The spath3 package: code

Andrew Stacey
loopspace@mathforge.org

v2.0 from 2021/01/19

1 Introduction

The `spath3` package is intended as a library for manipulating PGF's *soft paths*. In between defining a path and using it, PGF stores a path as a *soft path* where all the defining structure has been resolved into the basic operations but these have not yet been written to the output file. They can therefore still be manipulated by TEX, and as they have a very rigid form (and limited vocabulary), they are relatively easy to modify. This package provides some methods for working with these paths. It was originally not really intended for use by end users but as a foundation on which other packages can be built. However, over the years I've found myself using it at ever higher levels and so a set of interfaces has been designed using TikZ keys.

It also provides the engine that drives a few other packages, such as the `calligraphy`, `knot`, and `penrose` packages. The first two of these are subpackages of this one. The `calligraphy` package simulates a calligraphic pen stroking a path. The `knots` package can be used to draw knot (and similar) diagrams.

For usage, see the documentation of the following packages (`texdoc <package>`):

- `calligraphy`
- `knots`
- `penrose`
- `spath3` (*this* document is the code, there's another which focusses on usage)

2 Technical Details

The format of a soft path is a sequence of triples of the form `\macro {dimension}{dimension}`. The macro is one of a short list, the dimensions are coordinates in points. There are certain further restrictions, particularly that every path must begin with a `move to`, and Bézier curves consist of three triples.

In the original implementation, I wrapped this token list in a `prop` to store useful information along with the path. Over time, this additional structure has proved a little unwieldy and I've pared it back to working primarily with the original soft path as a token list.

A frequent use of this package is to break a path into pieces and do something with each of those pieces. To that end, there are various words that I use to describe the levels of the structure of a path.

At the top level is the path itself. At the bottom level are the triples of the form `\macro{dim}{dim}`, as described above. In between these are the *segments* and *components*.

A *segment* is a minimal drawing piece. Thus it might be a straight line or a Bézier curve. When a path is broken into segments then each segment is a complete path so it isn't simply a selection of triples from the original path.

A *component* is a minimal connected section of the path. So every component starts with a move command and continues until the next move command. For ease of implementation (and to enable a copperplate pen in the calligraphy package!), an isolated move is considered as a component. Thus the following path consists of three components:

```
\path (0,0) -- (1,0) (2,0) (3,0) to[out=0,in=90] (4,0);
```

3 Implementation

3.1 Initialisation

```
⟨@@=spath⟩
```

Load the LaTeX3 foundation and register us as a LaTeX3 package.

```
\NeedsTeXFormat{LaTeX2e}
\RequirePackage{expl3}
\RequirePackage{pgf}
\ProvidesExplPackage {spath3} {2021/01/19} {2.0} {Functions for
manipulating PGF soft paths}
\RequirePackage{xparse}
```

Utilities copied from https://github.com/loopspace/LaTeX3-Utilities for adding something in braces to a token list.

```
\cs_new_protected:Nn \__spath_tl_put_right_braced:Nn
{
  \tl_put_right:Nn #1 { { #2 } }
}
\cs_generate_variant:Nn \__spath_tl_put_right_braced:Nn { NV, cV, cv, Nx, cx }

\cs_new_protected:Nn \__spath_tl_gput_right_braced:Nn
{
  \tl_gput_right:Nn #1 { { #2 } }
}
\cs_generate_variant:Nn \__spath_tl_gput_right_braced:Nn { NV, cV, cv, Nx, cx }
\cs_new_protected:Nn \__spath_tl_put_left_braced:Nn
{
  \tl_put_left:Nn #1 { { #2 } }
}
\cs_generate_variant:Nn \__spath_tl_put_left_braced:Nn { NV, cV, cv, Nx, cx }

\cs_new_protected:Nn \__spath_tl_gput_left_braced:Nn
{
  \tl_gput_left:Nn #1 { { #2 } }
}
\cs_generate_variant:Nn \__spath_tl_gput_left_braced:Nn { NV, cV, cv, Nx, cx }
```

We need a slew of temporary variables.

```
\tl_new:N \l__spath_tmpa_tl
\tl_new:N \l__spath_tmpb_tl
\tl_new:N \l__spath_tmpc_tl
\tl_new:N \l__spath_tmpd_tl
\tl_new:N \l__spath_tmpe_tl

\seq_new:N \l__spath_tmpa_seq
\seq_new:N \l__spath_tmpb_seq

\tl_new:N \g__spath_output_tl
\int_new:N \g__spath_output_int
\seq_new:N \g__spath_output_seq

\dim_new:N \l__spath_tmpa_dim
\dim_new:N \l__spath_tmpb_dim
\dim_new:N \l__spath_move_x_dim
\dim_new:N \l__spath_move_y_dim
\fp_new:N \l__spath_tmpa_fp
\fp_new:N \l__spath_tmpb_fp
\int_new:N \l__spath_tmpa_int

\bool_new:N \g__spath_output_bool
\bool_new:N \l__spath_closed_bool
```

We need to be able to compare against the macros that can occur in a soft path so these token lists contain them. These are global constants so that they can be used in other packages.

```
\tl_const:Nn \c_spath_moveto_tl {\pgfsyssoftpath@movetotoken}
\tl_const:Nn \c_spath_lineto_tl {\pgfsyssoftpath@linetotoken}
\tl_const:Nn \c_spath_curveto_tl {\pgfsyssoftpath@curvetotoken}
\tl_const:Nn \c_spath_curvetoa_tl {\pgfsyssoftpath@curvetosupportatoken}
\tl_const:Nn \c_spath_curvetob_tl {\pgfsyssoftpath@curvetosupportbtoken}
\tl_const:Nn \c_spath_closepath_tl {\pgfsyssoftpath@closepathtoken}
```

We will want to be able to use anonymous spaths internally, so we create a global counter that we can use to refer to them.

```
\int_new:N \g__spath_anon_int
\int_gzero:N \g__spath_anon_int
```

Groups and iterations don’t mix well and I haven’t got a good scheme for protecting local calculations when iterating, so we do our best with iteration-specific variables.

```
\tl_new:N \l__spath_itera_tl
\tl_new:N \l__spath_iterb_tl
\tl_new:N \l__spath_iterc_tl
\tl_new:N \l__spath_iterd_tl
\tl_new:N \l__spath_iterp_tl
\dim_new:N \l__spath_itera_dim
\dim_new:N \l__spath_iterb_dim
\seq_new:N \l__spath_iter_seq
```

And some error messages

```
\msg_new:nnn { spath3 } { unknown path construction } { The~ path~ construction~ element~ #1
```

3.2 Functional Implementation

In the functional approach, we start with a token list containing a soft path and do something to it (either calculate some information or manipulate it in some fashion). We then store that information, or the manipulated path, in an appropriate macro. The macro to store it in is the first argument. These functions occur in two versions, the one with the `g` makes the assignment global.

`\spath_segments_to_seq:Nn`
`\spath_segments_gto_seq:Nn`

Splits a soft path into *segments*, storing the result in a sequence.

```
\cs_new_protected_nopar:Npn \__spath_segments_to_seq:n #1
{
  \group_begin:
  \tl_set:Nn \l__spath_itera_tl {#1}
  \tl_clear:N \l__spath_iterb_tl
  \seq_clear:N \l__spath_iter_seq
  \dim_zero:N \l__spath_itera_dim
  \dim_zero:N \l__spath_iterb_dim

  \bool_until_do:nn {
    \tl_if_empty_p:N \l__spath_itera_tl
  }
  {
    \tl_set:Nx \l__spath_iterc_tl {\tl_head:N \l__spath_itera_tl}
    \tl_set:Nx \l__spath_itera_tl {\tl_tail:N \l__spath_itera_tl}
    \tl_case:NnF \l__spath_iterc_tl
    {
      \c_spath_moveto_tl
      {
        \tl_set_eq:NN \l__spath_iterb_tl \c_spath_moveto_tl
        \tl_put_right:Nx \l__spath_iterb_tl {{\tl_head:N \l__spath_itera_tl}}
        \dim_set:Nn \l__spath_itera_dim {\tl_head:N \l__spath_itera_tl}
        \tl_set:Nx \l__spath_itera_tl {\tl_tail:N \l__spath_itera_tl}

        \tl_put_right:Nx \l__spath_iterb_tl {{\tl_head:N \l__spath_itera_tl}}
        \dim_set:Nn \l__spath_iterb_dim {\tl_head:N \l__spath_itera_tl}
        \tl_set:Nx \l__spath_itera_tl {\tl_tail:N \l__spath_itera_tl}

        \tl_set:Nx \l__spath_iterd_tl {\tl_head:N \l__spath_itera_tl}
        \tl_if_eq:NNF \l__spath_iterd_tl \c_spath_moveto_tl
        {
          \tl_clear:N \l__spath_iterb_tl
        }

      }

      \c_spath_lineto_tl
      {
        \tl_set_eq:NN \l__spath_iterb_tl \c_spath_moveto_tl
        \tl_put_right:Nx \l__spath_iterb_tl
        {
          {\dim_use:N \l__spath_itera_dim}
          {\dim_use:N \l__spath_iterb_dim}
        }
        \tl_put_right:NV \l__spath_iterb_tl \c_spath_lineto_tl

        \tl_put_right:Nx \l__spath_iterb_tl {{\tl_head:N \l__spath_itera_tl}}
        \dim_set:Nn \l__spath_itera_dim {\tl_head:N \l__spath_itera_tl}
        \tl_set:Nx \l__spath_itera_tl {\tl_tail:N \l__spath_itera_tl}

        \tl_put_right:Nx \l__spath_iterb_tl {{\tl_head:N \l__spath_itera_tl}}
        \dim_set:Nn \l__spath_iterb_dim {\tl_head:N \l__spath_itera_tl}
        \tl_set:Nx \l__spath_itera_tl {\tl_tail:N \l__spath_itera_tl}

      }

      \c_spath_curvetoa_tl
      {
        \tl_set_eq:NN \l__spath_iterb_tl \c_spath_moveto_tl
        \tl_put_right:Nx \l__spath_iterb_tl
        {
          {\dim_use:N \l__spath_itera_dim}
          {\dim_use:N \l__spath_iterb_dim}
        }
        \tl_put_right:NV \l__spath_iterb_tl \c_spath_curvetoa_tl

        \prg_replicate:nn {2} {
          \tl_put_right:Nx \l__spath_iterb_tl {{\tl_head:N \l__spath_itera_tl}}
          \tl_set:Nx \l__spath_itera_tl {\tl_tail:N \l__spath_itera_tl}
          \tl_put_right:Nx \l__spath_iterb_tl {{\tl_head:N                 \l__spath_itera_tl}
          \tl_set:Nx \l__spath_itera_tl {\tl_tail:N \l__spath_itera_tl}
          \tl_put_right:Nx \l__spath_iterb_tl {\tl_head:N               \l__spath_itera_tl}
          \tl_set:Nx \l__spath_itera_tl {\tl_tail:N \l__spath_itera_tl}
        }

        \tl_put_right:Nx \l__spath_iterb_tl {{\tl_head:N \l__spath_itera_tl}}
        \dim_set:Nn \l__spath_itera_dim {\tl_head:N \l__spath_itera_tl}
        \tl_set:Nx \l__spath_itera_tl {\tl_tail:N \l__spath_itera_tl}

        \tl_put_right:Nx \l__spath_iterb_tl {{\tl_head:N \l__spath_itera_tl}}
        \dim_set:Nn \l__spath_iterb_dim {\tl_head:N \l__spath_itera_tl}
        \tl_set:Nx \l__spath_itera_tl {\tl_tail:N \l__spath_itera_tl}

      }

      \c_spath_closepath_tl
      {
        \tl_set_eq:NN \l__spath_iterb_tl \c_spath_moveto_tl
        \tl_put_right:Nx \l__spath_iterb_tl
        {
          {\dim_use:N \l__spath_itera_dim}
          {\dim_use:N \l__spath_iterb_dim}
        }
        \tl_put_right:NV \l__spath_iterb_tl \c_spath_lineto_tl

        \tl_put_right:Nx \l__spath_iterb_tl {{\tl_head:N \l__spath_itera_tl}}
        \dim_set:Nn \l__spath_itera_dim {\tl_head:N \l__spath_itera_tl}
        \tl_set:Nx \l__spath_itera_tl {\tl_tail:N \l__spath_itera_tl}

        \tl_put_right:Nx \l__spath_iterb_tl {{\tl_head:N \l__spath_itera_tl}}
        \dim_set:Nn \l__spath_iterb_dim {\tl_head:N \l__spath_itera_tl}
        \tl_set:Nx \l__spath_itera_tl {\tl_tail:N \l__spath_itera_tl}

      }

    }
    {

      \tl_set_eq:NN \l__spath_iterb_tl \l__spath_iterc_tl
      \tl_put_right:Nx \l__spath_iterb_tl {{\tl_head:N \l__spath_itera_tl}}
      \dim_set:Nn \l__spath_itera_dim {\tl_head:N \l__spath_itera_tl}
      \tl_set:Nx \l__spath_itera_tl {\tl_tail:N \l__spath_itera_tl}

      \tl_put_right:Nx \l__spath_iterb_tl {{\tl_head:N \l__spath_itera_tl}}
      \dim_set:Nn \l__spath_iterb_dim {\tl_head:N \l__spath_itera_tl}
      \tl_set:Nx \l__spath_itera_tl {\tl_tail:N \l__spath_itera_tl}

    }

    \tl_if_empty:NF \l__spath_iterb_tl
    {
      \seq_put_right:NV \l__spath_iter_seq \l__spath_iterb_tl
    }
    \tl_clear:N \l__spath_iterb_tl
  }

  \seq_gclear:N \g__spath_output_seq
  \seq_gset_eq:NN \g__spath_output_seq \l__spath_iter_seq
  \group_end:
}
\cs_new_protected_nopar:Npn \spath_segments_to_seq:Nn #1#2
{
  \__spath_segments_to_seq:n {#2}
  \seq_clear_new:N #1
  \seq_set_eq:NN #1 \g__spath_output_seq
  \seq_gclear:N \g__spath_output_seq
}
\cs_generate_variant:Nn \spath_segments_to_seq:Nn {NV, cn, cV, Nv, cv}
\cs_new_protected_nopar:Npn \spath_segments_gto_seq:Nn #1#2
{
  \__spath_segments_to_seq:n {#2}
  \seq_clear_new:N #1
  \seq_gset_eq:NN #1 \g__spath_output_seq
  \seq_gclear:N \g__spath_output_seq
}
\cs_generate_variant:Nn \spath_segments_gto_seq:Nn {NV, cn, cV, Nv, cv}
```

(*End definition for* `\spath_segments_to_seq:Nn` *and* `\spath_segments_gto_seq:Nn`. *These functions are documented on page* **??**.)

`\spath_components_to_seq:Nn`
`\spath_components_gto_seq:Nn`
`\spath_components_to_clist:Nn`
`\spath_components_gto_clist:Nn`

Splits a soft path into *components*, storing the result in a sequence or a clist.

```
\cs_new_protected_nopar:Npn \__spath_components_to_seq:n #1
{
  \group_begin:
  \tl_set:Nn \l__spath_itera_tl {#1}
  \seq_clear:N \l__spath_iter_seq
  \tl_set:Nx \l__spath_itera_tl {\tl_tail:N \l__spath_itera_tl}
  \tl_put_right:NV \l__spath_itera_tl \c_spath_moveto_tl
  \tl_set_eq:NN \l__spath_iterb_tl \c_spath_moveto_tl
  \bool_do_until:nn {
    \tl_if_empty_p:N \l__spath_itera_tl
  }
  {
    \tl_set:Nx \l__spath_iterc_tl {\tl_head:N \l__spath_itera_tl}
    \tl_if_eq:NNT \l__spath_iterc_tl \c_spath_moveto_tl
    {
      \seq_put_right:NV \l__spath_iter_seq \l__spath_iterb_tl
      \tl_clear:N \l__spath_iterb_tl
    }
    \tl_if_single:NTF \l__spath_iterc_tl
    {
      \tl_put_right:NV \l__spath_iterb_tl \l__spath_iterc_tl
    }
    {
      \tl_put_right:Nx \l__spath_iterb_tl {{\l__spath_iterc_tl}}
    }
    \tl_set:Nx \l__spath_itera_tl {\tl_tail:N \l__spath_itera_tl}
  }

  \seq_gclear:N \g__spath_output_seq
  \seq_gset_eq:NN \g__spath_output_seq \l__spath_iter_seq
  \group_end:
}
\cs_new_protected_nopar:Npn \spath_components_to_seq:Nn #1#2
{
  \__spath_components_to_seq:n {#2}
  \seq_clear_new:N #1
  \seq_set_eq:NN #1 \g__spath_output_seq
  \seq_gclear:N \g__spath_output_seq
}
\cs_generate_variant:Nn \spath_components_to_seq:Nn {NV, cn, cV, cv, Nv}
\cs_new_protected_nopar:Npn \spath_components_gto_seq:Nn #1#2
{
  \__spath_components_to_seq:n {#2}
  \seq_clear_new:N #1
  \seq_gset_eq:NN #1 \g__spath_output_seq
  \seq_gclear:N \g__spath_output_seq
}
\cs_generate_variant:Nn \spath_components_gto_seq:Nn {NV, cn, cV, cv, Nv}
\cs_new_protected_nopar:Npn \spath_components_to_clist:Nn #1#2
{
  \__spath_components_to_seq:n {#2}
  \clist_clear_new:N #1
  \clist_set_from_seq:NN #1 \g__spath_output_seq
  \seq_gclear:N \g__spath_output_seq
}
\cs_generate_variant:Nn \spath_components_to_clist:Nn {NV, cn, cV, cv, Nv}
\cs_new_protected_nopar:Npn \spath_components_gto_clist:Nn #1#2
{
  \__spath_components_to_seq:n {#2}
  \clist_clear_new:N #1
  \clist_gset_from_seq:NN #1 \g__spath_output_seq
  \seq_gclear:N \g__spath_output_seq
}
\cs_generate_variant:Nn \spath_components_gto_clist:Nn {NV, cn, cV, cv, Nv}
```

(*End definition for* `\spath_components_to_seq:Nn` *and others. These functions are documented on page* **??**.)

`\spath_length:n`

Counts the number of triples in the path.

```
\cs_new_protected_nopar:Npn \spath_length:n #1
{
  \int_eval:n {\tl_count:n {#1} / 3}
}
\cs_generate_variant:Nn \spath_length:n {V}
```

(*End definition for* `\spath_length:n`. *This function is documented on page* **??**.)

`\spath_reallength:Nn`
`\spath_greallength:Nn`

The real length of a path is the number of triples that actually draw something (that is, the number of lines, curves, and closepaths).

```
\cs_new_protected_nopar:Npn \__spath_reallength:n #1
{
  \group_begin:
  \int_set:Nn \l__spath_tmpa_int {0}
  \tl_map_inline:nn {#1} {
    \tl_set:Nn \l__spath_tmpc_tl {##1}
    \tl_case:NnT \l__spath_tmpc_tl
    {
      \c_spath_lineto_tl {}
      \c_spath_curveto_tl {}
      \c_spath_closepath_tl {}
    }
    {
      \int_incr:N \l__spath_tmpa_int
    }
  }
  \int_gzero:N \g__spath_output_int
  \int_gset_eq:NN \g__spath_output_int \l__spath_tmpa_int
  \group_end:
}
\cs_new_protected_nopar:Npn \spath_reallength:Nn #1#2
{
  \__spath_reallength:n {#2}
  \int_set_eq:NN #1 \g__spath_output_int
  \int_gzero:N \g__spath_output_int
}
\cs_generate_variant:Nn \spath_reallength:Nn {NV, cn, cV, Nv, cv}
\cs_new_protected_nopar:Npn \spath_greallength:Nn #1#2
{
  \__spath_reallength:n {#2}
  \int_gset_eq:NN #1 \g__spath_output_int
  \int_gzero:N \g__spath_output_int
}
\cs_generate_variant:Nn \spath_greallength:Nn {NV, cn, cV}
```

(*End definition for* `\spath_reallength:Nn` *and* `\spath_greallength:Nn`. *These functions are documented on page* **??**.)

`\spath_numberofcomponents:Nn`
`\spath_gnumberofcomponents:Nn`

A component is a continuous segment of the path, separated by moves. Successive moves are not collapsed, and zero length moves count.

```
\cs_new_protected_nopar:Npn \__spath_numberofcomponents:n #1
{
  \group_begin:
  \int_set:Nn \l__spath_tmpa_int {0}
  \tl_map_inline:nn {#1} {
    \tl_set:Nn \l__spath_tmpa_tl {##1}
    \tl_case:Nn \l__spath_tmpa_tl
    {
      \c_spath_moveto_tl
      {
        \int_incr:N \l__spath_tmpa_int
      }
    }
  }
  \int_gzero:N \g__spath_output_int
  \int_gset_eq:NN \g__spath_output_int \l__spath_tmpa_int
  \group_end:
}
\cs_new_protected_nopar:Npn \spath_numberofcomponents:Nn #1#2
{
  \__spath_numberofcomponents:n {#2}
  \int_set_eq:NN #1 \g__spath_output_int
  \int_gzero:N \g__spath_output_int
}
\cs_generate_variant:Nn \spath_numberofcomponents:Nn {NV, cn, cV}
\cs_new_protected_nopar:Npn \spath_gnumberofcomponents:Nn #1#2
{
  \__spath_numberofcomponents:n {#2}
  \int_gset_eq:NN #1 \g__spath_output_int
  \int_gzero:N \g__spath_output_int
}
\cs_generate_variant:Nn \spath_gnumberofcomponents:Nn {NV, cn, cV}
```

(*End definition for `\spath_numberofcomponents:Nn` and `\spath_gnumberofcomponents:Nn`. These functions are documented on page* **??**.)

`\spath_initialpoint:Nn`
`\spath_ginitialpoint:Nn`

The starting point of the path.

```
\cs_new_protected_nopar:Npn \__spath_initialpoint:n #1
{
  \group_begin:
  \tl_clear:N \l__spath_tmpb_tl
  \tl_set:Nx \l__spath_tmpb_tl
  {
    { \tl_item:nn {#1} {2} }
    { \tl_item:nn {#1} {3} }
  }
  \tl_gset_eq:NN \g__spath_output_tl \l__spath_tmpb_tl
  \group_end:
}
\cs_new_protected_nopar:Npn \spath_initialpoint:Nn #1#2
{
  \__spath_initialpoint:n {#2}
  \tl_set_eq:NN #1 \g__spath_output_tl
  \tl_gclear:N \g__spath_output_tl
}
\cs_generate_variant:Nn \spath_initialpoint:Nn {NV, cn, cV, Nv}
\cs_new_protected_nopar:Npn \spath_ginitialpoint:Nn #1#2
{
  \__spath_initialpoint:n {#2}
  \tl_gset_eq:NN #1 \g__spath_output_tl
  \tl_gclear:N \g__spath_output_tl
}
\cs_generate_variant:Nn \spath_ginitialpoint:Nn {NV, cn, cV, Nv}
```

(*End definition for* `\spath_initialpoint:Nn` *and* `\spath_ginitialpoint:Nn`*. These functions are documented on page* **??**.)

