The spath package

Andrew Stacey
stacey@math.ntnu.no

v1.0 from 2011/06/03

1 Introduction

2 Implementation

Load the `pgf.oo` module.

```
\usepgfmodule{oo}
```

Define some auxilliary macros (should replace these with `etoolbox`

`\ge@addto@macro`

```
\long\def\ge@addto@macro#1#2{%
  \begingroup
  \toks@\expandafter\expandafter\expandafter{\expandafter#1#2}%
  \xdef#1{\the\toks@}%
  \endgroup}

\def\tikz@clear@foreach{%
\let\pgffor@beginhook=\pgfutil@empty
\let\pgffor@endhook=\pgfutil@empty
\let\pgffor@afterhook=\pgfutil@empty
}

```

`\ge@addbefore@macro`

```
\long\def\ge@addbefore@macro#1#2{%
  \begingroup
  \toks@\expandafter\expandafter\expandafter{\expandafter#2#1}%
  \xdef#1{\the\toks@}%
  \endgroup}
```

`\g@addbefore@macro`

```
\long\def\g@addbefore@macro#1#2{%
  \def\@temp{#2}%
  \ge@addbefore@macro{#1}{\@temp}}
```

```
\tikzset{%
  use path/.code={%
    \pgfsyssoftpath@setcurrentpath{#1}%
  }
}
```

This is the `spath` object. It is defined using the `pgf.oo` module. It represents a "soft path".

```
\pgfooclass{spath}{
```

Certain attributes of the soft path are saved to avoid having to recompute them for every operation. The `path` attribute is the soft path itself.

```
  \attribute path;
```

The `length` is the crudest measure of length: the number of soft path tokens in the path.

```
  \attribute length;
```

Since some path pieces consist of more than one token, the `length` is longer than what one might call the path length. The `real length` counts only the key tokens.

```
  \attribute real length;
```

A path can be viewed as being a collection of drawing tokens and moving tokens. We call a collection of subsequent drawing tokens a `component`. The `number of components` counts this. (Actually, it counts the number of `movetos`, which could be more if there are successive `movetos`.)

```
  \attribute number of components;
```

The `initial point` is where the path starts.

```
  \attribute initial point;
```

The `final point` is where the path ends.

```
  \attribute final point;
```

The `first action` is the first drawing action on the path (so not including the initial `moveto`.)

```
  \attribute first action;
```

The `last action` is the last action on the path (which might be a `moveto`.)

```
  \attribute last action;
```

To compute most of these attributes, we need to walk along the path. Obviously, we save their values to avoid having to do this walk too often. The `prepared` attribute tells us whether or not we've already done this walk.

```
  \attribute prepared;
```

For the path tapering, we need to know the line width and the width to taper to. The `taper line width` holds the latter.

```
  \attribute taper line width;
```

We need to keep track of our bounding box

```
  \attribute min bb;
  \attribute max bb;
```

Our last attribute is a “scratch” attribute for saving some macro that relates to the object in some way.

```
\attribute scratch pad;
```

Now we have the methods for the `spath` object.

We start with the creator. If given a path, we store it as the `path` attribute.

```
\method spath(#1) {%
  \let\spath@temp=#1\relax
  \ifx\spath@temp\relax
  \else
   \pgfoolet{path}{#1}%
  \fi
}
```

Set path from current path

```
\method use current path() {%
  \pgfsyssoftpath@getcurrentpath{\spath@temp}%
  \pgfoolet{path}{\spath@temp}%
  \let\spath@temp=\pgfutil@empty
}
```

Next, we have some generic attribute handling methods. These allow us to set, get, and let attributes, and also to show them in the logs.

We start with one that inserts the value of the given attribute in the token stream.

```
\method value(#1) {%
  \pgfoovalueof{#1}%
}
```

This sets the attribute (first argument) to the second argument.

```
\method set(#1,#2) {%
  \pgfooset{#1}{#2}%
}
```

This lets the attribute (first argument) to the second argument, which must be a macro.

```
\method let(#1,#2) {%
  \pgfoolet{#1}{#2}%
}
```

This sets the attribute (first argument) to the second argument if the attribute is not empty.

```
\method set if not empty(#1,#2) {%
\pgfooget{#1}{\spath@tmp}%
\ifx\spath@tmp\pgfutil@empty
  \pgfooset{#1}{#2}%
\fi
}
```

This lets the attribute (first argument) to the second argument, which must be a macro, if the attribute is not empty.

```
\method let if not empty(#1,#2) {%
\pgfooget{#1}{\spath@tmp}%
\ifx\spath@tmp\pgfutil@empty
  \pgfoolet{#1}{#2}%
\fi
}
```

This gets the attribute (first argument) into the second argument, which must be a macro.

```
\method get(#1,#2) {%
  \pgfooget{#1}{#2}%
}
```

This shows the attribute (argument) in the logs, via `\show`. (Should prefix with the attribute name to be more useful.)

```
\method show(#1) {%
  \pgfooget{#1}{\@temp}%
  \show\@temp
}
```

This says what we are

```
\method what am I() {%
   \message{I am an spath object}%
}
```

This clones the path to the macro passed to it. The macro must already exist (at the moment, though in thinking about it that's a little daft) but will be overwritten.

```
\method clone(#1) {%
  \let\spath@newpath=#1\relax
  \ifx\spath@newpath\relax
  \else
   \pgfoonew \spath@newpath =new spath()%
\tikz@clear@foreach
   \foreach \attribute in {
    path,
    length,
    real length,
    number of components,
    initial point,
    final point,
    first action,
    last action,
    prepared,
    taper line width,
    min bb,
    max bb%,
   } {
    \pgfooget{\attribute}{\spath@temp}%
    \spath@newpath.let(\attribute,\spath@temp)%
  }
```

```
    \let#1=\spath@newpath%
    \fi
 }
```

Now we begin the path-walking methods. These methods work by "evaluating" the path under certain conditions. A soft path consists of a series of commands which take two arguments, nominally an x-coordinate and a y-coordinate. By reseting the commands, we can make a soft path rewrite itself in a modified form, or to extract certain information from it. The actual implementation of these uses various auxilliary macros which are defined later.

This method translates the path by #2 in x and #3 in y. If #1 is given then it should be a macro and it will be turned into an object with the translated path, otherwise the current object is modified.

It works by redefining each soft path token to add the given dimensions to its arguments and then rewrite itself back to the path. Having done that, we have to adjust the various other attributes which are location-specific.

```
 \method translate path(#1,#2,#3) {%
   \let\spath@newpath=#1\relax
   \ifx\spath@newpath\relax
   \let\spath@tmppath=\pgfutil@empty
   \begingroup
    \spath@trx=#2\relax
    \spath@try=#3\relax
    \spath@translate@init
    \pgfoovalueof{path}%
    \pgfoolet{path}{\spath@tmppath}%
    \let\spath@tmppath=\pgfutil@empty
 \tikz@clear@foreach
    \foreach \attribute in {
     initial point,
     final point,
     min bb,
     max bb%,
    } {
     \pgfooget{\attribute}{\spath@temp}%
     \ifx\spath@temp\pgfutil@empty
     \else
      \spath@temp
      \advance\pgf@x by \spath@trx\relax
      \advance\pgf@y by \spath@try\relax
      \edef\spath@temp{\noexpand\pgfpoint{\the\pgf@x}{\the\pgf@y}}%
      \pgfoolet{\attribute}{\spath@temp}%
     \fi
   }
   \endgroup
   \let\spath@tmppath=\pgfutil@empty
  \else
   \pgfoothis.clone(\spath@newpath)%
   \begingroup
   \spath@newpath.translate path(,#2,#3)%
   \endgroup
   \let#1=\spath@newpath
  \fi
 }
```

Transform path according to an affine transformation

```
 \method transform path(#1,#2) {%
   \let\spath@newpath=#1\relax
   \ifx\spath@newpath\relax
   \let\spath@tmppath=\pgfutil@empty
   \begingroup
    \pgftransformreset
    \pgfsettransform{#2}%
    \spath@transform@init
    \pgfoovalueof{path}%
    \pgfoolet{path}{\spath@tmppath}%
    \let\spath@tmppath=\pgfutil@empty
 \tikz@clear@foreach
    \foreach \attribute in {
     initial point,
     final point,
     min bb,
     max bb,
     prepared%,
    } {
     \pgfoolet{\attribute}{\pgfutil@empty}%
    }%
   \endgroup
   \let\spath@tmppath=\pgfutil@empty
  \else
   \pgfoothis.clone(\spath@newpath)%
   \begingroup
   \spath@newpath.transform path(,#2)%
   \endgroup
   \let#1=\spath@newpath
  \fi
 }
```

Scale all coordinates

```
 \method scale path(#1,#2,#3) {%
   \let\spath@newpath=#1\relax
   \ifx\spath@newpath\relax
   \let\spath@tmppath=\pgfutil@empty
   \begingroup
    \edef\spath@scx{#2}%
    \edef\spath@scy{#3}%
    \ifx\spath@scy\pgfutil@empty
     \edef\spath@scy{#2}%
    \fi
    \spath@scale@init
    \pgfoovalueof{path}%
    \pgfoolet{path}{\spath@tmppath}%
    \let\spath@tmppath=\pgfutil@empty
 \tikz@clear@foreach
    \foreach \attribute in {
     initial point,
     final point,
     min bb,
     max bb%,
    } {
     \pgfooget{\attribute}{\spath@temp}%
     \ifx\spath@temp\pgfutil@empty
     \else
      \spath@temp
      \pgf@x=\spath@scx\pgf@x\relax
      \pgf@y=\spath@scy\pgf@y\relax
      \edef\spath@temp{\noexpand\pgfpoint{\the\pgf@x}{\the\pgf@y}}%
      \pgfoolet{\attribute}{\spath@temp}%
     \fi
   }
   \endgroup
   \let\spath@tmppath=\pgfutil@empty
  \else
   \pgfoothis.clone(\spath@newpath)%
   \begingroup
   \spath@newpath.scale path(,#2,#3)%
   \endgroup
   \let#1=\spath@newpath
  \fi
 }
```

Round all the corners on the path, ignoring any that are already set

```
\method round corners(#1,#2,#3) {%
  \let\spath@newpath=#1\relax
  \ifx\spath@newpath\relax
  \let\spath@tmppath=\pgfutil@empty
  \begingroup
   \spath@round@init
   \edef\spath@rndx{#2}%
   \edef\spath@rndy{#3}%
\ifx\spath@rndy\pgfutil@empty
 \let\spath@rndy\spath@rndx
\fi
   \pgfoovalueof{path}%
   \pgfoolet{path}{\spath@tmppath}%
  \endgroup
   \pgfooget{path}{\spath@tmppath}%
   \pgfprocessround\spath@tmppath\spath@temppath
   \pgfoolet{path}{\spath@temppath}%
 \else
```

```
  \pgfooget{path}{\spath@tmppath}%
  \spath@newpath =new spath(\spath@tmppath)%
  \begingroup
  \spath@newpath.round corners(,#2)%
  \endgroup
  \let#1=\spath@newpath
 \fi
}
```

This shrinks the path towards a particular point by a given number of points. It doesn't scale the path, rather it adds a number of points to each coordinate depending on which quadrant the point is in relative to the given point.

```
\method shrink path(#1,#2,#3,#4) {%
   \let\spath@newpath=#1\relax
   \ifx\spath@newpath\relax
   \let\spath@tmppath=\pgfutil@empty
   \begingroup
   \spath@trx=#2\relax
   \spath@try=#3\relax
   \edef\spath@shrinkby{#4}%
   \spath@shrink@init
   \pgfoovalueof{path}%
   \pgfoolet{path}{\spath@tmppath}%
   \let\spath@tmppath=\pgfutil@empty
\tikz@clear@foreach
   \foreach \attribute in {
    initial point,
    final point,
    min bb,
    max bb%,
   } {
    \pgfooget{\attribute}{\spath@temp}%
    \ifx\spath@temp\pgfutil@empty
    \else
     \spath@temp
     \pgfmathsetlength{\pgf@xa}{\pgf@x < \spath@trx ? \pgf@x + \spath@shrinkby : (\pgf@x > \s
     \pgfmathsetlength{\pgf@ya}{\pgf@y < \spath@try ? \pgf@y + \spath@shrinkby : (\pgf@y > \s
     \edef\spath@temp{\noexpand\pgfpoint{\the\pgf@xa}{\the\pgf@ya}}%
     \pgfoolet{\attribute}{\spath@temp}%
    \fi
  }
  \endgroup
  \let\spath@tmppath=\pgfutil@empty
 \else
  \pgfoothis.clone(\spath@newpath)%
  \begingroup
  \spath@newpath.shrink path(,#2,#3)%
  \endgroup
  \let#1=\spath@newpath
 \fi
}
```

This method prepares the path for a spirograph by applying an incremental rotation to each coordinate. If #1 is given then it should be a macro and it will be turned into an object with the transformed path, otherwise the current object is modified.

