The bez123 and multiply packages*

Author: Peter Wilson, Herries Press
Maintainer: LaTeX Project
https://github.com/LaTeX-Package-Repositories/herries-press

2026/10/11

Abstract

The bez123 package provides for the drawing of linear, cubic, and rational quadratic Bezier curves. The multiply package provides a command to multiply a length without numerical overflow.

Contents

1 Introduction **2**

2 Usage **2**

3 The bez123 package implementation **9**
3.1 Arithmetic in TeX 10
3.2 Linear Bezier curves 11
3.3 Cubic Bezier curves 12
3.4 Quadratic rational Bezier curve 15

4 Multiplication without overflow: The multiply package **21**

Index **23**

List of Tables

1 Conic forms of the rational quadratic Bezier curve 6

*This file (bez123.dtx) has version number v1.1c, last revised 2026/10/11.

List of Figures

1 Four sets of points and their convex hulls 4
2 Four sets of points, the cubic Bezier curves and their control polygons. Left — curves plotted with $N = 30$; Right — curves plotted with $N = 0$. 5
3 The angle β 6
4 The effect of weight variation ($W \geq 0$) on rational quadratic Bezier curves (`weightscale = 10000` (the default)) 7
5 The effect of `weightscale` on the drawing of rational quadratic Bezier curves 8
6 Rational quadratics with weights of ± 0.5 and an equilateral triangular convex hull (`weightscale = 50000`) 8
7 Three rational quadratics with weights of 0.5 (`weightscale = 10000`) . . 9
8 A rational quadratic that has gone negative; weights of ± 2 (`weightscale = 10000`) 10

1 Introduction

This document provides the commented source for a LaTeX package file that extends the LaTeX facilities for drawing Bezier curves. The package was originally developed as part of a suite designed for the typesetting of documents according to the rules for ISO international standards [Wil96]. This manual is typeset according to the conventions of the LaTeX DOCSTRIP utility which enables the automatic extraction of the LaTeX macro source files [GMS94].

Drawing a non-rational quadratic Bezier curve is provided as part of the standard LaTeX system. Section 2 provides the user manual for the new commands supplied by this package for drawing a variety of Bezier curves. These include commands for drawing linear and cubic non-rational Bezier curves and rational quadratic curves.

Section 3 describes the implementation of the package. As a side-effect of the implementation, a facility is also provided for performing multiplication in TeX without overflow. This is described in Section 4.

2 Usage

Leslie Lamport provided the means of drawing a quadratic Bezier curve *via* the LaTeX 2_ε `\qbezier` [Lam94, pp. 125–126] command. This package extends the Bezier facility by providing commands to draw linear, rational quadratic, and cubic Bezier curves.

Bezier curves are named after Pierre Bezier who invented them. They are widely used within Computer Aided Design (CAD) programs and other graphics systems; descriptions can be found in many places, with varying degrees of mathematical complexity, such as [FP81, Mor85, Far90].

The Bezier curve is a parameterized curve of degree n and can therefore be specified by $(n+1)$ points (i.e., point p_0 through p_n). Among its other properties, a Bezier curve of degree n passes through through the points p_0 and p_n and passes close to the other defining points. The general equation for a Bezier curve of degree n with parameter t is

$$p(t) = a_0 + a_1 t + a_2 t^2 + \cdots + a_n t^n \tag{1}$$

where the coefficients a_i depend on the defining points, and traditionally $0 \leq t \leq 1$.

For a linear (degree 1) curve, the equation is

$$p(t) = p_0 + (p_1 - p_0)t \tag{2}$$

By inspection, $p(0) = p_0$ and $p(1) = p_1$.

Rearranging equation (1) slightly we get

$$p(t) - p_0 = (p_1 - p_0)t \tag{3}$$

In other words, we can march along the curve from the starting point to the ending point by evaluating the right hand side of equation (3) for increasing values of the parameter t.

In order to shorten the equations slightly, and also make them more convenient to work with numerically, we will use the notation

$$l_{pq} = p_p - p_q$$

Thus, the final form for the linear Bezier curve is

$$p(t) - p_0 = l_{10}t \tag{4}$$

`\lbezier` The command `\lbezier[`⟨*N*⟩`](`⟨*p0*⟩`)(`⟨*p1*⟩`)` draws a linear Bezier curve with ⟨*N*⟩ plotted points from the point ⟨*p0*⟩ (with coordinates ⟨*x0,y0*⟩) to the point ⟨*p1*⟩ (with coordinates ⟨*x1,y1*⟩). ⟨*N*⟩ is an optional argument. If it is either not given or is given with a value of zero, then the command will calculate the number of points to be plotted, subject to a maximum number. There must be no spaces between the arguments to the `\lbezier` command; this restriction also applies to the other Bezier drawing commands provided by the `bez123` package.

Figure 3 shows an example of a dotted line drawn using the `\lbezier` command. The actual code used is:

```
\lbezier[50](15,30)(30,0)
```

thus drawing a straight line consisting of 50 points.

`\qbeziermax` The standard LaTeX command `\qbeziermax` sets a maximum limit on the number of points used to draw any of the Bezier curves.

`\thinlines` `\thicklines` `\linethickness` The 'points' used in drawing the Bezier curves are small squares. The size of these squares are controlled by the standard LaTeX `\thinlines`, `\thicklines` and/or `\linethickness` commands. Consult Lamport [Lam94] for descriptions of these, and `\qbeziermax`, commands.

It is convenient to introduce some general properties of Bezier curves at this point.

- A degree n Bezier curve is defined by $(n+1)$ points which we will label as p_0 through p_n. The lines joining the points $p_0, p_1, \ldots, p_n$ are called the *control polygon*. The Bezier curve is parameterized by a variable we will call t, with $0 \leq t \leq 1$.
- A degree n Bezier curve starts at point p_0 and ends at point p_n.
- At $t = 0$ the curve passes through p_0 and is tangent to the line $l_{10} = p_1 - p_0$.
- At $t = 1$ the curve passes through p_n and is tangent to the line $l_{(n)(n-1)} = p_n - p_{(n-1)}$.
- A *non-rational* Bezier curve lies within the *convex hull*[1] of the points p_0 through p_n. For examples of convex hulls see figure 1. Note that the shape of a convex hull is independent of the ordering of the points.