`\spath_finalpoint:Nn`
`\spath_gfinalpoint:Nn`

The final point of the path.

```
\cs_new_protected_nopar:Npn \__spath_finalpoint:n #1
{
  \group_begin:
  \tl_set:Nn \l__spath_tmpa_tl {#1}
  \tl_reverse:N \l__spath_tmpa_tl
  \tl_clear:N \l__spath_tmpb_tl
  \tl_set:Nx \l__spath_tmpb_tl
  {
    { \tl_item:Nn \l__spath_tmpa_tl {2} }
    { \tl_item:Nn \l__spath_tmpa_tl {1} }
  }
  \tl_gset_eq:NN \g__spath_output_tl \l__spath_tmpb_tl
  \group_end:
}
\cs_new_protected_nopar:Npn \spath_finalpoint:Nn #1#2
{
  \__spath_finalpoint:n {#2}
  \tl_set_eq:NN #1 \g__spath_output_tl
  \tl_gclear:N \g__spath_output_tl
}
\cs_generate_variant:Nn \spath_finalpoint:Nn {NV, cn, cV, Nv}
\cs_new_protected_nopar:Npn \spath_gfinalpoint:Nn #1#2
{
  \__spath_finalpoint:n {#2}
  \tl_gset_eq:NN #1 \g__spath_output_tl
  \tl_gclear:N \g__spath_output_tl
}
\cs_generate_variant:Nn \spath_gfinalpoint:Nn {NV, cn, cV, Nv}
```

(*End definition for* `\spath_finalpoint:Nn` *and* `\spath_gfinalpoint:Nn`*. These functions are documented on page* **??**.)

`\spath_reverse:Nn`
`\spath_greverse:Nn`

This computes the reverse of the path.

```
\cs_new_protected_nopar:Npn \__spath_reverse:n #1
{
  \group_begin:
  \tl_set:Nn \l__spath_tmpa_tl {#1}

  \tl_clear:N \l__spath_tmpb_tl
```

```
\tl_clear:N \l__spath_tmpd_tl
\tl_set:Nx \l__spath_tmpa_tl {\tl_tail:N \l__spath_tmpa_tl}
\dim_set:Nn \l__spath_tmpa_dim {\tl_head:N \l__spath_tmpa_tl}
\tl_set:Nx \l__spath_tmpa_tl {\tl_tail:N \l__spath_tmpa_tl}
\dim_set:Nn \l__spath_tmpb_dim {\tl_head:N \l__spath_tmpa_tl}
\tl_set:Nx \l__spath_tmpa_tl {\tl_tail:N \l__spath_tmpa_tl}

\tl_put_left:Nx \l__spath_tmpd_tl
{
  {\dim_use:N \l__spath_tmpa_dim}
  {\dim_use:N \l__spath_tmpb_dim}
}

\bool_set_false:N \l__spath_closed_bool

\bool_until_do:nn {
  \tl_if_empty_p:N \l__spath_tmpa_tl
}
{
  \tl_set:Nx \l__spath_tmpc_tl {\tl_head:N \l__spath_tmpa_tl}

  \tl_case:NnTF \l__spath_tmpc_tl
  {
    \c_spath_moveto_tl {

      \bool_if:NT \l__spath_closed_bool
      {
        \tl_put_right:NV \l__spath_tmpd_tl \c_spath_closepath_tl
        \tl_set:Nx \l__spath_tmpe_tl {\tl_tail:N \l__spath_tmpd_tl}
        \tl_put_right:Nx \l__spath_tmpd_tl
        {
          { \tl_head:N \l__spath_tmpd_tl }
          { \tl_head:N \l__spath_tmpe_tl }
        }
      }
      \bool_set_false:N \l__spath_closed_bool
      \tl_put_left:NV \l__spath_tmpd_tl \c_spath_moveto_tl
      \tl_put_left:NV \l__spath_tmpb_tl \l__spath_tmpd_tl
      \tl_clear:N \l__spath_tmpd_tl
    }
    \c_spath_lineto_tl {
      \tl_put_left:NV \l__spath_tmpd_tl \c_spath_lineto_tl
    }
    \c_spath_curveto_tl {
      \tl_put_left:NV \l__spath_tmpd_tl \c_spath_curvetoa_tl
    }
    \c_spath_curvetoa_tl {
      \tl_put_left:NV \l__spath_tmpd_tl \c_spath_curveto_tl
    }
    \c_spath_curvetob_tl {
      \tl_put_left:NV \l__spath_tmpd_tl \c_spath_curvetob_tl
    }
  }
  {
      \tl_set:Nx \l__spath_tmpa_tl {\tl_tail:N \l__spath_tmpa_tl}

      \dim_set:Nn \l__spath_tmpa_dim {\tl_head:N \l__spath_tmpa_tl}
      \tl_set:Nx \l__spath_tmpa_tl {\tl_tail:N \l__spath_tmpa_tl}
      \dim_set:Nn \l__spath_tmpb_dim {\tl_head:N \l__spath_tmpa_tl}
      \tl_set:Nx \l__spath_tmpa_tl {\tl_tail:N \l__spath_tmpa_tl}

      \tl_put_left:Nx \l__spath_tmpd_tl
      {
        {\dim_use:N \l__spath_tmpa_dim}
        {\dim_use:N \l__spath_tmpb_dim}
      }

    }
    {
      \tl_if_eq:NNTF \l__spath_tmpc_tl \c_spath_closepath_tl
      {
        \bool_set_true:N \l__spath_closed_bool
      }
      {
        \msg_warning:nnx { spath3 } { unknown path construction } {\l__spath_tmpc_tl }
      }

      \tl_set:Nx \l__spath_tmpa_tl {\tl_tail:N \l__spath_tmpa_tl}
      \tl_set:Nx \l__spath_tmpa_tl {\tl_tail:N \l__spath_tmpa_tl}
      \tl_set:Nx \l__spath_tmpa_tl {\tl_tail:N \l__spath_tmpa_tl}

    }
  }

  \bool_if:NT \l__spath_closed_bool
  {
    \tl_put_right:NV \l__spath_tmpd_tl \c_spath_closepath_tl
    \tl_set:Nx \l__spath_tmpe_tl {\tl_tail:N \l__spath_tmpd_tl}
    \tl_put_right:Nx \l__spath_tmpd_tl
    {
      { \tl_head:N \l__spath_tmpd_tl }
      { \tl_head:N \l__spath_tmpe_tl }
    }
  }

  \bool_set_false:N \l__spath_closed_bool
  \tl_put_left:NV \l__spath_tmpd_tl \c_spath_moveto_tl
  \tl_put_left:NV \l__spath_tmpb_tl \l__spath_tmpd_tl

  \tl_gset_eq:NN \g__spath_output_tl \l__spath_tmpb_tl
  \group_end:
}
\cs_new_protected_nopar:Npn \spath_reverse:Nn #1#2
{
  \__spath_reverse:n {#2}
  \tl_set_eq:NN #1 \g__spath_output_tl
  \tl_gclear:N \g__spath_output_tl
}
```

```
\cs_generate_variant:Nn \spath_reverse:Nn {NV, cn, cV, Nv}
\cs_new_protected_nopar:Npn \spath_reverse:N #1
{
  \spath_reverse:NV #1#1
}
\cs_generate_variant:Nn \spath_reverse:N {c}
\cs_new_protected_nopar:Npn \spath_greverse:Nn #1#2
{
  \__spath_reverse:n {#2}
  \tl_gset_eq:NN #1 \g__spath_output_tl
  \tl_gclear:N \g__spath_output_tl
}
\cs_generate_variant:Nn \spath_greverse:Nn {NV, cn, cV, Nv}
\cs_new_protected_nopar:Npn \spath_greverse:N #1
{
  \spath_greverse:NV #1#1
}
\cs_generate_variant:Nn \spath_greverse:N {c}
```

(*End definition for* `\spath_reverse:Nn` *and* `\spath_greverse:Nn`*. These functions are documented on page* **??**.)

`\spath_initialaction:Nn`
`\spath_ginitialaction:Nn`

This is the first thing that the path does (after the initial move).

```
\cs_new_protected_nopar:Npn \__spath_initialaction:n #1
{
  \group_begin:
  \tl_clear:N \l__spath_tmpa_tl
  \int_compare:nT
  {
    \tl_count:n {#1} > 3
  }
  {
    \tl_set:Nx \l__spath_tmpa_tl
    {
      \tl_item:Nn {#1} {4}
    }
  }
  \tl_gset_eq:NN \g__spath_output_tl \l__spath_tmpa_tl
  \group_end:
}
\cs_new_protected_nopar:Npn \spath_initialaction:Nn #1#2
{
  \__spath_initialaction:n {#2}
  \tl_set_eq:NN #1 \g__spath_output_tl
  \tl_gclear:N \g__spath_output_tl
}
\cs_generate_variant:Nn \spath_initialaction:Nn {NV}
\cs_new_protected_nopar:Npn \spath_ginitialaction:Nn #1#2
{
  \__spath_initialaction:n {#2}
  \tl_gset_eq:NN #1 \g__spath_output_tl
  \tl_gclear:N \g__spath_output_tl
}
\cs_generate_variant:Nn \spath_ginitialaction:Nn {NV}
```

(*End definition for* `\spath_initialaction:Nn` *and* `\spath_ginitialaction:Nn`*. These functions are documented on page* **??**.)

`\spath_finalaction:Nn`
`\spath_gfinalaction:Nn`

This is the last thing that the path does.

```
\cs_new_protected_nopar:Npn \__spath_finalaction:n #1
{
  \group_begin:
  \tl_clear:N \l__spath_tmpb_tl
  \int_compare:nT
  {
    \tl_count:n {#1} > 3
  }
  {
    \tl_set:Nn \l__spath_tmpa_tl {#1}
    \tl_reverse:N \l__spath_tmpa_tl
    \tl_set:Nx \l__spath_tmpb_tl
    {
      \tl_item:Nn \l__spath_tmpa_tl {3}
    }
    \tl_if_eq:NNT \l__spath_tmpb_tl \c_spath_curvetoa_tl
    {
      \tl_set_eq:NN \l__spath_tmpb_tl \c_spath_curveto_tl
    }
  }
  \tl_gset_eq:NN \g__spath_output_tl \l__spath_tmpb_tl
  \group_end:
}
\cs_new_protected_nopar:Npn \spath_finalaction:Nn #1#2
{
  \__spath_finalaction:n {#2}
  \tl_set_eq:NN #1 \g__spath_output_tl
  \tl_gclear:N \g__spath_output_tl
}
\cs_generate_variant:Nn \spath_finalaction:Nn {NV}
\cs_new_protected_nopar:Npn \spath_gfinalaction:Nn #1#2
{
  \__spath_finalaction:n {#2}
  \tl_gset_eq:NN #1 \g__spath_output_tl
  \tl_gclear:N \g__spath_output_tl
}
\cs_generate_variant:Nn \spath_gfinalaction:Nn {NV}
```

(*End definition for* `\spath_finalaction:Nn` *and* `\spath_gfinalaction:Nn`*. These functions are documented on page* **??**.)

`\spath_minbb:Nn`
`\spath_gminbb:Nn`

This computes the minimum (bottom left) of the bounding box of the path.

```
\cs_new_protected_nopar:Npn \__spath_minbb:n #1
{
  \group_begin:
  \tl_set:Nn \l__spath_tmpa_tl {#1}
  \tl_set:Nx \l__spath_tmpa_tl {\tl_tail:N \l__spath_tmpa_tl}
  \dim_set:Nn \l__spath_tmpa_dim {\tl_head:N \l__spath_tmpa_tl}
  \tl_set:Nx \l__spath_tmpa_tl {\tl_tail:N \l__spath_tmpa_tl}
  \dim_set:Nn \l__spath_tmpb_dim {\tl_head:N \l__spath_tmpa_tl}
  \tl_set:Nx \l__spath_tmpa_tl {\tl_tail:N \l__spath_tmpa_tl}
  \bool_until_do:nn {
    \tl_if_empty_p:N \l__spath_tmpa_tl
  }
  {
    \tl_set:Nx \l__spath_tmpa_tl {\tl_tail:N \l__spath_tmpa_tl}
    \dim_set:Nn \l__spath_tmpa_dim {\dim_min:nn {\tl_head:N \l__spath_tmpa_tl} {\l__spath_tm
    \tl_set:Nx \l__spath_tmpa_tl {\tl_tail:N \l__spath_tmpa_tl}
    \dim_set:Nn \l__spath_tmpb_dim {\dim_min:nn {\tl_head:N \l__spath_tmpa_tl} {\l__spath_tm
    \tl_set:Nx \l__spath_tmpa_tl {\tl_tail:N \l__spath_tmpa_tl}
  }
  \tl_clear:N \l__spath_tmpb_tl
  \tl_put_right:Nx \l__spath_tmpb_tl
  {
    {\dim_use:N \l__spath_tmpa_dim}
    {\dim_use:N \l__spath_tmpb_dim}
  }
  \tl_gset_eq:NN \g__spath_output_tl \l__spath_tmpb_tl
  \group_end:
}
\cs_new_protected_nopar:Npn \spath_minbb:Nn #1#2
{
  \__spath_minbb:n {#2}
  \tl_set_eq:NN #1 \g__spath_output_tl
  \tl_gclear:N \g__spath_output_tl
}
\cs_generate_variant:Nn \spath_minbb:Nn {NV, cn, cV}
\cs_new_protected_nopar:Npn \spath_gminbb:Nn #1#2
{
  \__spath_minbb:n {#2}
  \tl_gset_eq:NN #1 \g__spath_output_tl
  \tl_gclear:N \g__spath_output_tl
}
\cs_generate_variant:Nn \spath_gminbb:Nn {NV, cn, cV}
```

(*End definition for* `\spath_minbb:Nn` *and* `\spath_gminbb:Nn`*. These functions are documented on page* **??**.)

`\spath_maxbb:Nn`
`\spath_gmaxbb:Nn`

This computes the maximum (top right) of the bounding box of the path.

```
\cs_new_protected_nopar:Npn \__spath_maxbb:n #1
{
  \group_begin:
  \tl_set:Nn \l__spath_tmpa_tl {#1}
  \tl_set:Nx \l__spath_tmpa_tl {\tl_tail:N \l__spath_tmpa_tl}
  \dim_set:Nn \l__spath_tmpa_dim {\tl_head:N \l__spath_tmpa_tl}
  \tl_set:Nx \l__spath_tmpa_tl {\tl_tail:N \l__spath_tmpa_tl}
  \dim_set:Nn \l__spath_tmpb_dim {\tl_head:N \l__spath_tmpa_tl}
  \tl_set:Nx \l__spath_tmpa_tl {\tl_tail:N \l__spath_tmpa_tl}
  \bool_until_do:nn {
    \tl_if_empty_p:N \l__spath_tmpa_tl
  }
  {
    \tl_set:Nx \l__spath_tmpa_tl {\tl_tail:N \l__spath_tmpa_tl}
    \dim_set:Nn \l__spath_tmpa_dim {\dim_max:nn {\tl_head:N \l__spath_tmpa_tl} {\l__spath_tm
    \tl_set:Nx \l__spath_tmpa_tl {\tl_tail:N \l__spath_tmpa_tl}
    \dim_set:Nn \l__spath_tmpb_dim {\dim_max:nn {\tl_head:N \l__spath_tmpa_tl} {\l__spath_tm
    \tl_set:Nx \l__spath_tmpa_tl {\tl_tail:N \l__spath_tmpa_tl}
  }
  \tl_clear:N \l__spath_tmpb_tl
  \tl_put_right:Nx \l__spath_tmpb_tl
  {
    {\dim_use:N \l__spath_tmpa_dim}
    {\dim_use:N \l__spath_tmpb_dim}
  }
  \tl_gset_eq:NN \g__spath_output_tl \l__spath_tmpb_tl
  \group_end:
}
\cs_new_protected_nopar:Npn \spath_maxbb:Nn #1#2
{
  \__spath_maxbb:n {#2}
  \tl_set_eq:NN #1 \g__spath_output_tl
  \tl_gclear:N \g__spath_output_tl
}
\cs_generate_variant:Nn \spath_maxbb:Nn {NV, cn, cV}
\cs_new_protected_nopar:Npn \spath_gmaxbb:Nn #1#2
{
  \__spath_maxbb:n {#2}
  \tl_gset_eq:NN #1 \g__spath_output_tl
  \tl_gclear:N \g__spath_output_tl
}
\cs_generate_variant:Nn \spath_gmaxbb:Nn {NV, cn, cV}
```

(*End definition for* `\spath_maxbb:Nn` *and* `\spath_gmaxbb:Nn`*. These functions are documented on page* **??**.)

`\spath_save_to_aux:Nn`
`\spath_save_to_aux:N`

This saves a soft path to the auxfile. The first argument is the macro that will be assigned to the soft path when the aux file is read back in.

```
\int_set:Nn \l__spath_tmpa_int {\char_value_catcode:n {`@}}
\char_set_catcode_letter:N @
\cs_new_protected_nopar:Npn \spath_save_to_aux:Nn #1#2 {
  \tl_if_empty:nF {#2}
  {
    \tl_clear:N \l__spath_tmpa_tl
    \tl_put_right:Nn \l__spath_tmpa_tl {
      \ExplSyntaxOn
      \tl_clear:N #1
      \tl_set:Nn #1 {#2}
      \ExplSyntaxOff
    }
    \protected@write\@auxout{}{
      \tl_to_str:N \l__spath_tmpa_tl
    }
  }
}
\char_set_catcode:nn {`@} {\l__spath_tmpa_int}
\cs_generate_variant:Nn \spath_save_to_aux:Nn {cn, cV, NV}
\cs_new_protected_nopar:Npn \spath_save_to_aux:N #1
{
  \tl_if_exist:NT #1
  {
    \spath_save_to_aux:NV #1#1
  }
}
```

(*End definition for* `\spath_save_to_aux:Nn` *and* `\spath_save_to_aux:N`*. These functions are documented on page* **??**.)

3.3 Path Manipulation

These functions all manipulate a soft path. They come with a variety of different argument specifications. As a general rule, the first argument is the macro in which to store the modified path, the second is the path to manipulate, and the rest are the information about what to do. There is always a variant in which the path is specified by a macro and restored back in that same macro.

`\spath_translate:Nnnn`
`\spath_translate:Nnn`
`\spath_gtranslate:Nnnn`
`\spath_gtranslate:Nnn`

Translates a path.

```
\cs_new_protected_nopar:Npn \__spath_translate:nnn #1#2#3
{
  \group_begin:
  \tl_set:Nn \l__spath_tmpa_tl {#1}
  \tl_clear:N \l__spath_tmpb_tl
  \bool_until_do:nn {
    \tl_if_empty_p:N \l__spath_tmpa_tl
  }
  {
    \tl_put_right:Nx \l__spath_tmpb_tl {\tl_head:N \l__spath_tmpa_tl}
    \tl_set:Nx \l__spath_tmpa_tl {\tl_tail:N \l__spath_tmpa_tl}

    \dim_set:Nn \l__spath_tmpa_dim {\tl_head:N \l__spath_tmpa_tl + #2}
    \tl_set:Nx \l__spath_tmpa_tl {\tl_tail:N \l__spath_tmpa_tl}

    \dim_set:Nn \l__spath_tmpb_dim {\tl_head:N \l__spath_tmpa_tl + #3}
    \tl_set:Nx \l__spath_tmpa_tl {\tl_tail:N \l__spath_tmpa_tl}

    \tl_put_right:Nx \l__spath_tmpb_tl
    {
      {\dim_use:N \l__spath_tmpa_dim}
      {\dim_use:N \l__spath_tmpb_dim}
    }
  }
  \tl_gset_eq:NN \g__spath_output_tl \l__spath_tmpb_tl
  \group_end:
}
\cs_new_protected_nopar:Npn \spath_translate:Nnnn #1#2#3#4
{
  \__spath_translate:nnn {#2}{#3}{#4}
  \tl_set_eq:NN #1 \g__spath_output_tl
  \tl_gclear:N \g__spath_output_tl
}
\cs_generate_variant:Nn \spath_translate:Nnnn {NVxx, NVVV, NVnn}
\cs_new_protected_nopar:Npn \spath_translate:Nnn #1#2#3
{
  \spath_translate:NVnn #1#1{#2}{#3}
}
\cs_generate_variant:Nn \spath_translate:Nnn {NVV, cnn, cVV}
\cs_new_protected_nopar:Npn \spath_gtranslate:Nnnn #1#2#3#4
{
  \__spath_translate:nnn {#2}{#3}{#4}
  \tl_gset_eq:NN #1 \g__spath_output_tl
  \tl_gclear:N \g__spath_output_tl
}
\cs_generate_variant:Nn \spath_gtranslate:Nnnn {NVxx, NVVV, NVnn}
\cs_new_protected_nopar:Npn \spath_gtranslate:Nnn #1#2#3
{
  \spath_gtranslate:NVnn #1#1{#2}{#3}
}
\cs_generate_variant:Nn \spath_gtranslate:Nnn {NVV, cnn, cVV}
```

This variant allows for passing the coordinates as a single braced group as it strips off the outer braces of the second argument.

```
\cs_new_protected_nopar:Npn \spath_translate:Nn #1#2
{
  \spath_translate:Nnn #1 #2
}
\cs_generate_variant:Nn \spath_translate:Nn {NV}
\cs_new_protected_nopar:Npn \spath_gtranslate:Nn #1#2
{
  \spath_gtranslate:Nnn #1 #2
}
\cs_generate_variant:Nn \spath_gtranslate:Nn {NV}
```

(*End definition for* `\spath_translate:Nnnn` *and others. These functions are documented on page* **??**.)

`\spath_scale:Nnnn`
`\spath_scale:Nnn`
`\spath_gscale:Nnnn`
`\spath_gscale:Nnn`

Scale a path.

```
\cs_new_protected_nopar:Npn \__spath_scale:nnn #1#2#3
{
  \group_begin:
  \tl_set:Nn \l__spath_tmpa_tl {#1}
  \tl_clear:N \l__spath_tmpb_tl
  \bool_until_do:nn {
    \tl_if_empty_p:N \l__spath_tmpa_tl
  }
  {
    \tl_put_right:Nx \l__spath_tmpb_tl {\tl_head:N \l__spath_tmpa_tl}
    \tl_set:Nx \l__spath_tmpa_tl {\tl_tail:N \l__spath_tmpa_tl}

    \fp_set:Nn \l__spath_tmpa_fp {\tl_head:N \l__spath_tmpa_tl * #2}
    \tl_set:Nx \l__spath_tmpa_tl {\tl_tail:N \l__spath_tmpa_tl}

    \fp_set:Nn \l__spath_tmpb_fp {\tl_head:N \l__spath_tmpa_tl * #3}
    \tl_set:Nx \l__spath_tmpa_tl {\tl_tail:N \l__spath_tmpa_tl}

    \tl_put_right:Nx \l__spath_tmpb_tl
    {
      {\fp_to_dim:N \l__spath_tmpa_fp}
      {\fp_to_dim:N \l__spath_tmpb_fp}
    }
  }
  \tl_gset_eq:NN \g__spath_output_tl \l__spath_tmpb_tl
  \group_end:
}
\cs_new_protected_nopar:Npn \spath_scale:Nnnn #1#2#3#4
{
  \__spath_scale:nnn {#2}{#3}{#4}
  \tl_set_eq:NN #1 \g__spath_output_tl
  \tl_gclear:N \g__spath_output_tl
}
\cs_generate_variant:Nn \spath_scale:Nnnn {NVnn, Nnxx}
\cs_new_protected_nopar:Npn \spath_scale:Nnn #1#2#3
{
  \spath_scale:NVnn #1#1{#2}{#3}
}
\cs_generate_variant:Nn \spath_scale:Nnn {cnn, cVV, NVV}
\cs_new_protected_nopar:Npn \spath_gscale:Nnnn #1#2#3#4
{
  \__spath_scale:nnn {#2}{#3}{#4}
  \tl_gset_eq:NN #1 \g__spath_output_tl
  \tl_gclear:N \g__spath_output_tl
}
\cs_generate_variant:Nn \spath_gscale:Nnnn {NVnn, Nnxx}
\cs_new_protected_nopar:Npn \spath_gscale:Nnn #1#2#3
{
  \spath_gscale:NVnn #1#1{#2}{#3}
}
\cs_generate_variant:Nn \spath_gscale:Nnn {cnn, cVV, NVV}
```

This variant allows for passing the coordinates as a single braced group as it strips off the outer braces of the second argument.

```
\cs_new_protected_nopar:Npn \spath_scale:Nn #1#2
{
  \spath_scale:Nnn #1 #2
}

\cs_generate_variant:Nn \spath_scale:Nn {NV}
\cs_new_protected_nopar:Npn \spath_gscale:Nn #1#2
{
  \spath_gscale:Nnn #1 #2
}

\cs_generate_variant:Nn \spath_gscale:Nn {NV}
```

(*End definition for* `\spath_scale:Nnnn` *and others. These functions are documented on page* **??**.)