It works by redefining each soft path token to apply the given rotation to its arguments and then rewrite itself back to the path. Having done that, we have to adjust the various other attributes which are location-specific.

```
\method prepare spirograph(#1,#2,#3,#4) {%
  \pgfoothis.prepare()%
  \let\spath@newpath=#1\relax
  \ifx\spath@newpath\relax
  \let\spath@tmppath=\pgfutil@empty
  \begingroup
   \spath@trx=#2\relax
   \spath@try=#3\relax
   \def\spath@n{0}%
   \pgfooget{length}{\spath@temp}%
   \pgfmathsetmacro{\spath@gang}{#4/(\spath@temp-1)}%
   \spath@spirograph@init
   \pgfoovalueof{path}%
   \pgfoolet{path}{\spath@tmppath}%
   \let\spath@tmppath=\pgfutil@empty
   \pgfooget{final point}{\spath@temp}%
   \ifx\spath@temp\pgfutil@empty
   \else
    \spath@temp
    \pgf@xa=\pgf@x\relax
    \pgf@ya=\pgf@y\relax
    \advance\pgf@xa by -\spath@trx\relax
    \advance\pgf@ya by -\spath@try\relax
    \pgfmathsetmacro{\spath@gcos}{cos(#4)}%
    \pgfmathsetmacro{\spath@gsin}{sin(#4)}%
    \pgfmathsetlength{\pgf@xb}{\spath@gcos * \pgf@xa - \spath@gsin * \pgf@ya}%
    \pgfmathsetlength{\pgf@yb}{\spath@gsin * \pgf@xa + \spath@gcos * \pgf@ya}%
    \advance\pgf@xb by \spath@trx\relax
    \advance\pgf@yb by \spath@try\relax
    \edef\spath@temp{\noexpand\pgfpoint{\the\pgf@xb}{\the\pgf@yb}}%
    \pgfoolet{final point}{\spath@temp}%
   \fi
  \endgroup
  \let\spath@tmppath=\pgfutil@empty
 \else
  \pgfoothis.clone(\spath@newpath)%
  \begingroup
  \spath@newpath.prepare spirograph(,#2,#3,#4)%
  \endgroup
  \let#1=\spath@newpath
 \fi
 }
```

This method actually makes a spirograph. The arguments are the point of rotation and the number of iterations.

```
\method spirograph(#1,#2,#3,#4) {
    \pgfoothis.prepare()%
    \let\spath@newpath=#1\relax
    \ifx\spath@newpath\relax
    \begingroup
     \pgfmathsetmacro{\spath@gn}{360/#4}%
     \pgfoothis.prepare spirograph(,#2,#3,\spath@gn)%
     \pgfoothis.clone(\spath@tempa)%
     \pgfmathsetmacro{\spath@gna}{2*\spath@gn}%
     \pgfmathsetmacro{\spath@gnb}{360-\spath@gn}%
  \tikz@clear@foreach
     \foreach \k in {\spath@gn,\spath@gna,...,\spath@gnb} {%
      \spath@tempa.rotate path(,\spath@gn)%
      \pgfoothis.weld(,\spath@tempa)%
     }%
    \endgroup
   \else
    \pgfoothis.clone(\spath@newpath)%
    \begingroup
    \spath@newpath.spirograph(,#2,#3,#4)%
    \endgroup
    \let#1=\spath@newpath
   \fi
}
```

This method rotates the path by the given angle. If #1 is given then it should be a macro and it will be turned into an object with the transformed path, otherwise the current object is modified.

It works by redefining each soft path token to apply the given rotation to its arguments and then rewrite itself back to the path. Having done that, we have to adjust the various other attributes which are location-specific.

```
  \method rotate path(#1,#2) {%
    \let\spath@newpath=#1\relax
    \ifx\spath@newpath\relax
    \let\spath@tmppath=\pgfutil@empty
    \begingroup
     \pgfmathsetmacro{\spath@gcos}{cos(#2)}%
     \pgfmathsetmacro{\spath@gsin}{sin(#2)}%
     \spath@rotate@init
     \pgfoovalueof{path}%
     \pgfoolet{path}{\spath@tmppath}%
     \let\spath@tmppath=\pgfutil@empty
     \pgfooget{initial point}{\spath@temp}%
     \ifx\spath@temp\pgfutil@empty
     \else
      \spath@temp
      \pgf@xa=\pgf@x\relax
      \pgf@ya=\pgf@y\relax
      \pgfmathsetlength{\pgf@xb}{\spath@gcos * \pgf@xa - \spath@gsin * \pgf@ya}%
      \pgfmathsetlength{\pgf@yb}{\spath@gsin * \pgf@xa + \spath@gcos * \pgf@ya}%
      \edef\spath@temp{\noexpand\pgfpoint{\the\pgf@xb}{\the\pgf@yb}}%
      \pgfoolet{initial point}{\spath@temp}%
     \fi
     \pgfooget{final point}{\spath@temp}%
     \ifx\spath@temp\pgfutil@empty
     \else
      \spath@temp
      \pgf@xa=\pgf@x\relax
      \pgf@ya=\pgf@y\relax
      \pgfmathsetlength{\pgf@xb}{\spath@gcos * \pgf@xa - \spath@gsin * \pgf@ya}%
      \pgfmathsetlength{\pgf@yb}{\spath@gsin * \pgf@xa + \spath@gcos * \pgf@ya}%
      \edef\spath@temp{\noexpand\pgfpoint{\the\pgf@xb}{\the\pgf@yb}}%
      \pgfoolet{final point}{\spath@temp}%
     \fi
    \endgroup
    \let\spath@tmppath=\pgfutil@empty
   \else
    \pgfoothis.clone(\spath@newpath)%
    \begingroup
    \spath@newpath.rotate path(,#2)%
    \endgroup
    \let#1=\spath@newpath
   \fi
  }
```

Get the length of the path (number of pieces)

```
\method length() {%
  \pgfooget{length}{\spath@temp}%
  \ifx\spath@temp\pgfutil@empty
  \setcounter{spath@length}{0}%
  \begingroup
  \spath@length@init
  \pgfoovalueof{path}%
  \endgroup
   \edef\spath@temp{\the\value{spath@length}}%
   \pgfoolet{length}{\spath@temp}%
  \fi
  \spath@temp
}
```

Get the real length of the path (number of components that draw)

```
\method real length() {%
  \pgfooget{real length}{\spath@temp}%
  \ifx\spath@temp\pgfutil@empty
  \setcounter{spath@length}{0}%
  \begingroup
  \spath@reallength@init
  \pgfoovalueof{path}%
  \endgroup
   \edef\spath@temp{\the\value{spath@length}}%
   \pgfoolet{real length}{\spath@temp}%
  \fi
  \spath@temp
}
```

Get the number of components of the path

```
\method number of components() {%
  \pgfooget{number of components}{\spath@temp}%
  \ifx\spath@temp\pgfutil@empty
  \setcounter{spath@length}{0}%
  \begingroup
  \spath@components@init
  \pgfoovalueof{path}%
  \endgroup
   \edef\spath@temp{\the\value{spath@length}}%
   \pgfoolet{real length}{\spath@temp}%
  \fi
  \spath@temp
}
```

Get the initial point of the path

```
\method initial point() {%
  \pgfooget{initial point}{\spath@temp}%
  \ifx\spath@temp\pgfutil@empty
  \begingroup
  \spath@start@init
  \pgfoovalueof{path}%
  \edef\spath@temp{\noexpand\pgfpoint{\spath@sx}{\spath@sy}}%
  \pgfoolet{initial point}{\spath@temp}%
  \endgroup
  \pgfooget{initial point}{\spath@temp}%
  \fi
  \spath@temp
}
```

Get the final point of the path

```
\method final point() {%
  \pgfooget{final point}{\spath@temp}%
  \ifx\spath@temp\pgfutil@empty
  \begingroup
  \spath@end@init
  \pgfoovalueof{path}%
  \edef\spath@temp{\noexpand\pgfpoint{\spath@ex}{\spath@ey}}%
  \pgfoolet{final point}{\spath@temp}%
  \endgroup
  \pgfooget{final point}{\spath@temp}%
  \fi
  \spath@temp
}
```

Get the bounding box of the path

```
\method bounding box() {%
  \pgfooget{min bb}{\spath@temp}%
  \ifx\spath@temp\pgfutil@empty
  \else
   \pgfooget{min bb}{\spath@temp}%
  \fi
  \ifx\spath@temp\pgfutil@empty
   \begingroup
    \spath@boundingbox@init
    \pgfoovalueof{path}%
    \edef\spath@temp{\noexpand\pgfpoint{\spath@mix}{\spath@miy}}%
    \pgfoolet{min bb}{\spath@temp}%
    \edef\spath@temp{\noexpand\pgfpoint{\spath@max}{\spath@may}}%
    \pgfoolet{max bb}{\spath@temp}%
   \endgroup
  \fi
}
```

Get the bounding box of the path: minimum corner

```
\method min bb() {%
  \pgfooget{min bb}{\spath@temp}%
  \ifx\spath@temp\pgfutil@empty
   \pgfoothis.bounding box()%
   \pgfooget{min bb}{\spath@temp}%
  \fi
  \spath@temp
}
```

Get the bounding box of the path: maximum corner

```
\method max bb() {%
  \pgfooget{max bb}{\spath@temp}%
  \ifx\spath@temp\pgfutil@empty
   \pgfoothis.bounding box()%
   \pgfooget{max bb}{\spath@temp}%
  \fi
  \spath@temp
}
```

Get the mid point of the path (according to the bounding box)

```
\method mid point() {%
  \pgfoothis.min bb()%
  \pgf@xa=.5\pgf@x
  \pgf@ya=.5\pgf@y
  \pgfoothis.max bb()%
  \pgf@x=.5\pgf@x
  \pgf@y=.5\pgf@y
  \advance\pgf@xa by \pgf@x
  \advance\pgf@ya by \pgf@y
  \edef\spath@temp{\noexpand\pgfpoint{\the\pgf@xa}{\the\pgf@ya}}%
  \spath@temp
}
```

Reverse the path. If #1 is given then it should be a macro and it will be an object with the reversed path, otherwise the current object is modified.

```
\method reverse path(#1) {%
 \let\spath@newpath=#1\relax
 \ifx\spath@newpath\relax
  \let\spath@tmppath=\pgfutil@empty
  \pgfoothis.prepare()%
  \begingroup
  \spath@reverse@init
  \pgfoovalueof{path}%
  \g@addbefore@macro\spath@tmppath\pgfsyssoftpath@movetotoken
  \endgroup
  \pgfoolet{path}{\spath@tmppath}%
  \pgfooget{initial point}{\spath@temp}%
  \pgfooget{final point}{\spath@tempa}%
  \pgfoolet{final point}{\spath@temp}%
  \pgfoolet{initial point}{\spath@tempa}%
  \pgfooget{first action}{\spath@temp}%
  \pgfooget{last action}{\spath@tempa}%
  \pgfoolet{last action}{\spath@temp}%
  \pgfoolet{first action}{\spath@tempa}%
 \else
  \pgfoothis.clone(\spath@newpath)%
  \begingroup
  \spath@newpath.reverse()%
  \endgroup
  \let#1=\spath@newpath
 \fi
}
```

Prepare the path, filling out all the attributes. As many of the attributes require “walking” the path to figure them out, by doing them all in one go we can avoid too much duplication of effort.

```
\method prepare() {%
  \pgfooget{prepared}{\spath@temp}%
  \ifx\spath@temp\pgfutil@empty
  \let\spath@tmppath=\pgfutil@empty
  \begingroup
  \setcounter{spath@reallength}{0}%
  \setcounter{spath@components}{0}%
  \setcounter{spath@length}{0}%
  \spath@prepare@init
  \let\spath@first=\pgfutil@empty
  \pgfoovalueof{path}%
  \edef\spath@temp{\the\value{spath@components}}%
  \pgfoolet{number of components}{\spath@temp}%
```

```
   \edef\spath@temp{\the\value{spath@length}}%
   \pgfoolet{length}{\spath@temp}%
   \edef\spath@temp{\the\value{spath@reallength}}%
   \pgfoolet{real length}{\spath@temp}%
   \edef\spath@temp{\noexpand\pgfpoint{\spath@sx}{\spath@sy}}%
   \pgfoolet{initial point}{\spath@temp}%
   \edef\spath@temp{\noexpand\pgfpoint{\spath@ex}{\spath@ey}}%
   \pgfoolet{final point}{\spath@temp}%
   \edef\spath@temp{\noexpand\pgfpoint{\spath@mix}{\spath@miy}}%
   \pgfoolet{min bb}{\spath@temp}%
   \edef\spath@temp{\noexpand\pgfpoint{\spath@max}{\spath@may}}%
   \pgfoolet{max bb}{\spath@temp}%
   \pgfoolet{first action}{\spath@first}%
   \pgfoolet{last action}{\spath@last}%
   \pgfoolet{prepared}{1}%
   \endgroup
   \fi
 }
```

Now we have some other path manipulation methods that don't involve walking a path.

For tapering a path, it is important that the path has at least 3 components so that the first and last can be tapered. This method ensures that that is so by splitting the path if its real length is less than 3. If the real length is 1, it replaces the component by three. If the real length is 2, it splits both of them in half.

```
 \method at least three() {%
   \pgfoothis.prepare()%
   \pgfooget{real length}{\spath@temp}%
   \pgfooget{path}{\spath@tmppath}%
   \ifnum\spath@temp=1\relax
    \pgfooget{first action}{\spath@temp}%
    \ifx\spath@temp\spath@lineto
     \expandafter\spath@split@single@lineto\spath@tmppath\relax
     \pgfoolet{path}{\spath@tmppath}%
     \pgfooset{length}{4}%
    \else
     \expandafter\spath@split@single@curveto\spath@tmppath\relax
     \pgfoolet{path}{\spath@tmppath}%
     \pgfooset{length}{10}%
    \fi
    \pgfooset{real length}{3}%
   \else
    \ifnum\spath@temp=2\relax
     \pgfooget{first action}{\spath@temp}%
     \def\spath@tempa{0}%
     \ifx\spath@temp\spath@lineto
      \expandafter\spath@split@first@lineto\spath@tmppath\relax
      \def\spath@tempa{3}%
     \else
      \expandafter\spath@split@first@curveto\spath@tmppath\relax
      \def\spath@tempa{7}%
     \fi
     \pgfooget{last action}{\spath@temp}%
     \ifx\spath@temp\spath@lineto
      \expandafter\spath@split@second@lineto\spath@split@path@end\relax
      \pgfmathsetmacro{\spath@tempa}{\spath@tempa + 2}
     \else
      \expandafter\spath@split@second@curveto\spath@split@path@end\relax
      \pgfmathsetmacro{\spath@tempa}{\spath@tempa + 6}
     \fi
     \let\spath@tmppath=\spath@split@path@start
     \ge@addto@macro\spath@tmppath\spath@split@path@end
     \pgfoolet{path}{\spath@tmppath}%
     \pgfoolet{length}{\spath@tempa}%
     \pgfooset{real length}{4}%
    \fi
   \fi
 }
```

This is the tapering method. It actually only tapers the first (real) component of the path and throws away the rest. It should therefore be called after a path has been split (and after the path has been made to have at least 3 components).

```
\method taper out() {%
  \pgfoothis.prepare()%
  \pgfooget{first action}{\spath@temp}%
  \pgfooget{path}{\spath@tmppath}%
  \pgfooget{taper line width}{\taper@line@width}%
  \ifx\taper@line@width\pgfutil@empty
   \pgfmathsetmacro{\taper@line@width}{.5*\pgflinewidth}%
  \fi
  \ifx\spath@temp\spath@lineto
   \expandafter\spath@taper@lineto@out\spath@tmppath\relax
  \pgfoolet{path}{\spath@tapered@path}%
  \else
   \ifx\spath@temp\spath@curveto
    \expandafter\spath@taper@curveto@out\spath@tmppath\relax
    \pgfoolet{path}{\spath@tapered@path}%
   \fi
  \fi
}
```

The next few methods split a path by various conditions. The actual splitting routine is the same for all, the next few methods work by setting up the conditions necessary to determine the place to split the path.