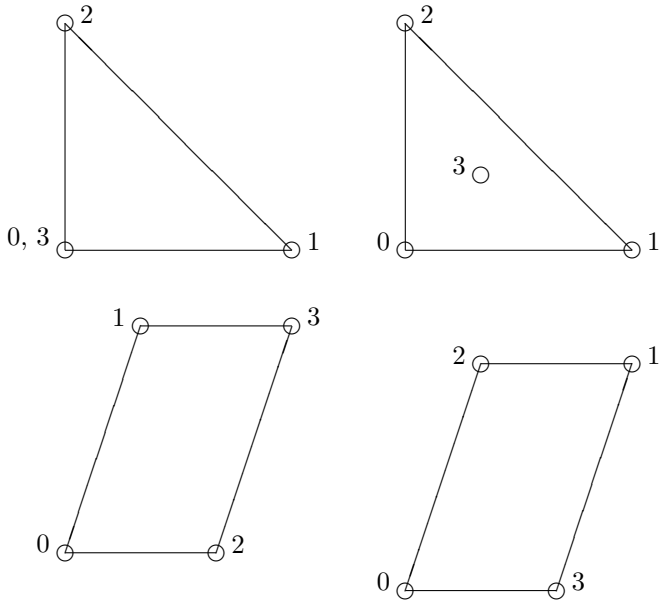


Figure 1: Four sets of points and their convex hulls

The equation for cubic Bezier curves is

$$p(t) - p_0 = 3l_{10}t + 3(l_{21} - l_{10})t^2 + (l_{30} - 3l_{21})t^3 \tag{5}$$

`\cbezier` The command `\cbezier[`$\langle N \rangle$`](`$\langle p0 \rangle$`)(`$\langle p1 \rangle$`)(`$\langle p2 \rangle$`)(`$\langle p3 \rangle$`)` draws a cubic Bezier curve, as defined by equation (5), from point $\langle p0 \rangle$ to point $\langle p3 \rangle$, where $\langle p1 \rangle$ and $\langle p2 \rangle$ are the intermediate points defining the control polygon.

Figure 2 shows four such cubic Bezier curves, their defining points and their control polygons. These are the same points that were used in figure 1 to illustrate convex hulls. It is easy to verify that a cubic Bezier curve does indeed lie within the convex hull of its defining points. The curves on the left of the figure were specified with a value of 30 for the argument $\langle N \rangle$, while those on the right had no value given for $\langle N \rangle$ and thus were drawn with the number of plotted points calculated by the drawing algorithm. The actual drawing commands used were:

```
\cbezier[30](0,0)(10,30)(20,0)(30,30)
\cbezier[30](0,0)(30,0)(0,30)(0,0)
\cbezier(0,0)(30,30)(10,30)(20,0)
\cbezier(0,0)(30,0)(0,30)(10,10)
```

Note that points are plotted along the curve at equidistant values of the of the parameter t. However, as the relationship between the actual distance in (x, y) coordinate space is a non-linear function of t, the separation between the plotted points is non-uniform.

The equation for a *non-rational* quadratic Bezier curve is

$$p(t) - p_0 = 2l_{10}t + (l_{20} - 2l_{10})t^2 \tag{6}$$

[1]The convex hull can be thought of as the shape that a rubber band will take if it is stretched around pins placed at each point.

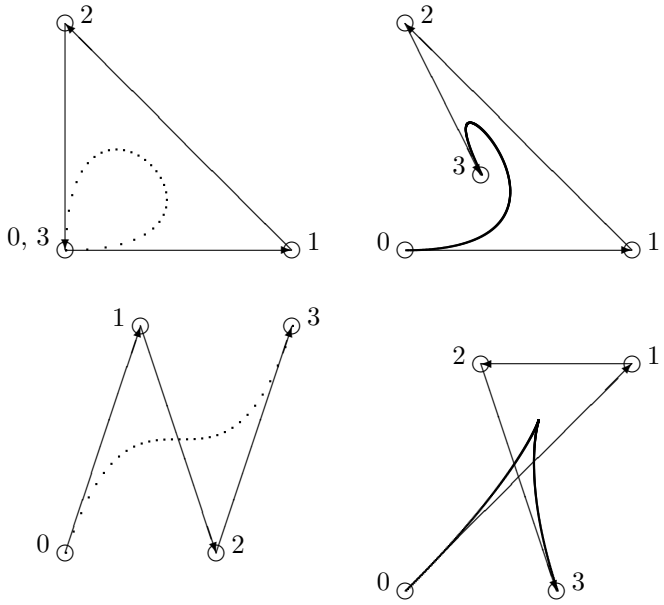


Figure 2: Four sets of points, the cubic Bezier curves and their control polygons. Left — curves plotted with $N = 30$; Right — curves plotted with $N = 0$

Using standard LaTeX this can be drawn by the `\qbezier` command. There is another form of a quadratic Bezier curve called a *rational* quadratic Bezier curve. Its equation is

$$p(t) - p_0 = \frac{2w_1 l_{10} t + (w_2 l_{20} - 2w_1 l_{10})t^2}{w_0 + 2\omega_{10} t + (\omega_{20} - \omega_{10})t^2} \tag{7}$$

where the w_i are the *weights* corresponding to the points p_i and $\omega_{pq} = w_p - w_q$. The shape of a non-rational curve can be changed by changing the positions of the defining points. The shape of a rational curve can also be modified by changing the values of the weights. A rational curve has the same general properties, outlined above, as a non-rational curve with the exception that the curve may lie outside the convex hull of the control polygon.

For the purposes at hand, we use a more restricted form of a rational quadratic Bezier curve, obtained by putting $W = w_1/w_0$ and then making $w_0 = w_2 = 1$ in equation (7). Performing these substitutions we end up with

$$p(t) - p_0 = \frac{2W l_{10} t + (l_{20} - 2W l_{10})t^2}{1 + 2(1 - W)t + 2(1 - W)t^2} \tag{8}$$

Note that when $W = 1$, (8) reduces to equation (6) and when $W = 0$ it effectively reduces to equation (4).

It turns out that a non-rational quadratic Bezier curve is an arc of a parabola, which is one of the conic curves. All the other conic curves can be represented by the rational quadratic Bezier curve described by equation (8) by suitable choices for the value of W. From now on, we will call W the *weight* of the rational quadratic Bezier curve. Table 1 lists the value, or value range, of W for the various forms of the conic curve.[2] For the

[2]We do not deal with the degenerate cases.

Table 1: Conic forms of the rational quadratic Bezier curve

Conic form	Weight (W)
Hyperbola	$\|W\| > 1$
Parabola	$\|W\| = 1$
Ellipse	$0 < \|W\| < 1$
Circle	$\|l_{10}\| = \|l_{21}\|$ and $W = \cos\beta$
Straight line	$W = 0$

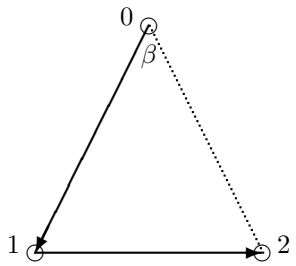


Figure 3: The angle β

case of a circle, β is the angle between the lines $l_{10} = (p_1 - p_0)$ and $l_{20} = (p_2 - p_0)$, as shown in figure 3.

`\rqbezier` The command `\rqbezier[`$\langle N\rangle$`](`$\langle p0\rangle$`)(`$\langle p1\rangle$`)(`$\langle p2\rangle$`)(`$\langle W\rangle$`)` draws a rational quadratic Bezier curve from $\langle p0\rangle$ to $\langle p2\rangle$ with weight $\langle W\rangle$, according to equation (8). As in the other Bezier commands, $\langle N\rangle$ is optional and controls the number of plotted points along the curve. Figure 4 shows several rational quadratic curves, all with the same control polygon but with differing values for the weight W. The code is:

```
\rqbezier[100](15,30)(0,0)(30,0)(4)
\rqbezier[100](15,30)(0,0)(30,0)(2)
\rqbezier(15,30)(0,0)(30,0)(1)
\rqbezier[100](15,30)(0,0)(30,0)(0.75)
\rqbezier[100](15,30)(0,0)(30,0)(0.5)
\rqbezier[100](15,30)(0,0)(30,0)(0.25)
\rqbezier(15,30)(0,0)(30,0)(0)
```

When $W > 1$ the curve is pulled toward the point p_1. Conversely, when $W < 1$ the curve is pushed away from the point p_1. In all cases, though, the curve starts and stops at p_0 and p_2 respectively.

