ANDERS THORUP: PSpictures for DVIPS, 10 September 2004

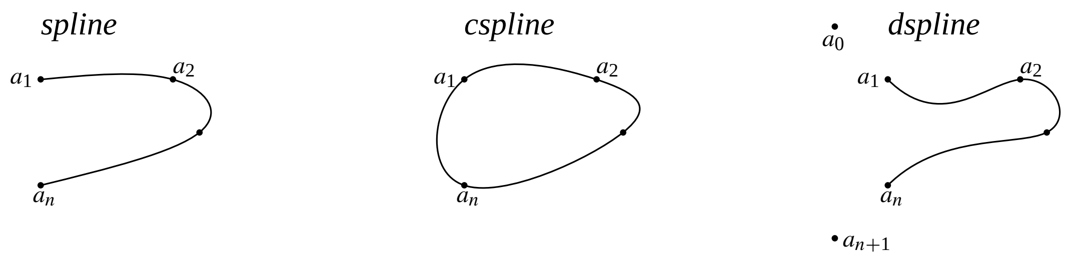
Introduction.

Let $a_1, \ldots, a_n$ be n points in the plane. A *spline* through the n points is a sequence of curves parameterized by polynomials $a_i(t)$ for $i = 1, \ldots, n-1$, such that the curve $a_i(t)$ begins at a_i and ends at a_{i+1} and such that the total curve from a_1 to a_n is smooth.

It would be desirable to be able to include in a TEX manuscript a spline with commands like the following:

`\spline{` $a_1 \ldots a_n$ `}`, `\cspline{` $a_1 \ldots a_n$ `}`, `\dspline{` $a_0\, a_1 \ldots a_{n+1}$ `}`,

where the n points a_i should be replaced by n pairs of coordinates. Naturally, the command should be packed into an environment giving an interpretation of the coordinates as multiples of a given unit length. The result of the first command should be a smooth curve through the points on the paper determined by the n pairs of coordinates. The second command should result in a closed spline, and the last in a directed spline: the points a_0 and a_{n+1} are control points determining the tangents at a_1 and a_n.



The Package.

This manual is the printout of a TEX file containing splines. The first part of the file is a package defining the necessary macros. It uses Bezier cubics as the polynomials $a_i(t)$, and it leaves the work of drawing the curves to the PostScript printer. To do so, PostScript needs, in addition to the coordinates of the n points, the coordinates of the so called control points. Each Bezier cubic is determined by two control points. For a spline through n points, the coordinates of the control points are determined by two systems of linear equations with n unknowns, see the section on splines at the end of the manual. The main innovation of the package is to let TEX solve the two systems of linear equations.

In addition, the package defines several other macros for curve drawing. To use the package as an input file in other TEX documents, find the following line in the file:

```
%\endinput
```

and erase the leading `%`. Then save the file in your favorite input directory. The macros of the package have been tested with plain TEX, with LaTeX, and with AmSTeX. It should be emphasized that the code resulting from the package is *not* portable. It assumes that DVIPS is used for transforming the dvi file into a PostScript printable file.

The Macros.

• The picture command sets up the overall environment in which all curve drawing commands have to be included. It has the following form:

`\PSpicture`{*xdimen*}{*ydimen*} *picture-material* `\endPSpicture`

The result of the command is a vertical (picture) box of width *xdimen* and height *ydimen* containing the *picture-material*. The *picture-material* consists of various picture commands setting objects in a coordinate system; by default, the origin of the coordinate system is the lower left vertex of the picture box.

• Coordinates (and lengths) inside the picture environment are interpreted as multiples of `\PSunit`. The default value is 1bp (1in=72bp),

```
\PSunit=1bp
```

It can be reset to any desired value outside the PSpicture environment. Inside the picture environment, coordinates and lengths are real numbers in decimal notation. Numerically, the maximal acceptable coordinate is $2^{15} - 1$. It is easy to let the two arguments, *xdimen* and *ydimen*, determining the dimensions of the picture box, depend on the `\PSunit`. For instance, `\PSpicture{3.5\PSunit}{2\PSunit}` is a valid start of the PSpicture command.

