex hull of the defining points.
- The graph starts at z_1 and stops at z_n.
- At the endpoints the tangents coincident with the directions $z_1 - z_2$ and $z_{n-1} - z_n$ correspondingly.

For $n = 2$ the Bernsteĭn polynomial $\mathcal{B}_1$ reduces to the linear form spanned by z_1 and z_2. The parametrization in t

$$\mathcal{B}_1[t] = (1-t)z_1 + tz_2 =: t[z_1, z_2]$$

is also known as *convex coordinates* for the segment $\overline{z_1 z_2}$.

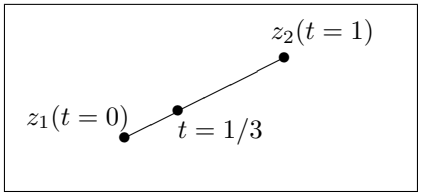

Figure 1: Line defined by two points

†There are also variants of this definitions with all coefficients $\equiv 1$.

For $n = 3$ the result is a (quadratic) parabola which can also be constructed as the convex hull of all tangents in the triangle $\Delta\, z_1 z_2 z_3$ (examplified in Fig. 2b).

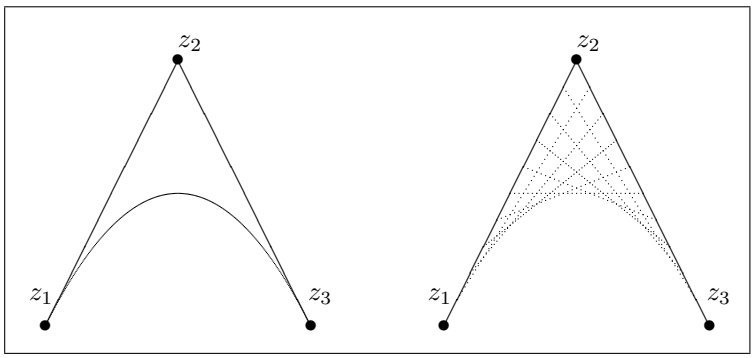

Figure 2: Quadratic parabola (a) as Bernsteĭn polynom of degree 2 and (b) as hull of tangents

For $n = 4$ finally we arrive at the cubic curves used e.g. in the METAFONT book[2].

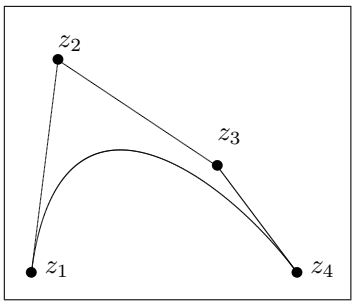

Figure 3: A simple cubic parabola.

We will not use more complicated polynomials for several reasons:

- Higher degree polynomials require more operations to calculate just one point of the graph.
- For sketches (and **not** exact graphs!) cubic splines are sufficient to scope with all kind of different curvature requirements.
- TEX can handle integers up to 2^{28}, and "real number" lengths are transformed to integers (multiples of scaled points: 1 pt=2^{16} sp) [1]. To stay within this restricted range even for cubic beziers we have to do calculations in the right order. Changing the order of multiplication and divisions will result very soon in arithmetic overflows. Also multiplication

with these pseudo-real numbers is not an associative operation (due to the range limits!).

- The maximum number of arguments for commands in TEX is limited to nine, which is just enough for four points and a number.

2 The Plotting Macros

2.1 Simple Beziers

There are two first level plot commands to be used in a LaTeX 2_ε `picture` environment:

```
\lbezier[n](x1,y1)(x2,y2)
\cbezier[n](x1,y1)(x2,y2)(x3,y3)(x4,y4)
```

The arguments in square brackets are optional! If they are omitted or $n = 0$ an adequate number will be calculated (cf. Section 8).

`\qbezier` `\lbezier` draws line segments from point (x_1, y_1) to (x_2, y_2), or more exactly, $n + 1$ intermediate points, while `\cbezier` is an implementation of the cubic variant. Just for completeness let me remind you that the quadratic variant—`\qbezier[n](x1,y1)(x2,y2)(x3,y3)`—is part of LaTeX 2_ε.

`\qbeziermax` n is always limited by the number `\qbeziermax` (=500).

You may change `\qbeziermax` by a command like (it is not a counter!) `\renewcommand{\qbeziermax}{1000}`.

2.1.1 lbezier

`\lbezier` `\lbezier` is straightforward defined as linear polynomial. It produces equally spaced points.

```
...
\put(0,25){\line(1,0){90}}
\lbezier[20](0,10)(90,10)
\lbezier[200](0,-5)(90,-5)
...
```

Use `\lbezier` only in cases where the line you want to plot is not within the scope of the `\line` command, i.e. the slope is not a small rational number and/or the length is too small.

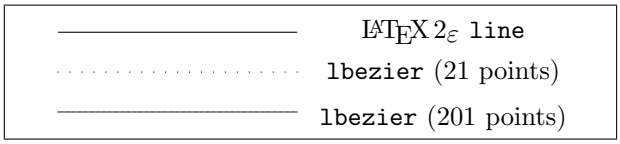

Figure 4: Different line commands

2.1.2 cbezier

`\cbezier` Just like the `\lbezier` macro `\cbezier` uses no tricks to generate the third order polynomial. The examples are from the METAFONT book (pp. 13)[2], where the influence of changing the order of the controlling points (z_1 up to z_4) is also demonstrated.

```
...
% z1=(0,16) z2=(40,84) z3=(136,96) z4=(250,0)
% z12=(20,50) z23=(88,90) z34=(193,48) z123=(54,70)
% z234=(140.5,69)
\cbezier[400](0,16)(40,84)(136,96)(250,0)
\lbezier[30](0,16)(40,84)
\lbezier[30](40,84)(136,96)
\lbezier[30](136,96)(250,0)
\lbezier[30](20,50)(88,90)
\lbezier[30](88,90)(193,48)
\lbezier[30](54,70)(140.5,69)
...
```

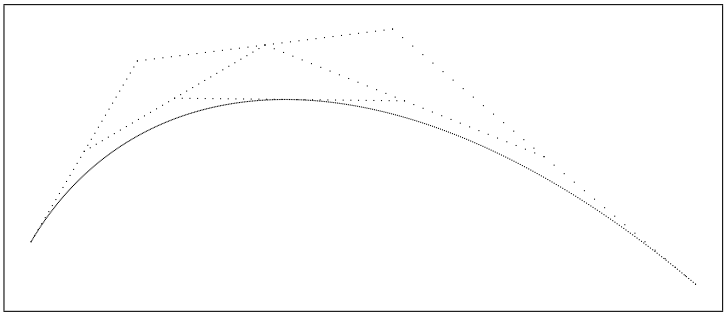

Figure 5: Iteration scheme for one point

`\Cbezier` The variant `\Cbezier` draws also dots and lines for the controlling points (see Fig. 6)[‡].

[‡]It resets also the plot symbol to the standard one; cf. Section 7

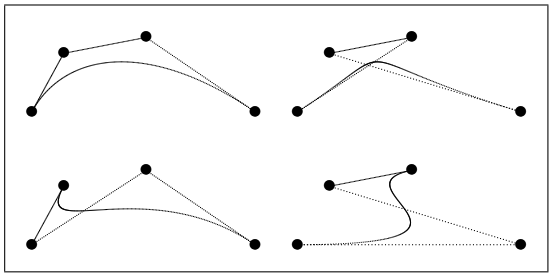

Figure 6: Examples for cubic curves with varying the order of the controlling points

2.2 Circles and Arcs

All complex plotting commands in this package use a variant of `\cbezier` as building block. As in the METAFONT book circles and arcs may be represented by `\cbezier`.

To illustrate the procedure of the macro we do one calculation explicitely.

E.g. we want to draw the upper right quarter of a circle with end points $z_1 = (0, r)$ and $z_4 = (r, 0)$. z_2 and z_3 determine the tangents. So we may introduce them as $z_2 = (h, r)$ and $z_3 = (r, h)$ with a—so far unspecified—parameter h.

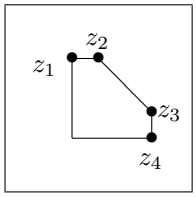

Figure 7: Sketch for the geometrical configuration

If we substitute all points in the formula for the Bernsteĭn polynomial for both components, we end at (for $t = 1/2$)

$$x[\frac{1}{2}] = y[\frac{1}{2}] = \frac{r}{2} + \frac{3h}{8}$$

These values should be $r/\sqrt{2}$ for a circle. So we arrive at

$$h = \frac{4}{3}\left(\sqrt{2} - 1\right).$$

`\cArc`
`\cCircle`

The plot commands are:

```
\cArc[n](xm,ym)(x1,y1)
\cCircle[n](xm,ym){r}[loc]
```

The optional qualifier n determines the number of plotted points (There are as before $n+1$ plotted points for arcs; for circles the number depends on the specifier *loc* and may be $n+1$, $2n+2$, or $4n+4$.).