Like the case of the cubic curve, points are plotted at equidistant values of the parameter t. The relationship between parameter value and coordinate positions in the rational case are highly non-linear. Thus the distance between the plotted points can vary quite remarkably. This is an inherent disadvantage with this type of curve. The user's remedy is to increase the number of points to be plotted, but this can lead to TEX running out of memory, not to mention the increased time to generate the drawing.

`\setweightscale` `\resetweightscale` Because of the way in which TEX performs arithmetic, and especially division, it is necessary to perform some scaling operations on the divisor when evaluating equation (8). The optimum value for the scaling is a complex function of the weight and the size and orientation of the control polygon. The algorithm uses a heuristic approach to

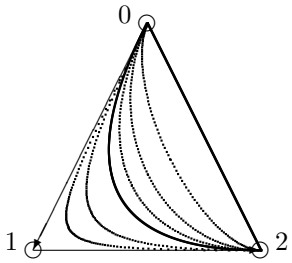


Figure 4: The effect of weight variation ($W \geq 0$) on rational quadratic Bezier curves (`weightscale = 10000` (the default))

calculate a 'good' value but is not always successful. The `\setweightscale{`⟨*number*⟩`}` command can be used to specify a scale factor. ⟨*number*⟩ must be a positive integer. The `\resetweightscale` command resets the scale factor to its default value, which is currently 10000 (ten thousand).

Figure 5 illustrates the effect on changing the `weightscale` used for drawing the same curves as shown in figure 4. Note that the `weightscale` has no effect when $W = 1$ or $W = 0$ as in these cases the curves are drawn using the algorithms for the `\qbezier` and `\lbezier` commands respectively.

It is obvious that some choices give very poorly formed curves. In other cases the curves may be poorly formed but do result in interesting cross-stitch like patterns.

Table 1 indicates that it is possible to draw circular arcs using a rational quadratic Bezier curves. The two legs of the control polygon define the tangents to the curve at the end points. Therefore, to draw a circular arc the two legs must be equal in length. That is, the convex hull is an isosceles triangle. In the special case when the convex hull forms an equilateral triangle, the required weight ($\cos\beta$, see figure 3) for drawing a circular arc is $\cos\beta = 0.5$. Further, for any given control polygon the curves drawn with weights of $\pm W$ are complementary. That is, the curve with weight $-W$ is the 'remainder' of the curve drawn with weight W. Thus, we have a simple means of drawing a complete circle, as shown in figure 6. The plotting commands of interest were:

```
\lbezier[25](0,0)(15,26)
\lbezier[25](0,0)(30,0)
\setweightscale{50000}
\rqbezier[100](15,26)(0,0)(30,0)(0.5)
\rqbezier[200](15,26)(0,0)(30,0)(-0.5)
\resetweightscale
```

where the `\lbezier` drawing commands were used to draw the dotted outline of the control polygon.

A more robust picture of the same circle is shown in figure 7 where the complete circle is pieced together from three non-complementary circular arcs. The drawing commands of interest were

```
\rqbezier[100](15,26)(0,0)(30,0)(0.5)
\rqbezier[100](30,0)(60,0)(45,26)(0.5)
\rqbezier[100](45,26)(30,52)(15,26)(0.5)
```

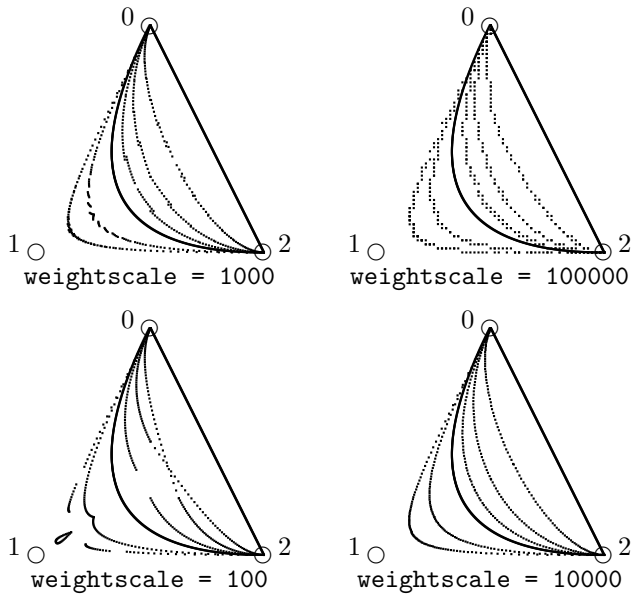


Figure 5: The effect of `weightscale` on the drawing of rational quadratic Bezier curves

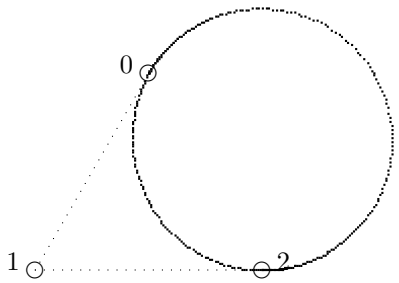


Figure 6: Rational quadratics with weights of ± 0.5 and an equilateral triangular convex hull (`weightscale = 50000`)

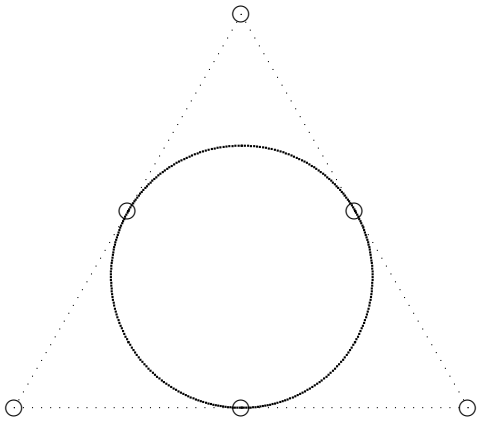


Figure 7: Three rational quadratics with weights of 0.5 (`weightscale = 10000`)

The astute reader will have realised that the divisor in equation (8) can go to zero, and can even be negative. This has interesting consequences, both when trying to do computer arithmetic, and also on the kind of curve that results. Essentially, the curve tends to ∞ as $W \to +0$. At $W = -0$ the curve is at $-\infty$ and then it tends to -0 as $W \to -\infty$. We will get a curve point at ∞ whenever $W = -1$ and a 'negative' curve for $W < -1$.

This effect is shown in figure 8 which draws the two branches of a hyperbola. The basic code for the illustration was

```
\lbezier[25](30,20)(0,10)
\lbezier[25](0,10)(30,0)
\rqbezier[100](30,20)(0,10)(30,0)(2)
\rqbezier[100](30,20)(0,10)(30,0)(-2)
```

where the control polygon was drawn using the `\lbezier` commands.