• Splines are drawn with `\spline`, `\cspline`, and `\dspline`:

`\spline`{*point-list*}, `\cspline`{*point-list*}, `\dspline`{*point-list*}

The *point-list* is a space separated list of the coordinates: *xcoord ycoord xcoord ycoord* ...*ycoord*. In particular, the number of coordinates has to be even. In addition, it has to be at least 6, corresponding to the coordinates of at least 3 points. The effect of `\spline` is to draw a natural spline through the points $a_1, \ldots, a_n$ given by the $2n$ coordinates in the list; `\cspline` draws a closed spline. Finally, `\dspline` draws a directed spline, see the section on splines.

The spline commands make heavy use of the TEX registers, and TEX will spend quite some time on the computations involved when the number of points is large. For a spline through n points, the number of dimension registers used is $4n + 5$. TEX has only 256 dimension registers, and some are in use when the spline command is started. So, depending on the TEX dialect in use, the maximal number of points for each spline command is less than 50. The spline commands will tell you if your number of points exceeds the maximum.

• Text may be inserted in the picture with `\LTX`, `\RTX`, `\CTX`:

`\LTX`{*TeX-stuff*}{*xcoord ycoord*}

The *TeX-stuff* is any horizontal TEX material. `\LTX` will set the *TeX-stuff* in an hbox left to the point with coordinates (*xcoord,ycoord*), and centered vertically. Similarly,

`\RTX`{*TeX-stuff*}{*xcoord ycoord*}, and `\CTX`{*TeX-stuff*}{*xcoord ycoord*}

will, respectively, set the *TeX-stuff* to the rigth of the point and centered horizontally around the point. Occasionally, the command `\TX`{*TeX-stuff-of-width-zero*}{*xcoord ycoord*} is useful. In addition, the *TeX-stuff* may be rotated with `\rotateRTX`:

`\rotateRTX`{*angle*} {*TeX-stuff*} {*xcoord ycoord*}

rotates the TeX-stuff, placed in an hbox with its lower left point at the given coordinates, clock-wise the specified angle around the point.

- Circles may be drawn with `\PScircle`:

  `\PScircle`{*xcoord ycoord radius*}

The center of the circle will be (*xcoord*,*ycoord*). Us usual, the *radius* is interpreted as a multiple of `\PSunit`.

- More generally, arcs may be drawn with `\PSarc`:

  `\PSarc`{$degree_1$ $degree_2$} {*xcoord ycoord radius*}

Thus `\PSarc{300 60}{0 0 1}` will draw the arc of the circle (counter clockwise oriented) from the point with angle $-\pi/3$ to the point with angle $\pi/3$. The command `\PScircle` is equivalent to `\PSarc{0 360}`. `\PSarcn` draws clockwise.

- Piece wise linear curves may be drawn by `\PSline` and `\PScline`:

  `\PSline`{*point-list*} `\PScline`{*point-list*}

The *point-list* has to consists of the coordinates of at least two points. For instance, `\PSline{1 0 0 1 0 0 1 1}`, in the picture environment and with `\PSunit=8pt`, produces this symbol: ⋉. With only two points in the list, that is, with four coordinates as argument, `\PSline` draws a straight line. It is an error to give only one point in the list. A closed polygon is drawn with `\PScline`.

- An arrow from one point to another may be drawn with `\PSarrow`:

  `\PSarrow`{$xcoord_1$ $ycoord_1$ $xcoord_2$ $ycoord_2$}

The result is an arrow with its tip at the point ($xcoord_2$,$ycoord_2$). The latter point has to be different from the first point in the list, since otherwise the direction of the arrowhead is undefined.

- Marks can be set at the points supporting the spline commands and the line commands.