`\cArc` plots a half circle with centre (x_m, y_m) and x-axis through (x_1, y_1) counterclockwise.

r is the radius of the circle, specified as decimal constant in terms of `\unitlength`.

`\cCircle` plots full, halves and quarters of circles by specifying *loc* (see the corresponding table).

Table 1: Location specifiers for `cCircle`s

loc	specifies ...
`f`	full circle
`l`	left half circle
`r`	right half circle
`b`	bottom half circle
`t`	top half circle
`lb` or `bl`	left bottom quarter of the circle
`lt` or `tl`	left top quarter of the circle
`rb` or `br`	right bottom quarter of the circle
`rt` or `tr`	right top quarter of the circle

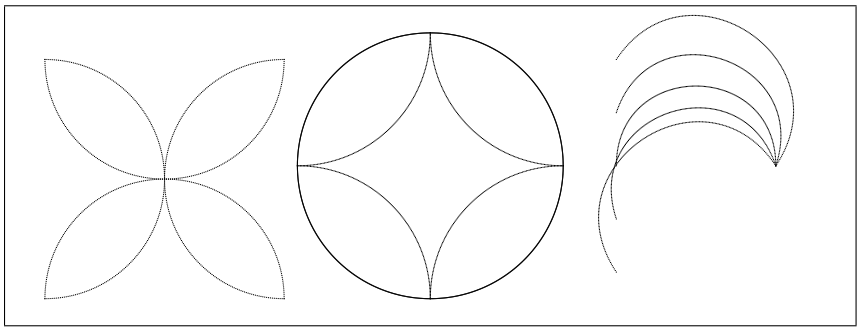

Figure 8: Examples for `cCircle` and `cArc`

3 Fitting Arcs

The quality of representating arcs by cubic bezier curves is quite satisfactory. The differences between circles and beziers may be estimated in two ways.

1. If we test the overall fit the area enclosed by the curves is a good metric: The area of `Carc` for the quarter circle is $1/30(-33+40\sqrt{2})r^2$ to be conferred with $\pi/4\,r^2$. This is an overshot by just 0.028%!

2. The pointwise fit is measured by the radial difference. The maximum is $\cong 0.00025\,r$ (at odd multiples of $\pi/8$), it is zero for all multiples of $\pi/4$.

4 Some TEXnical Notes

For the macros therein a lot of counters and lengths have to be declared.[§] Counters represent integer numbers, lengths are "real" numbers (actually they are just integer multiples of $1/65536 = 2^{-16}$). TEX has just a limited number of these stacks and therefore I use the same counters/lengths in all the macros.

One cannot store a real number for further use in these internal stacks just a multiplication of a *decimal constant* with a length is possible (counters may be multiplied also with real numbers but just the integer part of the decimal constant is used!)

The package calc introduced in the LATEX Companion[3] adds a new possiblity for multiplying lengths with the ratio of two lengths. This feature will be utilized furthermore.

5 Calculating Lengths

If I define lengths with respect to some `\unitlength` I can now define a product or fraction of two lengths:

```
\lengthc = \lengtha*\ratio{\lengthb}{\unitlength}
```

and

```
\lengthc = \unitlenght*\ratio{\lengtha}{\lengthb}
```

The dimension of `\lengthc` *in terms of* `\unitlength` (!) is the product, or factor respectively, of the two other lengths.

With these operations it is even possible to calculate square roots. Simply use the iteration scheme (m integer)

$$\xi_{m+1} = \frac{1}{2}\left(\xi_m + \frac{a}{\xi_m}\right)$$

which will converge fast (with accuracy `\eps`=1 sp) to $\sqrt{a}$ (starting with $\xi_0 = a > 0$).

Lengths (in a `picture` environment) are easily calculated too, one just has to care for the upper limits (the maximum length for TEX is roughly 16384 pt!).

`\LenMult` `\LenDiv` `\AbsLen` `\LenSqrt` `\Length` `\LenNorm`

The macros are:

[§] Although I reuse some internal lengths I had to declare some more to be used in function calls.

- `\LenMult#1#2#3` and `\LenDiv#1#2#3` with two input and one output length (`#3`).
- `\AbsLen#1` which returns the input length as positive length (TEX lengths can be negative!).
- `\LenSqrt#1#2` returns in the length `#2` the square root of length `#1` (to say it again: measured in terms of `\unitlength`).
- `\Length(#1,#2)(#3,#4)#5` stores in `#5` the length of the line segment between points `(#1,#2)` and `(#3,#4)` (coordinates may be decimal constants as in the `picture` commands).
- `\LenNorm#1#2#3` returns in `#3` the length of the hypothenuse of the rectangular triangle with catheti `#1` and `#2`.

`\eps` **All calculations** can be only exact up to the smallest length in TEX which is `\eps`=1 sp=2^{-16} pt=0.000015 pt.

Examples (`\unitlength`=1 pt):

```
Mult: \LenMult{3pt}{4.333333pt}{\PathLength}\the\PathLength
Div:  \LenDiv{3pt}{4.3333333pt}{\PathLength}\the\PathLength
Abs:  \setlength{\PathLength}{-10pt}\the\PathLength\
      \AbsLen{\PathLength}\the\PathLength
Sqrt: \LenSqrt{16pt}{\PathLength}\the\PathLength\
      \LenSqrt{2pt}{\PathLength}\the\PathLength\
      \Length(1.5,4.3)(2.7,5){\PathLength}\the\PathLength\
      \LenNorm{3pt}{4pt}{\PathLength}\the\PathLength
```

Mult: 12.99998pt (exact: 13 pt)

Div: 0.6923pt (exact: 0.692308 pt)

Abs: -10.0pt 10.0pt

Sqrt: 4.0pt (exact: 4 pt) 1.41422pt (exact: 1.414213 pt)
1.38919pt (exact: 1.389244 pt) 5.0pt(exact: 5 pt)

`\PathLengthQ` `\PathLengthC` `\PathLength` `\pathmax` Furthermore you can use these macros to evaluate the length of linear interpolations of the curves displayed by `\qbezier` and `\cbezier`. The syntax is `\PathLengthQ[n](x1,y1)(x2,y2)(x3,y3)` and `\PathLengthC[n](x1,y1)(x2,y2)(x3,y3)(x4,y4)` respectively. n is the number of interpolation points which is bounded by `\pathmax`=50. The length is stored in the —already defined and used—length `\PathLength`. Note: n is *not* optional for these two macros.

Example: For the cubic spline
`\cbezier[200](0,0)(50,100)(50,0)(100,100)` shown in Fig. 9 the results of the `\PathLength`
for n=2,5,10,20,30,40,50 are displayed below. You may increase the value of

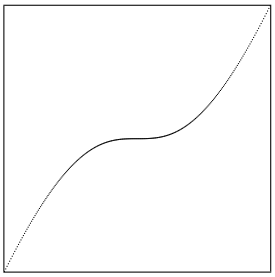

Figure 9: A nice cubic curve

`\pathmax` as for `\qbeziermax` but the result will due to the internal calculation problems not become sigificant better.

The results are: 141.42139pt, 148.87946pt, 149.71536pt, 149.92725pt, 149.96634pt, 149.9802pt, 149.9863pt. (An good numercial integration program will yield more accurate 149.999.)

6 More general arcs

`\cArcs` Finally you can plot an arc (i.e. a cubic approximation to the circle arc) between two points with given centre of the circle:
`\cArcs[n](xm,ym)(x1,y1)(x2,y2)`
with n+1 number of points (limited by `\qbeziermax` again) and centre (x_m, y_m).

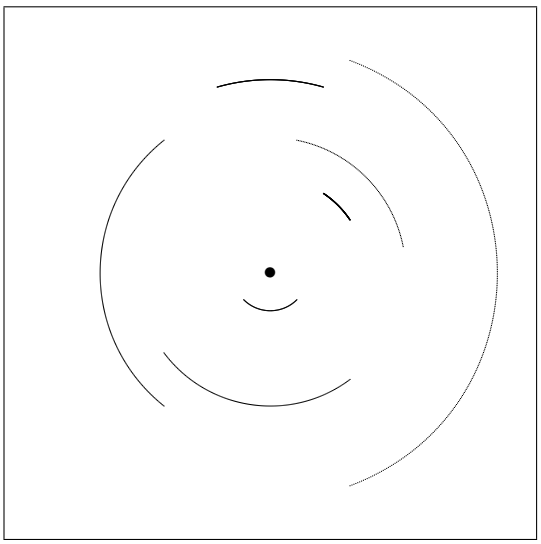

Figure 10: Some examples for arcs; the centre is marked by •

Limitations:

- The arc should be smaller than the half of a circle (The limit is handled by `\cArc` and is built-in again in `\cArcs`.) Otherwise the shape will become "elliptic" and ly in the wrong half plane.
- There is no check for consistency if $r_1^2 = (x_1 - x_m)^2 + (y_1 - y_m)^2$ and $r_2^2 = (x_2 - x_m)^2 + (y_2 - y_m)^2$ are really equal. The graph will contain in any case both points as border points.