The arguments are the same for each: macros to store the first and second parts of the path, and the number at which to split (negative counts from the end).

The first splits the path by number of tokens.

```
\method split path by length(#1,#2,#3) {%
  \pgfoothis.prepare()%
  \def\spath@test{\the\value{spath@length}}%
  \pgfooget{length}{\spath@length}%
  \pgfoothis.split(#1,#2,#3)%
}
```

This splits the path by number of drawing tokens.

```
\method split path by real length(#1,#2,#3) {%
  \pgfoothis.prepare()%
  \def\spath@test{\the\value{spath@reallength}}%
  \pgfooget{real length}{\spath@length}%
  \pgfoothis.split(#1,#2,#3)%
}
```

This splits the path by the number of components (`movetos`).

```
\method split path by component(#1,#2,#3) {%
  \pgfoothis.prepare()%
  \def\spath@test{\the\value{spath@components}}%
  \pgfooget{number of components}{\spath@length}%
  \pgfoothis.split(#1,#2,#3)%
}
```

This is the actual splitting code. It's another path-walker, counting as it goes and splitting when we get to the right place. Negative numbers mean working from the end. After the split, we have to ensure that the second part has a suitable `moveto` since all paths start with a move.

```
\method split(#1,#2,#3) {%
  \pgfmathsetmacro{\spath@splitat}{#3 < 0 ? \spath@length + #3: #3}%
  \pgfooget{path}{\spath@tmppath}%
  \let\spath@tmppatha=\pgfutil@empty
  \setcounter{spath@length}{0}%
  \setcounter{spath@reallength}{0}%
  \setcounter{spath@components}{-1}%
  \expandafter\spath@gobble\spath@tmppath\relax
  \pgfoonew #1 =new spath(\spath@tmppatha)%
  \pgfoonew #2 =new spath(\spath@tmppath)%
  #1.let(last action,\spath@last)%
  #2.let(first action,\spath@first)%
  \edef\spath@temp{\noexpand\pgfpoint{\spath@ex}{\spath@ey}}%
  #1.let(final point,\spath@temp)%
  \edef\spath@temp{\noexpand\pgfpoint{\spath@sx}{\spath@sy}}%
  #2.let(initial point,\spath@temp)%
  \pgfooget{first action}{\spath@temp}%
  #1.let(first action,\spath@temp)%
  \pgfooget{last action}{\spath@temp}%
  #2.let(last action,\spath@temp)%
  \pgfooget{initial point}{\spath@temp}%
  #1.let(initial point,\spath@temp)%
  \pgfooget{final point}{\spath@temp}%
  #2.let(final point,\spath@temp)%
  #1.let(length,\spath@splitat)%
  \edef\spath@temp{\the\value{spath@reallength}}
  #1.let(real length,\spath@temp)%
  \pgfmathsetmacro{\spath@temp}{\pgfoovalueof{real length} - \spath@temp}
  #2.let(real length,\spath@temp)%
  \edef\spath@temp{\the\value{spath@components}}
  #1.let(number of components,\spath@temp)%
  \pgfmathsetmacro{\spath@temp}{\pgfoovalueof{number of components} - \spath@temp}
  #2.let(number of components,\spath@temp)%
  \pgfooget{taper line width}{\spath@temp}%
  #1.let(taper line width,\spath@temp)%
  #2.let(taper line width,\spath@temp)%
  \pgfmathsetmacro{\spath@splitat}{\spath@length - \spath@splitat}%
  #2.let(length,\spath@splitat)%
  #1.set(prepared,1)%
  #2.set(prepared,1)%
}
%    \end{macrocode}
% This reprocesses the path (it's another walker).
% The purpose here is that we can set new conditions (via TikZ or PGF options) which might affe
% An obvious example is to turn on the \Verb+rounded corners+ option.
%    \begin{macrocode}
\method reprocess path() {%
  \begingroup
  \spath@reprocess@init
  \pgfsyssoftpath@setcurrentpath{\pgfutil@empty}%
  \pgfoovalueof{path}%
  \pgfsyssoftpath@getcurrentpath{\spath@tmppath}%
  \pgfoolet{path}{\spath@tmppath}%
  \endgroup
  \let\spath@tmppath\pgfutil@empty
}
```

Now we come to a family of methods that actually use the path. The first sets the path as the current path. This means, for example, that the next TikZ command will start work on this path rather than an empty path.

```
\method set as current path() {%
  \pgfooget{path}{\spath@tmppath}%
  \pgfsyssoftpath@setcurrentpath{\spath@tmppath}%
}
```

This replaces the path in the object by the current soft path in the TikZ/PGF system. As this replaces the path, all other attributes should be thrown away.

```
\method get from current path() {%
  \pgfsyssoftpath@getcurrentpath{\spath@tmppath}%
  \pgfooset{path}{\spath@tmppath}%
  \pgfoothis.set(prepared,0)%
}
```

This uses the path with the `\pgfusepath` command, so is a low-level access to the path.

```
\method use path(#1) {%
  \pgfooget{path}{\spath@tmppath}%
  \pgfoothis.min bb()%
  \pgf@protocolsizes\pgf@x\pgf@y
  \pgfoothis.max bb()%
  \pgf@protocolsizes\pgf@x\pgf@y
  \pgfsyssoftpath@setcurrentpath{\spath@tmppath}%
  \pgfsyssoftpath@flushcurrentpath
  \pgfusepath{#1}%
}
```

This uses the path via a TikZ-like command, so can use TikZ style options (passed as the argument).

```
\method use path with tikz(#1) {%
  \path[#1] \pgfextra{%
  \begingroup
  \spath@reprocess@init
  \pgfsyssoftpath@setcurrentpath{\pgfutil@empty}%
  \pgfoovalueof{path}%
  \endgroup};
}
```

This produces a string representation of the path (mainly for preloading paths)

```
\method to string() {%
   \begingroup
   \spath@string@init
   \pgfoovalueof{path}%
   \endgroup
}
```

Now we get some methods which act on two paths and join them in some fashion.

The first simply concatenates the paths. As each path starts with a `moveto`, the two paths are still somewhat separate.

```
\method concatenate(#1,#2) {%
  \let\spath@newpath=#1\relax
  \ifx\spath@newpath\relax
   \let\spath@other=#2\relax
   \ifx\spath@other\relax
   \else
    \spath@other.get(path,\spath@tmppath)%
    \pgfooget{path}{\spath@tmppatha}%
    \ge@addto@macro\spath@tmppatha\spath@tmppath
    \pgfoolet{path}{\spath@tmppatha}%
    \spath@other.get(final point,\spath@temp)%
    \pgfoolet{final point}{\spath@temp}%
    \spath@other.get(last action,\spath@temp)%
    \pgfoolet{last action}{\spath@temp}%
    \pgfooget{min bb}{\spath@temp}%
    \spath@temp
    \pgf@xa=\pgf@x
    \pgf@ya=\pgf@y
    \spath@other.get(min bb,\spath@temp)%
    \spath@temp
    \pgfmathsetmacro{\spath@mix}{min(\pgf@x,\pgf@xa)}%
    \pgfmathsetmacro{\spath@miy}{min(\pgf@y,\pgf@ya)}%
    \edef\spath@temp{\noexpand\pgfpoint{\spath@mix}{\spath@miy}}%
    \pgfoolet{min bb}{\spath@temp}%
    \pgfooget{max bb}{\spath@temp}%
    \spath@temp
    \pgf@xa=\pgf@x
    \pgf@ya=\pgf@y
    \spath@other.get(max bb,\spath@temp)%
    \spath@temp
    \pgfmathsetmacro{\spath@max}{max(\pgf@x,\pgf@xa)}%
    \pgfmathsetmacro{\spath@may}{max(\pgf@y,\pgf@ya)}%
    \edef\spath@temp{\noexpand\pgfpoint{\spath@mix}{\spath@miy}}%
    \pgfoolet{max bb}{\spath@temp}%
\tikz@clear@foreach
    \foreach \attribute in {
      length,
      real length,
      number of components%
    } {
      \spath@other.get(\attribute,\spath@temp)%
      \pgfooget{\attribute}{\spath@tempa}%
      \ifx\spath@temp\pgfutil@empty
      \else
      \ifx\spath@tempa\pgfutil@empty
      \else
       \pgfmathsetmacro{\spath@tempa}{\spath@temp + \spath@tempa}%
       \pgfoolet{\attribute}{\spath@tempa}%
      \fi
      \fi
    }%
  \fi
 \else
 \pgfoothis.clone(\spath@newpath)%
 \begingroup
 \spath@newpath.concatenate(,#2)%
 \endgroup
 \let#1=\spath@newpath
 \fi
}
```

This concatenates the paths but with a lineto instead of a moveto

```
\method concatenate with lineto(#1,#2) {%
  \let\spath@newpath=#1\relax
  \ifx\spath@newpath\relax
   \let\spath@other=#2\relax
   \ifx\spath@other\relax
   \else
    \spath@other.get(path,\spath@tmppath)%
    \expandafter\spath@movetoline\spath@tmppath\relax
    \pgfooget{path}{\spath@tmppatha}%
    \ge@addto@macro\spath@tmppatha\spath@tmppath
    \pgfoolet{path}{\spath@tmppatha}%
    \spath@other.get(final point,\spath@temp)%
    \pgfoolet{final point}{\spath@temp}%
    \spath@other.get(last action,\spath@temp)%
    \pgfoolet{last action}{\spath@temp}%
    \pgfooget{min bb}{\spath@temp}%
    \spath@temp
    \pgf@xa=\pgf@x
    \pgf@ya=\pgf@y
    \spath@other.get(min bb,\spath@temp)%
    \spath@temp
    \pgfmathsetmacro{\spath@mix}{min(\pgf@x,\pgf@xa)}%
    \pgfmathsetmacro{\spath@miy}{min(\pgf@y,\pgf@ya)}%
    \edef\spath@temp{\noexpand\pgfpoint{\spath@mix}{\spath@miy}}%
    \pgfoolet{min bb}{\spath@temp}%
    \pgfooget{max bb}{\spath@temp}%
    \spath@temp
    \pgf@xa=\pgf@x
    \pgf@ya=\pgf@y
    \spath@other.get(max bb,\spath@temp)%
    \spath@temp
    \pgfmathsetmacro{\spath@max}{max(\pgf@x,\pgf@xa)}%
    \pgfmathsetmacro{\spath@may}{max(\pgf@y,\pgf@ya)}%
    \edef\spath@temp{\noexpand\pgfpoint{\spath@mix}{\spath@miy}}%
    \pgfoolet{max bb}{\spath@temp}%
\tikz@clear@foreach
    \foreach \attribute in {
      length,
      real length,
      number of components%
    } {
      \spath@other.get(\attribute,\spath@temp)%
      \pgfooget{\attribute}{\spath@tempa}%
      \ifx\spath@temp\pgfutil@empty
      \else
      \ifx\spath@tempa\pgfutil@empty
      \else
       \pgfmathsetmacro{\spath@tempa}{\spath@temp + \spath@tempa}%
       \pgfoolet{\attribute}{\spath@tempa}%
      \fi
      \fi
    }%
  \fi
 \else
  \pgfoothis.clone(\spath@newpath)%
  \begingroup
  \spath@newpath.concatenate with lineto(,#2)%
  \endgroup
  \let#1=\spath@newpath
 \fi
}
```