  `\putPSmarks` `\putTeXmarks`

After `\putPSmarks`, a small circle is drawn around the points in the list of the following spline and line commands. After `\putTeXmarks`, a small black square is drawn. The default is to set no marks, and it can be obtained by `\nomarks`.

- A list of points, and in particular a single point, can be marked using

  `\TeXmark`{*point-list*} `\PSmark`{*point-list*}

The result is a mark set at each point in the list.

- By default, the lower left point of the picture box is the origin of the coordinate system. It can be reset with the command,

  `\llcoords`{*xcoord ycoord*}

Thus, after `\llcoords{2 -0.5}`, the lower left point of the picture box is the point with coordinates (2, −1/2). The change influences the following picture commands. Normally, the `\llcoords` command, if present, should be one of the first commands of the picture environment. However, if parts of the picture commands refer to different coordinate systems, it may be convenient to use several `\llcoords` commands.

- Curves resulting from the various picture commands have a thickness equal to the value of `\PSlinewidth`. By default,

```
\PSlinewidth=0.5pt
```

It can be given a different value globally. Inside the picture environment, it can be reset with `\setPSlinewidth`:

```
\setPSlinewidth{dimen}
```

The result is a change in the value of `\PSlinewidth`, and it influences the following picture commands in the same picture environment. Note that the argument to `\setPSlinewidth` is a usual TEX dimension.

- The curves generated by the picture commands can be dashed. The command,

```
\setdash{dimen}
```

will make the following curves in the picture environment appear as dashed curves with dashes of length equal to *dimen*. For instance, after `\setdash{5\PSlinewidth}`, the dashes will have a length equal to 5 times the line width. The default is reestablished with

```
\nodashes
```

- The size of the arrowhead produced with `\PSarrow` can be changed:

```
\setPSarrowheadlength{dimen}
```

As a default, the length of the arrow head is equal to 10 times the `\PSlinewidth`.

- With `\putTeXmarks` or `\TeXmark`, the points are actually marked with the content of `\markbox` defined as

```
\def\markbox{\hss\vrule height 3\PSlinewidth
                                  width 3\PSlinewidth\hss}
```

It is easy to redefine `\markbox` to produce different marks. With `\putPSmarks`, each point is marked with a small filled circle of radius equal to `\PSmarkradius`. The value of this dimension register is called whenever a PSmark is set, so you may change the value of `\PSmarkradius` at any time. The default value is set as follows:

```
\PSmarkradius=2\PSlinewidth,
```

but if you want this coupling between the size of the marks and the width of the curves, you have to repeat the above setting after each use of `\setPSlinewidth`.

- The PostScript interpreter can be addressed directly with the command `\PS`:

```
\PS{PostScript commands}
```

For instance, the command

```
    \PS{newpath 0 1 moveto 2 3 4 5 6 7 curveto stroke}
```

will draw a Bezier cubic from (0, 1) to (6, 7). It is easy to make syntax errors addressing PostScript directly. The result of a syntax error will most probably be that the printing process is stopped.

• Actually, each of the curve macros (`\spline`, `\PSline`, `\PSarc`, etc) described in this package results in PostScript code defining a path in the PostScript memory. The beginning of the code is the expansion of the control sequence `\newpath`, and the end of the code is the expansion of `\stroke`. The default effects of the expansions of these two control sequences are, respectively, to start a new path and to stroke the current path. In fact,

```
    \def\newpath{\PS{newpath}}   \def\stroke{\PS{stroke}}
```

However, with some knowledge of the PostScript language, you may redefine the two control sequences to obtain different effects. For instance, after

```
    \def\stroke{\PS{gsave 0.8 setgray fill grestore stroke}}
```

each of the following curves generated by the curve macros will have their enclosed area filled with gray before the curve is stroked. Also, with temporary settings like `\let\newpath= \relax`, `\let\stroke=\relax`, you may combine several curves into a single path object. Three commands deal directly with combinations of paths:

    `\subpaths`   `\longpath{`*xcoord ycoord*`}`   `\shortpaths`

After the first macro, `\subpaths`, each of the following curve macros will add a subpath to the current path. These curve macros are typically mixed with occurrences of the two other macros: After `\shortpaths`, each curve macro will contribute with a single subpath, and after `\longpath` each curve macro will append to the subpath started at the point (*xcoord, ycoord*), with the beginning of the curve connected with a straight line to the end of the previous curve. (Note that the arc defining commands need a little special care: even after the command `\shortpaths`, you have to specify where the arc begins: use `\PS{0 2 moveto}\PSarc{90 180}{0 0 2}}` to make sure that the arc starts at the point (0, 2) (as intended). Also, as the PSmarks are drawn with small circles, don't use them after `\shortpaths`.)

When the path has been created, you have to instruct PostScript explicitly on what to do with the path, for instance to fill it with `\PS{fill}` (or `\PS{eofill}`), or to stroke it with `\PS{stroke}`. In addition, you may want to restore the defaults with

```
    \normalpaths
```

For instance, in the following code

```
    \def\stroke{\PS{gsave 1 setgray fill grestore}}
    \PScircle{0 0 1}\PScline{0 0 1 0 1 2 0 2}\normalpaths
```

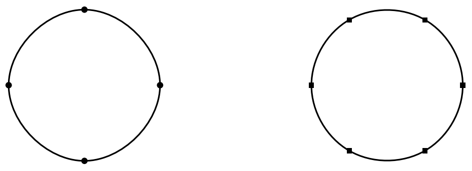
the `\PScircle` produces a white disk and the `\PScline` a white rectangle. Figures like this may be used to cover parts of previously set black material.

• The picture commands are meant to be used inside the picture environment. If used outside, most of them will provoke mysterious syntax errors, typically of the form

```
    ! Undefined control sequence.
                   ... \c@l
```

Examples.

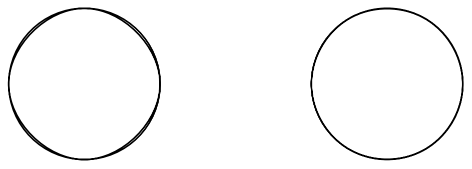
- A closed spline through 4 points on a circle, and through 6 points:



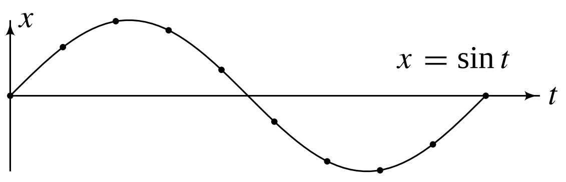
The drawing was produced with the following code:

```
\centerline{\PSunit=1truecm \PSpicture{6\PSunit}{2\PSunit}
\putPSmarks
\llcoords{-1 -1}
\cspline{
1 0
0 1
-1 0
0 -1}
\llcoords{-5 -1}
\putTeXmarks
\cspline{
1    0
0.5  0.8660
-0.5 0.8660
-1   0
-0.5 -0.8660
0.5  -0.8660}
\endPSpicture}
```

The `\centerline` centers the picture on the paper. Note the double use of the command `\llcoords`. With respect to the second circle, the lower left point of the picture box has coordinates $(-5, -1)$. The second spline is quite close to a circle, the first is not. The same code, with no marks, and an additional "true circle" drawn with `\PScircle{0 0 1}` for each of the two parts, produces the following:



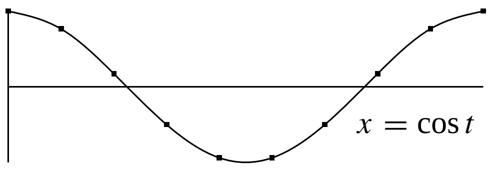
- The sine function from 0 to 2π, drawn as a natural spline through 10 points of distance $2\pi/9$. The coordinates of the 10 points were extracted from a table of the sine function.