I will shortly derive the formulas used in the code. The code is even more tricky due to the fact that I had just a limited number of lengths and the code reuses some lengths explicitely and implicitely by calling routines.

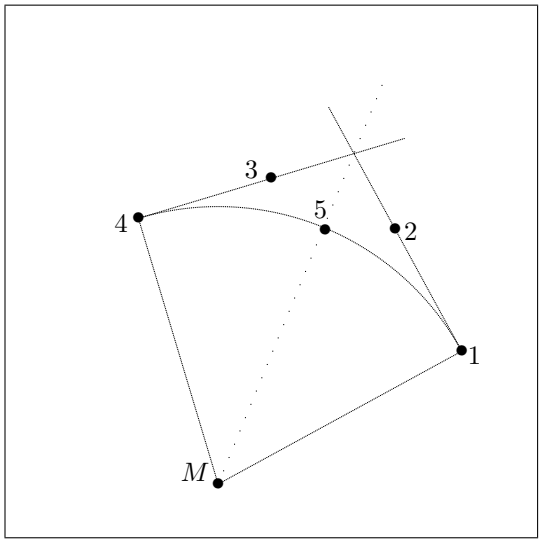

Figure 11: Sketch for the geometric situation

We know the coordinates for the points M, 1, and 4. The tangents $\overline{43}$ and $\overline{12}$ are normals to the radius in the corresponding points. The distances $\overline{43}$ and $\overline{12}$ should be equal. 5 lies on the symmetry axis (dotted line) with distance r from M.

Normal vectors: $\vec{n}_1 = (y_m - y_1, x_1 - x_m)$ and $\vec{n}_2 = (y_4 - y_m, x_m - x_4)$

Coordinate vectors: $\vec{2} = \vec{1} + \lambda\vec{n}_1$ and $\vec{3} = \vec{4} + \lambda\vec{n}_2$ (λ is the same because both normal vectors have length r)

Furthermore $\vec{5} = \mathcal{B}_4[1/2]$ (the cubic spline should also be symmetric and contain 5)

Now we have:

$$x[t] \quad = \quad (1-t)^3 x_1 + 3t(1-t)^2 x_2 + 3t^2(1-t)x_3 + t^3 x_4 \tag{1}$$

$$y[t] \quad = \quad (1-t)^3 y_1 + 3t(1-t)^2 y_2 + 3t^2(1-t)y_3 + t^3 y_4 \tag{2}$$

Substituting for x_2, y_2, x_3, and y_3 and $t \to 1/2$:

$$x_5 = x\left[\frac{1}{2}\right] \quad = \quad \frac{1}{2}(x_1 + x_4) + \frac{3}{8}\lambda(y_4 - y_1) \tag{3}$$

$$y_5 = y\left[\frac{1}{2}\right] \quad = \quad \frac{1}{2}(y_1 + y_4) + \frac{3}{8}\lambda(x_1 - x_4) \tag{4}$$

We could now calculate the norm of this point and set it equal to the radius $r^2 = (x_m - x_1)^2 + (y_m - y_1)^2$. This gives a quadratic equation for λ. But the result is a rather complex term with respect to our input parameters.

A nicer term can be found if we define

$$x_5 = x_m + \kappa(x_1 + x_4 - 2x_m) \quad y_5 = y_m + \kappa(y_1 + y_4 - 2y_m) \tag{5}$$

with aid of the symmetry vector. κ is simply r divided by the norm of the symmetry vector.

The resulting λ is now (using just the x-equation)

$$\lambda = \frac{4}{3}(-1 + 2\kappa)\frac{x_1 + x_4 - 2x_m}{y_4 - y_1} \tag{6}$$

Special cases:

- The symmetry vector is the null vector if $\overline{14}$ is a diameter of the circle. But this case is already solved by `\cArc`.
- For $y_4 = y_1$ one needs the equation for the y-component, i.e. we have as factor $(y_1 + y_4 - 2y_m)/(x_1 - x_4)$ in λ.

7 Varying the line thickness

There is another package, bez123[5], which introduces also linear and cubic bezier curves, even variants which plot exactly all kind of conic curves (ellipses, parabolas, and hyperbolas). There are two features in bez123, which I added in the third version of ebezier:

`\thinlines`
`\thicklines`
`\linethickness`
`\qbeziermax`

1. Changing the size of the plot squares by the LaTeX commands `\thinlines`, `\thicklines`, and/or `\linethickness`.
2. Calulation of an optimal number of plot points if n=0 instead of using the maximum `\qbeziermax` (see next section).

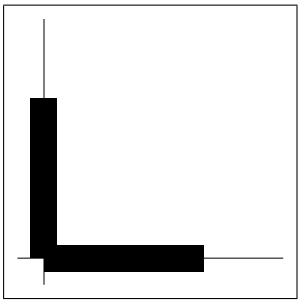

Figure 12: Axes with standard lines

If you look close to lines you will note some peculiarity. For instance the original LaTeX `\line` is in horizontal/vertical mode a simple `\ruler`.

Remark: The *line* is exactly as long as specified.

`\@wholewidth` But the plot point used by `\qbezier`, `bez123` and `ebezier` (until version 2!) is a small square which is not centered at the control points (dimension `\@wholewidth`)

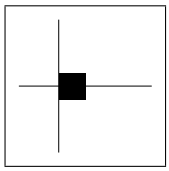

Figure 13: Old plot symbol

which results in a shifted y-axis and *lines* which are actually longer by an amount of one square (i.e. `\@wholewidth`)

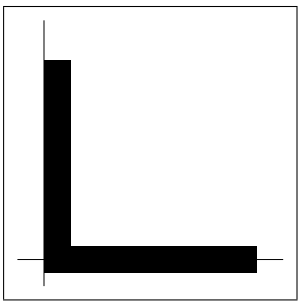

Figure 14: Axes with old plot symbol

or with hollow squares (• references to the end points).

This version uses centered plot symbols (standard is again a square)

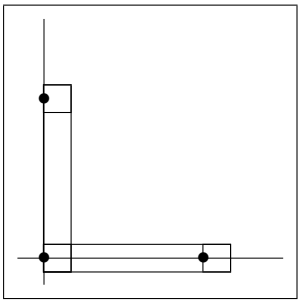

Figure 15: Axes with old plot symbol (hollow)

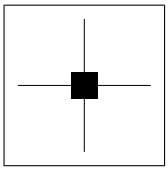

Figure 16: New standard plot symbol

which corrects the shift of the y-axis. The line is again longer but this time the excess is symmetrically on both ends

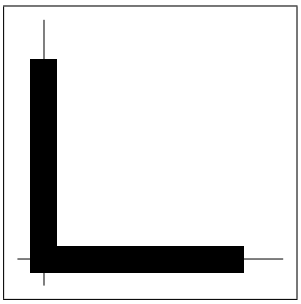

Figure 17: Axes with new standard plot symbol

or again with hollow squares.

`\DefOldPlotSymbol` `\Qbezier` To be consistent with the old version the command `\DefOldPlotSymbol` is supplied which uses the old form. Also a variant `\Qbezier` for `\qbezier` is defined which can use the new plot symbol.¶ ‖

The next point of consideration is the handling of slanted lines. In the

¶This command is just for convenience. A quadratic bezier can be plotted as cubic bezier as follows. If you want to plot `\qbezier[100](z1)(zm)(z4)` with $(z) = (x, y)$ you may calulate points $z_2 = 2/3[z_m, z_1]$ and $z_3 = 2/3[z_m, z_4]$. The cubic bezier `\cbezier[100](z1)(z2)(z3)(z4)` is exactly the same as the quadratic one!

‖It can also use the other new symbols defined later.

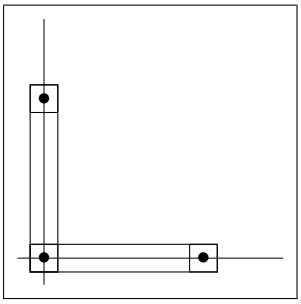

Figure 18: Axes with new standard plot symbol (hollow)

ordinary LaTeX-`picture` environment `\linethickness` has no effect on slanted lines. Now the change applies but a new problem occurs. If you plot a slanted line (slope angle φ) with squares

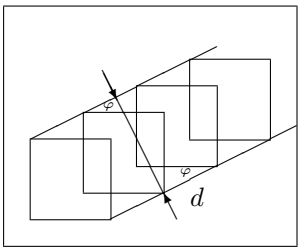

Figure 19: Effective thickness for slanted lines

your line gets effective thicker! The factor of enlargement is $\sin\varphi + \cos\varphi$ which has its maximum $\sqrt{2}$ with slope $\varphi_0 = 45^0$.