This concatenates the paths but removes the moveto

```
\method concatenate without moveto(#1,#2) {%
  \let\spath@newpath=#1\relax
  \ifx\spath@newpath\relax
   \let\spath@other=#2\relax
   \ifx\spath@other\relax
   \else
    \spath@other.get(path,\spath@tmppath)%
    \expandafter\spath@trimfirst\spath@tmppath\relax
    \pgfooget{path}{\spath@tmppatha}%
    \ge@addto@macro\spath@tmppatha\spath@tmppath
    \pgfoolet{path}{\spath@tmppatha}%
    \spath@other.get(final point,\spath@temp)%
    \pgfoolet{final point}{\spath@temp}%
    \spath@other.get(last action,\spath@temp)%
    \pgfoolet{last action}{\spath@temp}%
    \pgfooget{min bb}{\spath@temp}%
    \spath@temp
    \pgf@xa=\pgf@x
    \pgf@ya=\pgf@y
    \spath@other.get(min bb,\spath@temp)%
    \spath@temp
    \pgfmathsetmacro{\spath@mix}{min(\pgf@x,\pgf@xa)}%
    \pgfmathsetmacro{\spath@miy}{min(\pgf@y,\pgf@ya)}%
    \edef\spath@temp{\noexpand\pgfpoint{\spath@mix}{\spath@miy}}%
    \pgfoolet{min bb}{\spath@temp}%
    \pgfooget{max bb}{\spath@temp}%
    \spath@temp
    \pgf@xa=\pgf@x
    \pgf@ya=\pgf@y
    \spath@other.get(max bb,\spath@temp)%
    \spath@temp
    \pgfmathsetmacro{\spath@max}{max(\pgf@x,\pgf@xa)}%
    \pgfmathsetmacro{\spath@may}{max(\pgf@y,\pgf@ya)}%
    \edef\spath@temp{\noexpand\pgfpoint{\spath@mix}{\spath@miy}}%
    \pgfoolet{max bb}{\spath@temp}%
\tikz@clear@foreach
    \foreach \attribute in {
      length,
      real length,
      number of components%
    } {
      \spath@other.get(\attribute,\spath@temp)%
      \pgfooget{\attribute}{\spath@tempa}%
      \ifx\spath@temp\pgfutil@empty
      \else
```

```
        \ifx\spath@tempa\pgfutil@empty
        \else
         \pgfmathsetmacro{\spath@tempa}{\spath@temp + \spath@tempa}%
         \pgfoolet{\attribute}{\spath@tempa}%
        \fi
        \fi
      }%
      \pgfooget{length}{\spath@tempa}%
      \pgfmathsetmacro{\spath@tempa}{\spath@tempa - 1}%
      \pgfoolet{length}{\spath@tempa}%
     \fi
   \else
    \pgfoothis.clone(\spath@newpath)%
    \begingroup
    \spath@newpath.concatenate without moveto(,#2)%
    \endgroup
    \let#1=\spath@newpath
   \fi
  }
```

Welding is like concatenation except that the paths are brought together and the intervening moveto is removed

```
  \method weld(#1,#2) {%
    \let\spath@newpath=#1\relax
    \ifx\spath@newpath\relax
     \let\spath@other=#2\relax
     \ifx\spath@other\relax
     \else
      \pgfoothis.final point()%
      \pgf@xa=\pgf@x
      \pgf@ya=\pgf@y
      \spath@other.initial point()%
      \advance\pgf@xa by -\pgf@x
      \advance\pgf@ya by -\pgf@y
      \spath@other.translate path(\spath@tempo,\the\pgf@xa,\the\pgf@ya)%
%
      \spath@tempo.get(path,\spath@tmppath)%
      \expandafter\spath@trimfirst\spath@tmppath\relax
      \pgfooget{path}{\spath@tmppatha}%
      \ge@addto@macro\spath@tmppatha\spath@tmppath
      \pgfoolet{path}{\spath@tmppatha}%
%
      \spath@tempo.get(final point,\spath@temp)%
      \pgfoolet{final point}{\spath@temp}%
      \spath@tempo.get(last action,\spath@temp)%
      \pgfoolet{last action}{\spath@temp}%
      \pgfooget{min bb}{\spath@temp}%
      \spath@temp
      \pgf@xa=\pgf@x
      \pgf@ya=\pgf@y
      \spath@other.get(min bb,\spath@temp)%
      \spath@temp
      \pgfmathsetmacro{\spath@mix}{min(\pgf@x,\pgf@xa)}%
      \pgfmathsetmacro{\spath@miy}{min(\pgf@y,\pgf@ya)}%
      \edef\spath@temp{\noexpand\pgfpoint{\spath@mix}{\spath@miy}}%
      \pgfoolet{min bb}{\spath@temp}%
      \spath@temp
      \pgf@xa=\pgf@x
      \pgf@ya=\pgf@y
      \spath@other.get(max bb,\spath@temp)%
      \spath@temp
      \pgfmathsetmacro{\spath@max}{max(\pgf@x,\pgf@xa)}%
      \pgfmathsetmacro{\spath@may}{max(\pgf@y,\pgf@ya)}%
      \edef\spath@temp{\noexpand\pgfpoint{\spath@mix}{\spath@miy}}%
      \pgfoolet{max bb}{\spath@temp}%
\tikz@clear@foreach
      \foreach \attribute in {
        length,
        real length,
        number of components%
      } {
        \spath@tempo.get(\attribute,\spath@temp)%
        \pgfooget{\attribute}{\spath@tempa}%
        \ifx\spath@temp\pgfutil@empty
        \else
         \ifx\spath@tempa\pgfutil@empty
         \else
          \pgfmathsetmacro{\spath@tempa}{\spath@temp + \spath@tempa}%
          \pgfoolet{\attribute}{\spath@tempa}%
         \fi
        \fi
      }%
     \fi
   \else
    \pgfoothis.clone(\spath@newpath)%
    \begingroup
     \spath@newpath.weld(,#2)%
    \endgroup
    \let#1=\spath@newpath
   \fi
}
```

This library isn't great with closed paths, but we can certainly close up an existing path.

```
\method close() {%
  \pgfooget{path}{\spath@tmppath}%
  \g@addto@macro\spath@tmppath{\pgfsyssoftpath@closepathtoken{0pt}{0pt}}%
  \pgfoolet{path}{\spath@tmppath}%
  \pgfooget{length}{\spath@temp}%
  \pgfmathtruncatemacro{\spath@temp}{\spath@temp + 1}%
```

```
    \pgfoolet{length}{\spath@temp}%
    \pgfooget{real length}{\spath@temp}%
    \pgfmathtruncatemacro{\spath@temp}{\spath@temp + 1}%
    \pgfoolet{real length}{\spath@temp}%
    \pgfooget{number of components}{\spath@temp}%
    \pgfmathtruncatemacro{\spath@temp}{\spath@temp + 1}%
    \pgfoolet{number of components}{\spath@temp}%
    \pgfooget{initial point}{\spath@temp}%
    \pgfoolet{final point}{\spath@temp}%
    \pgfooset{last action}{closepath}%
    \pgfooset{prepared}{0}%
  }
```

This method steps through the path and splits curvetos that might possibly self-intersect.

```
  \method split self intersecting pieces() {%
  \begingroup
    \let\spath@tmppath=\pgfutil@empty
    \spath@selfintersect@init
    \pgfoovalueof{path}%
    \pgfoolet{path}{\spath@tmppath}%
  \endgroup
}
```

Explode the path into an array of drawing components

```
  \method explode(#1) {%
  \pgfooget{path}{\spath@tmp}%
  \spathexplode#1\spath@tmp
}
```

```
  \method at intersections(#1) {%
  \pgfooget{path}{\spath@tmppath}%
  #1.get(path,\spath@tmppatha)%
  \ifx\spath@tmppatha\pgfutil@empty
  \else
  \pgfoothis.get handle(\spath@firstpath)%
  \let\spath@secondpath=#1%
  \pgfintersectionofpaths{\pgfsetpath\spath@tmppatha}{\pgfsetpath\spath@tmppath}%
  \ifnum\pgfintersectionsolutions>0\relax
  \pgfsyssoftpath@setcurrentpath\pgfutil@empty
  \tikz@clear@foreach
  \foreach \spath@k in {1,...,\pgfintersectionsolutions} {
    \pgfpointintersectionsolution{\spath@k}%
    \edef\spath@ix{\the\pgf@x}%
    \edef\spath@iy{\the\pgf@y}%
    \pgfoothis.initial point()%
    \pgfmathtruncatemacro{\spath@first@nearstart}{(abs(\pgf@x - \spath@ix) + abs(\pgf@y - \spat
    \pgfoothis.final point()
    \pgfmathtruncatemacro{\spath@first@nearend}{(abs(\pgf@x - \spath@ix) + abs(\pgf@y - \spath@
    \pgfmathtruncatemacro{\spath@first@nearends}{\spath@first@nearstart || \spath@first@nearend
    \spath@secondpath.initial point()%
    \pgfmathtruncatemacro{\spath@second@nearstart}{(abs(\pgf@x - \spath@ix) + abs(\pgf@y - \spa
    \spath@secondpath.final point()
    \pgfmathtruncatemacro{\spath@second@nearend}{(abs(\pgf@x - \spath@ix) + abs(\pgf@y - \spath
    \pgfmathtruncatemacro{\spath@second@nearends}{\spath@second@nearstart || \spath@second@near
    \pgfmathtruncatemacro{\spath@nearends}{\spath@first@nearends || \spath@second@nearends}%
    \spath@execute@at@intersections
  }
  \fi
  \fi
}
```

Here endeth the `spath` object.

```
}
```

The `spath component` is designed to be part of an array of `spath` objects. Each `spath component` consists of an `spath` object and a link to a previous and a next `spath component` (either of which could be empty).

```
\pgfooclass{spath component}{
```

Our attributes are our `spath`, and the next and previous components.

```
  \attribute path;
  \attribute next component;
  \attribute previous component;
```

This is the creator. If called with an argument, that is assumed to be the previous component in the array.

```
  \method spath component(#1) {%
    \def\spath@temp{#1}%
    \ifx\spath@temp\pgfutil@empty
    \else
    \pgfoolet{previous component}{#1}%
    \fi
  }
```

Generic attribute handling, see `spath` class for details

```
  \method value(#1) {%
    \pgfoovalueof{#1}%
  }
  \method set(#1,#2) {%
    \pgfooset{#1}{#2}%
  }
  \method let(#1,#2) {%
    \pgfoolet{#1}{#2}%
  }
  \method get(#1,#2) {%
    \pgfooget{#1}{#2}%
  }
  \method show(#1) {%
    \pgfooget{#1}{\@temp}%
    \show\@temp
  }
```

This says what we are

```
\method what am I() {%
   \message{I am an spath component object}%
}
```

This sets the `path` attribute (as an instance of the `spath` class) from a saved path.

```
\method set path(#1) {%
  \pgfoonew \spath@tempa =new spath(#1)%
  \pgfoolet{path}{\spath@tempa}%
}
```

This recurses along the array applying an action to every object. Specifically, it applies the action to the `path` attribute and then calls the next component with the same arguments.

```
\method apply to paths(#1,#2) {%
  \pgfooget{path}{\spath@tmppath}%
  \ifx\spath@tmppath\pgfutil@empty
  \else
  \spath@tmppath.#1(#2)%
  \fi
  \pgfooget{next component}{\spath@temp}%
  \let\spath@next=\pgfutil@empty
  \ifx\spath@temp\pgfutil@empty
  \else
   \def\spath@next{\spath@temp.apply to paths(#1,{#2})}%
  \fi
  \spath@next
}
\method apply to previous paths(#1,#2) {%
  \pgfooget{path}{\spath@tmppath}%
  \ifx\spath@temppath\pgfutil@empty
  \else
  \spath@tmppath.#1(#2)%
  \fi
  \pgfooget{previous component}{\spath@temp}%
  \let\spath@next=\pgfutil@empty
  \ifx\spath@temp\pgfutil@empty
  \else
   \def\spath@next{\spath@temp.apply to previous paths(#1,{#2})}%
  \fi
  \spath@next%
}
\method apply for paths(#1) {%
  #1%
  \pgfooget{next component}{\spath@temp}%
  \let\spath@next=\pgfutil@empty
  \ifx\spath@temp\pgfutil@empty
  \else
   \def\spath@next{\spath@temp.apply for paths({#1})}%
  \fi
  \spath@next%
}
\method apply for previous paths(#1) {%
  #1%
  \pgfooget{previous component}{\spath@temp}%
  \let\spath@next=\pgfutil@empty
  \ifx\spath@temp\pgfutil@empty
  \else
   \def\spath@next{\spath@temp.apply for previous paths({#1})}%
  \fi
  \spath@next%
}
```

We look for intersections between paths. If #1 is given, we use that as one of the `spath` objects to intersect with and look for intersections with that starting with us and proceeding to all following components. If #1 is not given, we take our `spath` object as the one to use and look for intersections for all subsequent components. We carry the previous and next components around with us in case the intersection happens to be too close to an endpoint and we want to use the other components as well.

```
\method at intersections(#1,#2,#3) {%
\let\spath@next=\pgfutil@empty
\let\spath@tmp#1\relax%
\let\spath@secondpath@prev#2\relax
\let\spath@secondpath@next#3\relax
\ifx\spath@tmp\relax
 \pgfooget{path}{\spath@tmp}%
 \pgfooget{previous component}{\spath@temp}%
 \ifx\spath@temp\pgfutil@empty
 \else
 \spath@temp.get(path,\spath@secondpath@prev)%
 \fi
 \pgfooget{next component}{\spath@temp}%
 \ifx\spath@temp\pgfutil@empty
 \else
  \spath@temp.get(path,\spath@secondpath@next)%
  \def\spath@next{%
   \begingroup
   \spath@temp.at intersections(\spath@tmp,\spath@secondpath@prev,\spath@secondpath@next)%
   \endgroup
   \spath@temp.at intersections(,,)%
  }
 \fi
\else
 \begingroup
  \pgfooget{previous component}{\spath@temp}%
  \ifx\spath@temp\pgfutil@empty
  \else
   \spath@temp.get(path,\spath@firstpath@prev)%
  \fi
```

```
    \pgfooget{next component}{\spath@temp}%
    \ifx\spath@temp\pgfutil@empty
    \else
     \spath@temp.get(path,\spath@firstpath@next)%
    \fi
    \pgfooget{path}{\spath@temp}%
    \ifx\spath@temp\pgfutil@empty
    \else
     \spath@temp.at intersections(\spath@tmp)%
    \fi
   \endgroup
   \pgfooget{next component}{\spath@temp}%
   \ifx\spath@temp\pgfutil@empty
   \else
    \def\spath@next{%
     \spath@temp.at intersections(\spath@tmp,\spath@secondpath@prev,\spath@secondpath@next)%
    }
   \fi
  \fi
  \spath@next
  }
```

