1 Bounding Boxes for Bézier Curves

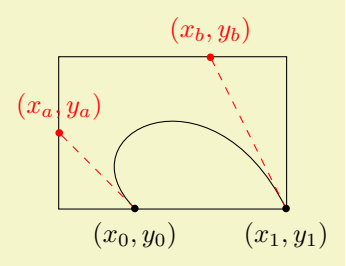
Ti*k*Z Library bbox

```
\usepgflibrary{bbox}  % LaTeX and plain TeX and pure pgf
\usepgflibrary[bbox]  % ConTeXt and pure pgf
\usetikzlibrary{bbox} % LaTeX and plain TeX when using TikZ
\usetikzlibrary[bbox] % ConTeXt when using TikZ
```

This library provides methods to determine tight bounding boxes for Bézier curves. This library loads and uses the `fpu` library.

1.1 Bounding box without the library

Ti*k*Z determines the bounding box of (cubic) Bezier curves by establishing the smallest rectangle that contains the end point and the two control points of the curve.




```
\begin{tikzpicture}[%
bullet/.style={circle,fill,
inner sep=1pt}]
 \draw (0,0) .. controls (-1,1)
 and (1,2) .. (2,0);
 \draw (current bounding box.south west)
 rectangle
  (current bounding box.north east);
 \draw[red,dashed]
 (0,0) -- (-1,1)
node[bullet,label=above:{$(x_a,y_a)$}]{}
    (2,0) -- (1,2)
node[bullet,label=above:{$(x_b,y_b)$}]{};
 \path (0,0)
 node[bullet,label=below:{$(x_0,y_0)$}]{}
 (2,0)
node[bullet,label=below:{$(x_1,y_1)$}]{};
\end{tikzpicture}
```

As one can see from this illustration, this may lead to drastic overestimates of the bounding box.

1.2 Computing the bounding box

Establishing the precise bounding box has been discussed in various places, the following discussion uses in part the results from https://pomax.github.io/bezierinfo/. What is a cubic Bézier curve? A cubic Bézier curve running from (x_0, y_0) to (x_1, y_1) with control points (x_a, y_a) and (x_b, y_b) can be parametrized by

$$\gamma(t) = \begin{pmatrix} x(t) \\ y(t) \end{pmatrix} = \begin{pmatrix} t^3x_1 + 3t^2(1-t)x_b + (1-t)^3x_0 + 3t(1-t)^2x_a \\ t^3y_1 + 3t^2(1-t)y_b + (1-t)^3y_0 + 3t(1-t)^2y_a \end{pmatrix}, \quad (1)$$

where t runs from 0 to 1 (and $\gamma(0) = (x_0, y_0)$ and $\gamma(1) = (x_1, y_1)$). Surely, the bounding box has to contain (x_0, y_0) and (x_1, y_1). If the functions $x(t)$ and $y(t)$ have extrema in the interval $[0, 1]$, then the bounding box will in general be larger than that. In order to determine the extrema of the curve, all we need to find the

extrema of the functions $x(t)$ and $y(t)$ for $0 \le t \le 1$. That is, we need to find the solutions of the quadratic equations

$$\frac{\mathrm{d}x}{\mathrm{d}t}(t) \;=\; 0 \quad \text{and} \quad \frac{\mathrm{d}y}{\mathrm{d}t}(t) \;=\; 0\,. \tag{2}$$

Let's discuss x first. If the discriminant

$$d \;:=\; x_0\,x_1 - x_1\,x_a + x_a\,x_a - x_0\,x_b - x_a x_b + x_b\,x_b \tag{3}$$

is greater than 0, there are two solutions

$$t_\pm \;=\; \frac{x_0 - 2x_a + x_b \pm \sqrt{d}}{x_0 - x_1 - 3(x_a - x_b)}\,. \tag{4}$$

If the denominator $x_0 - x_1 - 3(x_a - x_b)$ vanishes, one may use the l'Hospital rule to determine the solutions. In this case, we need to make sure that the bounding box contains, say $(x(t_-), y_0)$ and $(x(t_+), y_0)$. If $d \le 0$, the bounding box does not need to be increased in the x direction. On the other hand, if there are solutions, one needs include the points $\big(x(t_\pm), y_0\big)$ with $x(t)$ from (1) in the bounding box.

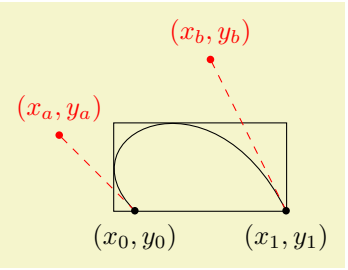
The analogous statements apply to $y(t)$.

1.3 Using the library

`/pgf/bezier bounding box=`⟨*boolean*⟩ (default `true`)

Turn the tight bounding box algorithm on and off. The initial value is `false`.

Caveat: As can be seen from the derivations, the necessary computations involve the squaring of lengths and taking ratios of lengths, which can easily lead to `dimension too large` errors. The library uses `fpu` to account for that, but errors may still occur.




```
\begin{tikzpicture}[bezier bounding box,%
bullet/.style={circle,fill,
inner sep=1pt}]
 \draw (0,0) .. controls (-1,1)
 and (1,2) .. (2,0);
 \draw (current bounding box.south west)
 rectangle
  (current bounding box.north east);
 \draw[red,dashed]
 (0,0) -- (-1,1)
node[bullet,label=above:{$(x_a,y_a)$}]{}
    (2,0) -- (1,2)
node[bullet,label=above:{$(x_b,y_b)$}]{};
 \path (0,0)
 node[bullet,label=below:{$(x_0,y_0)$}]{}
 (2,0)
node[bullet,label=below:{$(x_1,y_1)$}]{};
\end{tikzpicture}
```

A few comments are in order.

1. For paths with arrow heads one may need to load the `bending` library. This is because otherwise the quick arrow head distorts the path, and this happens after the bounding box has been computed. Even worse, arrow heads could get deformed.

2. If you shorten a path by some negative length, the bounding box will not be accurate either. However, this has nothing to do with curves, it also applies to straight lines. So this is not specific to the bbox library but something that one may want to keep in mind.

3. Let us also note that the computations can lead to `Dimension too large` errors. These errors do not come directly from the computations done by the library, which uses `fpu` for its computations, but from the aftermath. Many of these problems can be avoided by using the `fpu` library also for computing reciprocals. Using this key allows one to fix many `Dimension too large` errors in other libraries, which are not related to the present one, and also fixes inaccuracies of (inverse) transformations when the scale factors are not integer.

`/pgf/use fpu reciprocal`⟨*boolean*⟩ `(default true)` (no value)

This changes the computation of reciprocals from the standard pgf routine to an `fpu` variant. The initial value is `false`.