There are two possiblities to correct the thickness

- correct the line thickness of each line or
- use other plot symbols which behave better.

`\Lbezier` The first possibilitiy can be realized just for `\lbezier` and not `\cbezier` because the slope changes from point to point in the latter case. The solution is established by internally changing the `\linethickness` by the factor $\ell/(\Delta x + \Delta y)$ where ℓ denotes the length of the line ($= \sqrt{\Delta^2 x + \Delta^2 y}$) and Δx is the horizontal difference of the the points (Δy respectivelly for the vertical difference).

To use this line type call `\Lbezier[n](x1,y1)(x2,y2)`.

The second chance is to change the plot symbol to a disc. The smallest disk available is the character "." at 5pt. Unfortunately this method will implicitely restrict the `\linethickness` to some definite values (see the following table for the numbers in question).

Table 2: Dimensions for various plot symbols

Font	Size for (10pt)	Width	Heigth	Rule
`\vrm`	tiny	2.01392pt	0.61111pt	
`\virm`	tiny for 11/12pt	2.11108pt	0.72223pt	
`\viirm`	scriptsize	2.2639pt	0.80556pt	
`\viiirm`	footnotesize	2.36115pt	0.88889pt	
`\ixrm`	small	2.56943pt	0.97223pt	
`\xrm`	normalsize	2.77779pt	1.05554pt	
`\xirm`	normalsize 11pt	3.04167pt	1.15582pt	
`\xiirm`	large	3.26385pt	1.16666pt	
`\xivrm`	Large	4.0pt	1.51999pt	
`\xviirm`	LARGE	4.31383pt	1.41667pt	
`\xxrm`	huge	5.76112pt	2.18921pt	
`\xxvrm`	Huge	6.91112pt	2.6262pt	
`$\bullet$`		5.00002pt	4.44444pt	

`\DefPlotSymbol` With the aim of the command `\DefPlotSymbol{item}` you may define any *item* as your plot symbol**. It will be centered as the default plot square (otherwise an even larger shift of the y-axis would occur). Use explicit font selection with the names supplied in the table to ensure style independence (otherwise e.g. `\DefPlotSymbol{\tiny .}` would be different in 10pt and 11pt context).

`\DefShiftedPlotSymbol` This works for all *items* which have a vertical symmetry axis with respect to their defining bounding box (defined by METAFONT) and which ly on the baseline (or beyond if they have some defined depth). It will not work otherwise. For example the circles from the circle font have heigth and depth zero and their reference point is already the centre (i.e. the symbol extends backward). Or consider the "*"-sign. It does not ly on the baseline. For these cases a generalized command is supplied:
`\DefShiftedPlotSymbol{item}{x-shift}{y-shift}{height}`.

The shifts are applied to the left and downward. The supplied heigth will only have effect if you specify $n = 0$ for the number of plotting points.

Examples:

```
\DefShiftedPlotSymbol{\tencirc n}{0pt}{0pt}{1pt}
\DefShiftedPlotSymbol{\tencirc \char'176}{0pt}{0pt}{15pt}
\DefShiftedPlotSymbol{\fbox{\Huge *}}{0pt}{0pt}{25pt}
```

**A similar approach with centered symbols can be found in the packages epic[6] and PiCTEX[7].

```
%with bounding box
\setbox0=\hbox{*}
\DefShiftedPlotSymbol{*}{.5\wd0}{.7\ht0}{.6\ht0}
\lbezier[1](100,30)(100,30)
\lbezier[0](0,20)(125,20)
\DefShiftedPlotSymbol{*}{.5\wd0}{.7\ht0}{10\ht0}
\lbezier[0](0,10)(125,10)
```

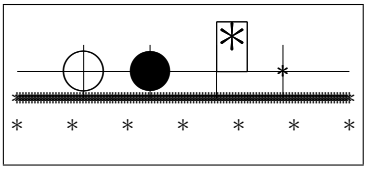

Figure 20: Examples for other plot symbols

Caution: The commands for changing the line thickness have implicit effects for plot symbols defined with `\DefPlotSymbol{item}` or `\DefShiftedPlotSymbol`. The implicit or explicit defined height is redefined. But the effect is only visible in case $n = 0$.

`\DefStandardPlotSymbol` In any case you may restore **default values** by stating

```
\DefStandardPlotSymbol
\thinlines
```

8 Estimation for the number of plotting points

As mentioned in the last section all plotting macros will calculate the number of plotting points if the value $n = 0$ is active. All calculations will use the actual length of the object which can be calculated with the aim of the calculation macros in Section *Calculating Lengths*. For all these calculations `\eps` is temporarily increased by a factor of 10 and for higher bezier curves just 5 intermediate points will be used. If the calculated number exceeds the specified maximum `\qbeziermax` an info in the log-file will be generated.

All macros for circles and arcs will use a simpler estimate due to their construction by an intrinsic call of the cubic bezier. It uses the length of the chord and the maximal deviation factor $\pi/2$ from the arc length.

9 Joining linear beziers

`\ljoin` A further command has been supplied to ease the drawing of polygon paths.

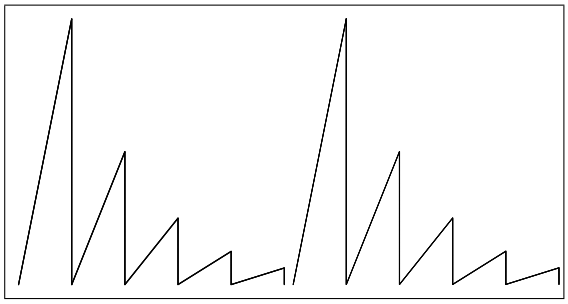

Figure 21: \ljoin versus \Ljoin

Instead of writing a sequence of \lbeziers with common vertices you can write compactly \ljoin[n](x1,y1)(x2,y2)(x3,y3)...(xm,ym)

Caution: There should be no spaces in the command, so break lines with % if necessary. There should be at least 2 points. The parameter n is optional, internally \lbezier[n](xk,yk)(xk+1,yk+1) will be executed.

\Ljoin There is also a variant \Ljoin which uses \Lbezier.

Versions

This is Version 4 from March 1, 2004.

Changes with regard to version 3:

- Bug-address changed.
- Error in defining (first) equation corrected (thanks to jens.schwaiger@uni-graz.at).
- Marginal corrections with regard to l2tabu (v1.8).
- Documentaion as pdf supplied.

Changes with regard to version 2:

- Implementing line thickness (\thinlines, \thicklines, and \setlength{\linethickness}{dimen}.
- Different plot symbols.
- \Lbezier for equally thick lines in all directions.
- \Qbezier implementation to be used with new plot symbols.
- Calculation of an optimal number of plot symbols (as default number for case n=0).

- Parameter n is for all *plot* commands optional.
- New macro for polygon paths.
- Style supplied in dtx-format.
- Minor style changes regarding numbers and lengths.

Changes with regard to Version 1:

- `\@tempa` replaced by `\@TempDim`. `\@tempa` was also used by other packages.
- Additionaly supplied `\RequirePackage{calc}`.
- Bug fixed for circles. The original macros did actually not support changes in `\unitlength`.

References

[1] D. E. Knuth: *The* TEX *Book*, Addison-Wesley, Reading MA, 1986.

[2] D. E. Knuth: *The* METAFONT *Book*, Addison-Wesley, Reading MA, 1986.

[3] M. Goossens, F. Mittelbach, A. Samarin: *The* LATEX *Companion*, Addison-Wesley, Reading MA, 1994.

[4] M. Goossens, S. Rahtz, F. Mittelbach: *The* LATEX *Graphics Companion*, Addison-Wesley, Reading MA, 1997.

[5] P. Wilson: *The* bez123 *and* multiply *packages*, 1998;
packages at CTAN/macros/latex/contrib/supported/bez123.

[6] S. Podar: *Enhancements to the Picture Environment in* LATEX, 1986;
package at CTAN/macros/latex/other/epic.

[7] M. J. Wichura: *The PiC*TEX *Manual*, 1992;
package at CTAN/graphics/pictex.

[8] R. Niepraschk, H. Gaesslein: The pict2e Package, 2003;
package at CTAN/macros/latex/contrib/pict2e.

[9] N. J. H. M. van Beurden: A LATEX picture editor for Windows, 2003;
package at CTAN/systems/win32/latexpix.