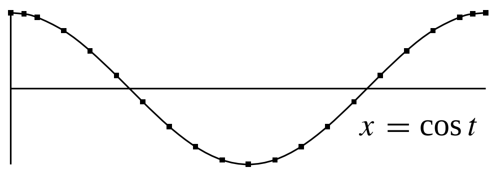
The spline for the sine function was produced by the following code:

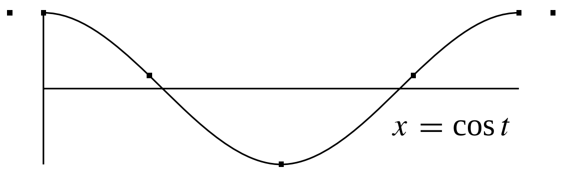
```
\centerline{\PSunit=1truecm
\PSpicture{7.2\PSunit}{2.4\PSunit}
\llcoords{0 -1.2}
\PSarrow{0 0 7 0}
\PSarrow{0 -1 0 1}
\putPSmarks
\spline{
0      0
0.6981 0.6428
           plus 7 more pairs of coordinates: t sin t
6.2830 0.0000
}
\TX{ $x$\hss}{0 1}
\TX{ $t$\hss}{7 0}
\TX{ $x=\sin t$\hss}{5 0.5}
\endPSpicture}
```

- The corresponding spline for the cosine is rather poor near the end points:

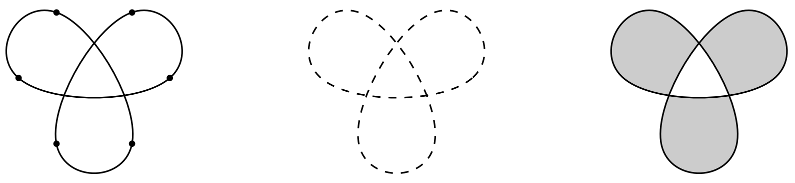


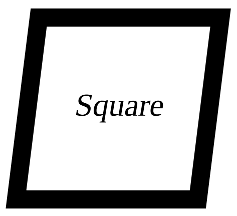
The misbehavior is caused by the fact that the cosine function has non vanishing second order derivatives at 0 and 2π. So the natural spline gives a poor approximation at the end points. With more supporting points the result is better. And with a directed spline, giving the horizontal tangents at the end points via two extra control points, fewer points are needed:





- The 6 points on the circle in a different order, with various strokes:

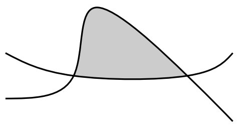




The square above was produced by the following code:

```
\centerline{\PSunit=5mm
\PSpicture{4.5\PSunit}{4\PSunit}
   \setPSlinewidth{2mm}
   \PScline{0 0 4 0 4.5 4 0.5 4}
   \TX{\hss \sl Square\hss}{2.25 2}
\endPSpicture}
```

- A dirty trick: The drawing



was made as follows: First, the two curves of the drawing were produced by two splines:

```
\spline{0 1 3 2 4 5 8 2 10 0} \spline{10 3 8 2 3 2 0 3}
```

Note that the two points of intersection, (3, 2) and (8, 2), appear in both point lists. When compiled, the two splines resulted in PostScript code, easily found in a special command in the dvi file:

```
newpath 0.0 1.0 moveto
 1.26195 1.006 2.52391 1.012 3.0 2.0 curveto
 3.47609 2.988 3.16669 4.95839 4.0 5.0 curveto
 4.83331 5.04161 6.80963 3.15477 8.0 2.0 curveto
 9.19037 0.84523 9.59546 0.42259 10.0 0.0 curveto
 stroke
newpath 10.0 3.0 moveto
 9.64455 2.60002 9.28911 2.20006 8.0 2.0 curveto
 6.71089 1.79994 4.48886 1.8 3.0 2.0 curveto
 1.51114 2.2 0.75554 2.6001 0.0 3.0 curveto
 stroke
```