`\spath_transform:Nnnnnnnn`
`\spath_transform:Nnnnnnn`
`\spath_gtransform:Nnnnnnnn`
`\spath_gtransform:Nnnnnnn`

Applies an affine (matrix and vector) transformation to path. The matrix is specified in rows first.

```
\cs_new_protected_nopar:Npn \__spath_transform:nnnnnnn #1#2#3#4#5#6#7
{
  \group_begin:
  \tl_set:Nn \l__spath_tmpa_tl {#1}
  \tl_clear:N \l__spath_tmpb_tl
  \bool_until_do:nn {
    \tl_if_empty_p:N \l__spath_tmpa_tl
  }
  {
    \tl_put_right:Nx \l__spath_tmpb_tl {\tl_head:N \l__spath_tmpa_tl}
    \tl_set:Nx \l__spath_tmpa_tl {\tl_tail:N \l__spath_tmpa_tl}
    \tl_set:Nx \l__spath_tmpc_tl {\tl_head:N \l__spath_tmpa_tl}
    \tl_set:Nx \l__spath_tmpa_tl {\tl_tail:N \l__spath_tmpa_tl}
    \tl_set:Nx \l__spath_tmpd_tl {\tl_head:N \l__spath_tmpa_tl}
    \tl_set:Nx \l__spath_tmpa_tl {\tl_tail:N \l__spath_tmpa_tl}

    \fp_set:Nn \l__spath_tmpa_fp {\l__spath_tmpc_tl * #2 + \l__spath_tmpd_tl * #4 + #6}
    \fp_set:Nn \l__spath_tmpb_fp {\l__spath_tmpc_tl * #3 + \l__spath_tmpd_tl * #5 + #7}
    \tl_put_right:Nx \l__spath_tmpb_tl
    {
      {\fp_to_dim:N \l__spath_tmpa_fp}
      {\fp_to_dim:N \l__spath_tmpb_fp}
    }
  }

  \tl_gset_eq:NN \g__spath_output_tl \l__spath_tmpb_tl
  \group_end:
}
\cs_new_protected_nopar:Npn \spath_transform:Nnnnnnnn #1#2#3#4#5#6#7#8
{
  \__spath_transform:nnnnnnn {#2}{#3}{#4}{#5}{#6}{#7}{#8}
  \tl_set_eq:NN #1 \g__spath_output_tl
  \tl_gclear:N \g__spath_output_tl
}
\cs_generate_variant:Nn \spath_transform:Nnnnnnnn {NVnnnnnn, Nnxxxxxx, cnnnnnnn}
\cs_new_protected_nopar:Npn \spath_transform:Nnnnnnn #1#2#3#4#5#6#7
{
  \spath_transform:NVnnnnnn #1#1{#2}{#3}{#4}{#5}{#6}{#7}
}
\cs_generate_variant:Nn \spath_transform:Nnnnnnn {cnnnnnn}
\cs_new_protected_nopar:Npn \spath_transform:Nnn #1#2#3
{
  \spath_transform:Nnnnnnnn #1{#2}#3
}
\cs_generate_variant:Nn \spath_transform:Nnn {cnn, cVn, NVn, NnV}
\cs_new_protected_nopar:Npn \spath_transform:Nn #1#2
{
  \spath_transform:NVnnnnnn #1#1#2
}
\cs_generate_variant:Nn \spath_transform:Nn {cn, cV, NV}

\cs_new_protected_nopar:Npn \spath_gtransform:Nnnnnnnn #1#2#3#4#5#6#7#8
{
  \__spath_transform:nnnnnnn {#2}{#3}{#4}{#5}{#6}{#7}{#8}
  \tl_gset_eq:NN #1 \g__spath_output_tl
  \tl_gclear:N \g__spath_output_tl
}
\cs_generate_variant:Nn \spath_gtransform:Nnnnnnnn {NVnnnnnn, Nnxxxxxx, cnnnnnnn}
\cs_new_protected_nopar:Npn \spath_gtransform:Nnnnnnn #1#2#3#4#5#6#7
{
  \spath_gtransform:NVnnnnnn #1#1{#2}{#3}{#4}{#5}{#6}{#7}
```

```
}
\cs_generate_variant:Nn \spath_gtransform:Nnnnnnn {cnnnnnn}
\cs_new_protected_nopar:Npn \spath_gtransform:Nnn #1#2#3
{
  \spath_gtransform:Nnnnnnnn #1{#2}#3
}
\cs_generate_variant:Nn \spath_gtransform:Nnn {cnn, cVn, NVn, NnV}
\cs_new_protected_nopar:Npn \spath_gtransform:Nn #1#2
{
  \spath_gtransform:NVnnnnnn #1#1#2
}
\cs_generate_variant:Nn \spath_gtransform:Nn {cn, cV, NV}
```

(*End definition for* `\spath_transform:Nnnnnnnn` *and others. These functions are documented on page* **??**.)

`\spath_weld:Nnn`
`\spath_weld:Nn`
`\spath_gweld:Nnn`
`\spath_gweld:Nn`

This welds one path to another, moving the second so that its initial point coincides with the first's final point. It is called a *weld* because the initial move of the second path is removed.

```
\cs_new_protected_nopar:Npn \__spath_weld:nn #1#2
{
  \group_begin:
  \tl_set:Nn \l__spath_tmpa_tl {#1}
  \tl_set:Nn \l__spath_tmpb_tl {#2}

  \spath_finalpoint:NV \l__spath_tmpc_tl \l__spath_tmpa_tl
  \spath_initialpoint:NV \l__spath_tmpd_tl \l__spath_tmpb_tl

  \dim_set:Nn \l__spath_tmpa_dim
  {
    \tl_item:Nn \l__spath_tmpc_tl {1}
    -
    \tl_item:Nn \l__spath_tmpd_tl {1}
  }
  \dim_set:Nn \l__spath_tmpb_dim
  {
    \tl_item:Nn \l__spath_tmpc_tl {2}
    -
    \tl_item:Nn \l__spath_tmpd_tl {2}
  }

  \spath_translate:NVV \l__spath_tmpb_tl \l__spath_tmpa_dim \l__spath_tmpb_dim

  \prg_replicate:nn {3}
  {
    \tl_set:Nx \l__spath_tmpb_tl {\tl_tail:N \l__spath_tmpb_tl}
  }

  \tl_put_right:NV \l__spath_tmpa_tl \l__spath_tmpb_tl
  \tl_gset_eq:NN \g__spath_output_tl \l__spath_tmpa_tl
  \group_end:
}
\cs_new_protected_nopar:Npn \spath_weld:Nnn #1#2#3
{
  \__spath_weld:nn {#2}{#3}
  \tl_set_eq:NN #1 \g__spath_output_tl
  \tl_gclear:N \g__spath_output_tl
}
\cs_generate_variant:Nn \spath_weld:Nnn {NVV,NVn}
\cs_new_protected_nopar:Npn \spath_weld:Nn #1#2
{
  \spath_weld:NVn #1#1{#2}
}
\cs_generate_variant:Nn \spath_weld:Nn {NV, Nv, cV, cv}
\cs_new_protected_nopar:Npn \spath_gweld:Nnn #1#2#3
{
  \__spath_weld:nn {#2}{#3}
  \tl_gset_eq:NN #1 \g__spath_output_tl
  \tl_gclear:N \g__spath_output_tl
}
\cs_generate_variant:Nn \spath_gweld:Nnn {NVV, NVn}
\cs_new_protected_nopar:Npn \spath_gweld:Nn #1#2
{
  \spath_gweld:NVn #1#1{#2}
}
\cs_generate_variant:Nn \spath_gweld:Nn {NV, Nv, cV, cv}
```

(*End definition for* `\spath_weld:Nnn` *and others. These functions are documented on page* **??**.)

`\spath_append_no_move:Nnn`
`\spath_append_no_move:Nn`
`\spath_gappend_no_move:Nnn`
`\spath_gappend_no_move:Nn`

Append the path from the second `spath` to the first, removing the adjoining move.

```
\cs_new_protected_nopar:Npn \__spath_append_no_move:nn #1#2
{
  \group_begin:
  \tl_set:Nn \l__spath_tmpa_tl {#1}
  \tl_set:Nn \l__spath_tmpb_tl {#2}
  \tl_set:Nx \l__spath_tmpb_tl {\tl_tail:N \l__spath_tmpb_tl}
  \tl_set:Nx \l__spath_tmpb_tl {\tl_tail:N \l__spath_tmpb_tl}
  \tl_set:Nx \l__spath_tmpb_tl {\tl_tail:N \l__spath_tmpb_tl}

  \tl_put_right:NV \l__spath_tmpa_tl \l__spath_tmpb_tl
  \tl_gset_eq:NN \g__spath_output_tl \l__spath_tmpa_tl
  \group_end:
}
\cs_new_protected_nopar:Npn \spath_append_no_move:Nnn #1#2#3
{
  \__spath_append_no_move:nn {#2}{#3}
  \tl_set_eq:NN #1 \g__spath_output_tl
  \tl_gclear:N \g__spath_output_tl
}
\cs_generate_variant:Nn \spath_append_no_move:Nnn {NVV, NVn}
\cs_new_protected_nopar:Npn \spath_append_no_move:Nn #1#2
{
  \spath_append_no_move:NVn #1#1{#2}
}
\cs_generate_variant:Nn \spath_append_no_move:Nn {NV, cv}
\cs_new_protected_nopar:Npn \spath_gappend_no_move:Nnn #1#2#3
{
  \__spath_append_no_move:nn {#2}{#3}
  \tl_gset_eq:NN #1 \g__spath_output_tl
  \tl_gclear:N \g__spath_output_tl
}
\cs_generate_variant:Nn \spath_gappend_no_move:Nnn {NVV, NVn}
\cs_new_protected_nopar:Npn \spath_gappend_no_move:Nn #1#2
{
  \spath_gappend_no_move:NVn #1#1{#2}
}
\cs_generate_variant:Nn \spath_gappend_no_move:Nn {NV, cv}
```

(*End definition for* `\spath_append_no_move:Nnn` *and others. These functions are documented on page* **??**.)

`\spath_append:Nnn`
`\spath_append:Nn`
`\spath_gappend:Nnn`
`\spath_gappend:Nn`

Prepend the path from the second `spath` to the first.

```
\cs_new_protected_nopar:Npn \spath_append:Nnn #1#2#3
{
  \tl_set:Nn #1 {#2}
  \tl_put_right:Nn #1 {#3}
}
\cs_generate_variant:Nn \spath_append:Nnn {NVV, NVn}
\cs_new_protected_nopar:Npn \spath_append:Nn #1#2
{
  \spath_append:NVn #1#1{#2}
}
\cs_generate_variant:Nn \spath_append:Nn {NV, Nv}
\cs_new_protected_nopar:Npn \spath_gappend:Nnn #1#2#3
{
  \tl_gset:Nn #1 {#2}
  \tl_gput_right:Nn #1 {#3}
}
\cs_generate_variant:Nn \spath_gappend:Nnn {NVV, NVn}
\cs_new_protected_nopar:Npn \spath_gappend:Nn #1#2
{
  \spath_gappend:NVn #1#1{#2}
}
\cs_generate_variant:Nn \spath_gappend:Nn {NV, Nv}
```

(*End definition for* `\spath_append:Nnn` *and others. These functions are documented on page* **??**.)

`\spath_prepend_no_move:Nnn`
`\spath_prepend_no_move:Nn`
`\spath_gprepend_no_move:Nnn`
`\spath_gprepend_no_move:Nn`

Prepend the path from the second `spath` to the first, removing the adjoining move.

```
\cs_new_protected_nopar:Npn \spath_prepend_no_move:Nnn #1#2#3
{
  \spath_append_no_move:Nnn #1{#3}{#2}
}
\cs_generate_variant:Nn \spath_prepend_no_move:Nnn {NVV, NVn}
\cs_new_protected_nopar:Npn \spath_prepend_no_move:Nn #1#2
{
  \spath_prepend_no_move:NVn #1#1{#2}
}
\cs_generate_variant:Nn \spath_prepend_no_move:Nn {NV, cv}
\cs_new_protected_nopar:Npn \spath_gprepend_no_move:Nnn #1#2#3
{
  \spath_gappend_no_move:Nnn #1{#3}{#2}
}
\cs_generate_variant:Nn \spath_gprepend_no_move:Nnn {NVV, NVn}
```

```
\cs_new_protected_nopar:Npn \spath_gprepend_no_move:Nn #1#2
{
  \spath_gprepend_no_move:NVn #1#1{#2}
}
\cs_generate_variant:Nn \spath_gprepend_no_move:Nn {NV, cv}
```

(*End definition for* `\spath_prepend_no_move:Nnn` *and others. These functions are documented on page* **??**.)

`\spath_prepend:Nnn`
`\spath_prepend:Nn`
`\spath_gprepend:Nnn`
`\spath_gprepend:Nn`

Prepend the path from the second `spath` to the first.

```
\cs_new_protected_nopar:Npn \spath_prepend:Nnn #1#2#3
{
  \spath_append:Nnn #1{#3}{#2}
}
\cs_generate_variant:Nn \spath_prepend:Nnn {NVV, NVn}
\cs_new_protected_nopar:Npn \spath_prepend:Nn #1#2
{
  \spath_prepend:NVn #1#1{#2}
}
\cs_generate_variant:Nn \spath_prepend:Nn {NV}
\cs_new_protected_nopar:Npn \spath_gprepend:Nnn #1#2#3
{
  \spath_gappend:Nnn #1{#3}{#2}
}
\cs_generate_variant:Nn \spath_gprepend:Nnn {NVV, NVn}
\cs_new_protected_nopar:Npn \spath_gprepend:Nn #1#2
{
  \spath_gprepend:NVn #1#1{#2}
}
\cs_generate_variant:Nn \spath_gprepend:Nn {NV}
```

(*End definition for* `\spath_prepend:Nnn` *and others. These functions are documented on page* **??**.)

`\spath_bake_round:Nn`
`\spath_bake_round:N`
`\spath_gbake_round:Nn`
`\spath_gbake_round:N`

The corner rounding routine is applied quite late in the process of building a soft path so this ensures that it is done.

```
\cs_new_protected_nopar:Npn \__spath_bake_round:n #1
{
  \group_begin:
  \tl_set:Nn \l__spath_tmpa_tl {#1}
  \pgf@@processround \l__spath_tmpa_tl\l__spath_tmpb_tl
  \tl_gset_eq:NN \g__spath_output_tl \l__spath_tmpb_tl
  \group_end:
}
\cs_new_protected_nopar:Npn \spath_bake_round:Nn #1#2
{
  \__spath_bake_round:n {#2}
  \tl_set_eq:NN #1 \g__spath_output_tl
  \tl_gclear:N \g__spath_output_tl
}
\cs_generate_variant:Nn \spath_bake_round:Nn {NV}
\cs_new_protected_nopar:Npn \spath_bake_round:N #1
{
  \spath_bake_round:NV #1#1
}
```

```
\cs_generate_variant:Nn \spath_bake_round:N {c}
\cs_new_protected_nopar:Npn \spath_gbake_round:Nn #1#2
{
  \__spath_bake_round:n {#2}
  \tl_gset_eq:NN #1 \g__spath_output_tl
  \tl_gclear:N \g__spath_output_tl
}
\cs_generate_variant:Nn \spath_gbake_round:Nn {NV}
\cs_new_protected_nopar:Npn \spath_gbake_round:N #1
{
  \spath_gbake_round:NV #1#1
}
\cs_generate_variant:Nn \spath_gbake_round:N {c}
```

(*End definition for* `\spath_bake_round:Nn` *and others. These functions are documented on page* **??**.)

`\spath_close:Nn`
`\spath_close:N`
`\spath_gclose:Nn`
`\spath_gclose:N`

Appends a close path to the end of the path. For now, the point is the initial or final point (respectively). To be future proof, it ought to be the point of the adjacent move to.

```
\cs_new_protected_nopar:Npn \__spath_close:n #1
{
  \group_begin:
  \tl_set:Nn \l__spath_tmpa_tl {#1}
  \spath_initialpoint:NV \l__spath_tmpb_tl \l__spath_tmpa_tl
  \tl_put_right:NV \l__spath_tmpa_tl \c_spath_closepath_tl
  \tl_put_right:NV \l__spath_tmpa_tl \l__spath_tmpb_tl
  \tl_gset_eq:NN \g__spath_output_tl \l__spath_tmpa_tl
  \group_end:
}
\cs_new_protected_nopar:Npn \spath_close:Nn #1#2
{
  \__spath_close:n {#2}
  \tl_set_eq:NN #1 \g__spath_output_tl
  \tl_gclear:N \g__spath_output_tl
}
\cs_generate_variant:Nn \spath_close:Nn {NV}
\cs_new_protected_nopar:Npn \spath_close:N #1
{
  \spath_close:NV #1#1
}
\cs_generate_variant:Nn \spath_close:N {c}
\cs_new_protected_nopar:Npn \spath_gclose:Nn #1#2
{
  \__spath_close:n {#2}
  \tl_gset_eq:NN #1 \g__spath_output_tl
  \tl_gclear:N \g__spath_output_tl
}
\cs_generate_variant:Nn \spath_gclose:Nn {NV}
\cs_new_protected_nopar:Npn \spath_gclose:N #1
{
  \spath_gclose:NV #1#1
}
\cs_generate_variant:Nn \spath_gclose:N {c}
```

(*End definition for* `\spath_close:Nn` *and others. These functions are documented on page* **??**.)

`\spath_open:Nn`
`\spath_open:N`
`\spath_gopen:Nn`
`\spath_gopen:N`

Removes all close paths from the path, replacing them by `lineto` if they move any distance.

```
\cs_new_protected_nopar:Npn \__spath_open:n #1
{
  \group_begin:
  \tl_set:Nn \l__spath_tmpa_tl {#1}
  \tl_clear:N \l__spath_tmpb_tl
  \bool_until_do:nn {
    \tl_if_empty_p:N \l__spath_tmpa_tl
  }
  {
    \tl_set:Nx \l__spath_tmpc_tl {\tl_head:N \l__spath_tmpa_tl}
    \tl_set:Nx \l__spath_tmpa_tl {\tl_tail:N \l__spath_tmpa_tl}

    \tl_case:NnF \l__spath_tmpc_tl
    {
      \c_spath_closepath_tl {

        \bool_if:nF
        {
          \dim_compare_p:n
          {
            \l__spath_move_x_dim == \l__spath_tmpa_dim
          }
          &&
          \dim_compare_p:n
          {
            \l__spath_move_y_dim == \l__spath_tmpb_dim
          }
        }
        {
          \tl_put_right:NV \l__spath_tmpb_tl \c_spath_lineto_tl

          \tl_put_right:Nx \l__spath_tmpb_tl {
            { \dim_use:N \l__spath_move_x_dim }
            { \dim_use:N \l__spath_move_y_dim }
          }
        }

        \dim_set:Nn \l__spath_tmpa_dim {\tl_head:N \l__spath_tmpa_tl}
        \tl_set:Nx \l__spath_tmpa_tl {\tl_tail:N \l__spath_tmpa_tl}
        \dim_set:Nn \l__spath_tmpb_dim {\tl_head:N \l__spath_tmpa_tl}
        \tl_set:Nx \l__spath_tmpa_tl {\tl_tail:N \l__spath_tmpa_tl}
      }

      \c_spath_moveto_tl {
        \tl_put_right:NV \l__spath_tmpb_tl \l__spath_tmpc_tl

        \dim_set:Nn \l__spath_move_x_dim {\tl_head:N \l__spath_tmpa_tl}
        \tl_set:Nx \l__spath_tmpa_tl {\tl_tail:N \l__spath_tmpa_tl}
        \dim_set:Nn \l__spath_move_y_dim {\tl_head:N \l__spath_tmpa_tl}
        \tl_set:Nx \l__spath_tmpa_tl {\tl_tail:N \l__spath_tmpa_tl}

        \tl_put_right:Nx \l__spath_tmpb_tl {
          { \dim_use:N \l__spath_move_x_dim }
          { \dim_use:N \l__spath_move_y_dim }
        }

        \dim_set_eq:NN \l__spath_tmpa_dim \l__spath_move_x_dim
        \dim_set_eq:NN \l__spath_tmpb_dim \l__spath_move_y_dim
      }
    }
    {
      \tl_put_right:NV \l__spath_tmpb_tl \l__spath_tmpc_tl

      \dim_set:Nn \l__spath_tmpa_dim {\tl_head:N \l__spath_tmpa_tl}
      \tl_set:Nx \l__spath_tmpa_tl {\tl_tail:N \l__spath_tmpa_tl}
      \dim_set:Nn \l__spath_tmpb_dim {\tl_head:N \l__spath_tmpa_tl}
      \tl_set:Nx \l__spath_tmpa_tl {\tl_tail:N \l__spath_tmpa_tl}

      \tl_put_right:Nx \l__spath_tmpb_tl {
        { \dim_use:N \l__spath_tmpa_dim }
        { \dim_use:N \l__spath_tmpb_dim }
      }
    }
  }
  \tl_gset_eq:NN \g__spath_output_tl \l__spath_tmpb_tl
  \group_end:
}
\cs_new_protected_nopar:Npn \spath_open:Nn #1#2
{
  \__spath_open:n {#2}
  \tl_set_eq:NN #1 \g__spath_output_tl
  \tl_gclear:N \g__spath_output_tl
}
\cs_generate_variant:Nn \spath_open:Nn {NV}
\cs_new_protected_nopar:Npn \spath_open:N #1
{
  \spath_open:NV #1#1
}
\cs_new_protected_nopar:Npn \spath_gopen:Nn #1#2
{
  \__spath_open:n {#2}
  \tl_gset_eq:NN #1 \g__spath_output_tl
  \tl_gclear:N \g__spath_output_tl
}
\cs_generate_variant:Nn \spath_gopen:Nn {NV}
\cs_new_protected_nopar:Npn \spath_gopen:N #1
{
  \spath_gopen:NV #1#1
}
```

(*End definition for* `\spath_open:Nn` *and others. These functions are documented on page* **??**.)

`\spath_remove_empty_components:Nn`
`\spath_remove_empty_components:N`
`\spath_gremove_empty_components:Nn`
`\spath_gremove_empty_components:N`

Remove any component that is simply a moveto.

```
\cs_new_protected_nopar:Npn \__spath_remove_empty_components:n #1
{
  \group_begin:
```

```
  \spath_components_to_seq:Nn \l__spath_tmpa_seq {#1}
  \tl_clear:N \l__spath_tmpa_tl
  \seq_map_inline:Nn \l__spath_tmpa_seq
  {
    \int_compare:nF
    {
      \tl_count:n {##1} == 3
    }
    {
      \tl_put_right:Nn \l__spath_tmpa_tl {##1}
    }
  }
  \tl_gset_eq:NN \g__spath_output_tl \l__spath_tmpa_tl
  \group_end:
}
\cs_new_protected_nopar:Npn \spath_remove_empty_components:Nn #1#2
{
  \__spath_remove_empty_components:n {#2}
  \tl_set_eq:NN #1 \g__spath_output_tl
  \tl_gclear:N \g__spath_output_tl
}
\cs_generate_variant:Nn \spath_remove_empty_components:Nn {NV}
\cs_new_protected_nopar:Npn \spath_remove_empty_components:N #1
{
  \spath_remove_empty_components:NV #1#1
}
\cs_generate_variant:Nn \spath_remove_empty_components:N {c}
\cs_new_protected_nopar:Npn \spath_gremove_empty_components:Nn #1#2
{
  \__spath_remove_empty_components:n {#2}
  \tl_gset_eq:NN #1 \g__spath_output_tl
  \tl_gclear:N \g__spath_output_tl
}
\cs_generate_variant:Nn \spath_gremove_empty_components:Nn {NV}
\cs_new_protected_nopar:Npn \spath_gremove_empty_components:N #1
{
  \spath_gremove_empty_components:NV #1#1
}
\cs_generate_variant:Nn \spath_gremove_empty_components:N {c}
```

(*End definition for* `\spath_remove_empty_components:Nn` *and others. These functions are documented on page* **??**.)