10 Implementation

The macros `\lbezier` and `\cbezier` are rather old, they existed since I realized the existence of bezier.sty more then ten years ago. Therefore the macros are written rather in pure TEX than in LATEX. Only the calculation macros demand for LATEX notation to use the package calc. But with this version the macros interact more and some LATEX part occurrs also in the plot macros.

```
⟨*package⟩
\NeedsTeXFormat{LaTeX2e}
\RequirePackage{calc}
%%
```

I define new font names because `cmr` may not be the standard font. They may be needed for plotting symbols.

```
\newfont{\vrm}{cmr5}
\newfont{\virm}{cmr6}
\newfont{\viirm}{cmr7}
\newfont{\viiirm}{cmr8}
\newfont{\ixrm}{cmr9}
\newfont{\xrm}{cmr10}
\newfont{\xiirm}{cmr12}
\newfont{\xviirm}{cmr17}
\newfont{\xirm}{cmr10 scaled \magstephalf}
\newfont{\xivrm}{cmr10 scaled \magstep2}
\newfont{\xxrm}{cmr10 scaled \magstep4}
\newfont{\xxvrm}{cmr10 scaled \magstep5}
%%
```

I need only three new counters,

```
\newcounter{@cnta}\newcounter{@cntb}\newcounter{@cntc}\newcounter{@cntd}
%%
```

but a lot of lengths. Packages like PiCTEX have problems by defining too many lengths, so I try to use as many already defined lengths (defined for usage in a plotting context).

```
%% \@TempDim#1#2#3{"count"|"dimen"|"box"|"skip"}{\myname}{\realname}
%%  allocate new one or alias is defined, so use it
%%
\def\@TempDim#1#2#3{%
  \ifx\@und@fined#3\csname new#1\endcsname#2%
  \else\let#2#3\fi}
%%
\@TempDim{dimen}\@X\@ovxx
\@TempDim{dimen}\@Xa\@ovdx
\@TempDim{dimen}\@Xb\@ovyy
\@TempDim{dimen}\@Xc\@ovdy
\@TempDim{dimen}\@Y\@ovro
\@TempDim{dimen}\@Ya\@ovri
```

```
\@TempDim{dimen}\@Yb\@xdim
\@TempDim{dimen}\@Yc\@ydim
\@TempDim{dimen}\@Z\@clnht
\@TempDim{dimen}\@Za\@clnwd
\@TempDim{dimen}\@Zb\@dashdim
\@TempDim{dimen}\@Zc\@tempdima
\@TempDim{dimen}\@Zd\@tempdimb
\@TempDim{dimen}\@Ze\@tempdimc
%%
\newlength{\@Zf}\newlength{\@Zg}\newlength{\@Zh}
\newlength{\@Zi}\newlength{\@Zj}
```

This special length will be used for the circle macros. The magic number is $0.55228474983 = 4/3(\sqrt{2} - 1)$.

```
\newlength{\magicnum}
\newcommand\set@magic{%
 \setlength{\magicnum}{0.55228474983\unitlength}}
%%
```

Another special one is `\eps`. It could be initialized by `\eps\@ne` but due to its context to the calculation part 1sp=1/65536pt is used.

```
\newlength{\eps}
\setlength{\eps}{1sp}
%%
```

The last one is `\PathLength`. It stores lengths which the user may need for further use.

```
\newlength{\PathLength}
%%
```

This two constants are needed in calculations, but I did not want to waste any additional counter. `\pathmax` may be redefined to exceed 256, so it is not defined by `\chardef`.

```
\chardef\x@=10
\newcommand{\pathmax}{50}
%%
```

This fundamental box will keep the plotting symbol.

```
\newsavebox{\@pt}
%%
```

I have to distinguish three cases: standard plot symbol, old standard plot symbol, or any new one. For this purpose I need two logicals.

```
\newif\if@other@symbol
\newif\if@standard@symbol
```

All plot symbols may be defined by the most general one, `\DefShiftedPlotSymbol`, but this way may be faster. The other important macro is `\set@width` which redefines the plot box due to changes which may have occurred (line thickness).

```
\newcommand{\DefStandardPlotSymbol}{%
 \@other@symbolfalse\@standard@symboltrue
 \setbox\@pt\hbox{\hskip -.5\wd0\vrule height\@halfwidth
 depth\@halfwidth width\@wholewidth}}
\newcommand{\DefOldPlotSymbol}{%
 \@other@symbolfalse\@standard@symbolfalse
 \setbox\@pt\hbox{\vrule height\@halfwidth
 depth\@halfwidth width\@wholewidth}}
\newcommand{\DefPlotSymbol}[1]{\setbox0=\hbox{#1}\@X\ht0\advance\@X-\dp0
 \@halfwidth.5\ht0\@wholewidth\ht0
 \@other@symboltrue\@standard@symbolfalse
 \setbox\@pt\hbox{\hskip -.5\wd0\lower.5\@X\copy0}}
\newcommand{\DefShiftedPlotSymbol}[4]{\setbox0=\hbox{#1}\@X #2\@Y #3
 \@wholewidth #4\@halfwidth.5\@wholewidth
 \@other@symboltrue\@standard@symbolfalse
 \setbox\@pt\hbox{\hskip-\@X\lower\@Y\copy0}}
\newcommand{\set@width}{%
 \if@other@symbol
  \relax
 \else
  \if@standard@symbol
    \@X-.5\@wholewidth
  \else
    \@X\z@
  \fi
  \setbox\@pt\hbox{\hskip\@X\vrule height\@halfwidth
  depth\@halfwidth width\@wholewidth}%
 \fi}
%%
```

The initialization is done here. Note that `\thinlines` is already default and needs not be specified here.

```
\DefStandardPlotSymbol
%%
```

All plot macros have an optional number. Therefore an additional internal macro is needed (it will have the same name with an extra @ in front of it.

Here is the simpliest one, the linear case.

```
\def\lbezier{\@ifnextchar [{\@lbezier}{\@lbezier[0]}}
\def\@lbezier[#1](#2,#3)(#4,#5){%
 \c@@cntc#1\relax
 \ifnum \c@@cntc<\@ne
```

I decrease the precision locally to speed up calculations. We need just an estimate.

```
  \multiply\eps\x@
  \Length(#2,#3)(#4,#5){\PathLength}%
  \divide\eps\x@
  \c@@cntc\PathLength
  \@X.5\@halfwidth \divide\c@@cntc\@X
  \ifnum \c@@cntc>\qbeziermax%
    \PackageInfo{ebezier}{\the\c@@cntc\space points needed exceeding %
                          qbeziermax=\qbeziermax!}\fi
\fi
\ifnum \c@@cntc>\qbeziermax
\c@@cntc\qbeziermax\relax
  \PackageWarning{ebezier}{Counter reset to qbeziermax=\qbeziermax!}\fi
\c@@cnta\c@@cntc\relax\advance\c@@cnta\@ne
\@Xa #4\unitlength \advance\@Xa-#2\unitlength \divide\@Xa\c@@cntc
\@Ya #5\unitlength \advance\@Ya-#3\unitlength \divide\@Ya\c@@cntc
\c@@cntb\z@\relax
\set@width
\put(#2,#3){\@whilenum{\c@@cntb<\c@@cnta}\do
   {\@X\c@@cntb \@Xa\@Y \c@@cntb\@Ya
   \raise\@Y\hbox to\z@{\hskip\@X\unhcopy\@pt\hss}%
   \advance\c@@cntb\@ne}}}
%%
```

`\Lbezier` changes the line thickness. It is stored in `\@Xb`.

```
\def\Lbezier{\@ifnextchar [{\@Lbezier}{\@Lbezier[0]}}
\def\@Lbezier[#1](#2,#3)(#4,#5){\c@@cntc#1\relax
 \@Xb\@wholewidth
 \@X #4\unitlength \advance\@X-#2\unitlength \AbsLen{\@X}%
 \@Y #5\unitlength \advance\@Y-#3\unitlength \AbsLen{\@Y}%
 \LenNorm{\@X}{\@Y}{\@Xc}\LenMult{\@Xc}{\@wholewidth}{\@Yb}%
 \LenDiv{\@Yb}{\@X+\@Y}{\@wholewidth}\@halfwidth .5\@wholewidth
 \ifnum \c@@cntc<\@ne
  \multiply\eps\x@
  \Length(#2,#3)(#4,#5){\PathLength}%
  \divide\eps\x@
  \c@@cntc\PathLength
  \@X.5\@halfwidth \divide\c@@cntc\@X
  \ifnum \c@@cntc>\qbeziermax%
    \PackageInfo{ebezier}{\the\c@@cntc\space points needed exceeding %
                          qbeziermax=\qbeziermax!}\fi
 \fi
 \ifnum \c@@cntc>\qbeziermax
 \c@@cntc\qbeziermax\relax
    \PackageWarning{ebezier}{Counter reset to qbeziermax=\qbeziermax!}\fi
 \c@@cnta\c@@cntc\relax \advance\c@@cnta\@ne
 \@Xa #4\unitlength \advance\@Xa-#2\unitlength \divide\@Xa\c@@cntc
 \@Ya #5\unitlength \advance\@Ya-#3\unitlength \divide\@Ya\c@@cntc
```

```
\c@@cntb\z@\relax
\set@width
\put(#2,#3){\@whilenum{\c@@cntb<\c@@cnta}\do
   {\@X\c@@cntb \@Xa\@Y \c@@cntb\@Ya
   \raise\@Y\hbox to\z@{\hskip\@X\unhcopy\@pt\hss}%
   \advance\c@@cntb\@ne}}
\@wholewidth\@Xb \@halfwidth .5\@Xb}
%%
```