3 The bez123 package implementation

LaTeX provides a facility for drawing quadratic Bezier curves. This package provides additional facilities for drawing linear, rational quadratic, and cubic Bezier curves.

Announce the name and version of the package, which requires LaTeX 2_ε.

```
⟨*bez⟩
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{bez123}[2026/10/11 v1.1c Bezier curves]
```

The package also requires the multiply package.

```
\RequirePackage{multiply}
⟨/bez⟩
```

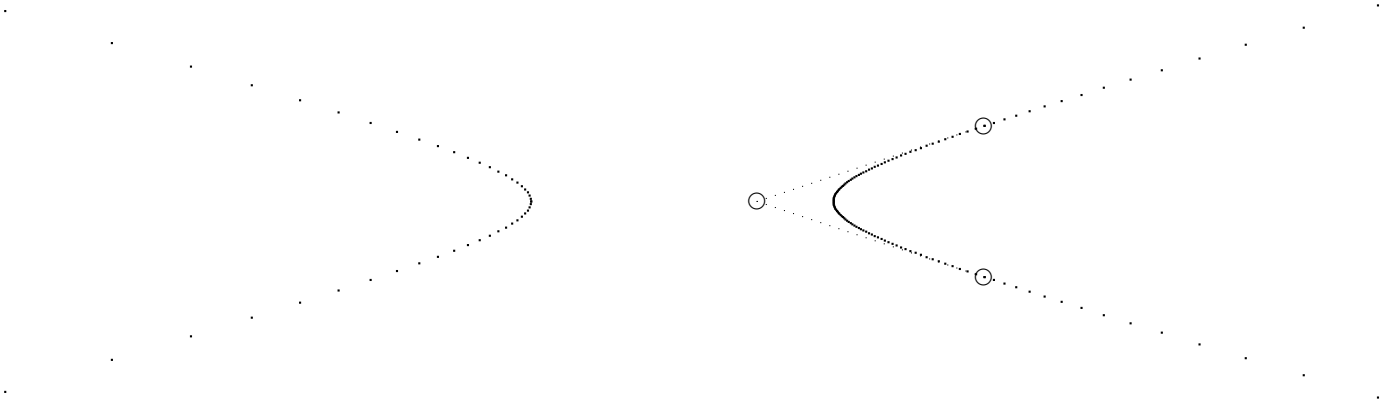


Figure 8: A rational quadratic that has gone negative; weights of ± 2 (`weightscale = 10000`)

3.1 Arithmetic in TEX

All arithmetic in TEX is based on integer arithmetic, with a maximum integer value of $M = 1073741823$. For example, $8/3 = 2$, $9/3 = 3$, and $10/3 = 3$. In other words, division always reduces the absolute value of the dividend, and also possibly truncates the value. One consequence of this is that the ordering of multiplication and division is important. For instance, $(8 \times 3)/3 = 8$ but $(8/3) \times 3 = 6$! Thus, in arithmetic calculations involving both multiplication and division, the dividend should be maximised and the divisor minimised, with multiplication preceding division; also remembering that there is a limit on the size of an integer. To avoid multiplication overflow when calculating say, $a \times b$, we must ensure that $\|a\| \leq \|M/b\|$.

When calculating polynomials, such as that in equation (1), we use a technique called Horner's schema, which is also known as nested multiplication. A general cubic equation, for example, can be written as:

$$p(t) - a_0 = t(a_1 + t(a_2 + ta_3)) \tag{9}$$

The following pseudo-code shows one way to implement Horner's schema for plotting N points in the interval $0 \leq t \leq 1$ of equation (9) using integer arithmetic.

```
procedure plot_cubic(a0, a1, a2, a3:vector; N:integer);
  local p:vector; end_local;
  a3 := a3/N;
  repeat i := 0 to N by 1;
    p := a3*i;
    p := p + a2; p := p/N; p := p*i;
    p := p + a1; p := p/N; p := p*i;
    draw(p + a0);
  end_repeat;
  return;
end_procedure;
```

We use the above algorithm, with suitable modifications according to the degree of the polynomial, for plotting the points along Bezier curves.

3.2 Linear Bezier curves

```
⟨*bez⟩
```

As a linear curve is simpler than a quadratic curve there is no need to declare extra variables from those used in the kernel by the `\qbezier` macro.

`\lbezier` The user command to draw a linear Bezier curve represented by equation (4). The form of the command is:
`\lbezier[`⟨*N*⟩`]{(`⟨*p0*⟩`)(`⟨*p1*⟩`)`
where ⟨*pN*⟩ is the comma separated X and Y coordinate values of point *pN*.

```
\newcommand{\lbezier}[2][0]{\@lbez{#1}#2}
```

(*End of definition for* `\lbezier`*. This function is documented on page 3.*)

`\@lbez` The drawing macro.

```
\gdef\@lbez#1(#2,#3)(#4,#5){%
%%%%\def\lbezier#1(#2,#3)(#4,#5){%
  \ifnum #1<\@ne
```

When the number of plotting points are not given, then we calculate how many are needed. First determine the X distance between the end points.

```
    \@ovxx = #4\unitlength
      \advance\@ovxx by -#2\unitlength
      \ifdim \@ovxx < \z@
        \@ovxx = -\@ovxx
      \fi
```

Similarly calculate the Y distance.

```
    \@ovyy = #5\unitlength
      \advance\@ovyy by -#3\unitlength
      \ifdim \@ovyy < \z@
        \@ovyy = -\@ovyy
      \fi
```

Temporarily store the maximum distance in `\@multicnt`.

```
    \ifdim \@ovxx > \@ovyy
      \@multicnt = \@ovxx
    \else
      \@multicnt = \@ovyy
    \fi
```

We use a small square as the visual representation of a point. Calculate the number of points required to give 50% overlap of adjacent squares, making sure that it doesn't exceed the limit. Store the result in `\@multicnt`.

```
    \@ovxx = 0.5\@halfwidth
    \divide\@multicnt by \@ovxx
    \ifnum \qbeziermax < \@multicnt
      \@multicnt = \qbeziermax\relax
    \fi
  \else
```

The number of points is given.

```
    \@multicnt = #1\relax
  \fi
```

Now we can prepare the constants for the plotting loop.

```
\@tempcnta = \@multicnt
\advance\@tempcnta by \@ne
\@ovdx = #4\unitlength
  \advance\@ovdx by -#2\unitlength
  \divide\@ovdx by \@multicnt
\@ovdy = #5\unitlength
  \advance\@ovdy by -#3\unitlength
  \divide\@ovdy by \@multicnt
```

The next bit of code defines the size of the square representing a point.

```
\setbox\@tempboxa\hbox{\vrule \@height\@halfwidth
                              \@depth \@halfwidth
                              \@width \@wholewidth}%
```

Start the plot at the first point.

```
\put(#2,#3){%
  \count@ = \z@
  \@whilenum{\count@ < \@tempcnta}\do
```

Evaluate the polynomial (simple in this case) using Horner's schema.

```
    {\@xdim = \count@\@ovdx
     \@ydim = \count@\@ovdy
```

Plot this point.

```
     \raise \@ydim
       \hb@xt@\z@{\kern\@xdim
                  \unhcopy\@tempboxa\hss}%
     \advance\count@\@ne}}%
```

The end of the definition of `\@lbez`.

```
}
```

(*End of definition for* `\@lbez`.)