That's all, folks. At least for the `spath component`

```
}
```

The reason that the above array structure was created was to be able to work on a soft path "component by component". The following macros take a soft path and split it at `movetos`.

```
\def\spathsplit#1#2{%
  \ifx#1\relax
  \pgfoonew #1 =new spath component({})%
  \else
  \message{\string#1\space already defined}
  \fi
  \let\spath@this@component=#1\relax
  \expandafter\spath@split@#2\relax
}
\def\spath@split@\pgfsyssoftpath@movetotoken#1\relax{%
  \spath@split@@#1\pgfsyssoftpath@movetotoken\relax
}
\def\spath@split@@#1\pgfsyssoftpath@movetotoken#2\relax{%
  \def\spath@tmppath{\pgfsyssoftpath@movetotoken#1}%
  \spath@this@component.set path(\spath@tmppath)%
  \def\spath@tmppath{#2}%
  \ifx\spath@tmppath\pgfutil@empty
  \else
   \pgfoonew \spath@next@component =new spath component(\spath@this@component)%
   \spath@this@component.let(next component,\spath@next@component)%
   \let\spath@this@component=\spath@next@component
   \spath@split@@#2\relax
  \fi
}
```

The following macros take a soft path and split it into drawing components.

```
\def\spathexplode#1#2{%
  \ifdefined#1
  \else
  \pgfoonew #1 =new spath component({})%
  \fi
  \let\spath@this@component=#1\relax
  \expandafter\spath@explode@#2\relax
}
\def\spath@explode@#1{%
  \let\spath@explode@next=\pgfutil@empty
  \ifx#1\pgfsyssoftpath@movetotoken
  \let\spath@explode@next=\spath@explode@savexy
  \else
  \ifx#1\pgfsyssoftpath@linetotoken
  \let\spath@explode@next=\spath@explode@lineto
  \else
  \ifx#1\pgfsyssoftpath@curvetosupportatoken
  \let\spath@explode@next=\spath@explode@curveto
  \else
  \ifx#1\pgfsyssoftpath@closepathtoken
  \let\spath@explode@next=\spath@explode@lineto
  \else
  \ifx#1\pgfsyssoftpath@recttoken
  \let\spath@explode@next=\spath@explode@rect
  \fi
  \fi
  \fi
  \fi
  \fi
  \spath@explode@next
}
\def\spath@explode@savexy#1#2{%
  \def\spath@ex{#1}%
  \def\spath@ey{#2}%
  \spath@explode@%
}
\def\spath@explode@lineto#1#2{%
  \edef\spath@tmppath{\noexpand\pgfsyssoftpath@movetotoken{\spath@ex}{\spath@ey}\noexpand\pgfsy
  \spath@this@component.set path(\spath@tmppath)%
  \pgfoonew \spath@next@component =new spath component(\spath@this@component)%
  \spath@this@component.let(next component,\spath@next@component)%
  \let\spath@this@component=\spath@next@component
  \def\spath@ex{#1}%
  \def\spath@ey{#2}%
  \spath@explode@%
```

```
}
\def\spath@explode@curveto#1#2\pgfsyssoftpath@curvetosupportbtoken#3#4\pgfsyssoftpath@curvetotol
  \edef\spath@tmppath{\noexpand\pgfsyssoftpath@movetotoken{\spath@ex}{\spath@ey}\noexpand\pgfsy
  \spath@this@component.set path(\spath@tmppath)%
  \pgfoonew \spath@next@component =new spath component(\spath@this@component)%
  \spath@this@component.let(next component,\spath@next@component)%
  \let\spath@this@component=\spath@next@component
  \def\spath@ex{#5}%
  \def\spath@ey{#6}%
  \spath@explode@%
}
% These are all our helper macros.
% The first batch are for defining the ``walker'' methods of the \Verb+spath+ class.
% For each of the methods we have to define some initialiser code which redefines the soft path
% As the ``something appropriate'' is often independent of the actual token macro, we set up a 
% These set the default actions which can be overridden afterwards.
%
% We start with translation.
% Each soft path token translates its arguments and then appends itself, with the translated ar
%    \begin{macrocode}
\def\spath@define@translate#1\relax{%
  \expandafter\gdef\csname spath@tr@#1\endcsname##1##2{%
    \pgf@xa=##1\relax
    \pgf@ya=##2\relax
    \advance\pgf@xa by \spath@trx
    \advance\pgf@ya by \spath@try
    \edef\spath@tmp{\expandafter\noexpand\csname pgfsyssoftpath@#1token\endcsname{\the\pgf@xa}{
    \ge@addto@macro\spath@tmppath\spath@tmp
  }
}
```

Apply a transformation to the path

```
\def\spath@define@transform#1\relax{%
  \expandafter\gdef\csname spath@trans@#1\endcsname##1##2{%
    \pgf@xa=##1\relax
    \pgf@ya=##2\relax
    \pgf@process{\pgfpointtransformed{\pgfqpoint{\pgf@xa}{\pgf@ya}}}%
    \edef\spath@tmp{\expandafter\noexpand\csname pgfsyssoftpath@#1token\endcsname{\the\pgf@x}{\
    \ge@addto@macro\spath@tmppath\spath@tmp
  }
}
```

Scale all coordinates

```
\def\spath@define@scale#1\relax{%
  \expandafter\gdef\csname spath@sc@#1\endcsname##1##2{%
    \pgf@xa=##1\relax
    \pgf@ya=##2\relax
    \pgf@xa=\spath@scx\pgf@xa\relax
    \pgf@ya=\spath@scy\pgf@ya\relax
    \edef\spath@tmp{\expandafter\noexpand\csname pgfsyssoftpath@#1token\endcsname{\the\pgf@xa}{
```

```
    \ge@addto@macro\spath@tmppath\spath@tmp
  }
}
```

Define the rounding macros, most do nothing

```
\def\spath@define@round#1\relax{%
  \expandafter\gdef\csname spath@rnd@#1\endcsname##1##2{%
    \edef\spath@tmp{\expandafter\noexpand\csname pgfsyssoftpath@#1token\endcsname{##1}{##2}}
    \ge@addto@macro\spath@tmppath\spath@tmp
  }
}
```

Each soft path token shrinks its arguments towards the given point and then appends itself to the temporary path.

```
\def\spath@define@shrink#1\relax{%
  \expandafter\gdef\csname spath@sh@#1\endcsname##1##2{%
    \pgfmathsetlength{\pgf@xa}{##1 < \spath@trx ? ##1 + \spath@shrinkby : (##1 > \spath@trx ? #
    \pgfmathsetlength{\pgf@ya}{##2 < \spath@try ? ##2 + \spath@shrinkby : (##2 > \spath@try ? #
    \edef\spath@tmp{\expandafter\noexpand\csname pgfsyssoftpath@#1token\endcsname{\the\pgf@xa}{
    \ge@addto@macro\spath@tmppath\spath@tmp
  }
}
```

Each soft path token rotates its arguments about the given origin and then appends itself to the temporary path. Then it increments the rotation counter.

```
\def\spath@define@spirograph#1\relax{%
  \expandafter\gdef\csname spath@gch@#1\endcsname##1##2{%
    \pgf@xa=##1\relax
    \pgf@ya=##2\relax
    \advance\pgf@xa by -\spath@trx
    \advance\pgf@ya by -\spath@try
    \pgfmathsetmacro\spath@gcos{cos(\spath@n * \spath@gang)}
    \pgfmathsetmacro\spath@gsin{sin(\spath@n * \spath@gang)}
    \pgfmathsetmacro\spath@n{\spath@n + 1}%
    \global\let\spath@n=\spath@n
    \pgfmathsetlength\pgf@xb{\spath@gcos * \pgf@xa - \spath@gsin * \pgf@ya}
    \pgfmathsetlength\pgf@yb{\spath@gsin * \pgf@xa + \spath@gcos * \pgf@ya}
    \advance\pgf@xb by \spath@trx
    \advance\pgf@yb by \spath@try
    \edef\spath@tmp{\expandafter\noexpand\csname pgfsyssoftpath@#1token\endcsname{\the\pgf@xb}{
    \ge@addto@macro\spath@tmppath\spath@tmp
  }
}
```

Each soft path token rotates its arguments and then appends itself to the temporary path.

```
\def\spath@define@rotate#1\relax{%
  \expandafter\gdef\csname spath@rot@#1\endcsname##1##2{%
    \pgf@xa=##1\relax
    \pgf@ya=##2\relax
    \pgfmathsetlength\pgf@xb{\spath@gcos * \pgf@xa - \spath@gsin * \pgf@ya}
    \pgfmathsetlength\pgf@yb{\spath@gsin * \pgf@xa + \spath@gcos * \pgf@ya}
    \edef\spath@tmp{\expandafter\noexpand\csname pgfsyssoftpath@#1token\endcsname{\the\pgf@xb}{
    \ge@addto@macro\spath@tmppath\spath@tmp
  }
}
```

This is for the length, we simply increment the length counter.

```
\def\spath@define@length#1\relax{%
  \expandafter\gdef\csname spath@len@#1\endcsname##1##2{%
    \stepcounter{spath@length}
  }
}
```

This is for the number of components, by default we do nothing. The special code for the `moveto` will be set up later.

```
\def\spath@define@components#1\relax{%
  \expandafter\gdef\csname spath@comp@#1\endcsname##1##2{%
  }
}
```

This is for the real length, by default we do nothing. The special code for the `lineto` and `curveto` will be set up later.

```
\def\spath@define@reallength#1\relax{%
  \expandafter\gdef\csname spath@rlen@#1\endcsname##1##2{%
  }
}
```

This is for reversing the path. By default, we place our coordinates *before* the current token. The special code for the `curveto` will be set up later.

```
\def\spath@define@reverse#1\relax{%
  \expandafter\gdef\csname spath@rev@#1\endcsname##1##2{%
    \edef\spath@tmp{{##1}{##2}\expandafter\noexpand\csname pgfsyssoftpath@#1token\endcsname}
    \ge@addbefore@macro\spath@tmppath\spath@tmp
  }
}
```

This is for recording the start of a path, so we save our coordinates and then reinitialise so that the rest of the path does nothing.

```
\def\spath@define@start#1\relax{%
  \expandafter\gdef\csname spath@start@#1\endcsname##1##2{%
    \edef\spath@sx{##1}
    \edef\spath@sy{##2}
    \spath@start@reinit
  }
}
```

This is for recording the end of a path, so we save our coordinates. Each component overwrites what the previous one saved so we're left with the final coordinates.

```
\def\spath@define@end#1\relax{%
  \expandafter\gdef\csname spath@end@#1\endcsname##1##2{%
    \edef\spath@ex{##1}
    \edef\spath@ey{##2}
  }
}
```

This is for splitting a soft path into components. The default action is to add ourself to the current temporary path.

```
\def\spath@define@array#1\relax{%
  \expandafter\gdef\csname spath@array@#1\endcsname##1##2{%
    \edef\spath@tmp{\expandafter\noexpand\csname       pgfsyssoftpath@#1token\endcsname{##1}{##
    \ge@addto@macro\spath@tmppath\spath@tmp
  }
}
```

This is for reprocessing the path.

```
\def\spath@define@reprocess#1\relax{%
  \expandafter\gdef\csname spath@rep@#1\endcsname##1##2{%
    \csname pgfpath#1\endcsname{\pgfpoint{##1}{##2}}%
  }%
}
```

This is for splitting self-intersecting pieces. Most pieces just record the endpoints. We only actually split `curvetos`.

```
\def\spath@define@selfintersect#1{%
  \expandafter\gdef\csname spath@selfintersect@#1\endcsname##1##2{%
    \edef\spath@ex{##1}%
    \edef\spath@ey{##2}%
    \edef\spath@tmp{\expandafter\noexpand\csname pgfsyssoftpath@#1token\endcsname{##1}{##2}}%
    \ge@addto@macro\spath@tmppath\spath@tmp
  }
}
```

This is for computing the bounding box

```
\def\spath@define@boundingbox#1{%
  \expandafter\gdef\csname spath@boundingbox@#1\endcsname##1##2{%
     \pgfmathsetmacro\spath@mix{min(\spath@mix,##1)}%
     \pgfmathsetmacro\spath@miy{min(\spath@miy,##2)}%
     \pgfmathsetmacro\spath@max{max(\spath@max,##1)}%
     \pgfmathsetmacro\spath@may{max(\spath@may,##2)}%
  }%
}%
```

This is for converting the path into a string

```
\def\spath@define@string#1\relax{%
  \expandafter\gdef\csname spath@str@#1\endcsname##1##2{%
  \pgfmathsetmacro\spath@sx{##1/1cm}%
  \pgfmathsetmacro\spath@sy{##2/1cm}%
    \expandafter\string\csname pgfpath#1\endcsname\{\string\pgfpointxy\{\spath@sx\}\{\spath@sy\
  }%
}
```

The next part of the code sets up the initialiser code for each of the routines. We start with empty initialiser code.

```
\let\spath@spirograph@init=\pgfutil@empty
```

```
\let\spath@rotate@init=\pgfutil@empty
\let\spath@round@init=\pgfutil@empty
\let\spath@translate@init=\pgfutil@empty
\let\spath@transform@init=\pgfutil@empty
\let\spath@scale@init=\pgfutil@empty
\let\spath@shrink@init=\pgfutil@empty
\let\spath@length@init=\pgfutil@empty
\let\spath@components@init=\pgfutil@empty
\let\spath@reallength@init=\pgfutil@empty
\let\spath@reverse@init=\pgfutil@empty
\let\spath@start@init=\pgfutil@empty
\let\spath@start@reinit=\pgfutil@empty
\let\spath@end@init=\pgfutil@empty
\let\spath@array@init=\pgfutil@empty
\let\spath@prepare@init=\pgfutil@empty
\let\spath@reprocess@init=\pgfutil@empty
\let\spath@string@init=\pgfutil@empty
\let\spath@selfintersect@init=\pgfutil@empty
\def\spath@boundingbox@init{%
  \def\spath@mix{16000pt}%
  \def\spath@miy{16000pt}%
  \def\spath@max{-16000pt}%
  \def\spath@may{-16000pt}%
}
```