`\spath_if_eq:nn` Test if two soft paths are equal, we allow a little tolerance on the calculations.

```
\prg_new_protected_conditional:Npnn \spath_if_eq:nn #1#2 { T, F, TF }
{
  \group_begin:
  \tl_set:Nn \l__spath_tmpa_tl {#1}
  \tl_set:Nn \l__spath_tmpb_tl {#2}
  \bool_gset_true:N \g__spath_tmpa_bool
  \int_compare:nNnTF {\tl_count:N \l__spath_tmpa_tl} = {\tl_count:N \l__spath_tmpb_tl}
  {
    \int_step_inline:nnnn {1} {3} {\tl_count:N \l__spath_tmpa_tl}
    {
      \tl_set:Nx \l__spath_tmpc_tl {\tl_item:Nn \l__spath_tmpa_tl {##1}}
      \tl_set:Nx \l__spath_tmpd_tl {\tl_item:Nn \l__spath_tmpb_tl {##1}}
      \tl_if_eq:NNF \l__spath_tmpc_tl \l__spath_tmpd_tl
      {
        \bool_gset_false:N \g__spath_tmpa_bool
      }
      \dim_set:Nn \l__spath_tmpa_dim {\tl_item:Nn \l__spath_tmpa_tl {##1+1}}
      \dim_set:Nn \l__spath_tmpb_dim {\tl_item:Nn \l__spath_tmpb_tl {##1+1}}
      \dim_compare:nF { \dim_abs:n { \l__spath_tmpa_dim - \l__spath_tmpb_dim} < 0.001pt }
      {
        \bool_gset_false:N \g__spath_tmpa_bool
      }
      \dim_set:Nn \l__spath_tmpa_dim {\tl_item:Nn \l__spath_tmpa_tl {##1+2}}
      \dim_set:Nn \l__spath_tmpb_dim {\tl_item:Nn \l__spath_tmpb_tl {##1+2}}
      \dim_compare:nF { \dim_abs:n { \l__spath_tmpa_dim - \l__spath_tmpb_dim} < 0.001pt }
      {
        \bool_gset_false:N \g__spath_tmpa_bool
      }
    }
  }
  {
    \bool_gset_false:N \g__spath_tmpa_bool
  }
  \group_end:
  \bool_if:NTF \g__spath_tmpa_bool
  {
    \prg_return_true:
  }
  {
    \prg_return_false:
  }
}
\prg_generate_conditional_variant:Nnn \spath_if_eq:nn {VV, Vn, nV, vv} {TF, T, F}
```

(*End definition for* `\spath_if_eq:nn`*. This function is documented on page* **??**.)

3.4 Splitting Commands

`\spath_split_curve:NNnn`
`\spath_gsplit_curve:NNnn`

Splits a Bezier cubic into pieces, storing the pieces in the first two arguments.

```
\cs_new_protected_nopar:Npn \__spath_split_curve:nn #1#2
{
  \group_begin:
  \tl_set_eq:NN \l__spath_tmpa_tl \c_spath_moveto_tl
  \tl_put_right:Nx \l__spath_tmpa_tl {
    {\tl_item:nn {#1} {2}}
    {\tl_item:nn {#1} {3}}
  }
  \tl_put_right:NV \l__spath_tmpa_tl \c_spath_curvetoa_tl
  \tl_put_right:Nx \l__spath_tmpa_tl
  {
    {\fp_to_dim:n
    {
      (1 - #2) * \tl_item:nn {#1} {2} + (#2) * \tl_item:nn {#1} {5}
    }}
    {\fp_to_dim:n
    {
      (1 - #2) * \tl_item:nn {#1} {3} + (#2) * \tl_item:nn {#1} {6}
    }}
  }

  \tl_put_right:NV \l__spath_tmpa_tl \c_spath_curvetob_tl
  \tl_put_right:Nx \l__spath_tmpa_tl
  {
    {\fp_to_dim:n
    {
      (1 - #2)^2 * \tl_item:nn {#1} {2} + 2 * (1 - #2) * (#2) * \tl_item:nn {#1} {5} + (#2)^
    }}
    {\fp_to_dim:n
    {
      (1 - #2)^2 * \tl_item:nn {#1} {3} + 2 * (1 - #2) * (#2) * \tl_item:nn {#1} {6} + (#2)^
    }}
  }

  \tl_put_right:NV \l__spath_tmpa_tl \c_spath_curveto_tl
  \tl_put_right:Nx \l__spath_tmpa_tl
  {
    {\fp_to_dim:n
      {
      (1 - #2)^3 * \tl_item:nn {#1} {2} + 3 * (1 - #2)^2 * (#2) * \tl_item:nn {#1} {5} + 3 *
    }}
    {\fp_to_dim:n
    {
      (1 - #2)^3 * \tl_item:nn {#1} {3} + 3 * (1 - #2)^2 * (#2) * \tl_item:nn {#1} {6} + 3 *
    }}
  }

  \tl_gclear:N \g__spath_output_tl
  \__spath_tl_gput_right_braced:NV \g__spath_output_tl \l__spath_tmpa_tl

  \tl_clear:N \l__spath_tmpa_tl
  \tl_set_eq:NN \l__spath_tmpa_tl \c_spath_moveto_tl
  \tl_put_right:Nx \l__spath_tmpa_tl
  {
    {\fp_to_dim:n
      {
      (1 - #2)^3 * \tl_item:nn {#1} {2} + 3 * (1 - #2)^2 * (#2) * \tl_item:nn {#1} {5} + 3 *
    }}
    {\fp_to_dim:n
    {
      (1 - #2)^3 * \tl_item:nn {#1} {3} + 3 * (1 - #2)^2 * (#2) * \tl_item:nn {#1} {6} + 3 *
    }}
  }

  \tl_put_right:NV \l__spath_tmpa_tl \c_spath_curvetoa_tl
  \tl_put_right:Nx \l__spath_tmpa_tl
  {
    {\fp_to_dim:n
    {
      (1 - #2)^2 * \tl_item:nn {#1} {5} + 2 * (1 - #2) * (#2) * \tl_item:nn {#1} {8} + (#2)^
    }}
    {\fp_to_dim:n
    {
      (1 - #2)^2 * \tl_item:nn {#1} {6} + 2 * (1 - #2) * (#2) * \tl_item:nn {#1} {9} + (#2)^
    }}
  }
  \tl_put_right:NV \l__spath_tmpa_tl \c_spath_curvetob_tl
  \tl_put_right:Nx \l__spath_tmpa_tl
  {
    {\fp_to_dim:n
    {
      (1 - #2) * \tl_item:nn {#1} {8} + (#2) * \tl_item:nn {#1} {11}
    }}
    {\fp_to_dim:n
    {
      (1 - #2) * \tl_item:nn {#1} {9} + (#2) * \tl_item:nn {#1} {12}
    }}
  }
  \tl_put_right:NV \l__spath_tmpa_tl \c_spath_curveto_tl
  \tl_put_right:Nx \l__spath_tmpa_tl {
    {\tl_item:nn {#1} {11}}
    {\tl_item:nn {#1} {12}}
  }

  \__spath_tl_gput_right_braced:NV \g__spath_output_tl \l__spath_tmpa_tl
  \group_end:
}
\cs_new_protected_nopar:Npn \spath_split_curve:NNnn #1#2#3#4
{
  \__spath_split_curve:nn {#3}{#4}
  \tl_set:Nx #1 {\tl_item:Nn \g__spath_output_tl {1}}
  \tl_set:Nx #2 {\tl_item:Nn \g__spath_output_tl {2}}
  \tl_gclear:N \g__spath_output_tl
}
\cs_generate_variant:Nn \spath_split_curve:NNnn {NNnV, NNVn, NNVV}
\cs_new_protected_nopar:Npn \spath_gsplit_curve:NNnn #1#2#3#4
{
  \__spath_split_curve:nn {#3}{#4}
  \tl_gset:Nx #1 {\tl_item:Nn \g__spath_output_tl {1}}
  \tl_gset:Nx #2 {\tl_item:Nn \g__spath_output_tl {2}}
  \tl_gclear:N \g__spath_output_tl
}
\cs_generate_variant:Nn \spath_gsplit_curve:NNnn {NNnV, NNVn, NNVV}
```

(*End definition for* `\spath_split_curve:NNnn` *and* `\spath_gsplit_curve:NNnn`*. These functions are documented on page* **??**.)

`\spath_split_line:NNnn`
`\spath_gsplit_line:NNnn`

Splits a line segment.

```
\cs_new_protected_nopar:Npn \__spath_split_line:nn #1#2
{
  \group_begin:
  \tl_set_eq:NN \l__spath_tmpa_tl \c_spath_moveto_tl
  \tl_put_right:Nx \l__spath_tmpa_tl {
    {\tl_item:nn {#1} {2}}
    {\tl_item:nn {#1} {3}}
  }
  \tl_put_right:NV \l__spath_tmpa_tl \c_spath_lineto_tl
  \tl_put_right:Nx \l__spath_tmpa_tl
  {
    {\fp_to_dim:n
    {
      (1 - #2) * \tl_item:nn {#1} {2} + (#2) * \tl_item:nn {#1} {5}
    }}
    {\fp_to_dim:n
    {
      (1 - #2) * \tl_item:nn {#1} {3} + (#2) * \tl_item:nn {#1} {6}
    }}
  }
  \tl_gclear:N \g__spath_output_tl
  \__spath_tl_gput_right_braced:NV \g__spath_output_tl \l__spath_tmpa_tl

  \tl_clear:N \l__spath_tmpa_tl
  \tl_set_eq:NN \l__spath_tmpa_tl \c_spath_moveto_tl
  \tl_put_right:Nx \l__spath_tmpa_tl
  {
    {\fp_to_dim:n
    {
      (1 - #2) * \tl_item:nn {#1} {2} + (#2) * \tl_item:nn {#1} {5}
    }}
    {\fp_to_dim:n
    {
      (1 - #2) * \tl_item:nn {#1} {3} + (#2) * \tl_item:nn {#1} {6}
    }}
  }
  \tl_put_right:NV \l__spath_tmpa_tl \c_spath_lineto_tl
  \tl_put_right:Nx \l__spath_tmpa_tl {
    {\tl_item:nn {#1} {5}}
    {\tl_item:nn {#1} {6}}
  }

  \__spath_tl_gput_right_braced:NV \g__spath_output_tl \l__spath_tmpa_tl
  \group_end:
}
\cs_new_protected_nopar:Npn \spath_split_line:NNnn #1#2#3#4
{
  \__spath_split_line:nn {#3}{#4}
  \tl_set:Nx #1 {\tl_item:Nn \g__spath_output_tl {1}}
  \tl_set:Nx #2 {\tl_item:Nn \g__spath_output_tl {2}}
  \tl_gclear:N \g__spath_output_tl
}
\cs_generate_variant:Nn \spath_split_line:NNnn {NNnV, NNVn, NNVV}
\cs_new_protected_nopar:Npn \spath_gsplit_line:NNnn #1#2#3#4
{
  \__spath_split_line:nn {#3}{#4}
  \tl_gset:Nx #1 {\tl_item:Nn \g__spath_output_tl {1}}
  \tl_gset:Nx #2 {\tl_item:Nn \g__spath_output_tl {2}}
  \tl_gclear:N \g__spath_output_tl
```

```
}
\cs_generate_variant:Nn \spath_gsplit_line:NNnn {NNnV, NNVn, NNVV}
```

(*End definition for* `\spath_split_line:NNnn` *and* `\spath_gsplit_line:NNnn`*. These functions are documented on page* **??**.)

```
\int_new:N \l__spath_split_int
\int_new:N \l__spath_splitat_int
\fp_new:N \l__spath_split_fp
\bool_new:N \l__spath_split_bool
\tl_new:N \l__spath_split_path_tl
\tl_new:N \l__spath_split_patha_tl
\tl_new:N \l__spath_split_pathb_tl
\tl_new:N \l__spath_split_intoa_tl
\tl_new:N \l__spath_split_intob_tl
\dim_new:N \l__spath_splitx_dim
\dim_new:N \l__spath_splity_dim
```

`\spath_split_at:NNnn`
`\spath_split_at:Nnn`
`\spath_split_at:Nn`
`\spath_gsplit_at:NNnn␣\spath_gsplit_at:Nnn`
`\spath_gsplit_at:Nn`

Split a path according to the parameter generated by the intersection routine. The versions with two `N` arguments stores the two parts in two macros, the version with a single `N` joins them back into a single path (as separate components).

```
\cs_new_protected_nopar:Npn \__spath_split_at:nn #1#2
{
  \group_begin:
  \int_set:Nn \l__spath_splitat_int {\fp_to_int:n {floor(#2) + 1}}
  \fp_set:Nn \l__spath_split_fp {#2 - floor(#2)}

  % Is split point near one end or other of a component?
  \fp_compare:nT
  {
    \l__spath_split_fp < 0.01
  }
  {
    % Near the start, so we'll place it at the start
    \fp_set:Nn \l__spath_split_fp {0}
  }
  \fp_compare:nT
  {
    \l__spath_split_fp > 0.99
  }
  {
    % Near the end, so we'll place it at the end
    \fp_set:Nn \l__spath_split_fp {0}
    \int_incr:N \l__spath_splitat_int
  }

  \int_zero:N \l__spath_split_int
  \bool_set_true:N \l__spath_split_bool

  \tl_set:Nn \l__spath_split_path_tl {#1}
  \tl_clear:N \l__spath_split_patha_tl

  \dim_zero:N \l__spath_splitx_dim
  \dim_zero:N \l__spath_splity_dim

```

```
\bool_until_do:nn {
  \tl_if_empty_p:N \l__spath_split_path_tl
  ||
  \int_compare_p:n { \l__spath_splitat_int == \l__spath_split_int  }
}
{
  \tl_set:Nx \l__spath_tmpc_tl {\tl_head:N \l__spath_split_path_tl}
  \tl_set:Nx \l__spath_split_path_tl {\tl_tail:N \l__spath_split_path_tl }
  \tl_case:Nn \l__spath_tmpc_tl
  {
    \c_spath_lineto_tl
    {
      \int_incr:N \l__spath_split_int
    }
    \c_spath_curvetoa_tl
    {
      \int_incr:N \l__spath_split_int
    }
  }
  \int_compare:nT { \l__spath_split_int < \l__spath_splitat_int  }
  {
    \tl_put_right:NV \l__spath_split_patha_tl \l__spath_tmpc_tl

    \tl_put_right:Nx \l__spath_split_patha_tl
    {{ \tl_head:N \l__spath_split_path_tl }}
    \dim_set:Nn \l__spath_splitx_dim {\tl_head:N \l__spath_split_path_tl}
    \tl_set:Nx \l__spath_split_path_tl {\tl_tail:N \l__spath_split_path_tl }

    \tl_put_right:Nx \l__spath_split_patha_tl
    {{ \tl_head:N \l__spath_split_path_tl }}
    \dim_set:Nn \l__spath_splity_dim {\tl_head:N \l__spath_split_path_tl}
    \tl_set:Nx \l__spath_split_path_tl {\tl_tail:N \l__spath_split_path_tl }

  }
}

\tl_clear:N \l__spath_split_pathb_tl
\tl_put_right:NV \l__spath_split_pathb_tl \c_spath_moveto_tl
\tl_put_right:Nx \l__spath_split_pathb_tl
{
  {\dim_use:N \l__spath_splitx_dim}
  {\dim_use:N \l__spath_splity_dim}
}

\fp_compare:nTF
{
  \l__spath_split_fp == 0
}
{
  \tl_set_eq:NN \l__spath_split_intob_tl \l__spath_split_pathb_tl
  \tl_if_empty:NF \l__spath_split_path_tl
  {
    \tl_put_right:NV \l__spath_split_intob_tl \l__spath_tmpc_tl
    \tl_put_right:NV \l__spath_split_intob_tl \l__spath_split_path_tl
  }
}
{

  \tl_case:Nn \l__spath_tmpc_tl
  {
    \c_spath_lineto_tl
    {
      \tl_put_right:NV \l__spath_split_pathb_tl \l__spath_tmpc_tl
      \tl_put_right:Nx \l__spath_split_pathb_tl
      {{ \tl_head:N \l__spath_split_path_tl }}
      \tl_set:Nx \l__spath_split_path_tl {\tl_tail:N \l__spath_split_path_tl }

      \tl_put_right:Nx \l__spath_split_pathb_tl
      {{ \tl_head:N \l__spath_split_path_tl }}
      \tl_set:Nx \l__spath_split_path_tl {\tl_tail:N \l__spath_split_path_tl }

      \spath_split_line:NNVV
      \l__spath_split_intoa_tl
      \l__spath_split_intob_tl
      \l__spath_split_pathb_tl
      \l__spath_split_fp

      \prg_replicate:nn {3} {
        \tl_set:Nx \l__spath_split_intoa_tl {\tl_tail:N \l__spath_split_intoa_tl}
      }

      \tl_put_right:NV \l__spath_split_patha_tl \l__spath_split_intoa_tl
      \tl_put_right:NV \l__spath_split_intob_tl \l__spath_split_path_tl
    }
    \c_spath_curvetoa_tl
    {
      \tl_put_right:NV \l__spath_split_pathb_tl \l__spath_tmpc_tl
      \tl_put_right:Nx \l__spath_split_pathb_tl
      {{ \tl_head:N \l__spath_split_path_tl }}
      \tl_set:Nx \l__spath_split_path_tl {\tl_tail:N \l__spath_split_path_tl }

      \tl_put_right:Nx \l__spath_split_pathb_tl
      {{ \tl_head:N \l__spath_split_path_tl }}
      \tl_set:Nx \l__spath_split_path_tl {\tl_tail:N \l__spath_split_path_tl }

      \prg_replicate:nn {2} {

        \tl_put_right:Nx \l__spath_split_pathb_tl
        { \tl_head:N \l__spath_split_path_tl }
        \tl_set:Nx \l__spath_split_path_tl {\tl_tail:N \l__spath_split_path_tl }

        \tl_put_right:Nx \l__spath_split_pathb_tl
        {{ \tl_head:N \l__spath_split_path_tl }}
        \tl_set:Nx \l__spath_split_path_tl {\tl_tail:N \l__spath_split_path_tl }

        \tl_put_right:Nx \l__spath_split_pathb_tl
        {{ \tl_head:N \l__spath_split_path_tl }}
        \tl_set:Nx \l__spath_split_path_tl {\tl_tail:N \l__spath_split_path_tl }
        }

        \spath_split_curve:NNVV
        \l__spath_split_intoa_tl
        \l__spath_split_intob_tl
        \l__spath_split_pathb_tl \l__spath_split_fp

        \prg_replicate:nn {3} {
          \tl_set:Nx \l__spath_split_intoa_tl {\tl_tail:N \l__spath_split_intoa_tl}
        }

        \tl_put_right:NV \l__spath_split_patha_tl \l__spath_split_intoa_tl
        \tl_put_right:NV \l__spath_split_intob_tl \l__spath_split_path_tl
      }
    }
  }

  \tl_gclear:N \g__spath_output_tl
  \__spath_tl_gput_right_braced:NV \g__spath_output_tl \l__spath_split_patha_tl
  \__spath_tl_gput_right_braced:NV \g__spath_output_tl \l__spath_split_intob_tl
  \group_end:
}
\cs_new_protected_nopar:Npn \spath_split_at:NNnn #1#2#3#4
{
  \__spath_split_at:nn {#3}{#4}
  \tl_set:Nx #1 {\tl_item:Nn \g__spath_output_tl {1}}
  \tl_set:Nx #2 {\tl_item:Nn \g__spath_output_tl {2}}
  \tl_gclear:N \g__spath_output_tl
}
\cs_generate_variant:Nn \spath_split_at:NNnn {NNVn, NNVV, NNnV}
\cs_new_protected_nopar:Npn \spath_gsplit_at:NNnn #1#2#3#4
{
  \__spath_split_at:nn {#3}{#4}
  \tl_gset:Nx #1 {\tl_item:Nn \g__spath_output_tl {1}}
  \tl_gset:Nx #2 {\tl_item:Nn \g__spath_output_tl {2}}
  \tl_gclear:N \g__spath_output_tl
}
\cs_generate_variant:Nn \spath_gsplit_at:NNnn {NNVn, NNVV, NNnV}
\cs_new_protected_nopar:Npn \spath_split_at:Nnn #1#2#3
{
  \__spath_split_at:nn {#2}{#3}
  \tl_set:Nx #1 {\tl_item:Nn \g__spath_output_tl {1} \tl_item:Nn \g__spath_output_tl {2}}
  \tl_gclear:N \g__spath_output_tl
}
\cs_generate_variant:Nn \spath_split_at:Nnn {NVn, NVV}
\cs_new_protected_nopar:Npn \spath_split_at:Nn #1#2
{
  \spath_split_at:NVn #1#1{#2}
}
\cs_new_protected_nopar:Npn \spath_gsplit_at:Nnn #1#2#3
{
  \__spath_split_at:nn {#2}{#3}
  \tl_gset:Nx #1 {\tl_item:Nn \g__spath_output_tl {1} \tl_item:Nn \g__spath_output_tl {2}}
  \tl_gclear:N \g__spath_output_tl
```

```
}
\cs_generate_variant:Nn \spath_gsplit_at:Nnn {NVn, NVV}
\cs_new_protected_nopar:Npn \spath_gsplit_at:Nn #1#2
{
  \spath_gsplit_at:NVn #1#1{#2}
}
```

(*End definition for* `\spath_split_at:NNnn` *and others. These functions are documented on page* **??**.)

3.5 Shortening Paths

This code relates to shortening paths. For curved paths, the routine uses the derivative at the end to figure out how far back to shorten. This means that the actual length that it shortens by is approximate, but it is guaranteed to be along its length.