3.3 Cubic Bezier curves

As cubic curves are more complex than quadratic curves we need some extra variables.

`\@wxc` `\@wyc` Lengths.

```
\newlength{\@wxc}
\newlength{\@wyc}
```

(*End of definition for* `\@wxc` *and* `\@wyc`.)

`\cbezier` The user command for drawing a cubic Bezier curve as represented by equation (5). It is called as:
`\cbezier[`⟨*N*⟩`](`⟨*p0*⟩`)(`⟨*p1*⟩`)(`⟨*p2*⟩`)(`⟨*p3*⟩`)`.

```
\newcommand{\cbezier}[2][0]{\@cbez{#1}#2}
```

(*End of definition for* `\cbezier`. *This function is documented on page* 4.)

`\@cbez` The drawing macro for cubic Bezier curves.

```
\gdef\@cbez#1(#2,#3)(#4,#5)(#6,#7)(#8,#9){%
  \ifnum #1<\@ne
```

We have to calculate the number of plotting points required. We will use the maximum of the box enclosing the convex hull as a measure. First do the X value, using `\@ovxx` to store the maximum X coordinate and `\@ovdx` the minimum.

```
\@ovxx = #2\unitlength
\@ovdx = \@ovxx
\@ovdy = #4\unitlength
\ifdim \@ovdy > \@ovxx
  \@ovxx = \@ovdy
\fi
\ifdim \@ovdy < \@ovdx
  \@ovdx = \@ovdy
\fi
\@ovdy = #6\unitlength
\ifdim \@ovdy > \@ovxx
  \@ovxx = \@ovdy
\fi
\ifdim \@ovdy < \@ovdx
  \@ovdx = \@ovdy
\fi
\@ovdy = #8\unitlength
\ifdim \@ovdy > \@ovxx
  \@ovxx = \@ovdy
\fi
\ifdim \@ovdy < \@ovdx
  \@ovdx = \@ovdy
\fi
```

Store the maximum X in `\@ovxx`.

```
\advance\@ovxx by -\@ovdx
```

Repeat the process for the maximum Y value, finally storing this in `\@ovyy`.

```
\@ovyy = #3\unitlength
\@ovdy = \@ovyy
\@ovdx = #5\unitlength
\ifdim \@ovdx > \@ovyy
  \@ovyy = \@ovdx
\fi
\ifdim \@ovdx < \@ovdy
  \@ovdy = \@ovdx
\fi
\@ovdx = #7\unitlength
\ifdim \@ovdx > \@ovyy
  \@ovyy = \@ovdx
\fi
\ifdim \@ovdx < \@ovdy
  \@ovdy = \@ovdx
\fi
\@ovdx = #9\unitlength
\ifdim \@ovdx > \@ovyy
  \@ovyy = \@ovdx
\fi
\ifdim \@ovdx < \@ovdy
  \@ovdy = \@ovdx
\fi
```

```
    \advance\@ovyy by -\@ovdy
```

Temporarily store the max of X and Y in `\@multicnt`.

```
    \ifdim \@ovxx > \@ovyy
      \@multicnt = \@ovxx
    \else
      \@multicnt = \@ovyy
    \fi
```

Calculate the number of points required to give 50% overlap, making sure that it doesn't exceed the limit. Store the number of points in `\@multicnt`.

```
    \@ovxx = 0.5\@halfwidth
    \divide\@multicnt by \@ovxx
    \ifnum \qbeziermax < \@multicnt
      \@multicnt = \qbeziermax\relax
    \fi
  \else
```

The number of points is given.

```
    \@multicnt = #1\relax
  \fi
```

Now we can prepare the constants for the plotting loop. First the control counts.

```
  \@tempcnta = \@multicnt
  \advance\@tempcnta by \@ne
```

Then the cubic coefficients, firstly for X.

```
  \@ovdx = #4\unitlength  \advance\@ovdx by -#2\unitlength
  \@ovxx = #6\unitlength  \advance\@ovxx by -\@ovdx
  \multiply\@ovdx by \thr@@
  \advance\@ovxx by -#4\unitlength  \multiply\@ovxx by \thr@@
  \@wxc = #4\unitlength  \advance\@wxc by -#6\unitlength
  \multiply\@wxc by \thr@@  \advance\@wxc by #8\unitlength
  \advance\@wxc by -#2\unitlength \divide\@wxc by \@multicnt
```

And similarly for Y.

```
  \@ovdy = #5\unitlength  \advance\@ovdy by -#3\unitlength
  \@ovyy = #7\unitlength  \advance\@ovyy by -\@ovdy
  \multiply\@ovdy by \thr@@
  \advance\@ovyy by -#5\unitlength  \multiply\@ovyy by \thr@@
  \@wyc = #5\unitlength  \advance\@wyc by -#7\unitlength
  \multiply\@wyc by \thr@@  \advance\@wyc by #9\unitlength
  \advance\@wyc by -#3\unitlength \divide\@wyc by \@multicnt
```

Set up the plotting box.

```
  \setbox\@tempboxa\hbox{\vrule \@height\@halfwidth
                                \@depth \@halfwidth
                                \@width \@wholewidth}%
```

Start the plot at the first point.

```
  \put(#2,#3){%
    \count@ = \z@
    \@whilenum{\count@ < \@tempcnta}\do
      {\@xdim = \count@\@wxc
```

```
144           \advance\@xdim by \@ovxx
145           \divide\@xdim by \@multicnt
146           \multiply\@xdim by \count@
147           \advance\@xdim by \@ovdx
148           \divide\@xdim by \@multicnt
149           \multiply\@xdim by \count@
150         \@ydim = \count@\@wyc
151           \advance\@ydim by \@ovyy
152           \divide\@ydim by \@multicnt
153           \multiply\@ydim by \count@
154           \advance\@ydim by \@ovdy
155           \divide\@ydim by \@multicnt
156           \multiply\@ydim by \count@
```

Plot the point.

```
157         \raise \@ydim
158           \hb@xt@\z@{\kern\@xdim
159                      \unhcopy\@tempboxa\hss}%
160         \advance\count@\@ne}}%
```

The end of the definition of `\@cbez`.

```
161 }
```

(*End of definition for* `\@cbez`.)

3.4 Quadratic rational Bezier curve

This is the most complex of the Bezier curves that we deal with. We need yet more variables.

`\@ww` `\@wwa` `\@wwb` `\@wwo` `\@wwi` Variables for the weight calculations.

```
162 \newlength{\@ww}
163 \newlength{\@wwa}
164 \newlength{\@wwb}
165 \newlength{\@wwo}
166 \newlength{\@wwi}
```

(*End of definition for* `\@ww` *and others.*)

`\c@@pntscale` Scale factor for points.

```
167 \newcounter{@pntscale}
```

(*End of definition for* `\c@@pntscale`.)

`\c@weightscale` Scale factor for divisor.

```
168 \newcounter{weightscale}
```

(*End of definition for* `\c@weightscale`.)

`\botscale` Scale factor for bottom weights.

```
169 \newlength{\botscale}
```

(*End of definition for* `\botscale`.)