It is easy to identify from this code the 4+3 Bezier cubics forming the two curves. In particular, it was easy to extract the PostScript code of the 3 Bezier cubics forming the boundary of the shaded region. In turn, this extract was inserted with shading commands directly in the tex file, before the two spline commands, as follows:

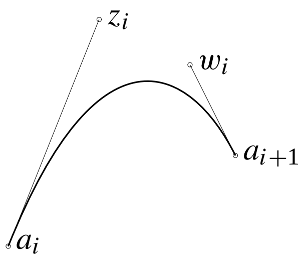
```
\PS{newpath 3.0 2.0 moveto
 3.47609 2.988 3.16669 4.95839 4.0 5.0 curveto
 4.83331 5.04161 6.80963 3.15477 8.0 2.0 curveto
 6.71089 1.79994 4.48886 1.8 3.0 2.0 curveto
 gsave 0.8 setgray fill grestore}
```

Splines.

A possible choice of the curves $a_i(t)$ in the general spline problem is to let each be a *Bezier cubic*. Assume given, in addition to the two points a_i and a_{i+1}, two control points z_i and w_i. Then, by definition, the Bezier cubic from a_i to a_{i+1} determined by the two control points is the cubic curve $a_i(t)$ for $0 \le t \le 1$ given by the polynomial,

$$a_i(t) = a_i(1-t)^3 + 3z_i t(1-t)^2 + 3w_i t^2(1-t) + a_{i+1}t^3.$$

At the extremal points, the values are $a_i(0) = a_i$ and $a_i(1) = a_{i+1}$, and the derivatives are $a_i'(0) = 3(z_i - a_i)$ and $a_i'(1) = 3(-w_i + a_{i+1})$. Thus the two control points determine the tangents at the extremal points. More precisely, the vector from a_i to z_i is one third of the derivative $a_i'(0)$ and the vector from w_i to a_{i+1} is one third of the derivative $a_i'(1)$.



For cubic splines, the condition of smoothness is that the total curve is two times continuously differentiable. For the Bezier cubic above, the derivatives are given by the equations,

$$a_i'(t)/3 = [z_i - a_i](1-t)^2 + 2[w_i - z_i]t(1-t) + [a_{i+1} - w_i]t^2,$$

$$a_i''(t)/6 = [-2z_i + w_i + a_i](1-t) + [z_i - 2w_i + a_{i+1}]t.$$

In particular, the condition of continuity for the first order derivatives, $a_i'(1) = a_{i+1}'(0)$ for $i = 1, \dots, n-2$, is the following set of $n-2$ equations,

$$w_i + z_{i+1} = 2a_{i+1}.$$

It is natural to define $z_n := 2a_n - w_{n-1}$. Then the $n-1$ control points w_i are determined by the n points z_i through the equations above. So, the sequences of Bezier cubics $a_i(t)$ forming a differentiable curve from a_1 to a_n are parameterized by the sets of n points z_i.

In terms of the z_i, the second order derivatives at the extremal points are given as follows, for $i = 1, \dots, n-1$,

$$a_i''(0)/6 = -2z_i - z_{i+1} + a_i + 2a_{i+1}, \quad a_i''(1)/6 = z_i + 2z_{i+1} - 3a_{i+1}.$$

Hence, the condition of continuity for the second order derivatives, $a_i''(1) = a_{i+1}''(0)$, is the following set of equations,

$$(*) \qquad z_i + 4z_{i+1} + z_{i+2} = 4a_{i+1} + 2a_{i+2}, \quad i = 1, \dots, n-2.$$