As in the previous section, there are various versions. In particular, there are versions where the path can be specified by a macro and is saved back into that macro.

```
\tl_new:N \l__spath_shorten_fa_tl
\tl_new:N \l__spath_shorten_path_tl
\tl_new:N \l__spath_shorten_last_tl
\tl_new:N \l__spath_shorten_lasta_tl
\tl_new:N \l__spath_shorten_lastb_tl
\int_new:N \l__spath_shorten_int
\fp_new:N \l__spath_shorten_x_fp
\fp_new:N \l__spath_shorten_y_fp
\fp_new:N \l__spath_shorten_len_fp
```

`\spath_shorten_at_end:Nnn` This macro shortens a path from the end by a dimension.

```
\cs_new_protected_nopar:Npn \__spath_shorten_at_end:nn #1#2
{
  \int_compare:nTF
  {
    \tl_count:n {#1} > 3
  }
  {
    \group_begin:
    \tl_set:Nn \l__spath_shorten_path_tl {#1}
    \tl_reverse:N \l__spath_shorten_path_tl

    \tl_set:Nx \l__spath_shorten_fa_tl {\tl_item:Nn \l__spath_shorten_path_tl {3}}

    \tl_clear:N \l__spath_shorten_last_tl
    \tl_if_eq:NNTF \l__spath_shorten_fa_tl \c_spath_curveto_tl
    {
      \int_set:Nn \l__spath_shorten_int {3}
    }
    {
      \int_set:Nn \l__spath_shorten_int {1}
    }

    \prg_replicate:nn { \l__spath_shorten_int }
    {
      \tl_put_right:Nx \l__spath_shorten_last_tl
      {
```

```
      {\tl_head:N \l__spath_shorten_path_tl}
    }
    \tl_set:Nx \l__spath_shorten_path_tl {\tl_tail:N \l__spath_shorten_path_tl}
    \tl_put_right:Nx \l__spath_shorten_last_tl
    {
      {\tl_head:N \l__spath_shorten_path_tl}
    }
    \tl_set:Nx \l__spath_shorten_path_tl {\tl_tail:N \l__spath_shorten_path_tl}
    \tl_put_right:Nx \l__spath_shorten_last_tl
    {
      \tl_head:N \l__spath_shorten_path_tl
    }
    \tl_set:Nx \l__spath_shorten_path_tl {\tl_tail:N \l__spath_shorten_path_tl}
  }

  \tl_put_right:Nx \l__spath_shorten_last_tl
  {
    {\tl_item:Nn \l__spath_shorten_path_tl {1}}
    {\tl_item:Nn \l__spath_shorten_path_tl {2}}
  }
  \tl_put_right:NV \l__spath_shorten_last_tl \c_spath_moveto_tl

  \tl_reverse:N \l__spath_shorten_path_tl

  \fp_set:Nn \l__spath_shorten_x_fp
  {
    \dim_to_fp:n {\tl_item:Nn \l__spath_shorten_last_tl {4}}
    -
    \dim_to_fp:n {\tl_item:Nn \l__spath_shorten_last_tl {1}}
  }

  \fp_set:Nn \l__spath_shorten_y_fp
  {
    \dim_to_fp:n {\tl_item:Nn \l__spath_shorten_last_tl {5}}
    -
    \dim_to_fp:n {\tl_item:Nn \l__spath_shorten_last_tl {2}}
  }

  \fp_set:Nn \l__spath_shorten_len_fp
  {
    sqrt( \l__spath_shorten_x_fp * \l__spath_shorten_x_fp +  \l__spath_shorten_y_fp *  \l_
  }

  \fp_compare:nTF
  {
    \l__spath_shorten_len_fp > #2
  }
  {

    \fp_set:Nn \l__spath_shorten_len_fp
    {
      (\l__spath_shorten_len_fp - #2)/ \l__spath_shorten_len_fp
    }

```

```
      \tl_reverse:N \l__spath_shorten_last_tl

      \tl_if_eq:NNTF \l__spath_shorten_fa_tl \c_spath_curveto_tl
      {
        \fp_set:Nn \l__spath_shorten_len_fp
        {
          1 - (1 -\l__spath_shorten_len_fp)/3
        }
        \spath_split_curve:NNVV
        \l__spath_shorten_lasta_tl
        \l__spath_shorten_lastb_tl
        \l__spath_shorten_last_tl
        \l__spath_shorten_len_fp
      }
      {
        \spath_split_line:NNVV
        \l__spath_shorten_lasta_tl
        \l__spath_shorten_lastb_tl
        \l__spath_shorten_last_tl
        \l__spath_shorten_len_fp
      }

      \prg_replicate:nn {3}
      {
        \tl_set:Nx \l__spath_shorten_lasta_tl {\tl_tail:N \l__spath_shorten_lasta_tl}
      }

      \tl_put_right:NV \l__spath_shorten_path_tl \l__spath_shorten_lasta_tl

    }
    {

      \int_compare:nT
      {
        \tl_count:N \l__spath_shorten_path_tl > 3
      }
      {
        \dim_set:Nn \l__spath_tmpa_dim {\fp_to_dim:n {#2 - \l__spath_shorten_len_fp } }
        \spath_shorten_at_end:NV \l__spath_shorten_path_tl \l__spath_tmpa_dim
      }
    }

    \tl_gset_eq:NN \g__spath_output_tl \l__spath_shorten_path_tl
    \group_end:
  }
  {
    \tl_gset:Nn \g__spath_output_tl {#1}
  }
}
\cs_new_protected_nopar:Npn \spath_shorten_at_end:Nnn #1#2#3
{
  \__spath_shorten_at_end:nn {#2}{#3}
  \tl_set_eq:NN #1 \g__spath_output_tl
  \tl_gclear:N \g__spath_output_tl
```

```
}
\cs_generate_variant:Nn \spath_shorten_at_end:Nnn {NVV, cnn, cVV, NVn}
\cs_new_protected_nopar:Npn \spath_shorten_at_end:Nn #1#2
{
  \spath_shorten_at_end:NVn #1#1{#2}
}
\cs_generate_variant:Nn \spath_shorten_at_end:Nn {cn, cV, NV}
\cs_new_protected_nopar:Npn \spath_gshorten_at_end:Nnn #1#2#3
{
  \__spath_shorten_at_end:nn {#2}{#3}
  \tl_gset_eq:NN #1 \g__spath_output_tl
  \tl_gclear:N \g__spath_output_tl
}
\cs_generate_variant:Nn \spath_gshorten_at_end:Nnn {NVV, cnn, cVV, NVn}
\cs_new_protected_nopar:Npn \spath_gshorten_at_end:Nn #1#2
{
  \spath_gshorten_at_end:NVn #1#1{#2}
}
\cs_generate_variant:Nn \spath_gshorten_at_end:Nn {cn, cV, NV}
```

(*End definition for* `\spath_shorten_at_end:Nnn`. *This function is documented on page* **??**.)

`\spath_shorten_at_start:Nnn`
`\spath_shorten_at_start:Nn`
`\spath_gshorten_at_start:Nnn`
`\spath_gshorten_at_start:Nn`

This macro shortens a path from the start by a dimension.

```
\cs_new_protected_nopar:Npn \__spath_shorten_at_start:nn #1#2
{
  \int_compare:nTF
  {
    \tl_count:n {#1} > 3
  }
  {
  \group_begin:
  \tl_set:Nn \l__spath_shorten_path_tl {#1}

  \tl_set:Nx \l__spath_shorten_fa_tl {\tl_item:Nn \l__spath_shorten_path_tl {4}}

    \tl_clear:N \l__spath_shorten_last_tl

  \tl_if_eq:NNTF \l__spath_shorten_fa_tl \c_spath_curvetoa_tl
  {
    \int_set:Nn \l__spath_shorten_int {3}
  }
  {
    \int_set:Nn \l__spath_shorten_int {1}
  }

  \tl_set_eq:NN \l__spath_shorten_last_tl \c_spath_moveto_tl
  \tl_set:Nx \l__spath_shorten_path_tl {\tl_tail:N \l__spath_shorten_path_tl }

  \prg_replicate:nn { \l__spath_shorten_int }
  {
    \__spath_tl_put_right_braced:Nx \l__spath_shorten_last_tl {\tl_item:Nn \l__spath_shorten
    \__spath_tl_put_right_braced:Nx \l__spath_shorten_last_tl {\tl_item:Nn \l__spath_shorten
    \tl_put_right:Nx \l__spath_shorten_last_tl {\tl_item:Nn \l__spath_shorten_path_tl {3}}

    \prg_replicate:nn {3}
    {
      \tl_set:Nx \l__spath_shorten_path_tl {\tl_tail:N \l__spath_shorten_path_tl }
    }
  }
  \__spath_tl_put_right_braced:Nx \l__spath_shorten_last_tl {\tl_item:Nn \l__spath_shorten_p
  \__spath_tl_put_right_braced:Nx \l__spath_shorten_last_tl {\tl_item:Nn \l__spath_shorten_p

  \fp_set:Nn \l__spath_shorten_x_fp
  {
    \dim_to_fp:n {\tl_item:Nn \l__spath_shorten_last_tl {5}}
    -
    \dim_to_fp:n {\tl_item:Nn \l__spath_shorten_last_tl {2}}
  }

  \fp_set:Nn \l__spath_shorten_y_fp
  {
    \dim_to_fp:n {\tl_item:Nn \l__spath_shorten_last_tl {6}}
    -
    \dim_to_fp:n {\tl_item:Nn \l__spath_shorten_last_tl {3}}
  }

  \fp_set:Nn \l__spath_shorten_len_fp
  {
    sqrt( \l__spath_shorten_x_fp * \l__spath_shorten_x_fp +  \l__spath_shorten_y_fp *  \l__s
  }

  \fp_compare:nTF
  {
    \l__spath_shorten_len_fp > #2
  }
  {

    \fp_set:Nn \l__spath_shorten_len_fp
    {
      #2/ \l__spath_shorten_len_fp
    }

    \tl_if_eq:NNTF \l__spath_shorten_fa_tl \c_spath_curvetoa_tl
    {
      \fp_set:Nn \l__spath_shorten_len_fp
      {
        \l__spath_shorten_len_fp/3
      }
      \spath_split_curve:NNVV
      \l__spath_shorten_lasta_tl
      \l__spath_shorten_lastb_tl
      \l__spath_shorten_last_tl
      \l__spath_shorten_len_fp
    }
    {
      \spath_split_line:NNVV
      \l__spath_shorten_lasta_tl
      \l__spath_shorten_lastb_tl
```

```
      \l__spath_shorten_last_tl
      \l__spath_shorten_len_fp
    }

    \prg_replicate:nn {2}
    {
      \tl_set:Nx \l__spath_shorten_path_tl {\tl_tail:N \l__spath_shorten_path_tl}
    }

    \tl_put_left:NV \l__spath_shorten_path_tl \l__spath_shorten_lastb_tl

  }
  {

    \tl_put_left:NV \l__spath_shorten_path_tl \c_spath_moveto_tl

    \int_compare:nT
    {
      \tl_count:N \l__spath_shorten_path_tl > 3
    }
    {
      \dim_set:Nn \l__spath_tmpa_dim {\fp_to_dim:n {#2 - \l__spath_shorten_len_fp } }
      \spath_shorten_at_start:NV \l__spath_shorten_path_tl \l__spath_tmpa_dim
    }
  }

  \tl_gset_eq:NN \g__spath_output_tl \l__spath_shorten_path_tl
  \group_end:
  }
  {
    \tl_gset:Nn \g__spath_output_tl {#1}
  }
}
\cs_new_protected_nopar:Npn \spath_shorten_at_start:Nnn #1#2#3
{
  \__spath_shorten_at_start:nn {#2}{#3}
  \tl_set_eq:NN #1 \g__spath_output_tl
  \tl_gclear:N \g__spath_output_tl
}
\cs_generate_variant:Nn \spath_shorten_at_start:Nnn {NVV, cnn, cVV, NVn}
\cs_new_protected_nopar:Npn \spath_shorten_at_start:Nn #1#2
{
  \spath_shorten_at_start:NVn #1#1{#2}
}
\cs_generate_variant:Nn \spath_shorten_at_start:Nn {cn, cV, NV}
\cs_new_protected_nopar:Npn \spath_gshorten_at_start:Nnn #1#2#3
{
  \__spath_shorten_at_start:nn {#2}{#3}
  \tl_gset_eq:NN #1 \g__spath_output_tl
  \tl_gclear:N \g__spath_output_tl
}
\cs_generate_variant:Nn \spath_gshorten_at_start:Nnn {NVV, cnn, cVV, NVn}
\cs_new_protected_nopar:Npn \spath_gshorten_at_start:Nn #1#2
{
  \spath_gshorten_at_start:NVn #1#1{#2}
}
\cs_generate_variant:Nn \spath_gshorten_at_start:Nn {cn, cV, NV}
```

(*End definition for* `\spath_shorten_at_start:Nnn` *and others. These functions are documented on page* **??**.)

3.6 Points on a Path

`\spath_point_at:Nnn`
`\spath_gpoint_at:Nnn`

Get the location of a point on a path, using the same location specification as the intersection library.

```
\cs_new_protected_nopar:Npn \__spath_point_at:nn #1#2
{
  \group_begin:
  \int_set:Nn \l__spath_tmpa_int {\fp_to_int:n {floor(#2) + 1}}
  \fp_set:Nn \l__spath_tmpa_fp {#2 - floor(#2)}

  \spath_segments_to_seq:Nn \l__spath_tmpa_seq {#1}

  \int_compare:nTF
  {
    \l__spath_tmpa_int < 1
  }
  {
    \spath_initialpoint:Nn \l__spath_tmpc_tl {#1}
  }
  {
    \int_compare:nTF
    {
      \l__spath_tmpa_int > \seq_count:N \l__spath_tmpa_seq
    }
    {
      \spath_finalpoint:Nn \l__spath_tmpc_tl {#1}
    }
    {

      \tl_set:Nx \l__spath_tmpa_tl {\seq_item:Nn \l__spath_tmpa_seq { \l__spath_tmpa_int} }

      \int_compare:nTF
      {
        \tl_count:N \l__spath_tmpa_tl > 3
      }
      {
        \tl_set:Nx \l__spath_tmpb_tl {\tl_item:Nn \l__spath_tmpa_tl {4}}
      }
      {
        \tl_set:Nx \l__spath_tmpb_tl {\tl_item:Nn \l__spath_tmpa_tl {1}}
      }

      \tl_clear:N \l__spath_tmpc_tl

      \tl_case:Nn \l__spath_tmpb_tl
      {
        \c_spath_moveto_tl
        {
          \tl_set:Nx \l__spath_tmpc_tl
          {
            {
              \tl_item:Nn \l__spath_tmpa_tl {2}
            }
            {
              \tl_item:Nn \l__spath_tmpa_tl {3}
            }
          }
        }

        \c_spath_lineto_tl
        {
          \tl_set:Nx \l__spath_tmpc_tl
          {
            {\fp_to_dim:n
              {
                (1 - \l__spath_tmpa_fp) * ( \tl_item:Nn \l__spath_tmpa_tl {2} )
                +
                \l__spath_tmpa_fp * ( \tl_item:Nn \l__spath_tmpa_tl {5} )
              }
            }
            {\fp_to_dim:n
              {
                (1 - \l__spath_tmpa_fp) * ( \tl_item:Nn \l__spath_tmpa_tl {3} )
                +
                \l__spath_tmpa_fp * ( \tl_item:Nn \l__spath_tmpa_tl {6} )
              }
            }
          }
        }

        \c_spath_closepath_tl
        {
          \tl_set:Nx \l__spath_tmpc_tl
          {
            {\fp_to_dim:n
              {
                (1 - \l__spath_tmpa_fp) * ( \tl_item:Nn \l__spath_tmpa_tl {2} )
                +
                \l__spath_tmpa_fp * ( \tl_item:Nn \l__spath_tmpa_tl {5} )
              }
            }
            {\fp_to_dim:n
              {
                (1 - \l__spath_tmpa_fp) * ( \tl_item:Nn \l__spath_tmpa_tl {3} )
                +
                \l__spath_tmpa_fp * ( \tl_item:Nn \l__spath_tmpa_tl {6} )
              }
            }
          }
        }

        \c_spath_curvetoa_tl
        {
          \tl_set:Nx \l__spath_tmpc_tl
          {
            {\fp_to_dim:n
              {
                (1 - \l__spath_tmpa_fp)^3 * \tl_item:Nn \l__spath_tmpa_tl {2} + 3 * (1 - \l_
            }}
            {\fp_to_dim:n
              {
                (1 - \l__spath_tmpa_fp)^3 * \tl_item:Nn \l__spath_tmpa_tl {3} + 3 * (1 - \l_
            }}
          }
        }
      }
    }
  }

  \tl_gclear:N \g__spath_output_tl
  \tl_gset_eq:NN \g__spath_output_tl \l__spath_tmpc_tl
  \group_end:
}
\cs_new_protected_nopar:Npn \spath_point_at:Nnn #1#2#3
{
  \__spath_point_at:nn {#2}{#3}
  \tl_set_eq:NN #1 \g__spath_output_tl
  \tl_gclear:N \g__spath_output_tl
}
\cs_generate_variant:Nn \spath_point_at:Nnn {NVn, NVV, NnV}
\cs_new_protected_nopar:Npn \spath_gpoint_at:Nnn #1#2#3
{
  \__spath_point_at:nn {#2}{#3}
  \tl_gset_eq:NN #1 \g__spath_output_tl
  \tl_gclear:N \g__spath_output_tl
}
\cs_generate_variant:Nn \spath_gpoint_at:Nnn {NVn, NVV, NnV}
```

(*End definition for* `\spath_point_at:Nnn` *and* `\spath_gpoint_at:Nnn`*. These functions are documented on page* **??**.)

`\spath_tangent_at:Nnn`
`\spath_gtangent_at:Nnn`

Get the tangent at a point on a path, using the same location specification as the intersection library.

```
\cs_new_protected_nopar:Npn \__spath_tangent_at:nn #1#2
{
  \group_begin:
  \int_set:Nn \l__spath_tmpa_int {\fp_to_int:n {floor(#2) + 1}}
  \fp_set:Nn \l__spath_tmpa_fp {#2 - floor(#2)}

  \spath_segments_to_seq:Nn \l__spath_tmpa_seq {#1}

  \int_compare:nTF
  {
    \l__spath_tmpa_int < 1
  }
```

```
{
  \spath_initialpoint:Nn \l__spath_tmpc_tl {#1}
}
{
  \int_compare:nTF
  {
    \l__spath_tmpa_int > \seq_count:N \l__spath_tmpa_seq
  }
  {
    \spath_finalpoint:Nn \l__spath_tmpc_tl {#1}
  }
  {

    \tl_set:Nx \l__spath_tmpa_tl {\seq_item:Nn \l__spath_tmpa_seq { \l__spath_tmpa_int} }

    \int_compare:nTF
    {
      \tl_count:N \l__spath_tmpa_tl > 3
    }
    {
      \tl_set:Nx \l__spath_tmpb_tl {\tl_item:Nn \l__spath_tmpa_tl {4}}
    }
    {
      \tl_set:Nx \l__spath_tmpb_tl {\tl_item:Nn \l__spath_tmpa_tl {1}}
    }

    \tl_clear:N \l__spath_tmpc_tl

    \tl_case:Nn \l__spath_tmpb_tl
    {
      \c_spath_moveto_tl
      {
        \tl_set:Nx \l__spath_tmpc_tl
        {
          {
            \tl_item:Nn \l__spath_tmpa_tl {2}
          }
          {
            \tl_item:Nn \l__spath_tmpa_tl {3}
          }
        }
      }

      \c_spath_lineto_tl
      {
        \tl_set:Nx \l__spath_tmpc_tl
        {
          {\fp_to_dim:n
            {
              ( \tl_item:Nn \l__spath_tmpa_tl {5} )
              -
              ( \tl_item:Nn \l__spath_tmpa_tl {2} )
            }
          }
          {\fp_to_dim:n
            {
              ( \tl_item:Nn \l__spath_tmpa_tl {6} )
              -
              ( \tl_item:Nn \l__spath_tmpa_tl {3} )
            }
          }
        }
      }

      \c_spath_closepath_tl
      {
        \tl_set:Nx \l__spath_tmpc_tl
        {
          {\fp_to_dim:n
            {
              ( \tl_item:Nn \l__spath_tmpa_tl {5} )
              -
              ( \tl_item:Nn \l__spath_tmpa_tl {2} )
            }
          }
          {\fp_to_dim:n
            {
              ( \tl_item:Nn \l__spath_tmpa_tl {6} )
              -
              ( \tl_item:Nn \l__spath_tmpa_tl {3} )
            }
          }
        }
      }

      \c_spath_curvetoa_tl
      {
        \tl_set:Nx \l__spath_tmpc_tl
        {
          {\fp_to_dim:n
            {
              3*(1 - \l__spath_tmpa_fp)^2 * (\tl_item:Nn \l__spath_tmpa_tl {5} - \tl_item:
              + 6 * (1 - \l__spath_tmpa_fp) * (\l__spath_tmpa_fp) *  (\tl_item:Nn \l__spat
              + 3*(\l__spath_tmpa_fp)^2 * (\tl_item:Nn \l__spath_tmpa_tl {11} - \tl_item:N
            }
          }
          {\fp_to_dim:n
            {
              3*(1 - \l__spath_tmpa_fp)^2 * (\tl_item:Nn \l__spath_tmpa_tl {6} - \tl_item:
              + 6 * (1 - \l__spath_tmpa_fp) * (\l__spath_tmpa_fp) *  (\tl_item:Nn \l__spat
              + 3*(\l__spath_tmpa_fp)^2 * (\tl_item:Nn \l__spath_tmpa_tl {12} - \tl_item:N
          }}
        }
      }
    }
  }
}

```

```
  \tl_gclear:N \g__spath_output_tl
  \tl_gset_eq:NN \g__spath_output_tl \l__spath_tmpc_tl
  \group_end:
}
\cs_new_protected_nopar:Npn \spath_tangent_at:Nnn #1#2#3
{
  \__spath_tangent_at:nn {#2}{#3}
  \tl_set_eq:NN #1 \g__spath_output_tl
  \tl_gclear:N \g__spath_output_tl
}
\cs_generate_variant:Nn \spath_tangent_at:Nnn {NVn, NVV, NnV}
\cs_new_protected_nopar:Npn \spath_gtangent_at:Nnn #1#2#3
{
  \__spath_tangent_at:nn {#2}{#3}
  \tl_gset_eq:NN #1 \g__spath_output_tl
  \tl_gclear:N \g__spath_output_tl
}
\cs_generate_variant:Nn \spath_gtangent_at:Nnn {NVn, NVV, NnV}
```

(*End definition for* `\spath_tangent_at:Nnn` *and* `\spath_gtangent_at:Nnn`*. These functions are documented on page* **??**.)