`\setweightscale` User level command `\setweightscale{`⟨*number*⟩`}` for setting the divisor scaling.

```
170 \newcommand{\setweightscale}[1]{\setcounter{weightscale}{#1}}
```

(*End of definition for* `\setweightscale`. *This function is documented on page* 6.)

`\resetweightscale` User level command for setting the divisor scaling to its default value (10^4). We also ensure that the scaling is set to this value.

```
\newcommand{\resetweightscale}{\setcounter{weightscale}{10000}}
\resetweightscale
```

(*End of definition for* `\resetweightscale`. *This function is documented on page* 6.)

`\rqbezier` The user level command for drawing a rational quadratic Bezier curve as represented by equation (8). The form of the command is
`\rqbezier[`⟨*N*⟩`](`⟨*p0*⟩`)(`⟨*p1*⟩`)(`⟨*p2*⟩`)(`⟨*W*⟩`)`
where the arguments are as per the other Bezier drawing commands, but with the final argument being the weight.

```
\newcommand{\rqbezier}[2][0]{\@rqbez{#1}#2}
```

(*End of definition for* `\rqbezier`. *This function is documented on page* 6.)

`\@rqbez` The drawing macro for a rational quadratic Bezier curve. If the weight is such that the curve is either rational quadratic ($W = 1$) or linear ($W = 0$), we use the simpler drawing macro.

```
\gdef\@rqbez#1(#2,#3)(#4,#5)(#6,#7)(#8){%
  \@ovxx = #8\unitlength
  \ifdim\@ovxx = \unitlength
    \PackageWarning{bez123}{Rational quadratic degenerates to quadratic}
    \qbezier[#1](#2,#3)(#4,#5)(#6,#7)
  \else
    \ifdim\@ovxx = \z@
      \PackageWarning{bez123}{Rational quadratic degenerates to linear}
      \lbezier[#1](#2,#3)(#6,#7)
    \else
```

Calculate the maximum length of the control polygon's bounding box. Store the result in `\@wwi`.

```
    \@ovxx = #4\unitlength
      \advance\@ovxx by -#2\unitlength
      \ifdim \@ovxx < \z@
        \@ovxx = -\@ovxx
      \fi
    \@ovdx = #6\unitlength
      \advance\@ovdx by -#4\unitlength
      \ifdim \@ovdx < \z@
        \@ovdx = -\@ovdx
      \fi
    \ifdim \@ovxx < \@ovdx
      \@ovxx = \@ovdx
    \fi
    \@ovyy = #5\unitlength
      \advance\@ovyy by -#3\unitlength
      \ifdim \@ovyy < \z@
        \@ovyy = -\@ovyy
      \fi
    \@ovdy = #7\unitlength
```

```
      \advance\@ovdy by -#5\unitlength
      \ifdim \@ovdy < \z@
        \@ovdy = -\@ovdy
      \fi
    \ifdim \@ovyy < \@ovdy
      \@ovyy = \@ovdy
    \fi
    \ifdim \@ovxx > \@ovyy
      \@multicnt = \@ovxx
    \else
      \@multicnt = \@ovyy
    \fi
    \@wwi = \@multicnt sp
```

Now determine the number of points to be plotted.

```
  \ifnum #1<\@ne
    \@ovxx = 0.5\@halfwidth
    \divide\@multicnt by \@ovxx
    \ifnum\qbeziermax < \@multicnt
      \@multicnt = \qbeziermax\relax
    \fi
  \else
```

Number of points is a given.

```
    \@multicnt = #1\relax
  \fi
```

We are going to plot the curve in two halves in an attempt to reduce roundoff problems. At a minimum this should at least make a symmetrical curve look symmetric about its mid point.

```
  \@tempcnta = \@multicnt
  \advance\@tempcnta by \@ne
  \divide\@tempcnta by \tw@
  \advance\@tempcnta by \@ne
```

We now have to deal with a possible multiplication overflow problem due to multiplication by the weight. In equation (8) the potentially largest term is the coefficient of t^2 (i.e., $(l_{20} - 2Wl_{10})$). The maximum length likely to be encountered is, say, 10 inches for a drawing on either A4 or US letterpaper. This is approximately 4.8×10^8sp. Doing a little arithmetic, and remembering that the maximum length in TEX is $M = 1073741823$sp, it means that we must have $\|W\| \leq 1$ to prevent overflow. However, a typical range for W is $-10 \leq W \leq 10$. Therefore we might have to do some scaling. Being pessimistic, we'll assume that $l_{20} = -l_{10}$ and that l_{10} is the largest dimension in the drawing. To prevent overflow we then have to meet the condition $\|W\| \leq (M - l_{20})/2l_{20}$, where all lengths are positive. We will use `\c@@pntscale` as a scale factor on W to meet this condition. Earlier we set `\@wwi` to be the positive value of the largest dimension in the drawing.
Set the distance scale factor. First evaluating the test condition.

```
  \@wwo = \maxdimen
    \advance\@wwo by -\@wwi
    \divide\@wwo by \tw@
    \divide\@wwo by \@wwi
```

Now perform the check and set the scale factor. We have to get a positive integer value for W as it may be a fraction. Actually, we only need to be concerned if $\|W\| > 1$.

```
\@wwi = 10sp
\@wwi = #8\@wwi
  \ifdim\@wwi < \z@
    \@wwi = -\@wwi
  \fi
  \divide\@wwi by 10\relax
\ifdim\@wwi < \@wwo
  \c@@pntscale = \@ne
\else
  \divide\@wwi by \tw@
  \ifdim\@wwi < \@wwo
    \c@@pntscale = \tw@
  \else
    \divide\@wwi by \tw@
    \ifdim\@wwi < \@wwo
      \c@@pntscale = 4\relax
    \else
      \divide\@wwi by \tw@
      \ifdim\@wwi < \@wwo
        \c@@pntscale = 8\relax
      \else
        \c@@pntscale = 16\relax
      \fi
    \fi
  \fi
\fi
```

Calculate the constants for the top line of the function.

```
\@ovxx = #4\unitlength \advance\@ovxx by -#2\unitlength
  \multiply\@ovxx by \tw@
  \divide\@ovxx by \c@@pntscale
  \@ovdx = #8\@ovxx
\@ovxx = #6\unitlength  \advance\@ovxx by -#2\unitlength
  \divide\@ovxx by \c@@pntscale
  \advance\@ovxx by -\@ovdx
  \divide\@ovxx by \@multicnt
\@ovyy = #5\unitlength \advance\@ovyy by -#3\unitlength
  \multiply\@ovyy by \tw@
  \divide\@ovyy by \c@@pntscale
  \@ovdy = #8\@ovyy
\@ovyy = #7\unitlength  \advance\@ovyy by -#3\unitlength
  \divide\@ovyy by \c@@pntscale
  \advance\@ovyy by -\@ovdy
  \divide\@ovyy by \@multicnt
```

Now the constants for the bottom line. We also need to do some scaling here. This scaling can be set by the user.

```
\setlength{\botscale}{\c@weightscale sp}
\@wwo = \botscale
\@wwi = #8\@wwo
\@wwa = \@wwo  \advance\@wwa by -\@wwi
  \multiply\@wwa by \tw@
```

```
  \@wwb = \@wwa
    \divide\@wwb by \@multicnt
```

