Documentation of 'pst-bezier.tex'

Tobias Nähring
www.tn-home.de

September 12, 2004

Contents

1 Introduction

The pstricks package provides (essentially) two main macros for drawing curves: \pscurve and \psbezier. Both macros employ Bezier splines.

The \pscurve macro takes multiple interpolated points as arguments. Thus, it is easy to draw long multiply bent curves. The problem with \pscurve is that there is no easy way[1] to change the automatically computed control points without simultaneously changing the interpolated points.

The \psbezier macro gives full control over the interpolation points and the control points of one Bezier polynominal of degree three (two interpolated points and two control points).

If one demands for the access to certain control points of one multiply bent curve one has to use multiple instances of the \psbezier macro. With this approache each inner interpolation point of the curve has to be input twice. Furthermore, if one needs smooth joints one has to compute control points symmetrically to the corresponding interpolation points for every joint even if one does not care so much about the exact tangential direction at some of those joints. That can be rather tedious.

The \bcurve macro of pst-bezier.sty is intented to demonstrate a way to combine the nice properties of the macros \pscurve and \psbezier. It provides an easy input format to describe 'arbitrarily' many interpolation points of a curve and to fix the control points at some freely selected interpolation points.

[1] Note that some control is possible via the curvature option.

Note, that `pst-bezier.sty` is **no final package** (e.g. the automatical computation of the control points is not as refined as that one for the macro `\pscurve`). To emphase that, the macro `\bcurve` does not fit into the name conventions of `pstricks`. I would be pleased if the features of `pst-bezier.tex` could be implemented into `pstricks-add.tex`. But, because of lack of spare time, I cannot promise to do any further work on that project.

2 Installation and usage of `pst-bezier.tex`

Installation: As prerequisites for `pst-bezier` you need resent working versions of LaTeX and `pstricks`. The files `pst-bezier.tex` and `pst-bezier.sty` must be somewhere in your TeX-input path. Further more, the file `pst-bezier.pro` must be in some path, where `dvips` can find it.

Usage: As usual, load the packages `pstricks` and `pst-bezier` in that order via the `\usepackage` macro.

Now you are ready to use the `\bcurve` macro within your document body. This macro is described in the next section with all its options.

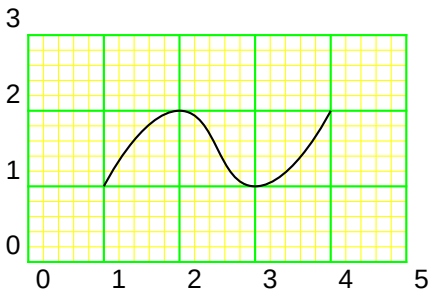
Whith the following simple LaTeX-source code you can test whether you have correctly installed the package:

```
\documentclass{minimal}
\usepackage{pstricks}
\usepackage{pst-bezier}
\begin{document}
 \begin{pspicture}(6,4)
   \bcurve(1,2)(5,2) % Draw just one straight line.
 \end{pspicture}
\end{document}
```

3 The `\bcurve` macro

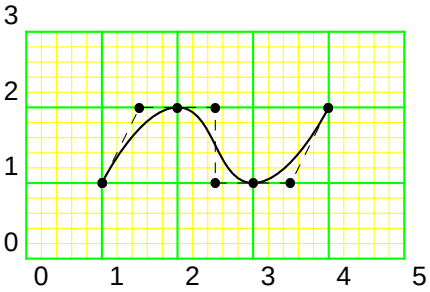
In the most simple form you can specify any number of interpolation points as the argument of `\bcurve`.

```
\bcurve(1,1)(2,2)(3,1)(4,2)
```



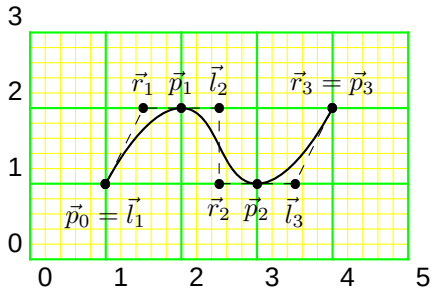
As usual, options can be specified within brackets.

```
\bcurve[showpoints=true]%
(1,1)(2,2)(3,1)(4,2)
```



As you can see in the above example, the `showpoints` feature works (partially) with `bcurve`.

The next figure shows again the curve from the first example. This time labels are added to the points (this is just for the following description, it is not a feature of `\bcurve`).



The points labelled with $\vec{p}_k$ $(k = 0, \ldots, 3)$ are the interpolation points, these ones labelled with $\vec{l}_1, \ldots, \vec{l}_3$, and these ones labelled with $\vec{r}_1, \ldots, \vec{r}_3$ are the left and right control points, respectively.