`\spath_transformation_at:Nnn`
`\spath_gtransformation_at:Nnn`

Gets a transformation that will align to a point on the path with the x-axis along the path.

```
\cs_new_protected_nopar:Npn \__spath_transformation_at:nn #1#2
{
  \group_begin:
  \tl_clear:N \l__spath_tmpa_tl
  \__spath_tangent_at:nn {#1}{#2}
  \tl_set_eq:NN \l__spath_tmpb_tl \g__spath_output_tl
  \fp_set:Nn \l__spath_tmpa_fp { sqrt( (\tl_item:Nn \l__spath_tmpb_tl {1})^2 +  (\tl_item:Nn
  \fp_compare:nTF {\l__spath_tmpa_fp = 0}
  {
    \fp_set:Nn \l__spath_tmpa_fp {1}
    \fp_set:Nn \l__spath_tmpb_fp {0}
  }
  {
    \fp_set:Nn \l__spath_tmpb_fp { (\tl_item:Nn \l__spath_tmpb_tl {2}) / \l__spath_tmpa_fp }
    \fp_set:Nn \l__spath_tmpa_fp { (\tl_item:Nn \l__spath_tmpb_tl {1}) / \l__spath_tmpa_fp }
  }
  \tl_set:Nx \l__spath_tmpa_tl
  {
    { \fp_to_decimal:n { \l__spath_tmpa_fp } }
    { \fp_to_decimal:n { \l__spath_tmpb_fp } }
    { \fp_to_decimal:n {- \l__spath_tmpb_fp } }
    { \fp_to_decimal:n { \l__spath_tmpa_fp } }
  }
  \__spath_point_at:nn {#1}{#2}
  \tl_put_right:NV \l__spath_tmpa_tl \g__spath_output_tl
  \tl_gset_eq:NN \g__spath_output_tl \l__spath_tmpa_tl
  \group_end:
}
\cs_new_protected_nopar:Npn \spath_transformation_at:Nnn #1#2#3
{
  \__spath_transformation_at:nn {#2}{#3}
  \tl_set_eq:NN #1 \g__spath_output_tl
  \tl_gclear:N \g__spath_output_tl
}
\cs_generate_variant:Nn \spath_transformation_at:Nnn {NVn, NVV, NnV, NvV}
\cs_new_protected_nopar:Npn \spath_gtransformation_at:Nnn #1#2#3
{
  \__spath_transformation_at:nn {#2}{#3}
  \tl_gset_eq:NN #1 \g__spath_output_tl
  \tl_gclear:N \g__spath_output_tl
}
\cs_generate_variant:Nn \spath_gtransformation_at:Nnn {NVn, NVV, NnV}
```

(*End definition for* `\spath_transformation_at:Nnn` *and* `\spath_gtransformation_at:Nnn`*. These functions are documented on page* **??***.*)

3.7 Intersection Routines

Note: I'm not consistent with number schemes. The intersection library is 0-based, but the user interface is 1-based (since if we "count" in a `\foreach` then it starts at 1). This should be more consistent.

```
\tl_new:N \l__spath_split_path_a_tl
\tl_new:N \l__spath_split_path_b_tl
\tl_new:N \l__spath_split_path_a_start_tl
\tl_new:N \l__spath_split_path_b_start_tl
\tl_new:N \l__spath_split_path_a_end_tl
\tl_new:N \l__spath_split_path_b_end_tl
\tl_new:N \l__spath_split_path_a_final_tl
\tl_new:N \l__spath_split_path_b_final_tl

\tl_new:N \l__spath_split_prev_first_tl
\tl_new:N \l__spath_split_prev_second_tl

\seq_new:N \l__spath_split_first_seq
\seq_new:N \l__spath_split_second_seq

\int_new:N \l__spath_split_segment_int
```

`\spath_intersect:NN`
`\spath_intersect:nn`

Pass two spaths to pgf's intersection routine.

```
\cs_new_protected_nopar:Npn \spath_intersect:NN #1#2
{
  \pgfintersectionofpaths%
  {%
    \pgfsetpath #1
  }{%
    \pgfsetpath #2
  }
}
\tl_new:N \l__spath_intersecta_tl
\tl_new:N \l__spath_intersectb_tl
\cs_new_protected_nopar:Npn \spath_intersect:nn #1#2
{
  \tl_set:Nn \l__spath_intersecta_tl {#1}
  \tl_set:Nn \l__spath_intersectb_tl {#2}
  \spath_intersect:NN \l__spath_intersecta_tl \l__spath_intersectb_tl
}
```

(*End definition for* `\spath_intersect:NN` *and* `\spath_intersect:nn`*. These functions are documented on page* **??**.)

`\spath_split_at_intersections:NNnn`
`\spath_split_at_intersections:NN`
`\spath_gsplit_at_intersections:NNnn`
`\spath_gsplit_at_intersections:NN`

Given two paths, split them at points where they intersect and store them in the macros.

```
\cs_new_protected_nopar:Npn \__spath_split_at_intersections:nn #1#2
{
  \group_begin:

  % Clear some token lists and sequences
  \tl_clear:N \l__spath_split_path_a_final_tl
  \tl_clear:N \l__spath_split_path_b_final_tl
  \seq_clear:N \l__spath_split_first_seq
  \seq_clear:N \l__spath_split_second_seq

  \pgfintersectionsortbyfirstpath
  % Find the intersections of these segments
  \tl_set:Nn \l__spath_split_path_a_tl {#1}
  \tl_set:Nn \l__spath_split_path_b_tl {#2}
  % Remove empty components
%  \spath_path_remove_empty_components:N \l__spath_split_path_a_tl
%  \spath_path_remove_empty_components:N \l__spath_split_path_b_tl

  \spath_intersect:NN \l__spath_split_path_a_tl \l__spath_split_path_b_tl

  % If we get intersections
  \int_compare:nT {\pgfintersectionsolutions > 0}
  {
    % Find the times of the intersections on each path
    \int_step_inline:nnnn {1} {1} {\pgfintersectionsolutions}
    {
    \pgfintersectiongetsolutiontimes{##1}{\l__spath_split_first_tl}{\l__spath_split_second_t
      \seq_put_left:NV \l__spath_split_first_seq \l__spath_split_first_tl
    }
  }

  \spath_intersect:NN \l__spath_split_path_b_tl \l__spath_split_path_a_tl

  % If we get intersections
  \int_compare:nT {\pgfintersectionsolutions > 0}
  {
    % Find the times of the intersections on each path
    \int_step_inline:nnnn {1} {1} {\pgfintersectionsolutions}
    {
    \pgfintersectiongetsolutiontimes{##1}{\l__spath_split_first_tl}{\l__spath_split_second_t
      \seq_put_left:NV \l__spath_split_second_seq \l__spath_split_first_tl
    }
  }

  \tl_set:Nn \l__spath_split_prev_first_tl {-1}

  \seq_map_inline:Nn \l__spath_split_first_seq
```

```
{
  \tl_set:Nn \l__spath_split_first_tl {##1}

  \tl_set_eq:NN \l__spath_tmpa_tl \l__spath_split_first_tl
  \int_compare:nT
  {
    \fp_to_int:n {floor( \l__spath_split_first_tl) }
    =
    \fp_to_int:n {floor( \l__spath_split_prev_first_tl) }
  }
  {
    \tl_set:Nx \l__spath_split_first_tl
    {
      \fp_eval:n {
        floor( \l__spath_split_first_tl )
        +
        ( \l__spath_split_first_tl - floor( \l__spath_split_first_tl) )
        /
        ( \l__spath_split_prev_first_tl - floor( \l__spath_split_prev_first_tl) )
      }
    }
  }
  \tl_set_eq:NN \l__spath_split_prev_first_tl \l__spath_tmpa_tl

  \spath_split_at:NNVV \l__spath_split_path_a_start_tl \l__spath_split_path_a_end_tl  \l__

  \tl_put_left:NV \l__spath_split_path_a_final_tl \l__spath_split_path_a_end_tl
  \tl_set_eq:NN \l__spath_split_path_a_tl \l__spath_split_path_a_start_tl

}

\tl_set:Nn \l__spath_split_prev_second_tl {-1}

\seq_map_inline:Nn \l__spath_split_second_seq
{
  \tl_set:Nn \l__spath_split_second_tl {##1}

  \tl_set_eq:NN \l__spath_tmpa_tl \l__spath_split_second_tl
  \int_compare:nT
  {
    \fp_to_int:n {floor( \l__spath_split_second_tl) }
    =
    \fp_to_int:n {floor( \l__spath_split_prev_second_tl) }
  }
  {
    \tl_set:Nx \l__spath_split_second_tl
    {
      \fp_eval:n {
        floor( \l__spath_split_second_tl )
        +
        ( \l__spath_split_second_tl - floor( \l__spath_split_second_tl) )
        /
        ( \l__spath_split_prev_second_tl - floor( \l__spath_split_prev_second_tl) )
      }
      }
    }
    \tl_set_eq:NN \l__spath_split_prev_second_tl \l__spath_tmpa_tl

    \spath_split_at:NNVV \l__spath_split_path_b_start_tl \l__spath_split_path_b_end_tl  \l__

    \tl_put_left:NV \l__spath_split_path_b_final_tl \l__spath_split_path_b_end_tl
    \tl_set_eq:NN \l__spath_split_path_b_tl \l__spath_split_path_b_start_tl

  }

  \tl_put_left:NV \l__spath_split_path_a_final_tl \l__spath_split_path_a_tl
  \tl_put_left:NV \l__spath_split_path_b_final_tl \l__spath_split_path_b_tl

  \tl_if_empty:NT \l__spath_split_path_a_final_tl
  {
    \tl_set_eq:NN \l__spath_split_path_a_final_tl \l__spath_split_path_a_tl
  }
  \tl_if_empty:NT \l__spath_split_path_b_final_tl
  {
    \tl_set_eq:NN \l__spath_split_path_b_final_tl \l__spath_split_path_b_tl
  }

  \spath_remove_empty_components:N \l__spath_split_path_a_final_tl
  \spath_remove_empty_components:N \l__spath_split_path_b_final_tl

  \tl_gclear:N \g__spath_output_tl
  \__spath_tl_gput_right_braced:NV \g__spath_output_tl \l__spath_split_path_a_final_tl
  \__spath_tl_gput_right_braced:NV \g__spath_output_tl \l__spath_split_path_b_final_tl
  \group_end:
}
\cs_new_protected_nopar:Npn \spath_split_at_intersections:NNnn #1#2#3#4
{
  \__spath_split_at_intersections:nn {#3}{#4}
  \tl_set:Nx #1 {\tl_item:Nn \g__spath_output_tl {1}}
  \tl_set:Nx #2 {\tl_item:Nn \g__spath_output_tl {2}}
  \tl_gclear:N \g__spath_output_tl
}
\cs_generate_variant:Nn \spath_split_at_intersections:NNnn {NNVV, ccVV, ccvv}
\cs_new_protected_nopar:Npn \spath_split_at_intersections:NN #1#2
{
  \spath_split_at_intersections:NNVV #1#2#1#2
}
\cs_generate_variant:Nn \spath_split_at_intersections:NN {cc}
\cs_new_protected_nopar:Npn \spath_gsplit_at_intersections:NNnn #1#2#3#4
{
  \__spath_split_at_intersections:nn {#3}{#4}
  \tl_gset:Nx #1 {\tl_item:Nn \g__spath_output_tl {1}}
  \tl_gset:Nx #2 {\tl_item:Nn \g__spath_output_tl {2}}
  \tl_gclear:N \g__spath_output_tl
}
\cs_generate_variant:Nn \spath_gsplit_at_intersections:NNnn {NNVV, ccVV, ccvv}
\cs_new_protected_nopar:Npn \spath_gsplit_at_intersections:NN #1#2
{
  \spath_gsplit_at_intersections:NNVV #1#2#1#2
}
\cs_generate_variant:Nn \spath_gsplit_at_intersections:NN {cc}
```

(*End definition for* `\spath_split_at_intersections:NNnn` *and others. These functions are documented on page* **??**.)

`\spath_split_at_self_intersections:Nn`
`\spath_split_at_self_intersections:N`
`\spath_gsplit_at_self_intersections:Nn`
`\spath_gsplit_at_self_intersections:N`

Given a path, split it at points where it self-intersects and store in the given macro.

```
\cs_new_protected_nopar:Npn \__spath_split_at_self_intersections:n #1
{
  \group_begin:

  % Copy the path
  \tl_set:Nn \l__spath_split_path_b_tl {#1}

  % Open the path
  \spath_open:N \l__spath_split_path_b_tl
  % Remove empty components
  \spath_remove_empty_components:N \l__spath_split_path_b_tl
  % Make a copy for later
  \tl_set_eq:NN \l__spath_split_path_b_final_tl \l__spath_split_path_b_tl

  % Clear some token lists and sequences
  \tl_clear:N \l__spath_split_path_a_tl
  \seq_clear:N \l__spath_split_first_seq
  \int_zero:N \l__spath_split_segment_int

  \pgfintersectionsortbyfirstpath

  \bool_do_until:nn
  {
    \int_compare_p:n
    {
      \tl_count:N \l__spath_split_path_b_tl < 4
    }
  }
  {
    \tl_clear:N \l__spath_split_path_a_tl
    \tl_put_right:Nx \l__spath_split_path_a_tl
    {
      \tl_item:Nn \l__spath_split_path_b_tl {1}
      {\tl_item:Nn \l__spath_split_path_b_tl {2}}
      {\tl_item:Nn \l__spath_split_path_b_tl {3}}
    }

    \tl_set:Nx \l__spath_tmpa_tl { \tl_item:Nn \l__spath_split_path_b_tl {4} }

    \tl_set:Nx \l__spath_split_path_b_tl {\tl_tail:N \l__spath_split_path_b_tl}

    \int_zero:N \l__spath_tmpa_int

    \tl_case:Nn \l__spath_tmpa_tl
    {
      \c_spath_moveto_tl
      {
        \tl_clear:N \l__spath_split_path_a_tl
        \tl_set:Nx \l__spath_split_path_b_tl {\tl_tail:N \l__spath_split_path_b_tl}
        \tl_set:Nx \l__spath_split_path_b_tl {\tl_tail:N \l__spath_split_path_b_tl}
        \tl_set:Nx \l__spath_split_path_b_tl {\tl_tail:N \l__spath_split_path_b_tl}
      }
      \c_spath_lineto_tl
      {
        \int_set:Nn \l__spath_tmpa_int {1}
      }
      \c_spath_curvetoa_tl
      {
        \int_set:Nn \l__spath_tmpa_int {3}
      }
    }

    \prg_replicate:nn { \l__spath_tmpa_int }
    {
      \tl_set:Nx \l__spath_split_path_b_tl {\tl_tail:N \l__spath_split_path_b_tl}
      \tl_set:Nx \l__spath_split_path_b_tl {\tl_tail:N \l__spath_split_path_b_tl}

      \tl_put_right:Nx \l__spath_split_path_a_tl
      {
        \tl_item:Nn \l__spath_split_path_b_tl {1}
        {\tl_item:Nn \l__spath_split_path_b_tl {2}}
        {\tl_item:Nn \l__spath_split_path_b_tl {3}}
      }

      \tl_set:Nx \l__spath_split_path_b_tl {\tl_tail:N \l__spath_split_path_b_tl}
    }

    \tl_put_left:NV \l__spath_split_path_b_tl \c_spath_moveto_tl

    \tl_if_empty:NF \l__spath_split_path_a_tl
    {
      % Intersect the current segment with the rest of the path
      \spath_intersect:NN \l__spath_split_path_a_tl \l__spath_split_path_b_tl


      % If we get intersections
      \int_compare:nT {\pgfintersectionsolutions > 0}
      {
        % Find the times of the intersections on each path
        \int_step_inline:nnnn {1} {1} {\pgfintersectionsolutions}
        {
          \pgfintersectiongetsolutiontimes{##1}{\l__spath_split_first_tl}{\l__spath_split_se
          \fp_compare:nT
          {
            \l__spath_split_first_tl < .99
          }
          {
            \tl_set:Nx \l__spath_tmpa_tl {\fp_to_decimal:n {\l__spath_split_first_tl +  \l__
            \seq_put_right:NV \l__spath_split_first_seq \l__spath_tmpa_tl
          }
        }
      }

      \spath_intersect:NN \l__spath_split_path_b_tl \l__spath_split_path_a_tl

      % If we get intersections
      \int_compare:nT {\pgfintersectionsolutions > 0}
      {
        % Find the times of the intersections on each path
        \int_step_inline:nnnn {1} {1} {\pgfintersectionsolutions}
        {
          \pgfintersectiongetsolutiontimes{##1}{\l__spath_split_first_tl}{\l__spath_split_se
          \fp_compare:nT
          {
            \l__spath_split_first_tl > .01
          }
          {
            \tl_set:Nx \l__spath_tmpa_tl {\fp_to_decimal:n {\l__spath_split_first_tl +  \l__
            \seq_put_right:NV \l__spath_split_first_seq \l__spath_tmpa_tl
          }
        }
      }

    }
    % Increment the segment counter
    \int_incr:N \l__spath_split_segment_int
  }

  % Sort the sequence by reverse order along the path
  \seq_sort:Nn \l__spath_split_first_seq
  {
    \fp_compare:nNnTF { ##1 } < { ##2 }
    { \sort_return_swapped: }
    { \sort_return_same: }
  }

  % Restore the original copy of the path
  \tl_set_eq:NN \l__spath_split_path_b_tl \l__spath_split_path_b_final_tl

  % Clear the token lists
  \tl_clear:N \l__spath_split_path_b_start_tl
  \tl_clear:N \l__spath_split_path_b_end_tl
  \tl_clear:N \l__spath_split_path_b_final_tl

  \tl_set:Nn \l__spath_split_prev_first_tl {-1}

  \seq_map_inline:Nn \l__spath_split_first_seq
  {
    \tl_set:Nn \l__spath_split_first_tl {##1}
    \tl_set_eq:NN \l__spath_tmpa_tl \l__spath_split_first_tl
    \int_compare:nT
    {
      \fp_to_int:n {floor( \l__spath_split_first_tl ) }
      =
      \fp_to_int:n {floor( \l__spath_split_prev_first_tl) }
    }
    {
      \tl_set:Nx \l__spath_split_first_tl
      {
        \fp_eval:n {
          floor( \l__spath_split_first_tl )
          +
          ( \l__spath_split_first_tl - floor( \l__spath_split_first_tl) )
          /
          ( \l__spath_split_prev_first_tl - floor( \l__spath_split_prev_first_tl) )
        }
      }
    }
    \tl_set_eq:NN \l__spath_split_prev_first_tl \l__spath_tmpa_tl

    \spath_split_at:NNVV \l__spath_split_path_b_start_tl \l__spath_split_path_b_end_tl  \l__

    \tl_put_left:NV \l__spath_split_path_b_final_tl \l__spath_split_path_b_end_tl
    \tl_set_eq:NN \l__spath_split_path_b_tl \l__spath_split_path_b_start_tl

  }

  \tl_put_left:NV \l__spath_split_path_b_final_tl \l__spath_split_path_b_tl

  \tl_if_empty:NT \l__spath_split_path_b_final_tl
  {
    \tl_set_eq:NN \l__spath_split_path_b_final_tl \l__spath_split_path_b_tl
  }

  \spath_remove_empty_components:N \l__spath_split_path_b_final_tl

  \tl_gclear:N \g__spath_output_tl
  \tl_gset_eq:NN \g__spath_output_tl \l__spath_split_path_b_final_tl
  \group_end:
}
\cs_new_protected_nopar:Npn \spath_split_at_self_intersections:Nn #1#2
{
  \__spath_split_at_self_intersections:n {#2}
  \tl_set_eq:NN #1 \g__spath_output_tl
  \tl_gclear:N \g__spath_output_tl
}
\cs_generate_variant:Nn \spath_split_at_self_intersections:Nn {NV}
\cs_new_protected_nopar:Npn \spath_split_at_self_intersections:N #1
{
  \spath_split_at_self_intersections:NV #1#1
}
\cs_generate_variant:Nn \spath_split_at_self_intersections:N {c}
\cs_new_protected_nopar:Npn \spath_gsplit_at_self_intersections:Nn #1#2
{
  \__spath_split_at_self_intersections:n {#2}
  \tl_gset_eq:NN #1 \g__spath_output_tl
  \tl_gclear:N \g__spath_output_tl
}
```

```
\cs_generate_variant:Nn \spath_gsplit_at_self_intersections:Nn {NV}
\cs_new_protected_nopar:Npn \spath_gsplit_at_self_intersections:N #1
{
  \spath_gsplit_at_self_intersections:NV #1#1
}
\cs_generate_variant:Nn \spath_gsplit_at_self_intersections:N {c}
```

(*End definition for* `\spath_split_at_self_intersections:Nn` *and others. These functions are documented on page* **??**.)

`\spath_join_component:Nnn`
`\spath_join_component:Nn`
`\spath_gjoin_component:Nnn`
`\spath_gjoin_component:Nn`

Join the specified component of the spath to its predecessor.

```
\tl_new:N \l__spath_tmpj_tl
\cs_new_protected_nopar:Npn \__spath_join_component:nn #1#2
{
  \group_begin:
  \spath_components_to_seq:Nn \l__spath_tmpa_seq {#1}
  \int_compare:nT
  {
    #2 == 1
  }
  {
    \tl_clear:N \l__spath_tmpj_tl
    \seq_pop_left:NN \l__spath_tmpa_seq \l__spath_tmpj_tl

    \prg_replicate:nn {3}
    {
      \tl_set:Nx \l__spath_tmpj_tl {\tl_tail:N \l__spath_tmpj_tl}
    }

    \seq_put_right:NV \l__spath_tmpb_seq \l__spath_tmpj_tl
  }
  \bool_if:nT
  {
    \int_compare_p:n
    {
      #2 > 1
    }
    &&
    \int_compare_p:n
    {
      #2 <= \seq_count:N \l__spath_tmpa_seq
    }
  }
  {

    \seq_clear:N \l__spath_tmpb_seq
    \seq_map_indexed_inline:Nn \l__spath_tmpa_seq
    {
      \tl_set:Nn \l__spath_tmpj_tl {##2}
      \int_compare:nT {##1 = #2}
      {
        \prg_replicate:nn {3}
        {
          \tl_set:Nx \l__spath_tmpj_tl {\tl_tail:N \l__spath_tmpj_tl}
        }
      }
      \seq_put_right:NV \l__spath_tmpb_seq \l__spath_tmpj_tl
    }

    \seq_set_eq:NN \l__spath_tmpa_seq \l__spath_tmpb_seq
  }

  \tl_gset:Nx \g__spath_output_tl {\seq_use:Nn \l__spath_tmpa_seq {} }
  \group_end:
}
\cs_new_protected_nopar:Npn \spath_join_component:Nnn #1#2#3
{
  \__spath_join_component:nn {#2}{#3}
  \tl_set_eq:NN #1 \g__spath_output_tl
  \tl_gclear:N \g__spath_output_tl
}
\cs_generate_variant:Nn \spath_join_component:Nnn {NVn, NVV}
\cs_new_protected_nopar:Npn \spath_join_component:Nn #1#2
{
  \spath_join_component:NVn #1#1{#2}
}
\cs_generate_variant:Nn \spath_join_component:Nn {cn, NV, cV}
\cs_new_protected_nopar:Npn \spath_gjoin_component:Nnn #1#2#3
{
  \__spath_join_component:nn {#2}{#3}
  \tl_gset_eq:NN #1 \g__spath_output_tl
  \tl_gclear:N \g__spath_output_tl
}
\cs_generate_variant:Nn \spath_gjoin_component:Nnn {NVn, NVV}
\cs_new_protected_nopar:Npn \spath_gjoin_component:Nn #1#2
{
  \spath_gjoin_component:NVn #1#1{#2}
}
\cs_generate_variant:Nn \spath_gjoin_component:Nn {cn, NV, cV}
```

(*End definition for* `\spath_join_component:Nnn` *and others. These functions are documented on page* **??**.)