The two joining macros need two internal steps to process an implicit list.

```
\def\ljoin{\@ifnextchar [{\@ljoin}{\@ljoin[0]}}
\def\@ljoin[#1](#2,#3){\@ifnextchar ({\l@join[#1](#2,#3)}{\relax}}
\def\l@join[#1](#2,#3)(#4,#5){%
 \lbezier[#1](#2,#3)(#4,#5)%
 \ljoin[#1](#4,#5)}
%%
\def\Ljoin{\@ifnextchar [{\@Ljoin}{\@Ljoin[0]}}
\def\@Ljoin[#1](#2,#3){\@ifnextchar ({\L@join[#1](#2,#3)}{\relax}}
\def\L@join[#1](#2,#3)(#4,#5){%
 \Lbezier[#1](#2,#3)(#4,#5)%
 \Ljoin[#1](#4,#5)}
%%
```

`\Qbezier` is defined, because `\qbezier` uses an other plot box. The original macro is a little bit more complicated to handle extra spaces but I hope this will suffice.

```
\def\Qbezier{\@ifnextchar [{\@Qbezier}{\@Qbezier[0]}}
\def\@Qbezier[#1](#2,#3)(#4,#5)(#6,#7){\c@@cntc#1\relax
 \ifnum \c@@cntc<\@ne
  \multiply\eps\x@
  \PathLengthQ[5](#2,#3)(#4,#5)(#6,#7)%
  \divide\eps\x@
  \c@@cntc\PathLength
  \@X.5\@halfwidth \divide\c@@cntc\@X
  \ifnum \c@@cntc>\qbeziermax%
    \PackageInfo{ebezier}{\the\c@@cntc\space points needed exceeding %
                          qbeziermax=\qbeziermax!}\fi
 \fi
 \ifnum \c@@cntc>\qbeziermax
 \c@@cntc\qbeziermax\relax
    \PackageWarning{ebezier}{Counter reset to qbeziermax=\qbeziermax!}\fi
 \c@@cnta\c@@cntc\relax \advance\c@@cnta\@ne
 \@Xa #4\unitlength \advance\@Xa-#2\unitlength \multiply\@Xa\tw@
 \@Xb #6\unitlength \advance\@Xb-#2\unitlength
 \advance\@Xb-\@Xa \divide\@Xb\c@@cntc
 \@Ya #5\unitlength \advance\@Ya-#3\unitlength \multiply\@Ya\tw@
 \@Yb #7\unitlength \advance\@Yb-#3\unitlength
 \advance\@Yb-\@Ya \divide\@Yb\c@@cntc
 \c@@cntb\z@\relax
```

```
\set@width
\put(#2,#3){\@whilenum{\c@@cntb<\c@@cnta}\do
    {\@X\c@@cntb \@Xb\@Y \c@@cntb\@Yb
     \advance\@X\@Xa \advance\@Y\@Ya
     \divide\@X\c@@cntc \divide\@Y\c@@cntc
     \multiply\@X\c@@cntb \multiply\@Y\c@@cntb
     \raise \@Y \hb@xt@\z@{\kern\@X\unhcopy\@pt\hss}%
     \advance\c@@cntb\@ne}}}
%%
```

\cbezier is the most complex command. All calculations have to be done in the correct order to minimize overflow conditions.

```
\def\cbezier{\@ifnextchar [{\@cbezier}{\@cbezier[0]}}
\def\@cbezier[#1](#2,#3)(#4,#5)(#6,#7)(#8,#9){%
 \c@@cntc#1\relax
 \ifnum \c@@cntc<\@ne
  \multiply\eps\x@
  \PathLengthC[5](#2,#3)(#4,#5)(#6,#7)(#8,#9)%
  \divide\eps\x@
  \c@@cntc\PathLength
  \@X = 0.5\@halfwidth
  \divide\c@@cntc\@X
  \ifnum \c@@cntc>\qbeziermax%
    \PackageInfo{ebezier}{\the\c@@cntc\space  points needed exceeding %
                          qbeziermax=\qbeziermax!}\fi
 \fi
 \ifnum \c@@cntc>\qbeziermax
 \c@@cntc\qbeziermax\relax
   \PackageWarning{ebezier}{Counter reset to qbeziermax=\qbeziermax!}\fi
  \c@@cnta\c@@cntc\relax \advance\c@@cnta\@ne
  \@Xa #4\unitlength \advance\@Xa-#2\unitlength \multiply\@Xa\thr@@
  \@Xb #6\unitlength \advance\@Xb-#2\unitlength \multiply\@Xb\thr@@
  \advance\@Xb -2\@Xa
  \@Xc #8\unitlength \advance\@Xc-#2\unitlength
  \advance\@Xc-\@Xa \advance\@Xc-\@Xb
  \@Ya #5\unitlength \advance\@Ya-#3\unitlength \multiply\@Ya\thr@@
  \@Yb #7\unitlength \advance\@Yb-#3\unitlength \multiply\@Yb\thr@@
  \advance\@Yb-2\@Ya
  \@Yc #9\unitlength \advance\@Yc-#3\unitlength
  \advance\@Yc-\@Ya \advance\@Yc-\@Yb
  \divide\@Xc\c@@cntc \divide\@Yc\c@@cntc
  \c@@cntb\z@\relax
 \set@width
  \put(#2,#3){\@whilenum{\c@@cntb<\c@@cnta}\do
      {\@X\c@@cntb \@Xc\@Y \c@@cntb\@Yc
      \advance\@X\@Xb \advance\@Y\@Yb
      \divide\@X\c@@cntc \divide\@Y\c@@cntc
      \multiply\@X\c@@cntb \multiply\@Y\c@@cntb
      \advance\@X\@Xa  \advance\@Y\@Ya
```

```
      \divide\@X\c@@cntc \divide\@Y\c@@cntc
      \multiply\@X\c@@cntb \multiply\@Y\c@@cntb
      \raise \@Y \hbox to \z@{\hskip \@X\unhcopy\@pt\hss}%
      \advance\c@@cntb\@ne}}}
%%
```

`\Cbezier` changes the plot symbol so a restore is needed. But it will not keep the original one!

```
\def\Cbezier{\@ifnextchar [{\@Cbezier}{\@Cbezier[0]}}
\def\@Cbezier[#1](#2,#3)(#4,#5)(#6,#7)(#8,#9){%
 \cbezier[#1](#2,#3)(#4,#5)(#6,#7)(#8,#9)%
 \c@@cntc#1\relax\divide\c@@cntc\thr@@
 \lbezier[\c@@cntc](#2,#3)(#4,#5)%
 \lbezier[\c@@cntc](#4,#5)(#6,#7)%
 \lbezier[\c@@cntc](#6,#7)(#8,#9)%
 \DefPlotSymbol{$\bullet$}
 \lbezier[1](#2,#3)(#2,#3)
 \lbezier[1](#4,#5)(#4,#5)
 \lbezier[1](#6,#7)(#6,#7)
 \lbezier[1](#8,#9)(#8,#9)
 \DefStandardPlotSymbol
 \thinlines}
%%
```