Between each consecutive pair $\vec{p}_{k-1}, \vec{p}_k$ of interpolation points the `\bcurve` macro draws a cubic Bezier spline. The control points $\vec{l}_k$ and $\vec{r}_k$ determine the tangential direction of the bezier spline at the interpolation points. More exactly, the bezier spline from $\vec{p}_{k-1}$ to $\vec{p}_k$ is tangent to the vector $\vec{l}_k - \vec{p}_{k-1}$ at the point $\vec{p}_{k-1}$ and tantengial to the vektor $\vec{r}_k - \vec{p}_k$ at the point $\vec{p}_k$.

Without any optional modifier arguments (described later in this text) the control points are computed automatically from the interpolation points by the

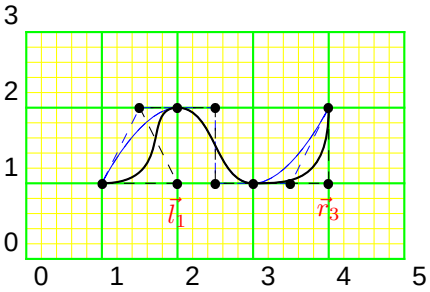
formulas[2]

$$
\begin{aligned}
\vec{l}_1 &= \vec{p}_0 \\
\vec{l}_k &= t_k(\vec{p}_k - \vec{p}_{k-2}) && \text{for } k = 2, \ldots, n \\
\vec{r}_k &= t_k(\vec{p}_{k-1} - \vec{p}_{k+1}) && \text{for } k = 1, \ldots, n-1 \\
\vec{r}_n &= \vec{p}_n
\end{aligned}
$$

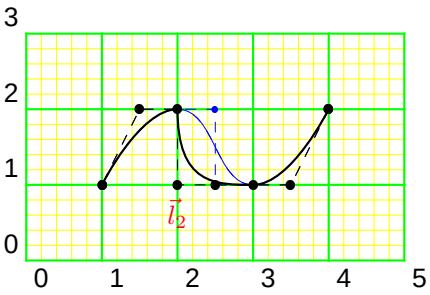
where t_k $(k = 1, \ldots, n)$ are real coefficients which are called tension and which default to the value `bcurveTension` $= 0.25$.

You can change the appearance of the curve by several modifiers. First of all you can directly set the left and right control points via the modifiers `l(x,y)` and `r(x,y)`, resp., as shown in the next two examples. The unmodified curve is drawn in the background in blue color.

```
\psset{showpoints=true}
\bcurve(1,1)l(2,1)%
   (2,2)(3,1)r(4,1)(4,2)
```

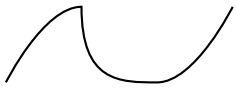



```
\bcurve(1,1)%
  (2,2)l(2,1)(3,1)(4,2)
```



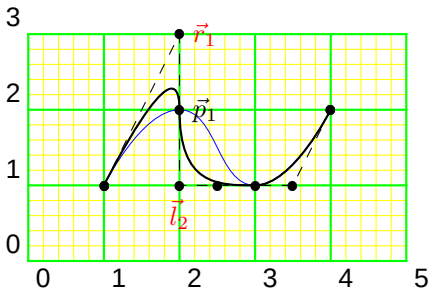
[2]Note that this method is very crude. To compute the curve such that the curvature is continuous would require solving a nonlinear system of equations. That is not implemented yet.

On the right hand side the last example is shown once more without grid and with `showpoints=false`. There, you see that there is a corner at the second interpolation point.



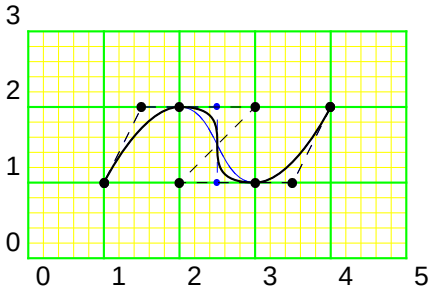
If you change some left control point $\vec{l}_k$ with the help of the `L(x,y)` modifier then the control point $\vec{r}_{k-1}$ is set symmetrically to $\vec{l}_k$ with respect to the interpolation point $\vec{p}_{k-1}$. In that way you get a smooth joint as demonstrated in the next example.

```
\bcurve(1,1)%
  (2,2)L(2,1)(3,1)(4,2)
```



With the `t{t}` modifier you can change the tension of the automatically computed control points of the current Bezier spline.