We now loop over the possible soft path tokens and define the different types of action each will expand to in the different circumstances. These are the default actions, as defined by the above macros. Specific actions can be redefined afterwards.

```
  \tikz@clear@foreach
\foreach \spath@cpt in {
  moveto,
  lineto,
  curvetosupporta,
  curvetosupportb,
  curveto,
  rectcorner,
  rectsize,
  closepath,
  specialround%
} {
```

Save token names for comparison

```
\expandafter\xdef\csname spath@\spath@cpt\endcsname{\expandafter\string\csname pgfsyssoftpath@\
```

Define the translation macros.

```
\expandafter\spath@define@translate\spath@cpt\relax
```

Add the “redefinition” code to the initialiser

```
\edef\spath@tmp{\noexpand\let\expandafter\noexpand\csname pgfsyssoftpath@\spath@cpt token\endcs

```

```
\ge@addto@macro\spath@translate@init\spath@tmp
```

Define the transformation macros.

```
\expandafter\spath@define@transform\spath@cpt\relax
```

Add the “redefinition” code to the initialiser

```
\edef\spath@tmp{\noexpand\let\expandafter\noexpand\csname pgfsyssoftpath@\spath@cpt token\endcs

\ge@addto@macro\spath@transform@init\spath@tmp
```

Define the scaling macros.

```
\expandafter\spath@define@scale\spath@cpt\relax
```

Add the “redefinition” code to the initialiser

```
\edef\spath@tmp{\noexpand\let\expandafter\noexpand\csname pgfsyssoftpath@\spath@cpt token\endcs

\ge@addto@macro\spath@scale@init\spath@tmp
```

Define the rounding macros.

```
\expandafter\spath@define@round\spath@cpt\relax
```

Add the “redefinition” code to the initialiser

```
\edef\spath@tmp{\noexpand\let\expandafter\noexpand\csname pgfsyssoftpath@\spath@cpt token\endcs

\ge@addto@macro\spath@round@init\spath@tmp
```

Define the shrinking macros.

```
\expandafter\spath@define@shrink\spath@cpt\relax
```

Add the “redefinition” code to the initialiser

```
\edef\spath@tmp{\noexpand\let\expandafter\noexpand\csname pgfsyssoftpath@\spath@cpt token\endcs

\ge@addto@macro\spath@shrink@init\spath@tmp
```

Define the spirograph macros.

```
\expandafter\spath@define@spirograph\spath@cpt\relax
```

Add the “redefinition” code to the initialiser

```
\edef\spath@tmp{\noexpand\let\expandafter\noexpand\csname pgfsyssoftpath@\spath@cpt token\endcs

\ge@addto@macro\spath@spirograph@init\spath@tmp
```

Define the rotation macros.

```
\expandafter\spath@define@rotate\spath@cpt\relax
```

Add the “redefinition” code to the initialiser

```
\edef\spath@tmp{\noexpand\let\expandafter\noexpand\csname pgfsyssoftpath@\spath@cpt token\endcs

\ge@addto@macro\spath@rotate@init\spath@tmp
```

Now do the same for counting the total length.

```
\expandafter\spath@define@length\spath@cpt\relax

\edef\spath@tmp{\noexpand\let\expandafter\noexpand\csname pgfsyssoftpath@\spath@cpt token\endcs

\ge@addto@macro\spath@length@init\spath@tmp
```

Now do the same for counting the components.

```
\expandafter\spath@define@components\spath@cpt\relax

\edef\spath@tmp{\noexpand\let\expandafter\noexpand\csname pgfsyssoftpath@\spath@cpt token\endcs

\ge@addto@macro\spath@components@init\spath@tmp
```

Now do the same for counting the real length.

```
\expandafter\spath@define@reallength\spath@cpt\relax

\edef\spath@tmp{\noexpand\let\expandafter\noexpand\csname pgfsyssoftpath@\spath@cpt token\endcs

\ge@addto@macro\spath@reallength@init\spath@tmp
```

This is for reversing a path.

```
\expandafter\spath@define@reverse\spath@cpt\relax

\edef\spath@tmp{\noexpand\let\expandafter\noexpand\csname pgfsyssoftpath@\spath@cpt token\endcs

\ge@addto@macro\spath@reverse@init\spath@tmp
```

This is for the initial coordinates

```
\expandafter\spath@define@start\spath@cpt\relax

\edef\spath@tmp{\noexpand\let\expandafter\noexpand\csname pgfsyssoftpath@\spath@cpt token\endcs

\ge@addto@macro\spath@start@init\spath@tmp
```

Once we have the initial coordinates, we reinitialise our path tokens so that they just gobble their code.

```
\edef\spath@tmp{\noexpand\let\expandafter\noexpand\csname pgfsyssoftpath@\spath@cpt token\endcs

\ge@addto@macro\spath@start@reinit\spath@tmp
```

This is for the final coordinates.

```
\expandafter\spath@define@end\spath@cpt\relax

\edef\spath@tmp{\noexpand\let\expandafter\noexpand\csname pgfsyssoftpath@\spath@cpt token\endcs

\ge@addto@macro\spath@end@init\spath@tmp
```

Split in to array

```
\expandafter\spath@define@array\spath@cpt\relax

\edef\spath@tmp{\noexpand\let\expandafter\noexpand\csname pgfsyssoftpath@\spath@cpt token\endcs

\ge@addto@macro\spath@array@init\spath@tmp
```

Prepare a path, figuring out all the data. Actions are too complicated to specify a template so just create initialisation code

```
\edef\spath@tmp{\noexpand\let\expandafter\noexpand\csname pgfsyssoftpath@\spath@cpt token\endcs

\ge@addto@macro\spath@prepare@init\spath@tmp
```

Reprocess a path

```
\expandafter\spath@define@reprocess\spath@cpt\relax

\edef\spath@tmp{\noexpand\let\expandafter\noexpand\csname pgfsyssoftpath@\spath@cpt token\endcs

\ge@addto@macro\spath@reprocess@init\spath@tmp
```

Split self-intersecting pieces

```
\expandafter\spath@define@selfintersect\expandafter{\spath@cpt}

\edef\spath@tmp{\noexpand\let\expandafter\noexpand\csname pgfsyssoftpath@\spath@cpt token\endcs

\ge@addto@macro\spath@selfintersect@init\spath@tmp
```

Bounding box

```
\expandafter\spath@define@boundingbox\expandafter{\spath@cpt}

\edef\spath@tmp{\noexpand\let\expandafter\noexpand\csname pgfsyssoftpath@\spath@cpt token\endcs

\ge@addto@macro\spath@boundingbox@init\spath@tmp
```

To string

```
\expandafter\spath@define@string\spath@cpt\relax

\edef\spath@tmp{\noexpand\let\expandafter\noexpand\csname pgfsyssoftpath@\spath@cpt token\endcs

\ge@addto@macro\spath@string@init\spath@tmp
```

Phew!

```
}
```

Now we add the corrections for the above: where an action deviates from the "default" we need to redefine it.

Correction for counting components: count movetos

```
\def\spath@comp@moveto#1#2{%
  \stepcounter{spath@length}%
}
```

Correction for counting real length: count linetos and curvetos

```
\def\spath@rlen@lineto#1#2{%
  \stepcounter{spath@length}%
}
\def\spath@rlen@curveto#1#2{%
  \stepcounter{spath@length}%
}
```

Correction for reversing initial moveto

```
\def\spath@rev@moveto#1#2{
  \ifx\spath@tmppath\pgfutil@empty
  \edef\spath@tmp{{#1}{#2}}%
  \else
  \edef\spath@tmp{{#1}{#2}\noexpand\pgfsyssoftpath@movetotoken}%
  \fi
  \ge@addbefore@macro\spath@tmppath\spath@tmp
}
```

Correction for reversing curvetos

```
\def\spath@rev@curvetosupporta#1#2{%
  \edef\spath@tmp{{#1}{#2}\noexpand\pgfsyssoftpath@curvetotoken}%
  \ge@addbefore@macro\spath@tmppath\spath@tmp
}
\def\spath@rev@curvetosupportb#1#2{%
  \edef\spath@tmp{{#1}{#2}\noexpand\pgfsyssoftpath@curvetosupportbtoken}%
  \ge@addbefore@macro\spath@tmppath\spath@tmp
}
\def\spath@rev@curveto#1#2{%
  \edef\spath@tmp{{#1}{#2}\noexpand\pgfsyssoftpath@curvetosupportatoken}%
  \ge@addbefore@macro\spath@tmppath\spath@tmp
}
```

Correction for reversing closepaths

```
\def\spath@rev@closepath#1#2{%
  \edef\spath@tmp{\noexpand\pgfsyssoftpath@closepathtoken{#1}{#2}}%
  \ge@addto@macro\spath@tmppath\spath@tmp
}
```

Correction for reprocessing path: curvetos

```
\def\spath@rep@curvetosupporta#1#2\pgfsyssoftpath@curvetosupportbtoken#3#4\pgfsyssoftpath@curve
    \pgfpathcurveto{\pgfpoint{#1}{#2}}{\pgfpoint{#3}{#4}}{\pgfpoint{#5}{#6}}%
  }%
```

Correction for converting path to string: curvetos

```
\def\spath@str@curvetosupporta#1#2\pgfsyssoftpath@curvetosupportbtoken#3#4\pgfsyssoftpath@curve
  \pgfmathsetmacro\spath@sx{#1/1cm}%
  \pgfmathsetmacro\spath@sy{#2/1cm}%
    \string\pgfpathcurveto\{\string\pgfpointxy\{\spath@sx\}\{\spath@sy\}\}%
  \pgfmathsetmacro\spath@sx{#3/1cm}%
  \pgfmathsetmacro\spath@sy{#4/1cm}%
  \{\string\pgfpointxy\{\spath@sx\}\{\spath@sy\}\}%
  \pgfmathsetmacro\spath@sx{#5/1cm}%
  \pgfmathsetmacro\spath@sy{#6/1cm}%
  \{\string\pgfpointxy\{\spath@sx\}\{\spath@sy\}\}\\%
  }%
```

Closepath

```
\def\spath@str@closepath#1#2{%
  \string\pgfclosepath\\}
```

Correction for splitting in to an array

```
\def\spath@array@moveto#1#2{%
  \ifx\spath@tmppath\pgfutil@empty
  \else
   \expandafter\global\expandafter\let\csname spath@array@\spath@path@name @\the\value{spath@ar
   \def\spath@tmppath{\pgfsyssoftpath@movetotoken{#1}{#2}}%
  \fi
  \stepcounter{spath@array}%
}
```

Correction for rounding corners: lineto and curvetosupporta need to add the extra token

```
\def\spath@rnd@lineto#1#2{%
  \edef\spath@tmp{%
    \noexpand\pgfsyssoftpath@specialroundtoken{\spath@rndx}{\spath@rndy}%
    \noexpand\pgfsyssoftpath@linetotoken{#1}{#2}}%
    \ge@addto@macro\spath@tmppath\spath@tmp
}
\def\spath@rnd@closepath#1#2{%
  \edef\spath@tmp{%
    \noexpand\pgfsyssoftpath@specialroundtoken{\spath@rndx}{\spath@rndy}%
    \noexpand\pgfsyssoftpath@closepathtoken{#1}{#2}}%
    \ge@addto@macro\spath@tmppath\spath@tmp
}
\def\spath@rnd@curvetosupportatoken#1#2{%
  \edef\spath@tmp{%
    \noexpand\pgfsyssoftpath@specialroundtoken{\spath@rndx}{\spath@rndy}%
    \noexpand\pgfsyssoftpath@curvetosupportatoken{#1}{#2}}%
    \ge@addto@macro\spath@tmppath\spath@tmp
}
```

Whereas existing special round tokens should take the maximum of their rounding and the specified rounding, and also ensure that the next corner is not *doubly* rounded.

```
\def\spath@rnd@specialround#1#2#3#4#5{%
  \pgfmathsetmacro{\spath@srndx}{max(#1,\spath@rndx)}%
  \pgfmathsetmacro{\spath@srndy}{max(#1,\spath@rndy)}%
  \edef\spath@tmp{%
    \noexpand\pgfsyssoftpath@specialroundtoken{\spath@srndx pt}{\spath@srndy pt}%
    \noexpand#3{#4}{#5}}%
    \ge@addto@macro\spath@tmppath\spath@tmp
}
```

Correction for special rounds: almost nothing should touch these

```
\def\spath@tr@specialround#1#2{%
  \edef\spath@tmp{\noexpand\pgfsyssoftpath@specialroundtoken{#1}{#2}}%
  \ge@addto@macro\spath@tmppath\spath@tmp
}
\def\spath@rev@specialround#1#2{%
  \edef\spath@tmp{\noexpand\pgfsyssoftpath@specialroundtoken{#1}{#2}}%
  \ge@addto@macro\spath@tmppath\spath@tmp
}
\def\spath@sh@specialround#1#2{%
  \edef\spath@tmp{\noexpand\pgfsyssoftpath@specialroundtoken{#1}{#2}}%
  \ge@addto@macro\spath@tmppath\spath@tmp
}
\def\spath@rot@specialround#1#2{%
```

```
  \edef\spath@tmp{\noexpand\pgfsyssoftpath@specialroundtoken{#1}{#2}}%
  \ge@addto@macro\spath@tmppath\spath@tmp
}
```