`\l@put` is like `\put` but its arguments are lengths and not decimal constants. It will be used in `\l@cbezier` which also has lengths as arguments. All complex plotting commands use this form. Just for the calculation of plotting points four more lengths are needed. I use the "scratch" dimens from TEX.

```
\long\gdef\l@put(#1,#2)#3{%
 \@killglue\raise#2\hb@xt@\z@{\kern#1#3\hss}\ignorespaces}
%%
\long\gdef\l@cbezier[#1](#2,#3)(#4,#5)(#6,#7)(#8,#9){%
 \c@@cntc#1\relax
 \dimen1#2\dimen3#3
%%
 \@Xa #4 \advance\@Xa-#2 \multiply\@Xa\thr@@
 \@Xb #6 \advance\@Xb-#2 \multiply\@Xb\thr@@
 \advance\@Xb-2\@Xa
 \@Xc #8 \advance\@Xc-#2
 \advance\@Xc-\@Xa \advance\@Xc-\@Xb
 \@Ya #5 \advance\@Ya-#3 \multiply\@Ya\thr@@
 \@Yb #7 \advance\@Yb-#3 \multiply\@Yb\thr@@
 \advance\@Yb-2\@Ya
 \@Yc #9 \advance\@Yc-#3
 \advance\@Yc-\@Ya \advance\@Yc-\@Yb
%%
%% assume half arc
%%
 \ifnum \c@@cntc <\@ne
```

```
  \multiply\eps\x@
  \dimen5#2 \advance\dimen5-#8 \AbsLen{\dimen5}%
  \dimen7#3 \advance\dimen7-#9 \AbsLen{\dimen7}%
  \LenNorm{\dimen5}{\dimen7}{\PathLength}%
  \divide\eps\x@
  \c@@cntc\PathLength
  \dimen5.5\@halfwidth
  \divide\c@@cntc\dimen5
%%
%%  11/7 \approx \pi/2
%%
  \divide\c@@cntc 7 \multiply\c@@cntc 11
  \ifnum \c@@cntc>\qbeziermax
    \PackageInfo{ebezier}{\the\c@@cntc\space points needed exceeding %
                          qbeziermax=\qbeziermax!}\fi
\fi
\ifnum\c@@cntc>\qbeziermax
  \c@@cntc\qbeziermax\relax
    \PackageWarning{ebezier}{Counter reset to qbeziermax=\qbeziermax!}\fi
\c@@cnta\c@@cntc\relax\advance\c@@cnta\@ne%
\divide\@Xc\c@@cntc \divide\@Yc\c@@cntc
\c@@cntb\z@\relax
\set@width
\l@put(\dimen1,\dimen3){\@whilenum{\c@@cntb<\c@@cnta}\do
     {\@X\c@@cntb \@Xc\@Y \c@@cntb\@Yc
     \advance\@X\@Xb \advance\@Y\@Yb
     \divide\@X\c@@cntc \divide\@Y\c@@cntc
     \multiply\@X\c@@cntb \multiply\@Y\c@@cntb
     \advance\@X\@Xa \advance\@Y\@Ya
     \divide\@X\c@@cntc \divide\@Y\c@@cntc
     \multiply\@X\c@@cntb \multiply\@Y\c@@cntb
     \raise\@Y\hbox to\z@{\hskip\@X\unhcopy\@pt\hss}%
     \advance\c@@cntb\@ne}}}
%%
```

The building blocks for the circles are the four quarters. Each is defined separately and will be combined by the `\cCircle` macro.

```
\def\@circle@rt[#1](#2,#3)#4{%
 \set@magic
 \@Z #4\magicnum\@Za #2\unitlength\@Zb #3\unitlength
 \@Zc #2\unitlength \advance\@Zc\@Z
 \@Zd #3\unitlength \advance\@Zd\@Z
 \@Ze #4\unitlength \advance\@Ze\@Za
 \@Zf #4\unitlength \advance\@Zf\@Zb
 \l@cbezier[#1](\@Ze,\@Zb)(\@Ze,\@Zd)(\@Zc,\@Zf)(\@Za,\@Zf)}
%%
\def\@circle@lt[#1](#2,#3)#4{%
 \set@magic
 \@Z #4\magicnum\@Za #2\unitlength\@Zb #3\unitlength
```

```
 \@Zc #2\unitlength \advance\@Zc-\@Z
 \@Zd #3\unitlength \advance\@Zd\@Z
 \@Ze -#4\unitlength \advance\@Ze\@Za
 \@Zf #4\unitlength \advance\@Zf\@Zb
 \l@cbezier[#1](\@Za,\@Zf)(\@Zc,\@Zf)(\@Ze,\@Zd)(\@Ze,\@Zb)}
%%
\def\@circle@rb[#1](#2,#3)#4{%
 \set@magic
 \@Z #4\magicnum\@Za #2\unitlength\@Zb #3\unitlength
 \@Zc #2\unitlength \advance\@Zc\@Z
 \@Zd #3\unitlength \advance\@Zd-\@Z
 \@Ze #4\unitlength \advance\@Ze\@Za
 \@Zf -#4\unitlength \advance\@Zf\@Zb
 \l@cbezier[#1](\@Za,\@Zf)(\@Zc,\@Zf)(\@Ze,\@Zd)(\@Ze,\@Zb)}
%%
\def\@circle@lb[#1](#2,#3)#4{%
 \set@magic
 \@Z #4\magicnum\@Za #2\unitlength\@Zb #3\unitlength
 \@Zc #2\unitlength \advance\@Zc-\@Z
 \@Zd #3\unitlength \advance\@Zd-\@Z
 \@Ze -#4\unitlength \advance\@Ze\@Za
 \@Zf -#4\unitlength \advance\@Zf\@Zb
 \l@cbezier[#1](\@Ze,\@Zb)(\@Ze,\@Zd)(\@Zc,\@Zf)(\@Za,\@Zf)}
%%
```

I use the logicals from the `\oval` defined in LaTeX. So I need just one more logical `\if@ovf`.

```
\newif\if@ovf
\def\cCircle{\@ifnextchar [{\@cCircle}{\@cCircle[0]}}
\def\@cCircle[#1](#2,#3)#4[#5]{%
 \@ovtfalse\@ovbfalse\@ovlfalse\@ovrfalse\@ovffalse
 \c@@cnta#1\relax
 \@tfor\reserved@a:=#5\do{\csname @ov\reserved@a true\endcsname}%
 \if@ovf\@ovttrue \divide\c@@cnta\tw@\fi
 \if@ovt
  \if@ovr
     \@circle@rt[\c@@cnta](#2,#3){#4}
  \else\if@ovl
     \@circle@lt[\c@@cnta](#2,#3){#4}
  \else\divide\c@@cnta\tw@
     \@circle@rt[\c@@cnta](#2,#3){#4}
     \@circle@lt[\c@@cnta](#2,#3){#4}
  \fi\fi
  \if@ovf
     \@circle@rb[\c@@cnta](#2,#3){#4}
     \@circle@lb[\c@@cnta](#2,#3){#4}
  \fi
  \else\if@ovb
  \if@ovr
```

```
    \@circle@rb[\c@@cnta](#2,#3){#4}
  \else\if@ovl
    \@circle@lb[\c@@cnta](#2,#3){#4}
  \else\divide\c@@cnta\tw@
    \@circle@rb[\c@@cnta](#2,#3){#4}
    \@circle@lb[\c@@cnta](#2,#3){#4}
  \fi\fi
 \else
   \divide\c@@cnta\tw@
   \if@ovr
    \@circle@rb[\c@@cnta](#2,#3){#4}
    \@circle@rt[\c@@cnta](#2,#3){#4}
  \else
  \if@ovl
    \@circle@lb[\c@@cnta](#2,#3){#4}
    \@circle@lt[\c@@cnta](#2,#3){#4}
  \else
   \PackageError{Ebezier}{Missing or illegal position specifier in cCircle}
  \fi\fi\fi\fi}
%%
\def\cArc{\@ifnextchar [{\@cArc}{\@cArc[0]}}
\def\@cArc[#1](#2,#3)(#4,#5){%
 \c@@cntc#1\relax
 \@X #2\unitlength \@Y #3\unitlength
 \@Za #4\unitlength \@Zb #5\unitlength
 \@Zc 2\@X \advance\@Zc-\@Za \@Zd 2\@Y \advance\@Zd-\@Zb
 \@Xa\@Y \advance\@Xa-\@Zb \@Ya\@Za \advance\@Ya-\@X
 \multiply\@Xa 4 \divide\@Xa\thr@@ \multiply\@Ya 4 \divide\@Ya\thr@@
 \@Ze\@Za \advance\@Ze\@Xa \@Zf\@Zb \advance\@Zf\@Ya
 \@Zg\@Zc \advance\@Zg\@Xa \@Zh\@Zd \advance\@Zh\@Ya
 \l@cbezier[#1](\@Za,\@Zb)(\@Ze,\@Zf)(\@Zg,\@Zh)(\@Zc,\@Zd)}
%%
```

Historically from this point starts the calculation part. The notation will be more LaTeX convenient.