\spath_spot_weld_components:Nn
\spath_spot_weld_components:N
\spath_spot_gweld_components:Nn
\spath_spot_gweld_components:N

Weld together any components where the last point of one is at the start point of the next (within a tolerance).

```
\cs_new_protected_nopar:Npn \__spath_spot_weld_components:n #1
{
  \group_begin:
  \dim_zero:N \l__spath_move_x_dim
  \dim_zero:N \l__spath_move_y_dim

  \spath_components_to_seq:Nn \l__spath_tmpa_seq {#1}
  \seq_clear:N \l__spath_tmpb_seq
  \dim_set:Nn \l__spath_move_x_dim {\tl_item:nn {#1} {2} + 10 pt}
  \dim_set:Nn \l__spath_move_y_dim {\tl_item:nn {#1} {3} + 10 pt}

  \int_set:Nn \l__spath_tmpa_int {\seq_count:N \l__spath_tmpa_seq}

```

```
\seq_map_inline:Nn \l__spath_tmpa_seq
{
  \tl_set:Nn \l__spath_tmpa_tl {##1}
  \bool_if:nT
  {
    \dim_compare_p:n
    {
      \dim_abs:n {\l__spath_move_x_dim - \tl_item:Nn \l__spath_tmpa_tl {2} } < 0.01pt
    }
    &&
    \dim_compare_p:n
    {
      \dim_abs:n {\l__spath_move_y_dim - \tl_item:Nn \l__spath_tmpa_tl {3} } < 0.01pt
    }
  }
  {
    \prg_replicate:nn {3}
    {
      \tl_set:Nx \l__spath_tmpa_tl {\tl_tail:N \l__spath_tmpa_tl}
    }
    \int_decr:N \l__spath_tmpa_int
  }
  \tl_reverse:N \l__spath_tmpa_tl
  \dim_set:Nn \l__spath_move_x_dim {\tl_item:Nn \l__spath_tmpa_tl {2}}
  \dim_set:Nn \l__spath_move_y_dim {\tl_item:Nn \l__spath_tmpa_tl {1}}
  \tl_reverse:N \l__spath_tmpa_tl
  \seq_put_right:NV \l__spath_tmpb_seq \l__spath_tmpa_tl
}

\tl_set:Nx \l__spath_tmpa_tl {\seq_use:Nn \l__spath_tmpb_seq {} }
\spath_components_to_seq:NV \l__spath_tmpb_seq \l__spath_tmpa_tl


\spath_initialpoint:Nn \l__spath_tmpa_tl {#1}
\spath_finalpoint:Nn \l__spath_tmpb_tl {#1}

\bool_if:nT
{
  \dim_compare_p:n
  {
    \dim_abs:n {\tl_item:Nn \l__spath_tmpa_tl {1} - \tl_item:Nn \l__spath_tmpb_tl {1} } <
  }
  &&
  \dim_compare_p:n
  {
    \dim_abs:n {\tl_item:Nn \l__spath_tmpa_tl {2} - \tl_item:Nn \l__spath_tmpb_tl {2} } <
  }
}
{
  \int_compare:nTF
  {
    \seq_count:N \l__spath_tmpb_seq > 1
  }
  {
      \seq_pop_left:NN \l__spath_tmpb_seq \l__spath_tmpb_tl

      \prg_replicate:nn {3}
      {
        \tl_set:Nx \l__spath_tmpb_tl {\tl_tail:N \l__spath_tmpb_tl}
      }
      \seq_put_right:NV \l__spath_tmpb_seq \l__spath_tmpb_tl
    }
    {
      \tl_set:NV \l__spath_tmpb_tl \c_spath_closepath_tl
      \tl_put_right:Nx \l__spath_tmpb_tl
      {
        { \tl_item:Nn \l__spath_tmpa_tl {1} }
        { \tl_item:Nn \l__spath_tmpa_tl {2} }
      }
      \seq_put_right:NV \l__spath_tmpb_seq \l__spath_tmpb_tl
    }
  }

  \tl_gset:Nx \g__spath_output_tl {\seq_use:Nn \l__spath_tmpb_seq {}}
  \group_end:
}
\cs_new_protected_nopar:Npn \spath_spot_weld_components:Nn #1#2
{
  \__spath_spot_weld_components:n {#2}
  \tl_set_eq:NN #1 \g__spath_output_tl
  \tl_gclear:N \g__spath_output_tl
}
\cs_generate_variant:Nn \spath_spot_weld_components:Nn {NV, cV, cn}
\cs_new_protected_nopar:Npn \spath_spot_weld_components:N #1
{
  \spath_spot_weld_components:NV #1#1
}
\cs_generate_variant:Nn \spath_spot_weld_components:N {c}
\cs_new_protected_nopar:Npn \spath_spot_gweld_components:Nn #1#2
{
  \__spath_spot_weld_components:n {#2}
  \tl_gset_eq:NN #1 \g__spath_output_tl
  \tl_gclear:N \g__spath_output_tl
}
\cs_generate_variant:Nn \spath_spot_gweld_components:Nn {NV, cV, cn}
\cs_new_protected_nopar:Npn \spath_spot_gweld_components:N #1
{
  \spath_spot_gweld_components:NV #1#1
}
\cs_generate_variant:Nn \spath_spot_gweld_components:N {c}
```

(*End definition for* `\spath_spot_weld_components:Nn` *and others. These functions are documented on page* **??**.)

3.8 Exporting Commands

`\spath_convert_to_svg:Nn`
`\spath_gconvert_to_svg:Nn`

Convert the soft path to an SVG document.

```
\cs_new_protected_nopar:Npn \__spath_convert_to_svg:n #1
```

```
{
  \group_begin:
  \tl_clear:N \l__spath_tmpa_tl
  \tl_put_right:Nn \l__spath_tmpa_tl
  {
    <?xml~ version="1.0"~ standalone="no"?>
    \iow_newline:
    <!DOCTYPE~ svg~ PUBLIC~ "-//W3C//DTD SVG 1.1//EN"~
    "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
    \iow_newline:
    <svg~ xmlns="http://www.w3.org/2000/svg"~ version="1.1"~viewBox="
  }

  \spath_minbb:Nn \l__spath_tmpb_tl {#1}
  \spath_maxbb:Nn \l__spath_tmpc_tl {#1}
  \tl_put_right:Nx \l__spath_tmpa_tl
  {
    \dim_to_decimal:n
    {
      \tl_item:Nn \l__spath_tmpb_tl {1} - 10pt
    }
    \exp_not:n {~}
    \dim_to_decimal:n
    {
      \tl_item:Nn \l__spath_tmpb_tl {2} - 10pt
    }
    \exp_not:n {~}
    \dim_to_decimal:n
    {
      \tl_item:Nn \l__spath_tmpc_tl {1}
      -
      \tl_item:Nn \l__spath_tmpb_tl {1}
      + 20pt
    }
    \exp_not:n {~}
    \dim_to_decimal:n
    {
      \tl_item:Nn \l__spath_tmpc_tl {2}
      -
      \tl_item:Nn \l__spath_tmpb_tl {2}
      + 20pt
    }
  }

  \tl_put_right:Nn \l__spath_tmpa_tl
  {
    ">
    \iow_newline:
    <path~ d="
  }
  \tl_set:Nn \l__spath_tmpc_tl {use:n}
  \tl_map_inline:Nn #1
  {
    \tl_set:Nn \l__spath_tmpb_tl {##1}
    \tl_case:NnF \l__spath_tmpb_tl
    {
      \c_spath_moveto_tl
      {
        \tl_put_right:Nn \l__spath_tmpa_tl {M~}
        \tl_set:Nn \l__spath_tmpc_tl {use:n}
      }
      \c_spath_lineto_tl
      {
        \tl_put_right:Nn \l__spath_tmpa_tl {L~}
        \tl_set:Nn \l__spath_tmpc_tl {use:n}
      }
      \c_spath_closepath_tl
      {
        \tl_put_right:Nn \l__spath_tmpa_tl {Z~}
        \tl_set:Nn \l__spath_tmpc_tl {use_none:n}
      }
      \c_spath_curvetoa_tl
      {
        \tl_put_right:Nn \l__spath_tmpa_tl {C~}
        \tl_set:Nn \l__spath_tmpc_tl {use:n}
      }
      \c_spath_curvetob_tl {
        \tl_set:Nn \l__spath_tmpc_tl {use:n}
      }
      \c_spath_curveto_tl {
        \tl_set:Nn \l__spath_tmpc_tl {use:n}
      }
    }
    {
      \tl_put_right:Nx \l__spath_tmpa_tl {\use:c { \l__spath_tmpc_tl } {\dim_to_decimal:n {#
    }
  }
  \tl_put_right:Nn \l__spath_tmpa_tl
  {
    "~ fill="none"~ stroke="black"~ />
    \iow_newline:
    </svg>
    \iow_newline:
  }
  \tl_gset_eq:NN \g__spath_output_tl \l__spath_tmpa_tl
  \group_end:
}
\cs_new_protected_nopar:Npn \spath_convert_to_svg:Nn #1#2
{
  \__spath_convert_to_svg:n {#2}
  \tl_set_eq:NN #1 \g__spath_output_tl
  \tl_gclear:N \g__spath_output_tl
}
\cs_new_protected_nopar:Npn \spath_gconvert_to_svg:Nn #1#2
{
  \__spath_convert_to_svg:n {#2}
  \tl_gset_eq:NN #1 \g__spath_output_tl
  \tl_gclear:N \g__spath_output_tl
}
```

(*End definition for* `\spath_convert_to_svg:Nn` *and* `\spath_gconvert_to_svg:Nn`*. These functions are documented on page* **??**.)

`\spath_export_to_svg:nn` Save a soft path to an SVG file.

```
\iow_new:N \g__spath_stream
\cs_new_protected_nopar:Npn \spath_export_to_svg:nn #1#2
{
  \spath_convert_path_to_svg:Nn \l__spath_iterp_tl {#2}
  \iow_open:Nn \g__spath_stream {#1 .svg}
  \iow_now:Nx \g__spath_stream
  {
    \tl_use:N \l__spath_iterp_tl
  }
  \iow_close:N \g__spath_stream
}
```

(*End definition for* `\spath_export_to_svg:nn`*. This function is documented on page* **??**.)

`\spath_show:n` Displays the soft path on the terminal.

```
\cs_new_protected_nopar:Npn \spath_show:n #1
{
  \int_step_inline:nnnn {1} {3} {\tl_count:n {#1}}
  {
    \iow_term:x {
      \tl_item:Nn \l__spath_tmpa_tl {##1}
      {\tl_item:Nn \l__spath_tmpa_tl {##1+1}}
      {\tl_item:Nn \l__spath_tmpa_tl {##1+2}}
    }
  }
}
\cs_generate_variant:Nn \spath_show:n {V, v}
```

(*End definition for* `\spath_show:n`*. This function is documented on page* **??**.)

3.8.1 PGF and TikZ Interface Functions

Spaths come from PGF so we need some functions that get and set spaths from the pgf system.

`\spath_get_current_path:N`
`\spath_gget_current_path:N`

Grab the current soft path from PGF.

```
\cs_new_protected_nopar:Npn \spath_get_current_path:N #1
{
  \pgfsyssoftpath@getcurrentpath #1
}
```

```
\cs_new_protected_nopar:Npn \spath_gget_current_path:N #1
{
  \pgfsyssoftpath@getcurrentpath #1
  \tl_gset_eq:NN #1 #1
}
```

(*End definition for* `\spath_get_current_path:N` *and* `\spath_gget_current_path:N`*. These functions are documented on page* **??**.)

`\spath_protocol_path:n`

This feeds the bounding box of the soft path to PGF to ensure that its current bounding box contains the soft path.

```
\cs_new_protected_nopar:Npn \spath_protocol_path:n #1
{
  \spath_minbb:Nn \l__spath_tmpa_tl {#1}
  \exp_last_unbraced:NV \pgf@protocolsizes\l__spath_tmpa_tl

  \spath_maxbb:Nn \l__spath_tmpa_tl {#1}
  \exp_last_unbraced:NV \pgf@protocolsizes\l__spath_tmpa_tl
}
\cs_generate_variant:Nn \spath_protocol_path:n {V}
```

(*End definition for* `\spath_protocol_path:n`*. This function is documented on page* **??**.)

`\spath_set_current_path:n`
`\spath_set_current_path:N`

Sets the current path to the specified soft path.

```
\cs_new_protected_nopar:Npn \spath_set_current_path:n #1
{
  \spath_protocol_path:n {#1}
  \tl_set:Nn \l__spath_tmpa_tl {#1}
  \pgfsyssoftpath@setcurrentpath\l__spath_tmpa_tl
}
\cs_new_protected_nopar:Npn \spath_set_current_path:N #1
{
  \spath_protocol_path:V #1
  \pgfsyssoftpath@setcurrentpath #1
}
\cs_generate_variant:Nn \spath_set_current_path:N {c}
```

(*End definition for* `\spath_set_current_path:n` *and* `\spath_set_current_path:N`*. These functions are documented on page* **??**.)

`\spath_use_path:nn`

Uses the given soft path at the PGF level.

```
\cs_new_protected_nopar:Npn \spath_use_path:nn #1#2
{
  \spath_set_current_path:n {#1}
  \pgfusepath{#2}
}
```

(*End definition for* `\spath_use_path:nn`*. This function is documented on page* **??**.)

`\spath_tikz_path:nn`

Uses the given soft path at the TikZ level.

```
\cs_new_protected_nopar:Npn \spath_tikz_path:nn #1#2
{
  \path[#1] \pgfextra{
    \spath_set_current_path:n {#2}
    \tl_put_right:Nn \tikz@preactions {\def\tikz@actions@path{#2}}
  };
}
\cs_generate_variant:Nn \spath_tikz_path:nn {Vn, VV, nv, Vv, nV}
```

(*End definition for* `\spath_tikz_path:nn`*. This function is documented on page* **??**.)

`\spath_set_tikz_coords:n` Sets the `\tikz@lastx` and other coordinates from the soft path.

```
\cs_new_protected_nopar:Npn \spath_set_tikz_coords:n #1
{
  \spath_finalpoint:Nn \l__spath_tmpa_tl {#1}
  \tl_set:Nx \l__spath_tmpa_tl
  {
    \exp_not:c {tikz@lastx}=\tl_item:Nn \l__spath_tmpa_tl {1}
    \exp_not:c {tikz@lasty}=\tl_item:Nn \l__spath_tmpa_tl {2}
    \exp_not:c {tikz@lastxsaved}=\tl_item:Nn \l__spath_tmpa_tl {1}
    \exp_not:c {tikz@lastysaved}=\tl_item:Nn \l__spath_tmpa_tl {2}
  }
  \tl_use:N \l__spath_tmpa_tl
}
\cs_generate_variant:Nn \spath_set_tikz_coords:n {V, v}
```

(*End definition for* `\spath_set_tikz_coords:n`. *This function is documented on page* **??**.)

4 The TikZ interface

This provides an interface to the soft path manipulation routines via a series of TikZ keys. They all live in the `spath` family.

```
\RequirePackage{spath3}
\RequirePackage{expl3}
\ExplSyntaxOn

\tl_new:N \l__spath_current_tl
\tl_new:N \l__spath_reverse_tl
\tl_new:N \l__spath_prefix_tl
\tl_new:N \l__spath_suffix_tl
\tl_new:N \g__spath_smuggle_tl
\seq_new:N \g__spath_tmpa_seq
\seq_new:N \g__spath_tmpb_seq
\bool_new:N \l__spath_draft_bool
```

When saving a soft path, by default we use a naming convention that is compatible with the intersections library so that paths saved here and paths saved by the `name path` facility of the intersections library are mutually exchangeable.

```
\tl_set:Nn \l__spath_prefix_tl {tikz@intersect@path@name@}
\tl_set:Nn \l__spath_suffix_tl {}
```

When a soft path is grabbed from TikZ we're usually deep in a group so I've adapted the code from the intersections library to dig the definition out of the group without making everything global.

```
\tl_new:N \g__spath_tikzfinish_tl
\cs_new_protected_nopar:Npn \spath_at_end_of_path:
{
  \tl_use:N \g__spath_tikzfinish_tl
  \tl_gclear:N \g__spath_tikzfinish_tl
}
\tl_put_right:Nn \tikz@finish {\spath_at_end_of_path:}

\cs_new_protected_nopar:Npn \spath_save_path:Nn #1#2
```

```
{
  \tl_gput_right:Nn \g__spath_tikzfinish_tl
  {
    \tl_clear_new:N #1
    \tl_set:Nn #1 {#2}
  }
}
\cs_generate_variant:Nn \spath_save_path:Nn {cn, NV, cV}

\cs_new_protected_nopar:Npn \spath_gsave_path:Nn #1#2
{
  \tl_gput_right:Nn \g__spath_tikzfinish_tl
  {
    \tl_gclear_new:N #1
    \tl_gset:Nn #1 {#2}
  }
}
\cs_generate_variant:Nn \spath_gsave_path:Nn {cn, NV, cV}
```

Now we define all of our keys.

```
\tikzset{
```

We're in the `spath` key family.

```
  spath/.is~family,
  spath/.cd,
```

We provide for saving soft paths with a specific prefix and suffix in the name. The default is to make it compatible with the intersections library.

```
  set~ prefix/.store~ in=\l__spath_prefix_tl,
  prefix/.is~choice,
  prefix/default/.style={
    /tikz/spath/set~ prefix=tikz@intersect@path@name@
  },
  set~ suffix/.store~ in=\l__spath_suffix_tl,
  suffix/.is~choice,
  suffix/default/.style={
    /tikz/spath/set~ suffix={}
  },
  set~ name/.style={
    /tikz/spath/prefix=#1,
    /tikz/spath/suffix=#1
  },
```

Keys for saving and cloning a soft path.

```
  save/.code={
    \tikz@addmode{
      \spath_get_current_path:N \l__spath_tmpa_tl
      \spath_bake_round:NV \l__spath_tmpa_tl \l__spath_tmpa_tl
      \spath_save_path:cV {\tl_use:N \l__spath_prefix_tl #1 \tl_use:N \l__spath_suffix_tl} \
    }
  },
  clone/.code~ 2~ args={
    \tl_if_exist:cT {\tl_use:N \l__spath_prefix_tl #2 \tl_use:N \l__spath_suffix_tl}
    {
      \tl_clear_new:c
      {\tl_use:N \l__spath_prefix_tl #1 \tl_use:N \l__spath_suffix_tl}
      \tl_set_eq:cc
      {\tl_use:N \l__spath_prefix_tl #1 \tl_use:N \l__spath_suffix_tl}
      {\tl_use:N \l__spath_prefix_tl #2 \tl_use:N \l__spath_suffix_tl}
    }
  },
  clone~ global/.code~ 2~ args={
    \tl_if_exist:cT {\tl_use:N \l__spath_prefix_tl #2 \tl_use:N \l__spath_suffix_tl}
    {
      \tl_gclear_new:c
      {\tl_use:N \l__spath_prefix_tl #1 \tl_use:N \l__spath_suffix_tl}
      \tl_gset_eq:cc
      {\tl_use:N \l__spath_prefix_tl #1 \tl_use:N \l__spath_suffix_tl}
      {\tl_use:N \l__spath_prefix_tl #2 \tl_use:N \l__spath_suffix_tl}
    }
  },
  save~ global/.code={
    \tikz@addmode{
      \spath_get_current_path:N \l__spath_tmpa_tl
      \spath_bake_round:NV \l__spath_tmpa_tl \l__spath_tmpa_tl
      \spath_gsave_path:cV
      {\tl_use:N \l__spath_prefix_tl #1 \tl_use:N \l__spath_suffix_tl}
      \l__spath_tmpa_tl
    }
  },
```

Saves a soft path to the aux file.

```
  save~ to~ aux/.code={
    \tl_if_exist:cT {\tl_use:N \l__spath_prefix_tl #1 \tl_use:N \l__spath_suffix_tl}
    {
      \spath_save_to_aux:c
      {\tl_use:N \l__spath_prefix_tl #1 \tl_use:N \l__spath_suffix_tl}
    }
  },
```

Restores a soft path to the current path.

```
  restore/.code={
    \tl_if_exist:cT {\tl_use:N \l__spath_prefix_tl #1 \tl_use:N \l__spath_suffix_tl}
    {
      \spath_set_current_path:c
      {\tl_use:N \l__spath_prefix_tl #1 \tl_use:N \l__spath_suffix_tl}
      \spath_set_tikz_coords:v
      {\tl_use:N \l__spath_prefix_tl #1 \tl_use:N \l__spath_suffix_tl}
    }
  },
```

Diagnostic, show the current path in the terminal and log.

```
  show~current~path/.code={
    \tikz@addmode{
      \pgfsyssoftpath@getcurrentpath\l__spath_tmpa_tl
      \iow_term:n {----~ current~ soft~ path~ ---}
      \spath_show:V \l__spath_tmpa_tl
    }
  },
```

Diagnostic, show the named soft path in the terminal and log.

```
  show/.code={
    \tl_if_exist:cT {\tl_use:N \l__spath_prefix_tl #1 \tl_use:N \l__spath_suffix_tl}
    {
      \iow_term:n {---~ soft~ path~ #1~ ---}
      \spath_show:v {\tl_use:N \l__spath_prefix_tl #1 \tl_use:N \l__spath_suffix_tl}
    }
  },
```

Appends the named path to the current path with a weld.

```
  append/.code={
    \tl_if_exist:cT {\tl_use:N \l__spath_prefix_tl #1 \tl_use:N \l__spath_suffix_tl}
    {
      \spath_get_current_path:N \l__spath_current_tl
      \spath_weld:Nv
      \l__spath_current_tl
      {\tl_use:N \l__spath_prefix_tl #1 \tl_use:N \l__spath_suffix_tl}
      \spath_set_current_path:N \l__spath_current_tl
      \spath_set_tikz_coords:V \l__spath_current_tl
    }
  },
```

Joins the second named path to the first.

```
  join~ with/.code~ 2~ args={
    \tl_if_exist:cT {\tl_use:N \l__spath_prefix_tl #1 \tl_use:N \l__spath_suffix_tl}
    {
      \tl_if_exist:cT {\tl_use:N \l__spath_prefix_tl #2 \tl_use:N \l__spath_suffix_tl}
      {
        \spath_append:cv
        {\tl_use:N \l__spath_prefix_tl #1 \tl_use:N \l__spath_suffix_tl}
        {\tl_use:N \l__spath_prefix_tl #2 \tl_use:N \l__spath_suffix_tl}
      }
    }
  },
```

Does a "spot weld" on a soft path, which means that any components that start where the previous component ends are welded together.

```
  spot~ weld/.code={
    \tl_if_exist:cT {\tl_use:N \l__spath_prefix_tl #1 \tl_use:N \l__spath_suffix_tl}
    {
      \spath_spot_weld_components:c
      {\tl_use:N \l__spath_prefix_tl #1 \tl_use:N \l__spath_suffix_tl}
    }
  },
```

Reverses the named path.

```
  reverse/.code={
    \tl_if_exist:cT {\tl_use:N \l__spath_prefix_tl #1 \tl_use:N \l__spath_suffix_tl}
    {
      \spath_reverse:c
      {\tl_use:N \l__spath_prefix_tl #1 \tl_use:N \l__spath_suffix_tl}
    }
  },
```

Appends the reversal of the path to the current path.

```
  append~ reverse/.code={
    \tl_if_exist:cT {\tl_use:N \l__spath_prefix_tl #1 \tl_use:N \l__spath_suffix_tl}
    {
      \spath_reverse:Nv
      \l__spath_reverse_tl
      {\tl_use:N \l__spath_prefix_tl #1 \tl_use:N \l__spath_suffix_tl}
      \spath_get_current_path:N \l__spath_current_tl
      \spath_weld:NV \l__spath_current_tl \l__spath_reverse_tl
      \spath_set_current_path:N \l__spath_current_tl
      \spath_set_tikz_coords:V \l__spath_current_tl
    }
  },
```

Inserts the named path into the current path as-is, meaning without transforming or welding it.

```
  insert/.code={
    \tl_if_exist:cT {\tl_use:N \l__spath_prefix_tl #1 \tl_use:N \l__spath_suffix_tl}
    {
      \spath_get_current_path:N \l__spath_current_tl
      \spath_append:Nv
      \l__spath_current_tl
      {\tl_use:N \l__spath_prefix_tl #1 \tl_use:N \l__spath_suffix_tl}
      \spath_set_current_path:N \l__spath_current_tl
      \spath_set_tikz_coords:V \l__spath_current_tl
    }
  },
```

Inserts the reverse of the named path.

```
  insert~ reverse/.code={
    \tl_if_exist:cT {\tl_use:N \l__spath_prefix_tl #1 \tl_use:N \l__spath_suffix_tl}
    {
      \spath_reverse:Nv
      \l__spath_reverse_tl
      {\tl_use:N \l__spath_prefix_tl #1 \tl_use:N \l__spath_suffix_tl}
      \spath_get_current_path:N \l__spath_current_tl
      \spath_append:NV \l__spath_current_tl \l__spath_reverse_tl
      \spath_set_current_path:N \l__spath_current_tl
      \spath_set_tikz_coords:V \l__spath_current_tl
    }
  },
```