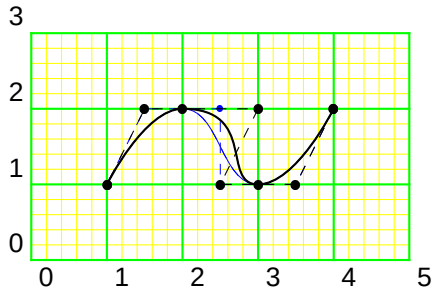
```
\bcurve(1,1)%
  (2,2)t{0.5}(3,1)(4,2)
```



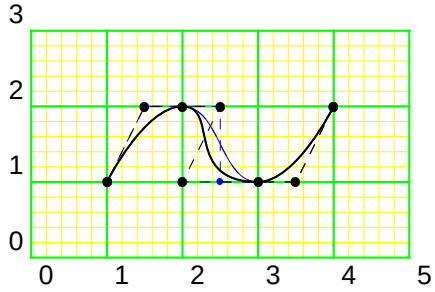
As you can see from the example both control points of the current spline are affected by the `t{t}` modifier.

If you want to change the tension of just the left or right control point you can use the `tl{t}` or `tr{t}` modifier, respectively, as demonstrated in the following two examples.

```
\bcurve(1,1)%
  (2,2)tl{0.5}(3,1)(4,2)
```

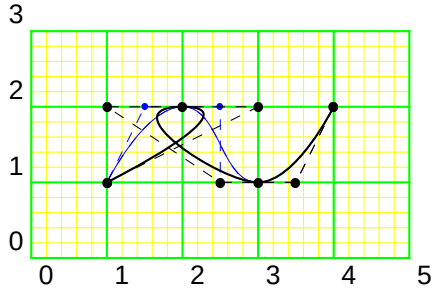



```
\bcurve(1,1)%
  (2,2)tr{0.5}(3,1)(4,2)
```



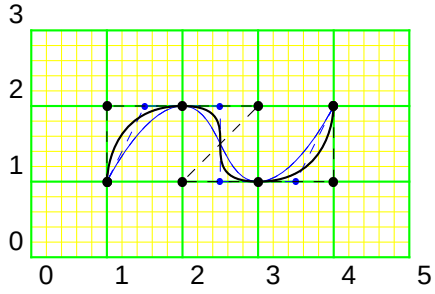
The `ts{`*t*`}` modifier changes the tension of the left and right control points next to the interpolation point which stands in front of the modifier. In the next example a negative tension value leads to a rather surprising effect.

```
\bcurve(1,1)%
  (2,2)ts{-0.5}(3,1)(4,2)
```



The default value of the tension can be set with the option `bcurveTension` as in the following example.

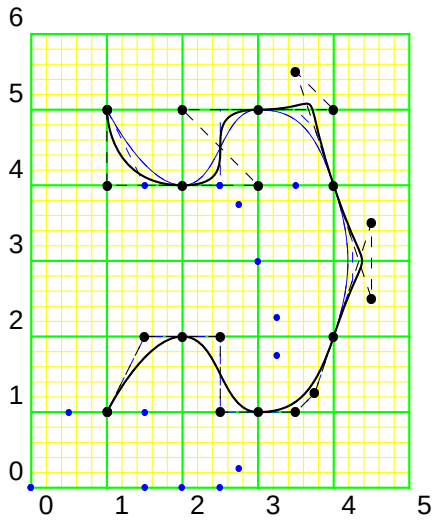
```
\bcurve[bcurveTension=0.5]%
  (1,1)(2,2)(3,1)(4,2)
```



You can set this option also with the help of the `\psset` macro.

It is even possible to change the value of `bcurveTension` in the middle of a `bcurve`. Just use the modifier `T{`*t*`}` for that purpose as shown in the following example.

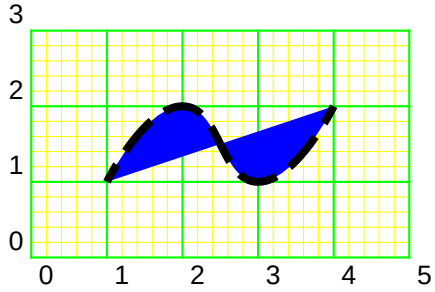
```
\bcurve(1,1)(2,2)(3,1)(4,2)%
  T{0.5}(4,4)(3,5)(2,4)(1,5)
```



Certainly, you can use the `T{`*t*`}` modifier several times in one curve. (Try it for yourself.)

The `linestyle` and `fillstyle` options (and several more) are respected by `\bcurve` as the following example shows.

```
\bcurve[linestyle=dashed,%
  linewidth=3pt,%
  dash=0.5 0.2,%
  fillstyle=solid,%
  fillcolor=blue](1,1)%
  (2,2)(3,1)(4,2)
\endpspicture
```



4 Things that do not work ('known bugs')

As already mentioned this project is something like an experiment. So, there are many things that do not work.

- Newlines inside the argument list are not ignored.
- The control points are computed in a rather crude way (see above). The `curvature` option is not recognised.
- If `fillstyle` is set to `solid` and `showpoints` to `true` then the fill color covers the interpolation and control points.
- Arrowheads do not work.