All square roots are calculated by the same iteration. To keep numbers small enough some scaling has to be done (factor `\c@@cntd`).

```
\def\LenMult#1#2#3{\setlength{#3}{#1*\ratio{#2}{\unitlength}}}
%%
\def\LenDiv#1#2#3{\setlength{#3}{\unitlength*\ratio{#1}{#2}}}
%%
\def\AbsLen#1{\ifdim#1<\z@\setlength{#1}{-#1}\fi}
%%
\def\LenSqrt#1#2{%
 \setlength{\@Za}{#1}%
 \ifdim\@Za>\eps\loop\setlength{\@Zb}{(\@Za+\unitlength*\ratio{#1}{\@Za})/2}%
 \setlength{\@Zc}{\@Za-\@Zb}\AbsLen{\@Zc}%
 \ifdim\@Zc>\eps\setlength{\@Za}{\@Zb}\repeat\fi%
 \setlength{#2}{\@Za}}
```

```
%%
\def\Length(#1,#2)(#3,#4)#5{%
 \setlength{\@Zd}{#3\unitlength-#1\unitlength}%
 \setlength{\@Ze}{#4\unitlength-#2\unitlength}%
 \setcounter{@cntd}{1}%
 \setlength{\@Zf}{\@Zd}\ifdim\@Ze>\@Zd\setlength{\@Zf}{\@Ze}\fi
 \loop\setlength{\@Zd}{\@Zd/2}\setlength{\@Ze}{\@Ze/2}\setlength{\@Zf}{\@Zf/2}%
  \multiply\c@@cntd\tw@\ifdim\@Zf>\x@ pt\repeat
 \LenMult{\@Zd}{\@Zd}{\@Zg}\LenMult{\@Ze}{\@Ze}{\@Zh}\setlength{\@Zf}{\@Zg+\@Zh}%
 \LenSqrt{\@Zf}{\@Zg}\setlength{#5}{\@Zg*\value{@cntd}}}
%%
\def\LenNorm#1#2#3{%
 \setlength{\@Zd}{#1}\setlength{\@Ze}{#2}\setcounter{@cntd}{1}%
 \setlength{\@Zf}{\@Zd}\ifdim\@Ze>\@Zd\setlength{\@Zf}{\@Ze}\fi
 \loop\setlength{\@Zd}{\@Zd/2}\setlength{\@Ze}{\@Ze/2}\setlength{\@Zf}{\@Zf/2}%
  \multiply\c@@cntd\tw@\ifdim\@Zf>\x@ pt\repeat
 \LenMult{\@Zd}{\@Zd}{\@Zg}\LenMult{\@Ze}{\@Ze}{\@Zh}\setlength{\@Zf}{\@Zg+\@Zh}%
 \LenSqrt{\@Zf}{\@Zg}\setlength{#3}{\@Zg*\value{@cntd}}}
%%
\def\PathLengthQ[#1](#2,#3)(#4,#5)(#6,#7){%
 \PathLength\z@\c@@cntc#1\relax
 \ifnum \c@@cntc<\@ne \c@@cntc\pathmax\relax\fi
 \ifnum \c@@cntc>\pathmax \c@@cntc\pathmax\relax
    \PackageWarning{ebezier}{Counter reset to pathmax=\pathmax!}\fi
 \@Za\z@ \@Zb\z@ \c@@cntb\c@@cntc\relax \advance\c@@cntb\@ne
 \@Xb #4\unitlength \advance\@Xb-#2\unitlength \multiply\@Xb\tw@
 \@Yb #5\unitlength \advance\@Yb-#3\unitlength \multiply\@Yb\tw@
 \@Xa #6\unitlength \advance\@Xa-#2\unitlength
 \advance\@Xa-\@Xb \divide\@Xa\c@@cntc
 \@Ya #7\unitlength \advance\@Ya-#3\unitlength
 \advance\@Ya-\@Yb \divide\@Ya\c@@cntc \c@@cnta\@ne\relax
 \@whilenum{\c@@cnta<\c@@cntb}\do
  {\@X\c@@cnta\@Xa \advance\@X\@Xb \divide\@X\c@@cntc \multiply\@X\c@@cnta
  \@Y\c@@cnta\@Ya \advance\@Y\@Yb \divide\@Y\c@@cntc \multiply\@Y\c@@cnta
  \@Zi\@X\@Zj\@Y
  \advance\@X-\@Za \advance\@Y-\@Zb \LenNorm{\@X}{\@Y}{\@Z}%
  \advance\PathLength\@Z
  \@Za\@Zi\@Zb\@Zj \advance\c@@cnta\@ne}}
%%
\def\PathLengthC[#1](#2,#3)(#4,#5)(#6,#7)(#8,#9){%
  \PathLength\z@ \c@@cntc#1\relax
 \ifnum \c@@cntc<\@ne \c@@cntc\pathmax\relax\fi
 \ifnum \c@@cntc>\pathmax \c@@cntc\pathmax\relax
    \PackageWarning{ebezier}{Counter reset to pathmax=\pathmax!}\fi
 \@Za\z@ \@Zb\z@ \c@@cnta\c@@cntc\relax \advance\c@@cnta\@ne
 \@Xa #4\unitlength \advance\@Xa-#2\unitlength \multiply\@Xa\thr@@
 \@Xb #6\unitlength \advance\@Xb-#2\unitlength \multiply\@Xb\thr@@
 \advance\@Xb-2\@Xa
 \@Xc #8\unitlength \advance\@Xc-#2\unitlength
 \advance\@Xc-\@Xa \advance\@Xc-\@Xb
```

```
 \@Ya #5\unitlength \advance\@Ya-#3\unitlength \multiply\@Ya\thr@@
 \@Yb #7\unitlength \advance\@Yb-#3\unitlength \multiply\@Yb\thr@@
 \advance\@Yb-2\@Ya
 \@Yc #9\unitlength \advance\@Yc-#3\unitlength
 \advance\@Yc-\@Ya \advance\@Yc-\@Yb
 \divide\@Xc\c@@cntc \divide\@Yc\c@@cntc
 \c@@cntb\@ne\relax
 \@whilenum{\c@@cntb<\c@@cnta}\do
  {\@X\c@@cntb\@Xc \@Y\c@@cntb\@Yc \advance\@X\@Xb \advance\@Y\@Yb
  \divide\@X\c@@cntc \divide\@Y\c@@cntc
  \multiply\@X\c@@cntb \multiply\@Y\c@@cntb
  \advance\@X\@Xa \advance\@Y\@Ya
  \divide\@X\c@@cntc \divide\@Y\c@@cntc
  \multiply\@X\c@@cntb \multiply\@Y\c@@cntb
  \@Zi\@X\@Zj\@Y
  \advance\@X-\@Za \advance\@Y-\@Zb \LenNorm{\@X}{\@Y}{\@Z}%
  \advance\PathLength\@Z
  \@Za\@Zi\@Zb\@Zj\advance\c@@cntb\@ne}}
%%
```

The most complex macro is explained in the text. The exception is handled by the logical `\if@ovf`.

```
\def\cArcs{\@ifnextchar [{\@cArcs}{\@cArcs[0]}}
\def\@cArcs[#1](#2,#3)(#4,#5)(#6,#7){%
 \c@@cntc#1\relax
 \@ovffalse
 \@X#2\unitlength\@Y#3\unitlength
 \@Zi#6\unitlength\@Zj#7\unitlength
 \setlength{\@Xa}{\@X-\@Zi}\setlength{\@Ya}{\@Y-\@Zj}%
 \LenNorm{\@Xa}{\@Ya}{\@Xb}%
 \@Xa#4\unitlength \advance\@Xa\@Zi \advance\@Xa-2\@X
 \@Ya#5\unitlength \advance\@Ya\@Zj \advance\@Ya-2\@Y
 \@Xc\@Xa\AbsLen{\@Xc}\@Yc\@Ya\AbsLen{\@Yc}%
 \ifdim\@Xc<\eps\ifdim\@Yc<\eps\@ovftrue\fi\fi
 \if@ovf
  \cArc[#1](#2,#3)(#4,#5)%
 \else
  \LenNorm{\@Xa}{\@Ya}{\@Yb}%
  \setlength{\@Xc}{\unitlength*\ratio{\@Xb}{\@Yb}}%
  \setlength{\@Yc}{(-\unitlength+\@Xc*2)*4/3}%
  \@Xb-#5\unitlength \advance\@Xb\@Zj
  \@Z\@Xb\AbsLen{\@Z}%
  \ifdim\@Z<100\eps \@Xb#4\unitlength \advance\@Xb-\@Zi \@Xa\@Ya\fi
  \setlength{\@Z}{\@Yc*\ratio{\@Xa}{\@Xb}}%
  \@Xa#4\unitlength\@Ya#5\unitlength
  \setlength{\@Za}{\@Y-\@Ya}\setlength{\@Zb}{\@Xa-\@X}%
  \setlength{\@Zc}{\@Zj-\@Y}\setlength{\@Zd}{\@X-\@Zi}%
  \@Xb\@Xa \LenMult{\@Z}{\@Za}{\@Zh}\advance\@Xb\@Zh
  \@Yb\@Ya\LenMult{\@Z}{\@Zb}{\@Zh}\advance\@Yb\@Zh
```

```
  \@Xc\@Zi\LenMult{\@Z}{\@Zc}{\@Zh}\advance\@Xc\@Zh
  \@Yc\@Zj\LenMult{\@Z}{\@Zd}{\@Zh}\advance\@Yc\@Zh
  \@Z\@Xa\@Za\@Ya\@Zb\@Xb\@Zc\@Yb\@Zd\@Xc\@Ze\@Yc
  \l@cbezier[#1](\@Z,\@Za)(\@Zb,\@Zc)(\@Zd,\@Ze)(\@Zi,\@Zj)%
\fi}
```

⟨/package⟩

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.