The “prepare” routine is too complicated to have defaults, so we need to set up them all here. At the moment, this reverses the path as well. This is now implemented in a different way so should be removed.

```
\def\spath@prepare@moveto#1#2{%
  \ifx\spath@tmppath\pgfutil@empty
  \g@addbefore@macro\spath@tmppath{{#1}{#2}}%
  \edef\spath@sx{#1}%
  \edef\spath@sy{#2}%
  \edef\spath@mix{#1}%
  \edef\spath@miy{#2}%
  \edef\spath@max{#1}%
  \edef\spath@may{#2}%
  \else
  \g@addbefore@macro\spath@tmppath{{#1}{#2}\pgfsyssoftpath@movetotoken}%
  \pgfmathsetmacro{\spath@mix}{min(\spath@mix,#1)}%
  \pgfmathsetmacro{\spath@max}{max(\spath@max,#1)}%
  \pgfmathsetmacro{\spath@miy}{min(\spath@miy,#2)}%
  \pgfmathsetmacro{\spath@may}{max(\spath@may,#2)}%
  \fi
  \edef\spath@ex{#1}%
  \edef\spath@ey{#2}%
  \stepcounter{spath@length}%
  \stepcounter{spath@components}%
  \let\spath@last\spath@moveto
}

\def\spath@prepare@lineto#1#2{%
  \g@addbefore@macro\spath@tmppath{{#1}{#2}\pgfsyssoftpath@linetotoken}%
  \edef\spath@ex{#1}%
  \edef\spath@ey{#2}%
  \pgfmathsetmacro{\spath@mix}{min(\spath@mix,#1)}%
  \pgfmathsetmacro{\spath@max}{max(\spath@max,#1)}%
  \pgfmathsetmacro{\spath@miy}{min(\spath@miy,#2)}%
  \pgfmathsetmacro{\spath@may}{max(\spath@may,#2)}%
  \stepcounter{spath@length}%
  \stepcounter{spath@reallength}%
  \let\spath@last\spath@lineto
  \ifx\spath@first\pgfutil@empty
  \let\spath@first\spath@lineto
  \fi
}

\def\spath@prepare@curveto#1#2{%
  \g@addbefore@macro\spath@tmppath{{#1}{#2}\pgfsyssoftpath@curvetosupportatoken}%
  \edef\spath@ex{#1}%
  \edef\spath@ey{#2}%
  \pgfmathsetmacro{\spath@mix}{min(\spath@mix,#1)}%
  \pgfmathsetmacro{\spath@max}{max(\spath@max,#1)}%
  \pgfmathsetmacro{\spath@miy}{min(\spath@miy,#2)}%
  \pgfmathsetmacro{\spath@may}{max(\spath@may,#2)}%
  \stepcounter{spath@length}%
  \stepcounter{spath@reallength}%
  \let\spath@last\spath@curveto
  \ifx\spath@first\pgfutil@empty
  \let\spath@first\spath@curveto
  \fi
}

\def\spath@prepare@curvetosupporta#1#2{%
  \g@addbefore@macro\spath@tmppath{{#1}{#2}\pgfsyssoftpath@curvetotoken}%
  \stepcounter{spath@length}%
  \pgfmathsetmacro{\spath@mix}{min(\spath@mix,#1)}%
  \pgfmathsetmacro{\spath@max}{max(\spath@max,#1)}%
  \pgfmathsetmacro{\spath@miy}{min(\spath@miy,#2)}%
  \pgfmathsetmacro{\spath@may}{max(\spath@may,#2)}%
}

\def\spath@prepare@curvetosupportb#1#2{%
  \g@addbefore@macro\spath@tmppath{{#1}{#2}\pgfsyssoftpath@curvetosupportbtoken}%
  \stepcounter{spath@length}%
  \pgfmathsetmacro{\spath@mix}{min(\spath@mix,#1)}%
  \pgfmathsetmacro{\spath@max}{max(\spath@max,#1)}%
  \pgfmathsetmacro{\spath@miy}{min(\spath@miy,#2)}%
  \pgfmathsetmacro{\spath@may}{max(\spath@may,#2)}%
}

\def\spath@prepare@closepath#1#2{%
  \g@addbefore@macro\spath@tmppath{{#1}{#2}\pgfsyssoftpath@closepathtoken}%
  \stepcounter{spath@length}%
}

\def\spath@prepare@specialround#1#2{%
  \g@addbefore@macro\spath@tmppath{{#1}{#2}\pgfsyssoftpath@specialroundtoken}%
  \stepcounter{spath@length}%
}
```

At the moment, this doesn't handle the closepath variants very well (if at all). This needs implementing.

This splits a `curveto` if there is a possibility that it self-intersects.

```
\def\spath@selfintersect@curvetosupporta#1#2\pgfsyssoftpath@curvetosupportbtoken#3#4\pgfsyssoft
  \pgfmathsetmacro{\spath@ax}{\spath@ex/28}
  \pgfmathsetmacro{\spath@bx}{#1/28}
  \pgfmathsetmacro{\spath@cx}{#3/28}
  \pgfmathsetmacro{\spath@dx}{#5/28}
  \pgfmathsetmacro{\spath@ay}{\spath@ey/28}
  \pgfmathsetmacro{\spath@by}{#2/28}
  \pgfmathsetmacro{\spath@cy}{#4/28}
  \pgfmathsetmacro{\spath@dy}{#6/28}
  \pgfmathsetmacro{\spath@enum}{%
    ( (\spath@ay - 3 * \spath@by + 3 * \spath@cy - \spath@dy) * (3 *
\spath@cx - 3 * \spath@dx)
    - (\spath@ax - 3 * \spath@bx + 3 * \spath@cx - \spath@dx) * (3 *
\spath@cy - 3 * \spath@dy) )}
  \pgfmathsetmacro{\spath@denum}{%
    ( (\spath@ax - 3 * \spath@bx + 3 * \spath@cx - \spath@dx) * (3 *
\spath@by - 6 * \spath@cy + 3 * \spath@dy)
    - (\spath@ay - 3 * \spath@by + 3 * \spath@cy - \spath@dy) * (3 *
\spath@bx - 6 * \spath@cx + 3 * \spath@dx) )}
  \pgfmathtruncatemacro{\spath@split}{\spath@enum > 0 ? (\spath@enum <
2 * \spath@denum) : (\spath@enum > 2 * \spath@denum)}
  \ifnum\spath@split=1\relax
  \pgfmathsetmacro{\spath@splitt}{.5*(\spath@enum)/(\spath@denum)}%
  \pgfsyssoftpath@setcurrentpath{\spath@tmppath}%
  \pgfpathcurvebetweentimecontinue{0}{\spath@splitt}{\pgfqpoint{\spath@ex}{\spath@ey}}{\pgfqpoi
  \pgfpathcurvebetweentimecontinue{\spath@splitt}{1}{\pgfqpoint{\spath@ex}{\spath@ey}}{\pgfqpoi
  \pgfsyssoftpath@getcurrentpath{\spath@tmppath}%
  \else
    \g@addto@macro\spath@tmppath{%
    \pgfsyssoftpath@curvetosupportatoken{#1}{#2}%
    \pgfsyssoftpath@curvetosupportbtoken{#3}{#4}%
    \pgfsyssoftpath@curvetototoken{#5}{#6}%
  }%
  \fi
  \edef\spath@ex{#5}%
  \edef\spath@ey{#6}%
}
```

Now we have some helper macros. This gets rid of the initial `moveto`, leaving the rest in `\spath@tmppath`.

```
\def\spath@trimfirst#1#2#3#4\relax{%
  \edef\spath@this@action{\string#1}%
  \ifx\spath@this@action\spath@moveto
   \def\spath@tmppath{#4}%
  \else
   \def\spath@tmppath{#1{#2}{#3}#4}%
  \fi
}
```

This replaces the initial `moveto` with a `lineto`, leaving the whole path in `\spath@tmppath`.

```
\def\spath@movetoline#1#2#3#4\relax{%
  \edef\spath@this@action{\string#1}%
  \ifx\spath@this@action\spath@moveto
   \def\spath@tmppath{\pgfsyssoftpath@linetotoken{#2}{#3}#4}%
  \else
   \def\spath@tmppath{#1{#2}{#3}#4}%
  \fi
}
```

This is our gobbling macro for splitting a path according to some criterion (length, real length, or number of components).

```
\def\spath@gobble#1#2#3{%
  \stepcounter{spath@length}%
  \edef\spath@this@action{\string#1}%
  \ifx\spath@this@action\spath@lineto
   \stepcounter{spath@reallength}%
  \fi
  \ifx\spath@this@action\spath@curveto
   \stepcounter{spath@reallength}%
  \fi
  \ifx\spath@this@action\spath@moveto
   \stepcounter{spath@components}%
  \fi
  \g@addto@macro\spath@tmppatha{#1{#2}{#3}}%
  \ifnum\spath@test=\spath@splitat\relax
   \def\spath@tmppath{\pgfsyssoftpath@movetotoken{#2}{#3}}%
   \edef\spath@last{\string#1}%
   \ifx\spath@last\spath@curvetosupporta
    \let\spath@last=\spath@curveto
   \fi
   \edef\spath@ex{#2}%
   \edef\spath@ey{#3}%
   \let\spath@next=\spath@lastgobble
  \else
   \let\spath@next=\spath@gobble
  \fi
  \pgfutil@ifnextchar\relax{%
    \ifx\spath@next\spath@gobble
     \def\spath@tmppath{\pgfsyssoftpath@movetotoken{#2}{#3}}%
     \edef\spath@last{\string#1}%
     \ifx\spath@last\spath@curvetosupporta
      \let\spath@last=\spath@curveto
     \fi
     \edef\spath@ex{#2}%
     \edef\spath@ey{#3}%
    \fi
    \edef\spath@sx{#2}%
    \edef\spath@sy{#3}%
    \let\spath@first\spath@moveto
    }{\spath@next}%
}
\def\spath@lastgobble#1#2#3#4\relax{%
  \g@addto@macro\spath@tmppath{#1{#2}{#3}#4}%
   \edef\spath@first{\string#1}%
  \ifx\spath@first\spath@curvetosupporta
  \let\spath@first=\spath@curveto
  \fi
   \edef\spath@sx{#2}%
   \edef\spath@sy{#3}%
}
```

We need a few dimensions and counters to keep track of things.

```
\newdimen\spath@trx
\newdimen\spath@try
\newcounter{spath@length}
\newcounter{spath@reallength}
\newcounter{spath@components}
\newcounter{spath@array}
```

`\spath@taper@lineto@out` This macro sets things up for tapering a `lineto`.

```
\def\spath@taper@lineto@out#1#2#3#4#5#6#7\relax{%
```

#1 is `\pgfsyssoftpath@movetotoken` #2 is x-coord of starting point #3 is y-coord of starting point #4 is `\pgfsyssoftpath@linetotoken` #5 is x-coord of ending point #6 is y-coord of ending point #7 shouldn't have anything in

```
  \pgfmathsetmacro{\spath@sx}{.7 * #2 + .3 * #5}
  \pgfmathsetmacro{\spath@sy}{.7 * #3 + .3 * #6}
  \pgfmathsetmacro{\spath@ex}{.3 * #2 + .7 * #5}
  \pgfmathsetmacro{\spath@ey}{.3 * #3 + .7 * #6}
  \edef\spath@tmp{\noexpand\spath@taper@path{#2}{#3}{\spath@sx pt}{\spath@sy pt}{\spath@ex pt}{
  \spath@tmp
}
```

`\spath@taper@curveto@out` This macro sets things up for tapering a `curveto`.

```
\def\spath@taper@curveto@out\pgfsyssoftpath@movetotoken#1#2\pgfsyssoftpath@curvetosupportatoken
  \spath@taper@path{#1}{#2}{#3}{#4}{#5}{#6}{#7}{#8}
}
```

`\spath@taper@path` This is the actual tapering macro.

```
\def\spath@taper@path#1#2#3#4#5#6#7#8{
```

#1 is x-coord of starting point #2 is y-coord of starting point #3 is x-coord of first control point #4 is y-coord of first control point #5 is x-coord of second control point #6 is y-coord of second control point #7 is x-coord of ending point #8 is y-coord of ending point

```
  \edef\spath@sx{#1}
  \edef\spath@sy{#2}
  \edef\spath@cx{#3}
  \edef\spath@cy{#4}
  \edef\spath@dx{#5}
  \edef\spath@dy{#6}
  \edef\spath@ex{#7}
  \edef\spath@ey{#8}

```

Orthogonal vector at end

```
\pgfmathsetmacro{\spath@ox}{-\spath@ey + \spath@dy}
\pgfmathsetmacro{\spath@oy}{\spath@ex - \spath@dx}

```

Orthogonal vector at start

```
\pgfmathsetmacro{\spath@sox}{\spath@sy - \spath@cy}
\pgfmathsetmacro{\spath@soy}{-\spath@sx + \spath@cx}

```

Adjust length to half the line width

```
\pgfmathsetmacro{\spath@oox}{.5*\spath@ox * \pgflinewidth / veclen(\spath@ox,\spath@oy)}
\pgfmathsetmacro{\spath@ooy}{.5*\spath@oy * \pgflinewidth / veclen(\spath@ox,\spath@oy)}

```

Adjust length to half the thinner line width

```
\pgfmathsetmacro{\spath@soox}{.5*\spath@sox * \taper@line@width / veclen(\spath@sox,\spath@so
\pgfmathsetmacro{\spath@sooy}{.5*\spath@soy * \taper@line@width / veclen(\spath@sox,\spath@so
```

Shift the start point, the control points, and the end points

```
\pgfmathsetmacro{\spath@sx}{\spath@sx + \spath@soox}
\pgfmathsetmacro{\spath@sy}{\spath@sy + \spath@sooy}
\pgfmathsetmacro{\spath@cx}{\spath@cx + \spath@soox}
\pgfmathsetmacro{\spath@cy}{\spath@cy + \spath@sooy}
\pgfmathsetmacro{\spath@dx}{\spath@dx + \spath@oox}
\pgfmathsetmacro{\spath@dy}{\spath@dy + \spath@ooy}
\pgfmathsetmacro{\spath@ex}{\spath@ex + \spath@oox}
\pgfmathsetmacro{\spath@ey}{\spath@ey + \spath@ooy}
```