Prepare for the drawing.

```
  \@wwi = \botscale
  \setbox\@tempboxa\hbox{\vrule \@height\@halfwidth
                                \@depth \@halfwidth
                                \@width \@wholewidth}%
```

Draw the first half of the curve.

```
  \put(#2,#3){%
    \count@ = \z@
    \@whilenum{\count@ < \@tempcnta}\do
      {\@xdim = \count@\@ovxx
         \advance\@xdim by \@ovdx
         \divide\@xdim by \@multicnt
         \multiply\@xdim by \count@
       \@ydim = \count@\@ovyy
         \advance\@ydim by \@ovdy
         \divide\@ydim by \@multicnt
         \multiply\@ydim by \count@
       \@ww = \count@\@wwb
         \advance\@ww by -\@wwa
         \divide\@ww by \@multicnt
         \multiply\@ww by \count@
         \advance\@ww by \@wwo
         \divide\@ww by \c@@pntscale
         \ifdim\@ww = \z@
```

We are about to divide by `\@ww` which is zero. Treat `\@ww` as unity.

```
         \else
           \divide\@xdim by \@ww
           \divide\@ydim by \@ww
         \fi
```

For reasons I don't understand, the % signs at the end of the next few lines are important!

```
         \multnooverflow{\@xdim}{\botscale}%
         \multnooverflow{\@ydim}{\botscale}%
       \raise \@ydim
         \hb@xt@\z@{\kern\@xdim
                    \unhcopy\@tempboxa\hss}%
       \advance\count@\@ne}}
```

We now repeat the above process for plotting the second half of the curve, starting at the end point.
Calculate the constants for the top line of the function.

```
  \@ovxx = #4\unitlength \advance\@ovxx by -#6\unitlength
    \multiply\@ovxx by \tw@
    \divide\@ovxx by \c@@pntscale
    \@ovdx = #8\@ovxx
  \@ovxx = #2\unitlength  \advance\@ovxx by -#6\unitlength
    \divide\@ovxx by \c@@pntscale
    \advance\@ovxx by -\@ovdx
    \divide\@ovxx by \@multicnt
```

```
\@ovyy = #5\unitlength \advance\@ovyy by -#7\unitlength
  \multiply\@ovyy by \tw@
  \divide\@ovyy by \c@@pntscale
  \@ovdy = #8\@ovyy
\@ovyy = #3\unitlength  \advance\@ovyy by -#7\unitlength
  \divide\@ovyy by \c@@pntscale
  \advance\@ovyy by -\@ovdy
  \divide\@ovyy by \@multicnt
```

The constants for the bottom line are the same as before as the function is symmetric. Similarly we don't need to recalculate the size of the rule box.
Draw the second half of the curve.

```
\put(#6,#7){%
  \count@ = \z@
  \@whilenum{\count@ < \@tempcnta}\do
    {\@xdim = \count@\@ovxx
       \advance\@xdim by \@ovdx
       \divide\@xdim by \@multicnt
       \multiply\@xdim by \count@
     \@ydim = \count@\@ovyy
       \advance\@ydim by \@ovdy
       \divide\@ydim by \@multicnt
       \multiply\@ydim by \count@
     \@ww = \count@\@wwb
       \advance\@ww by -\@wwa
       \divide\@ww by \@multicnt
       \multiply\@ww by \count@
       \advance\@ww by \@wwo
       \divide\@ww by \c@@pntscale
       \ifnum\@ww = \z@
```

We are about to divide by `\@ww` which is zero. Treat `\@ww` as unity.

```
       \else
         \divide\@xdim by \@ww
         \divide\@ydim by \@ww
       \fi
```

For reasons I don't understand, the % signs at the end of the next few lines are important!

```
       \multnooverflow{\@xdim}{\botscale}%
       \multnooverflow{\@ydim}{\botscale}%
     \raise \@ydim
       \hb@xt@\z@{\kern\@xdim
                  \unhcopy\@tempboxa\hss}%
     \advance\count@\@ne}}
```

End of definition of `\@rqbez`.

```
\fi\fi}
```

(*End of definition for* `\@rqbez`.)

The end of this package.

```
⟨/bez⟩
```

4 Multiplication without overflow: The **multiply** package

TEX provides for integer arithmetic, subject to an upper limit given by `\maxdim`. For at least the bez123 package we need to be able to multiply without overflow.

Announce the name of the package.

```
⟨*mult⟩
\ProvidesPackage{multiply}[1998/10/14 v1.1 Multiplication of lengths without overflow]
```

`\n@fl@wa` `\n@fl@wb` `\n@fl@wc` `\ifch@nge`

We need three length variables for this function. We also need a boolean flag for dealing with negative numbers.

```
\newlength{\n@fl@wa}
\newlength{\n@fl@wb}
\newlength{\n@fl@wc}
\newif\ifch@nge
```

(*End of definition for* `\n@fl@wa` *and others.*)

`\multnooverflow`

The routine `\multnooverflow{⟨a⟩}{⟨b⟩}` sets a to the minimum of ab and `\maxdimen`, preserving signs. $\langle a \rangle$ must be a length; it must not be a number literal.

```
\newcommand{\multnooverflow}[2]{%
  \n@fl@wa = #1\relax%
  \n@fl@wb = #2\relax%
  \ch@ngefalse%
```

Easy if $-1 \leq b \leq 1$.

```
  \ifnum\n@fl@wb = \@ne%
  \else%
    \ifnum\n@fl@wb = \z@%
      \n@fl@wa = \z@%
    \else%
      \ifnum\n@fl@wb = \m@ne%
        \ch@ngetrue%
      \else%
```

Also easy if $-1 \leq a \leq 1$.

```
        \ifnum\n@fl@wa = \z@%
        \else%
          \ifnum\n@fl@wa = \@ne%
            \n@fl@wa = \n@fl@wb%
          \else%
            \ifnum\n@fl@wa = \m@ne%
              \n@fl@wa = -\n@fl@wb%
            \else%
```

We have to check for potential overflow. First make sure that we deal only with positive values.

```
              \ifnum\n@fl@wa < \z@%
                \ch@ngetrue%
                \n@fl@wa = -\n@fl@wa%
              \fi%
              \ifnum\n@fl@wb < \z@%
```

```
                \n@fl@wb = -\n@fl@wb%
                \ifch@nge%
                  \ch@ngefalse%
                \else%
                  \ch@ngetrue%
                \fi%
              \fi%
```

Check for overflow.

```
              \n@fl@wc = \maxdimen%
              \divide\n@fl@wc by \n@fl@wb%
              \advance\n@fl@wc by -1sp% \m@ne
              \ifnum\n@fl@wa > \n@fl@wc%
```

We have overflow. Set the multiplication result to `\maxdimen`.

```
                \n@fl@wa = \maxdimen%
                \PackageWarning{multiply}{Multiplication overflow}%
              \else%
```

It is safe to do the multiplication.

```
                \multiply\n@fl@wa by \n@fl@wb%
              \fi%
            \fi%
          \fi%
        \fi%
      \fi%
    \fi%
  \fi%
```

The result of ab is in `\n@fl@wa`. Adjust the sign if necessary.

```
  \ifch@nge%
    \n@fl@wa = -\n@fl@wa%
  \fi%
```

Return the result in the first argument variable.

```
  #1 = \n@fl@wa%
}
```

(*End of definition for* `\multnooverflow`.)