To determine the spline completely, we need two more independent conditions on the n control points z_i. One natural choice is to require that the second order derivatives vanish at the extremal points a_1 and a_n, that is, $a_1''(0) = 0$ and $a_{n-1}''(1) = 0$. It imposes the following two conditions on the z_i:

$$(**) \qquad 2z_1 + z_2 = a_1 + 2a_2, \qquad z_{n-1} + 2z_n = 3a_n.$$

The spline through $a_1, \dots, a_n$ determined by the equations (*) and (**) is called the *natural spline*. It should be emphasized that there are other natural conditions to impose on a spline. For instance, the spline is determined by giving the tangents at the two extremal points, or, equivalently, by giving the first control point z_1 and the last control point $w_{n-1} = 2a_n - z_n$. We will call this spline the *directed spline*. In the package, the two control points are given by adding two extra points, a_0 and a_{n+1}, to the list of points a_i, with the interpretation that $z_1 - a_1 = a_1 - a_0$ and $z_n = a_{n+1}$; so we add the following equations:

$$(***) \qquad z_1 = 2a_1 - a_0, \quad z_n = a_{n+1}.$$

A third possibility is to determine the spline by the condition that the second order derivatives at the extremal points a_1 and a_n are, respectively, equal to the second order derivatives at the (nearby) points a_2 and a_{n-2}. The latter condition replaces (***) by the following equations:

$$z_1 + z_2 = (a_1 + 5a_2)/3, \quad z_{n-1} + z_n = (a_{n-1} + 5a_n)/3.$$

This third possibility is not implemented in the package.

A different problem is to look for a *closed spline*, that is, to look for an additional cubic $a_n(t)$ connecting a_n to a_1. The further condition is then that the total closed curve is smooth. Reading the indices modulo n, the further condition simply adds the two equations (*) for $i = n-1$ and $i = n$.

There are several reasons for choosing (Bezier) cubics to define the spline. The most important is simply that the PostScript printer is able to draw a Bezier cubic given the coordinates of the four points. This package implements the natural spline and the closed spline.

Solution of the equations.

The set of linear equations for the control points z_i, for the natural spline, for the directed spline, and for the closed spline, have a unique solution. For the natural spline, the equations are (*) and (**), and the matrix of coefficients is tridiagonal:

$$\begin{bmatrix} 2 & 1 & & & \\ 1 & 4 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & 4 & 1 \\ & & & 1 & 2 \end{bmatrix}$$

The equations (*) and (**) are solved by Gaussian elimination resulting in the following algorithm (where the reciprocals of the diagonal elements are stored in the factors f_i):

```
SPLINE:          input $n,\ a_1, \dots, a_n$
                 registers $i,\ z_1, \dots, z_n,\ f_1, \dots, f_n$
    (initialize) $f_1 \leftarrow 1/2\ ,\ \ z_1 \leftarrow a_1 + 2a_2$
  (forward loop) for $i = 2, \dots, n-1$ do
                     $z_i \leftarrow 4a_i + 2a_{i+1} - z_{i-1}f_{i-1}$
                     $f_i \leftarrow 1/(4 - f_{i-1})$
                 end
        (finish) $z_n \leftarrow 3a_n - z_{n-1}f_{n-1}\ ,\ \ f_n \leftarrow 1/(2 - f_{n-1})$
         (solve) $z_n \leftarrow z_n f_n$
          (loop) for $i = n-1, \dots, 2, 1$ do
                     $z_i \leftarrow (z_i - z_{i+1}) f_i$
                 end
                 output $z_1, \dots, z_n$
```

In the loop, the factor f_i converges to $2 - \sqrt{3} \approx 0.3$. So the multiplications by f_i and the inversions of $4 - f_i$ cause no problems of arithmetic overflow.

For the closed spline, the equations (*) modulo n give a more complicated matrix:

$$\begin{bmatrix} 4 & 1 & & & 1 \\ 1 & 4 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & 4 & 1 \\ 1 & & & 1 & 4 \end{bmatrix}$$