Add the first pieces

```
\let\spath@tapered@path=\pgfutil@empty
\edef\spath@to@add{\noexpand\pgfsyssoftpath@movetotoken{\spath@sx pt}{\spath@sy pt}}
\ge@addto@macro\spath@tapered@path\spath@to@add
\edef\spath@to@add{\noexpand\pgfsyssoftpath@curvetosupportatoken{\spath@cx pt}{\spath@cy pt}}
\ge@addto@macro\spath@tapered@path\spath@to@add
\edef\spath@to@add{\noexpand\pgfsyssoftpath@curvetosupportbtoken{\spath@dx pt}{\spath@dy pt}}
\ge@addto@macro\spath@tapered@path\spath@to@add
\edef\spath@to@add{\noexpand\pgfsyssoftpath@curvetotoken{\spath@ex pt}{\spath@ey pt}}
\ge@addto@macro\spath@tapered@path\spath@to@add
```

Make the end roughly round

```
\pgfmathsetmacro{\spath@fx}{\spath@ex + 1.32*\spath@ooy}
\pgfmathsetmacro{\spath@fy}{\spath@ey - 1.32*\spath@oox}
\edef\spath@to@add{\noexpand\pgfsyssoftpath@curvetosupportatoken{\spath@fx pt}{\spath@fy pt}}
\ge@addto@macro\spath@tapered@path\spath@to@add
```

Shift the end points and the control points again

```
\pgfmathsetmacro{\spath@fx}{\spath@fx - 2*\spath@oox}
\pgfmathsetmacro{\spath@fy}{\spath@fy - 2*\spath@ooy}
\pgfmathsetmacro{\spath@dx}{\spath@dx - 2*\spath@oox}
\pgfmathsetmacro{\spath@dy}{\spath@dy - 2*\spath@ooy}
\pgfmathsetmacro{\spath@ex}{\spath@ex - 2*\spath@oox}
```

```
  \pgfmathsetmacro{\spath@ey}{\spath@ey - 2*\spath@ooy}
  \pgfmathsetmacro{\spath@cx}{\spath@cx - 2*\spath@soox}
  \pgfmathsetmacro{\spath@cy}{\spath@cy - 2*\spath@sooy}
  \pgfmathsetmacro{\spath@sx}{\spath@sx - 2*\spath@soox}
  \pgfmathsetmacro{\spath@sy}{\spath@sy - 2*\spath@sooy}
  \edef\spath@to@add{\noexpand\pgfsyssoftpath@curvetosupportbtoken{\spath@fx pt}{\spath@fy pt}}
  \ge@addto@macro\spath@tapered@path\spath@to@add
  \edef\spath@to@add{\noexpand\pgfsyssoftpath@curvetotoken{\spath@ex pt}{\spath@ey pt}}
  \ge@addto@macro\spath@tapered@path\spath@to@add
  \edef\spath@to@add{\noexpand\pgfsyssoftpath@curvetosupportatoken{\spath@dx pt}{\spath@dy pt}}
  \ge@addto@macro\spath@tapered@path\spath@to@add
  \edef\spath@to@add{\noexpand\pgfsyssoftpath@curvetosupportbtoken{\spath@cx pt}{\spath@cy pt}}
  \ge@addto@macro\spath@tapered@path\spath@to@add
  \edef\spath@to@add{\noexpand\pgfsyssoftpath@curvetotoken{\spath@sx pt}{\spath@sy pt}}
  \ge@addto@macro\spath@tapered@path\spath@to@add
```

Make the end roughly round

```
  \pgfmathsetmacro{\spath@fx}{\spath@sx - 1.32*\spath@sooy}
  \pgfmathsetmacro{\spath@fy}{\spath@sy + 1.32*\spath@soox}
  \edef\spath@to@add{\noexpand\pgfsyssoftpath@curvetosupportatoken{\spath@fx pt}{\spath@fy pt}}
  \ge@addto@macro\spath@tapered@path\spath@to@add
  \pgfmathsetmacro{\spath@fx}{\spath@fx + 2*\spath@soox}
  \pgfmathsetmacro{\spath@fy}{\spath@fy + 2*\spath@sooy}
  \edef\spath@to@add{\noexpand\pgfsyssoftpath@curvetosupportbtoken{\spath@fx pt}{\spath@fy pt}}
  \ge@addto@macro\spath@tapered@path\spath@to@add
  \pgfmathsetmacro{\spath@sx}{\spath@sx + 2*\spath@soox}
  \pgfmathsetmacro{\spath@sy}{\spath@sy + 2*\spath@sooy}
  \edef\spath@to@add{\noexpand\pgfsyssoftpath@curvetotoken{\spath@sx pt}{\spath@sy pt}}
  \ge@addto@macro\spath@tapered@path\spath@to@add
```

Close the path, ought to make it rounded

```
%  \g@addto@macro{\spath@tapered@path}{\pgfsyssoftpath@closepathtoken}
}
```

The following have been modified to fit the new routines

`\spath@single@split`

```
\def\spath@single@split{.3}
```

`\spath@split@single@lineto` This splits a single `lineto` into three pieces.

```
\def\spath@split@single@lineto\pgfsyssoftpath@movetotoken#1#2\pgfsyssoftpath@linetotoken#3#4#5\
  \pgfmathsetmacro{\spath@sx}{(1-\spath@single@split)*#1 + \spath@single@split * #3}
  \pgfmathsetmacro{\spath@sy}{(1-\spath@single@split)*#2 + \spath@single@split * #4}
  \pgfmathsetmacro{\spath@ex}{(1-\spath@single@split)*#3 + \spath@single@split * #1}
  \pgfmathsetmacro{\spath@ey}{(1-\spath@single@split)*#4 + \spath@single@split * #2}
  \edef\spath@tmppath{\noexpand\pgfsyssoftpath@movetotoken{#1}{#2}\noexpand\pgfsyssoftpath@line
}
```

`\spath@split@single@curveto` This splits a single `curveto` into three pieces.

```
\def\spath@split@single@curveto\pgfsyssoftpath@movetotoken#1#2\pgfsyssoftpath@curvetosupportato
```

```
  \pgfmathsetmacro{\spath@cx}{(1 - \spath@single@split)*#1 + \spath@single@split * #3}
  \pgfmathsetmacro{\spath@cy}{(1 - \spath@single@split)*#2 + \spath@single@split * #4}
  \pgfmathsetmacro{\spath@dx}{(1 - \spath@single@split)^2 * #1 + 2*\spath@single@split * (1 - \
  \pgfmathsetmacro{\spath@dy}{(1 - \spath@single@split)^2 * #2 + 2*\spath@single@split * (1 - \
  \pgfmathsetmacro{\spath@ex}{(1 - \spath@single@split)^3 * #1 + 3*\spath@single@split * (1 - \
  \pgfmathsetmacro{\spath@ey}{(1 - \spath@single@split)^3 * #2 + 3*\spath@single@split * (1 - \
  \edef\spath@tmppath{\noexpand\pgfsyssoftpath@movetotoken{#1}{#2}\noexpand\pgfsyssoftpath@curv
```

Should be some sort of optimisation to do here

```
  \pgfmathsetmacro{\spath@cx}{(1 - \spath@single@split)^2*\spath@single@split * #1 + (1 - 3 * \
  \pgfmathsetmacro{\spath@cy}{(1 - \spath@single@split)^2*\spath@single@split * #2 + (1 - 3 * \
  \pgfmathsetmacro{\spath@dx}{(1 - \spath@single@split)^2*\spath@single@split * #7 + (1 - 3 * \
  \pgfmathsetmacro{\spath@dy}{(1 - \spath@single@split)^2*\spath@single@split * #8 + (1 - 3 * \
  \pgfmathsetmacro{\spath@ex}{\spath@single@split^3 * #1 + 3 * \spath@single@split^2 * (1 - \sp
  \pgfmathsetmacro{\spath@ey}{\spath@single@split^3 * #2 + 3 * \spath@single@split^2 * (1 - \sp
  \edef\spath@to@add{\noexpand\pgfsyssoftpath@curvetosupportatoken{\spath@cx     pt}{\spath@cy 
  \ge@addto@macro\spath@tmppath\spath@to@add
  \pgfmathsetmacro{\spath@dx}{(1 - \spath@single@split)*#7 + \spath@single@split * #5}
  \pgfmathsetmacro{\spath@dy}{(1 - \spath@single@split)*#8 + \spath@single@split * #6}
  \pgfmathsetmacro{\spath@cx}{(1 - \spath@single@split)^2 * #7 + 2*\spath@single@split * (1 - \
  \pgfmathsetmacro{\spath@cy}{(1 - \spath@single@split)^2 * #8 + 2*\spath@single@split * (1 - \
  \edef\spath@to@add{\noexpand\pgfsyssoftpath@curvetosupportatoken{\spath@cx     pt}{\spath@cy 
  \ge@addto@macro\spath@tmppath\spath@to@add
}
```

The next routines split two path tokens into four, they therefore split each one in half.

\spath@double@split

```
\def\spath@double@split{.5}
```

\spath@split@first@lineto This splits the first token if it is a lineto.

```
\def\spath@split@first@lineto\pgfsyssoftpath@movetotoken#1#2\pgfsyssoftpath@linetotoken#3#4#5\r
  \pgfmathsetmacro{\spath@sx}{(1-\spath@single@split)*#1 + \spath@single@split * #3}
  \pgfmathsetmacro{\spath@sy}{(1-\spath@single@split)*#2 + \spath@single@split * #4}
  \edef\spath@split@path@start{\noexpand\pgfsyssoftpath@movetotoken{#1}{#2}\noexpand\pgfsyssoft
  \edef\spath@split@path@end{\noexpand\pgfsyssoftpath@movetotoken{#3}{#4}}
  \g@addto@macro\spath@split@path@end{#5}
}
```

\spath@split@second@lineto This splits the second token if it is a lineto.

```
\def\spath@split@second@lineto\pgfsyssoftpath@movetotoken#1#2\pgfsyssoftpath@linetotoken#3#4#5\
  \pgfmathsetmacro{\spath@sx}{(1-\spath@single@split)*#3 + \spath@single@split * #1}
  \pgfmathsetmacro{\spath@sy}{(1-\spath@single@split)*#4 + \spath@single@split * #2}
  \edef\spath@split@path@end{\noexpand\pgfsyssoftpath@linetotoken{\spath@sx pt}{\spath@sy pt}\n
}
```

\spath@split@first@curveto This splits the first token if it is a curveto.

```
\def\spath@split@first@curveto\pgfsyssoftpath@movetotoken#1#2\pgfsyssoftpath@curvetosupportatok
  \pgfmathsetmacro{\spath@cx}{(1 - \spath@double@split)*#1 + \spath@double@split * #3}
  \pgfmathsetmacro{\spath@cy}{(1 - \spath@double@split)*#2 + \spath@double@split * #4}
  \pgfmathsetmacro{\spath@dx}{(1 - \spath@double@split)^2 * #1 + 2*\spath@double@split * (1 - \
  \pgfmathsetmacro{\spath@dy}{(1 - \spath@double@split)^2 * #2 + 2*\spath@double@split * (1 - \
  \pgfmathsetmacro{\spath@ex}{(1 - \spath@double@split)^3 * #1 + 3*\spath@double@split * (1 - \
  \pgfmathsetmacro{\spath@ey}{(1 - \spath@double@split)^3 * #2 + 3*\spath@double@split * (1 - \
  \edef\spath@split@path@start{\noexpand\pgfsyssoftpath@movetotoken{#1}{#2}\noexpand\pgfsyssoft
  \pgfmathsetmacro{\spath@dx}{(1 - \spath@double@split)*#5 + \spath@double@split * #7}
  \pgfmathsetmacro{\spath@dy}{(1 - \spath@double@split)*#6 + \spath@double@split * #8}
  \pgfmathsetmacro{\spath@cx}{(1 - \spath@double@split)^2 * #3 + 2*\spath@double@split * (1 - \
  \pgfmathsetmacro{\spath@cy}{(1 - \spath@double@split)^2 * #4 + 2*\spath@double@split * (1 - \

  \edef\spath@to@add{\noexpand\pgfsyssoftpath@curvetosupportatoken{\spath@cx     pt}{\spath@cy
  \ge@addto@macro\spath@split@path@start\spath@to@add
  \edef\spath@split@path@end{\noexpand\pgfsyssoftpath@movetotoken{#7}{#8}}
  \g@addto@macro\spath@split@path@end{#9}
}
```

`\spath@split@second@curveto` This splits the second token if it is a `curveto`.

```
\def\spath@split@second@curveto\pgfsyssoftpath@movetotoken#1#2\pgfsyssoftpath@curvetosupportato
  \pgfmathsetmacro{\spath@cx}{(1 - \spath@double@split)*#1 + \spath@double@split * #3}
  \pgfmathsetmacro{\spath@cy}{(1 - \spath@double@split)*#2 + \spath@double@split * #4}
  \pgfmathsetmacro{\spath@dx}{(1 - \spath@double@split)^2 * #1 + 2*\spath@double@split * (1 - \
  \pgfmathsetmacro{\spath@dy}{(1 - \spath@double@split)^2 * #2 + 2*\spath@double@split * (1 - \
  \pgfmathsetmacro{\spath@ex}{(1 - \spath@double@split)^3 * #1 + 3*\spath@double@split * (1 - \
  \pgfmathsetmacro{\spath@ey}{(1 - \spath@double@split)^3 * #2 + 3*\spath@double@split * (1 - \
  \edef\spath@split@path@end{\noexpand\pgfsyssoftpath@curvetosupportatoken{\spath@cx     pt}{\s
  \pgfmathsetmacro{\spath@dx}{(1 - \spath@double@split)*#5 + \spath@double@split * #7}
  \pgfmathsetmacro{\spath@dy}{(1 - \spath@double@split)*#6 + \spath@double@split * #8}
  \pgfmathsetmacro{\spath@cx}{(1 - \spath@double@split)^2 * #3 + 2*\spath@double@split * (1 - \
  \pgfmathsetmacro{\spath@cy}{(1 - \spath@double@split)^2 * #4 + 2*\spath@double@split * (1 - \

  \edef\spath@to@add{\noexpand\pgfsyssoftpath@curvetosupportatoken{\spath@cx     pt}{\spath@cy
  \ge@addto@macro\spath@split@path@end\spath@to@add
}
```