The end of this package.

```
⟨/mult⟩
```

References

[Far90] Gerald Farin. *Curves and Surfaces for Computer Aided Geometric Design — A Practical Guide*. Academic Press, Inc., second edition, 1990.

[FP81] I. D. Faux and M. J. Pratt. *Computational Geometry for Design and Manufacture*. Ellis Horwood, 1981.

[GMS94] Michel Goossens, Frank Mittelbach, and Alexander Samarin. *The LaTeX Companion*. Addison-Wesley Publishing Company, 1994.

[Lam94] Leslie Lamport. *LaTeX: A Document Preparation System*. Addison-Wesley Publishing Company, second edition, 1994.

[Mor85] Michael E. Mortenson. *Geometric Modeling*. John Wiley & Sons, Inc., 1985.

[Wil96] Peter R. Wilson. *LaTeX for standards: The LaTeX package files user manual*. NIST Report NISTIR, June 1996.

Index

The italic numbers denote the pages where the corresponding entry is described, numbers underlined point to the definition, all others indicate the places where it is used.

B

`\botscale` 169, 275, 276, 282, 308, 309, 352, 353

C

`\cbezier` *4*
`\cbezier` 57

D

`\do` 47, 142, 288, 332

L

`\lbezier` *3*
`\lbezier` 7, 9, 182
`\linethickness` *3*

M

`\maxdimen` 229, 398, 402
`\multiply` 125, 126, 128, 132, 133, 135, 146, 149, 153, 156, 260, 268, 279, 292, 296, 300, 315, 323, 336, 340, 344, 405
`\multnooverflow` . . . 308, 309, 352, 353, 366

N

`\newif` 365

P

`\PackageWarning` 177, 181, 403
`\ProvidesPackage` 3, 361
`\put` 45, 140, 286, 330

Q

`\qbezier` 178
`\qbeziermax` *3*
`\qbeziermax` 28, 29, 115, 116, 219, 220

R

`\raise` 50, 157, 310, 354
`\RequirePackage` 4
`\resetweightscale` *6*
`\resetweightscale` 171
`\rqbezier` *6*
`\rqbezier` 173

S

`\setweightscale` *6*
`\setweightscale` 170

T

TEX and LATEX 2ε commands:
`\@cbez` 57, 58
`\@depth` 43, 138, 284
`\@halfwidth` 26, 42, 43, 113, 137, 138, 217, 283, 284
`\@height` 42, 137, 283
`\@lbez` 7, 8
`\@multicnt` 22, 24, 27, 28, 29, 32, 34, 38, 41, 109, 111, 114, 115, 116, 119, 121, 129, 136, 145, 148, 152, 155, 211, 213, 215, 218, 219, 220, 223, 225, 266, 274, 281, 291, 295, 299, 321, 329, 335, 339, 343
`\@ovdx` 36, 37, 38, 48, 61, 66, 67, 73, 74, 80, 81, 83, 86, 87, 88, 90, 91, 93, 94, 95, 97, 98, 100, 101, 102, 104, 105, 123, 124, 125, 147, 189, 190, 191, 192, 194, 195, 262, 265, 290, 317, 320, 334
`\@ovdy` 39, 40, 41, 49, 62, 63, 64, 66, 67, 69, 70, 71, 73, 74, 76, 77, 78, 80, 81, 85, 90, 91, 97, 98, 104, 105, 107, 130, 131, 132, 154, 202, 203, 204, 205, 207, 208, 270, 273, 294, 325, 328, 338
`\@ovxx` 11, 12, 13, 14, 21, 22, 26, 27, 60, 61, 63, 64, 70, 71, 77, 78, 83, 108, 109, 113, 114, 124, 126, 144, 175, 176,

180, 184, 185, 186, 187, 194, 195, 210, 211, 217, 218, 259, 260, 261, 262, 263, 264, 265, 266, 289, 314, 315, 316, 317, 318, 319, 320, 321, 333
\@ovyy 16, 17, 18, 19, 21, 24, 84, 85, 87, 88, 94, 95, 101, 102, 107, 108, 111, 131, 133, 151, 197, 198, 199, 200, 207, 208, 210, 213, 267, 268, 269, 270, 271, 272, 273, 274, 293, 322, 323, 324, 325, 326, 327, 328, 329, 337
\@rqbez 173, 174
\@whilenum 47, 142, 288, 332
\@wholewidth 44, 139, 285
\@width 44, 139, 285
\@ww 162, 297, 298, 299, 300, 301, 302, 303, 305, 306, 341, 342, 343, 344, 345, 346, 347, 349, 350
\@wwa 162, 278, 279, 280, 298, 342
\@wwb 162, 280, 281, 297, 341
\@wwi 162, 215, 230, 232, 233, 234, 235, 236, 238, 239, 242, 243, 246, 247, 250, 251, 277, 278, 282
\@wwo . . 162, 229, 230, 231, 232, 239, 243, 247, 251, 276, 277, 278, 301, 345
\@wxc 55, 127, 128, 129, 143
\@wyc 55, 134, 135, 136, 150
\@xdim 48, 51, 143, 144, 145, 146, 147, 148, 149, 158, 289, 290, 291, 292, 305, 308, 311, 333, 334, 335, 336, 349, 352, 355
\@ydim 49, 50, 150, 151, 152, 153, 154, 155, 156, 157, 293, 294, 295, 296, 306, 309, 310, 337, 338, 339, 340, 350, 353, 354
\c@@pntscale 167, 240, 244, 248, 252, 254, 261, 264, 269, 272, 302, 316, 319, 324, 327, 346
\c@weightscale 168, 275
\ch@ngefalse 369, 393
\ch@ngetrue 376, 387, 395
\count@ 46, 47, 48, 49, 53, 141, 142, 143, 146, 149, 150, 153, 156, 160, 287, 288, 289, 292, 293, 296, 297, 300, 313, 331, 332, 333, 336, 337, 340, 341, 344, 357
\hb@xt@ 51, 158, 311, 355
\ifch@nge 362, 392, 413
\n@fl@wa 362, 367, 373, 378, 380, 381, 383, 384, 386, 388, 401, 402, 405, 414, 416
\n@fl@wb 362, 368, 370, 372, 375, 381, 384, 390, 391, 399, 405
\n@fl@wc 362, 398, 399, 400, 401
\thr@@ 125, 126, 128, 132, 133, 135
\thicklines *3*
\thinlines *3*

U

\unhcopy 52, 159, 312, 356
\unitlength 11, 12, 16, 17, 36, 37, 39, 40, 60, 62, 69, 76, 84, 86, 93, 100, 123, 124, 126, 127, 128, 129, 130, 131, 133, 134, 135, 136, 175, 176, 184, 185, 189, 190, 197, 198, 202, 203, 259, 263, 267, 271, 314, 318, 322, 326

V

\vrule 42, 137, 283