and a more complicated algorithm (the factor f_n is inverted at the end of the forward loop):

```
CSPLINE:        input $n,\ a_1, \ldots, a_n$
                registers $i,\ z_1, \ldots, z_n,\ f_1, \ldots, f_n,\ q_1, \ldots, q_n$
   (initialize) $f_1 \leftarrow 1/2\ ,\ \ q_1 \leftarrow 1\ ,\ \ z_1 \leftarrow 4a_1 + 2a_2\ ,\ \ f_n \leftarrow 4\ ,\ \ z_n \leftarrow 4a_n + 2a_1$
 (forward loop) for $i = 2, \ldots, n-1$ do
                    $z_i \leftarrow 4a_i + 2a_{i+1} - z_{i-1} f_{i-1}$
                    $q_i \leftarrow -q_{i-1} f_{i-1}\ ,\ \ f_i \leftarrow 1/(4 - f_{i-1})$
                    $f_n \leftarrow f_n + q_i q_{i-1}\ ,\ \ z_n \leftarrow z_n + q_i z_{i-1}$
                end
       (finish) $q_n \leftarrow -(1 + q_{n-1}) f_{n-1},\ z_n \leftarrow z_n + q_n z_{n-1}$
                $f_n \leftarrow f_n + q_n(1 + q_{n-1})\ ,\ \ f_n \leftarrow 1/f_n$
        (solve) $z_n \leftarrow z_n f_n$
         (loop) for $i = n-1, \ldots, 2, 1$ do
                    $z_i \leftarrow (z_i - z_{i+1} - q_i z_n) f_i$
                end
                output $z_1, \ldots, z_n$
```

Finally, for the directed spline, the equations (*) and (***) correspond to the following matrix:

$$\begin{bmatrix} 1 & & & & \\ 1 & 4 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & 4 & 1 \\ & & & & 1 \end{bmatrix}$$

The equations (*) and (***) are solved by the following algorithm:

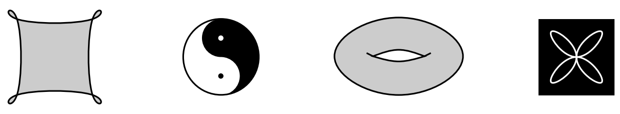
```
DSPLINE:        input $n,\ a_0, a_1, \ldots, a_n, a_{n+1}$
                registers $i,\ z_1, \ldots, z_n,\ f_2, \ldots, f_{n-1}$
   (initialize) $f_2 \leftarrow 1/4,\ z_1 \leftarrow 2a_1 - a_0,\ z_2 \leftarrow 4a_2 + 2a_3 - z_1$
 (forward loop) for $i = 3, \ldots, n-1$ do
                    $z_i \leftarrow 4a_i + 2a_{i+1} - z_{i-1} f_{i-1}$
                    $f_i \leftarrow 1/(4 - f_{i-1})$
                end
       (finish) $z_n \leftarrow a_{n+1}$
         (loop) for $i = n-1, \ldots, 3, 2$ do
                    $z_i \leftarrow (z_i - z_{i+1}) f_i$
                end
                output $z_1, \ldots, z_n$
```

Implementation.

The three algorithms are implemented in the package. First, while reading the arguments to `\spline`, the arguments are stored in dimension registers A allocated dynamically. The factors f appearing in the algorithms are stored in count registers F as the integers $F := f * B$, where the base is $B = 2^{15}$. This choice of base allows the square B^2 to be a valid TeX integer, and it allows the following procedure for the multiplications $z * f$ in the algorithm: split z into $z = zh * B + zl$ where the two "digits" zh and zl are less than B. Then $zf = zh * F + zl * F/B$. Similarly, the inversion $f \leftarrow (m - f)^{-1}$ (where $m = 2$ or $m = 4$) is obtained as $F \leftarrow B^2/(mB - F)$. The factors q appearing in the other algorithms are treated similarly.

The count registers F and the dimension registers Z are allocated during the forward loop. Finally, when the equations are solved, the contents of the relevant registers are passed to the PostScript interpreter in a `\special`.

The main part of the TeX-code was written in 1995. The code implementing the DSPLINE algorithm was written by Ulrik Buchholtz in September 2003. The code for determining the PSmarkradius was changed as described above in October 2003. Compared to the previous version, the default circles produced by `\PSmark` and `\putPSmarks` have doubled their sizes.



Anders Thorup

Anders Thorup