These keys shorten the path.

```
  shorten~ at~ end/.code~ 2~ args={
    \tl_if_exist:cT {\tl_use:N \l__spath_prefix_tl #1 \tl_use:N \l__spath_suffix_tl}
    {
      \spath_shorten_at_end:cn
      {\tl_use:N \l__spath_prefix_tl #1 \tl_use:N \l__spath_suffix_tl} {#2}
    }
  },
  shorten~ at~ start/.code~ 2~ args ={
    \tl_if_exist:cT {\tl_use:N \l__spath_prefix_tl #1 \tl_use:N \l__spath_suffix_tl}
    {
      \spath_shorten_at_start:cn
      {\tl_use:N \l__spath_prefix_tl #1 \tl_use:N \l__spath_suffix_tl} {#2}
    }
  },
  shorten~ at~ both~ ends/.code~ 2~ args={
    \tl_if_exist:cT {\tl_use:N \l__spath_prefix_tl #1 \tl_use:N \l__spath_suffix_tl}
    {
      \spath_shorten_at_end:cn
      {\tl_use:N \l__spath_prefix_tl #1 \tl_use:N \l__spath_suffix_tl} {#2}
      \spath_shorten_at_start:cn
      {\tl_use:N \l__spath_prefix_tl #1 \tl_use:N \l__spath_suffix_tl} {#2}
    }
  },
```

This translates the named path.

```
  translate/.code~ n~ args={3}{
    \tl_if_exist:cT {\tl_use:N \l__spath_prefix_tl #1 \tl_use:N \l__spath_suffix_tl}
    {
      \spath_translate:cnn
      {\tl_use:N \l__spath_prefix_tl #1 \tl_use:N \l__spath_suffix_tl}{#2}{#3}
    }
  },
```

Exports the path as an SVG file.

```
  export~ to~ svg/.code={
    \tl_if_exist:cT {\tl_use:N \l__spath_prefix_tl #1 \tl_use:N \l__spath_suffix_tl}
    {
      \spath_export_to_svg:c
      {\tl_use:N \l__spath_prefix_tl #1 \tl_use:N \l__spath_suffix_tl}
    }
  },
```

Transforms the named path using TikZ transformation specifications.

```
  transform/.code~ 2~ args={
    \tl_if_exist:cT {\tl_use:N \l__spath_prefix_tl #1 \tl_use:N \l__spath_suffix_tl}
    {
      \group_begin:
      \pgftransformreset
      \tikzset{#2}
      \pgfgettransform \l__spath_tmpa_tl
      \tl_gset:Nn \g__spath_smuggle_tl
      {
        \spath_transform:cnnnnnn
        {\tl_use:N \l__spath_prefix_tl #1 \tl_use:N \l__spath_suffix_tl}
      }
      \tl_gput_right:NV \g__spath_smuggle_tl \l__spath_tmpa_tl
      \group_end:
      \tl_use:N \g__spath_smuggle_tl
    }
  },
  transform~global/.code~ 2~ args={
    \tl_if_exist:cT {\tl_use:N \l__spath_prefix_tl #1 \tl_use:N \l__spath_suffix_tl}
    {
      \group_begin:
      \pgftransformreset
      \tikzset{#2}
      \pgfgettransform \l__spath_tmpa_tl
      \tl_gset:Nn \g__spath_smuggle_tl
      {
        \spath_gtransform:cnnnnnn
        {\tl_use:N \l__spath_prefix_tl #1 \tl_use:N \l__spath_suffix_tl}
      }
      \tl_gput_right:NV \g__spath_smuggle_tl \l__spath_tmpa_tl
      \group_end:
      \tl_use:N \g__spath_smuggle_tl
    }
  },
```

Splits two paths at their mutual intersections.

```
  split~ at~ intersections/.code~ n~ args={2}{
    \tl_if_exist:cT {\tl_use:N \l__spath_prefix_tl #1 \tl_use:N \l__spath_suffix_tl}
    {
      \tl_if_exist:cT {\tl_use:N \l__spath_prefix_tl #2 \tl_use:N \l__spath_suffix_tl}
      {
        \spath_split_at_intersections:cc
        {\tl_use:N \l__spath_prefix_tl #1 \tl_use:N \l__spath_suffix_tl}
        {\tl_use:N \l__spath_prefix_tl #2 \tl_use:N \l__spath_suffix_tl}
      }
    }
  },
```

Splits a path at its self-intersections.

```
  split~ at~ self~ intersections/.code={
    \tl_if_exist:cT {\tl_use:N \l__spath_prefix_tl #1 \tl_use:N \l__spath_suffix_tl}
    {
      \spath_split_at_self_intersections:c
      {\tl_use:N \l__spath_prefix_tl #1 \tl_use:N \l__spath_suffix_tl}
    }
  },
```

Extract the components of a path into a comma separated list (suitable for using in a `\foreach` loop).

```
  get~ components~ of/.code~ 2~ args={
    \tl_if_exist:cT {\tl_use:N \l__spath_prefix_tl #1 \tl_use:N \l__spath_suffix_tl}
    {
      \clist_clear_new:N #2
      \spath_components_to_seq:Nv
      \l__spath_tmpa_seq
      {\tl_use:N \l__spath_prefix_tl #1 \tl_use:N \l__spath_suffix_tl}
      \seq_map_inline:Nn \l__spath_tmpa_seq
      {
        \tl_new:c
        {\tl_use:N \l__spath_prefix_tl anonymous_\int_use:N \g__spath_anon_int \tl_use:N \l_
        \tl_set:cn
        {\tl_use:N \l__spath_prefix_tl anonymous_\int_use:N \g__spath_anon_int \tl_use:N \l_
        \clist_put_right:Nx #2 {anonymous_\int_use:N \g__spath_anon_int}
        \int_gincr:N \g__spath_anon_int
      }
    }
  },
```

Loop through the components of a soft path and render each as a separate TikZ path so that they can be individually styled.

```
render~ components/.code={
  \tl_if_exist:cT {\tl_use:N \l__spath_prefix_tl #1 \tl_use:N \l__spath_suffix_tl}
  {
    \group_begin:
    \spath_components_to_seq:Nv
    \l__spath_tmpa_seq
    {\tl_use:N \l__spath_prefix_tl #1 \tl_use:N \l__spath_suffix_tl}
    \seq_map_indexed_inline:Nn \l__spath_tmpa_seq
    {
      \spath_tikz_path:nn
      {
        every~ spath~ component/.try,
        spath ~component~ ##1/.try,
        spath ~component/.try={##1},
        every~ #1~ component/.try,
        #1 ~component~ ##1/.try,
        #1 ~component/.try={##1},
      }
      {
        ##2
      }
    }
    \group_end:
  }
},
```

This puts gaps between components of a soft path. The list of components is passed through a `\foreach` loop so can use the shortcut syntax from those loops.

```
insert~ gaps~ after~ components/.code~ n~ args={3}{
  \tl_if_exist:cT {\tl_use:N \l__spath_prefix_tl #1 \tl_use:N \l__spath_suffix_tl}
  {
    \group_begin:
    \seq_gclear:N \g__spath_tmpa_seq
    \seq_gclear:N \g__spath_tmpb_seq
    \foreach \l__spath_tmpa_tl in {#3}
    {
      \seq_gput_right:NV \g__spath_tmpa_seq \l__spath_tmpa_tl
      \seq_gput_right:Nx \g__spath_tmpb_seq {\int_eval:n { \l__spath_tmpa_tl + 1 }}
    }
    \spath_components_to_seq:Nv
    \l__spath_tmpa_seq
    {\tl_use:N \l__spath_prefix_tl #1 \tl_use:N \l__spath_suffix_tl}
    \seq_clear:N \l__spath_tmpb_seq
    \seq_map_indexed_inline:Nn \l__spath_tmpa_seq
    {
      \tl_set:Nn \l__spath_tmpa_tl {##2}
      \seq_if_in:NnT \g__spath_tmpa_seq {##1}
      {
        \spath_shorten_at_end:Nn \l__spath_tmpa_tl {#2/2}
      }
      \seq_if_in:NnT \g__spath_tmpb_seq {##1}
      {
        \spath_shorten_at_start:Nn \l__spath_tmpa_tl {#2/2}
      }
      \seq_put_right:NV \l__spath_tmpb_seq \l__spath_tmpa_tl
    }
    \tl_gset:Nx \g__spath_output_tl {\seq_use:Nn \l__spath_tmpb_seq {} }
    \group_end:
    \tl_set_eq:cN
    {\tl_use:N \l__spath_prefix_tl #1 \tl_use:N \l__spath_suffix_tl}
    \g__spath_output_tl
    \tl_gclear:N \g__spath_output_tl
  }
},
```

Join the specified components together, joining each to its previous one.

```
join~ components/.code~ 2~ args={
  \tl_if_exist:cT {\tl_use:N \l__spath_prefix_tl #1 \tl_use:N \l__spath_suffix_tl}
  {
    \seq_gclear:N \g__spath_tmpa_seq
    \foreach \l__spath_tmpa_tl in {#2}
    {
      \seq_gput_right:NV \g__spath_tmpa_seq \l__spath_tmpa_tl
    }
    \seq_gsort:Nn \g__spath_tmpa_seq
    {
      \int_compare:nNnTF {##1} > {##2}
      { \sort_return_same: }
      { \sort_return_swapped: }
    }
    \seq_map_inline:Nn \g__spath_tmpa_seq
    {
      \spath_join_component:cn
      {\tl_use:N \l__spath_prefix_tl #1 \tl_use:N \l__spath_suffix_tl}{##1}
    }
  }
},
```

Remove all components of the path that don't actually draw anything.

```
remove~ empty~ components/.code={
  \tl_if_exist:cT {\tl_use:N \l__spath_prefix_tl #1 \tl_use:N \l__spath_suffix_tl}
  {
    \spath_remove_empty_components:c
    {\tl_use:N \l__spath_prefix_tl #1 \tl_use:N \l__spath_suffix_tl}
  }
},
```

This puts a conditional around the `spot weld` key because when figuring out a knot drawing then we will initially want to render it without the spot weld to keep the number of components constant.

```
draft~ mode/.is~ choice,
draft~ mode/true/.code={
  \bool_set_true:N \l__spath_draft_bool
},
draft~ mode/false/.code={
  \bool_set_false:N \l__spath_draft_bool
},
maybe~ spot~ weld/.code={
  \bool_if:NF \l__spath_draft_bool
  {
    \tl_if_exist:cT {\tl_use:N \l__spath_prefix_tl #1 \tl_use:N \l__spath_suffix_tl}
    {
      \spath_spot_weld_components:c
      {\tl_use:N \l__spath_prefix_tl #1 \tl_use:N \l__spath_suffix_tl}
    }
  }
},
```

Set the transformation to lie along a path.

```
  transform~ to/.code~ 2~ args={
    \group_begin:
    \tl_if_exist:cTF
    {
      \tl_use:N \l__spath_prefix_tl
      #1
      \tl_use:N \l__spath_suffix_tl
    }
    {
      \spath_reallength:Nv
      \l__spath_tmpa_int
      {
        \tl_use:N \l__spath_prefix_tl
        #1
        \tl_use:N \l__spath_suffix_tl
      }

      \tl_set:Nx \l__spath_tmpb_tl
      {\fp_to_decimal:n {(#2) * (\l__spath_tmpa_int)}}
      \spath_transformation_at:NvV \l__spath_tmpc_tl
      {
        \tl_use:N \l__spath_prefix_tl
        #1
        \tl_use:N \l__spath_suffix_tl
      }
      \l__spath_tmpb_tl
      \tl_gset_eq:NN \g__spath_smuggle_tl \l__spath_tmpc_tl
    }
    {
      \tl_gset_eq:NN \g__spath_smuggle_tl { {1}{0}{0}{1}{0pt}{0pt} }
    }
    \group_end:
    \exp_last_unbraced:NV \pgfsettransformentries \g__spath_smuggle_tl
    \tl_gclear:N \g__spath_smuggle_tl
  },
```

As above, but with a possible extra 180° rotation if needed to ensure that the new y–axis points vaguely upwards.

```
  upright~ transform~ to/.code~ 2~ args={
    \group_begin:
    \tl_if_exist:cTF
    {
      \tl_use:N \l__spath_prefix_tl
      #1
      \tl_use:N \l__spath_suffix_tl
    }
    {
      \spath_reallength:Nv
      \l__spath_tmpa_int
      {
        \tl_use:N \l__spath_prefix_tl
        #1
        \tl_use:N \l__spath_suffix_tl
      }

      \tl_set:Nx \l__spath_tmpb_tl
      {\fp_to_decimal:n {(#2) * (\l__spath_tmpa_int)}}
      \spath_transformation_at:NvV \l__spath_tmpc_tl
      {
        \tl_use:N \l__spath_prefix_tl
        #1
        \tl_use:N \l__spath_suffix_tl
      }
      \l__spath_tmpb_tl
      \tl_gset_eq:NN \g__spath_smuggle_tl \l__spath_tmpc_tl
    }
    {
      \tl_gset_eq:NN \g__spath_smuggle_tl { {1}{0}{0}{1}{0pt}{0pt} }
    }
    \fp_compare:nT { \tl_item:Nn \g__spath_smuggle_tl {4} < 0}
    {
      \tl_gset:Nx \g__spath_smuggle_tl
      {
        { \fp_eval:n { - (\tl_item:Nn \g__spath_smuggle_tl {1})} }
        { \fp_eval:n { - (\tl_item:Nn \g__spath_smuggle_tl {2})} }
        { \fp_eval:n { - (\tl_item:Nn \g__spath_smuggle_tl {3})} }
        { \fp_eval:n { - (\tl_item:Nn \g__spath_smuggle_tl {4})} }
        { \tl_item:Nn \g__spath_smuggle_tl {5} }
        { \tl_item:Nn \g__spath_smuggle_tl {6} }
      }
    }
    \group_end:
    \exp_last_unbraced:NV \pgfsettransformentries \g__spath_smuggle_tl
    \tl_gclear:N \g__spath_smuggle_tl
  },
```

This is a useful set of styles for drawing a knot diagram.

```
  knot/.style~ n~ args={3}{
    spath/.cd,
    split~ at~ self~ intersections=#1,
    insert~ gaps~ after~ components={#1}{#2}{#3},
    maybe~ spot~ weld=#1,
    render~ components=#1
  },
}
```

This defines a coordinate system that finds a position on a soft path.

```
\tikzdeclarecoordinatesystem{spath}{%
  \group_begin:
  \tl_set:Nn \l__spath_tmpa_tl {#1}
  \tl_trim_spaces:N \l__spath_tmpa_tl

  \seq_set_split:NnV \l__spath_tmpa_seq {~} \l__spath_tmpa_tl
  \seq_pop_right:NN \l__spath_tmpa_seq \l__spath_tmpb_tl

  \tl_set:Nx \l__spath_tmpa_tl { \seq_use:Nn \l__spath_tmpa_seq {~} }
  \tl_if_exist:cTF
  {
    \tl_use:N \l__spath_prefix_tl
    \tl_use:N \l__spath_tmpa_tl
    \tl_use:N \l__spath_suffix_tl
  }
  {

    \tl_set_eq:Nc
    \l__spath_tmpa_tl
    {
      \tl_use:N \l__spath_prefix_tl
      \tl_use:N \l__spath_tmpa_tl
      \tl_use:N \l__spath_suffix_tl
    }

    \tl_if_empty:NTF \l__spath_tmpa_tl
    {
      \tl_gset_eq:NN \g__spath_smuggle_tl \pgfpointorigin
    }
    {
      \spath_reallength:NV \l__spath_tmpa_int \l__spath_tmpa_tl
      \tl_set:Nx \l__spath_tmpb_tl
      {\fp_to_decimal:n {(\l__spath_tmpb_tl) * (\l__spath_tmpa_int)}}
      \spath_point_at:NVV \l__spath_tmpc_tl \l__spath_tmpa_tl \l__spath_tmpb_tl

      \tl_clear:N \l__spath_tmpd_tl
      \tl_put_right:Nn \l__spath_tmpd_tl {\pgf@x=}
      \tl_put_right:Nx \l__spath_tmpd_tl {\tl_item:Nn \l__spath_tmpc_tl {1}}
      \tl_put_right:Nn \l__spath_tmpd_tl {\relax}
      \tl_put_right:Nn \l__spath_tmpd_tl {\pgf@y=}
      \tl_put_right:Nx \l__spath_tmpd_tl {\tl_item:Nn \l__spath_tmpc_tl {2}}
      \tl_put_right:Nn \l__spath_tmpd_tl {\relax}
      \tl_gset_eq:NN \g__spath_smuggle_tl \l__spath_tmpd_tl
    }
  }
  {
    \tl_gset_eq:NN \g__spath_smuggle_tl \pgfpointorigin
  }
    \group_end:
    \tl_use:N \g__spath_smuggle_tl
}

\ExplSyntaxOff
```

5 The Calligraphy Package

```
⟨@@=cal⟩
```

5.1 Initialisation

```
\RequirePackage{spath3}
\ExplSyntaxOn

\tl_new:N \l__cal_tmpa_tl
\tl_new:N \l__cal_tmpb_tl
\tl_new:N \l__cal_tmp_path_tl
\tl_new:N \l__cal_tmp_rpath_tl
\tl_new:N \l__cal_tmp_rpathb_tl
\tl_new:N \l__cal_tmp_patha_tl

\seq_new:N \l__cal_tmpa_seq

\int_new:N \l__cal_tmpa_int
\int_new:N \l__cal_tmpb_int
\int_new:N \g__cal_path_component_int
\int_new:N \g__cal_label_int

\fp_new:N \l__cal_tmpa_fp
\fp_new:N \l__cal_tmpb_fp
\fp_new:N \l__cal_tmpc_fp
\fp_new:N \l__cal_tmpd_fp
\fp_new:N \l__cal_tmpe_fp

\dim_new:N \l__cal_tmpa_dim
\dim_new:N \l__cal_tmpb_dim
\dim_new:N \l__cal_tmpc_dim
\dim_new:N \l__cal_tmpd_dim
\dim_new:N \l__cal_tmpe_dim
\dim_new:N \l__cal_tmpf_dim
\dim_new:N \l__cal_tmpg_dim
\dim_new:N \l__cal_tmph_dim

\bool_new:N \l__cal_annotate_bool
\bool_new:N \l__cal_taper_start_bool
\bool_new:N \l__cal_taper_end_bool
\bool_new:N \l__cal_taperable_bool

\dim_new:N \l__cal_taper_width_dim
\dim_new:N \l__cal_line_width_dim

\bool_set_true:N \l__cal_taper_start_bool
\bool_set_true:N \l__cal_taper_end_bool

\cs_generate_variant:Nn \tl_put_right:Nn {Nv}

\msg_new:nnn { calligraphy } { undefined pen } { The~ pen~ "#1"~ is~ not~ defined. }
```

5.2 TikZ Keys

The public interface to this package is through TikZ keys and styles.

```
\tikzset{
  define~pen/.code={
    \tikzset{pen~name=#1}
    \pgf@relevantforpicturesizefalse
    \tikz@addmode{
      \pgfsyssoftpath@getcurrentpath\l__cal_tmpa_tl
      \spath_components_to_seq:NV \l__cal_tmpa_seq \l__cal_tmpa_tl
      \seq_gclear_new:c {g__cal_pen_\pgfkeysvalueof{/tikz/pen~name}_seq}
      \seq_gset_eq:cN {g__cal_pen_\pgfkeysvalueof{/tikz/pen~name}_seq} \l__cal_tmpa_seq
      \pgfusepath{discard}%
    }
  },
  define~pen/.default={default},
  use~pen/.code={
    \tikzset{pen~name=#1}
    \int_gzero:N \g__cal_path_component_int
    \cs_set_eq:NN \pgfpathmoveto \cal_moveto:n
    \tikz@addmode{
      \pgfsyssoftpath@getcurrentpath\l__cal_tmpa_tl
      \spath_components_to_seq:NV \l__cal_tmpa_seq \l__cal_tmpa_tl
      \tl_if_exist:cTF {g__cal_pen_\pgfkeysvalueof{/tikz/pen~name}_seq}
      {
        \cal_path_create:Nc \l__cal_tmpa_seq {g__cal_pen_\pgfkeysvalueof{/tikz/pen~name}_seq
      }
      {
        \msg_warning:nnx { calligraphy } { undefined pen } { \pgfkeysvalueof{/tikz/pen~name}
      }
    }
  },
  use~pen/.default={default},
  pen~name/.initial={default},
  copperplate/.style={pen~name=copperplate},
  pen~colour/.initial={black},
  weight/.is~choice,
  weight/heavy/.style={
    line~width=\pgfkeysvalueof{/tikz/heavy~line~width},
    taper~width=\pgfkeysvalueof{/tikz/light~line~width},
  },
  weight/light/.style={
    line~width=\pgfkeysvalueof{/tikz/light~line~width},
    taper~width=0pt,
  },
  heavy/.style={
    weight=heavy
  },
  light/.style={
    weight=light
  },
  heavy~line~width/.initial=2pt,
  light~line~width/.initial=1pt,
  taper/.is~choice,
  taper/.default=both,
  taper/none/.style={
    taper~start=false,
    taper~end=false,
  },
  taper/both/.style={
    taper~start=true,
    taper~end=true,
  },
  taper/start/.style={
    taper~start=true,
    taper~end=false,
  },
  taper/end/.style={
    taper~start=false,
    taper~end=true,
  },
  taper~start/.code={
    \tl_if_eq:nnTF {#1} {true}
    {
      \bool_set_true:N \l__cal_taper_start_bool
    }
    {
      \bool_set_false:N \l__cal_taper_start_bool
    }
  },
  taper~start/.default={true},
  taper~end/.code={
    \tl_if_eq:nnTF {#1} {true}
    {
      \bool_set_true:N \l__cal_taper_end_bool
    }
    {
      \bool_set_false:N \l__cal_taper_end_bool
    }
  },
  taper~end/.default={true},
  taper~width/.code={\dim_set:Nn \l__cal_taper_width_dim {#1}},
  nib~style/.code~2~args={
    \tl_clear_new:c {l__cal_nib_style_#1}
    \tl_set:cn {l__cal_nib_style_#1} {#2}
  },
  stroke~style/.code~2~args={
    \tl_clear_new:c {l__cal_stroke_style_#1}
    \tl_set:cn {l__cal_stroke_style_#1} {#2}
  },
  this~stroke~style/.code={
    \tl_clear_new:c {l__cal_stroke_inline_style_ \int_use:N \g__cal_path_component_int}
    \tl_set:cn {l__cal_stroke_inline_style_ \int_use:N \g__cal_path_component_int} {#1}
  },
  annotate/.style={
    annotate~if,
    annotate~reset,
    annotation~style/.update~value={#1},
  },
  annotate~if/.default={true},
  annotate~if/.code={
    \tl_if_eq:nnTF {#1} {true}
    {
      \bool_set_true:N \l__cal_annotate_bool
    }
    {
      \bool_set_false:N \l__cal_annotate_bool
    }
  },
  annotate~reset/.code={
    \int_gzero:N \g__cal_label_int
  },
  annotation~style/.initial={draw,->},
  annotation~shift/.initial={(0,1ex)},
  every~annotation~node/.initial={anchor=south~west},
  annotation~node~style/.code~2~args={
    \tl_clear_new:c {l__cal_annotation_style_ #1 _tl}
    \tl_set:cn {l__cal_annotation_style_ #1 _tl}{#2}
  },
  tl~use:N/.code={
    \exp_args:NV \pgfkeysalso #1
  },
  tl~use:c/.code={
    \tl_if_exist:cT {#1}
    {
      \exp_args:Nv \pgfkeysalso {#1}
    }
  },
  /handlers/.update~style/.code={
    \tl_if_eq:nnF {#1} {\pgfkeysnovalue}
    {
      \pgfkeys{\pgfkeyscurrentpath/.code=\pgfkeysalso{#1}}
    }
  },
  /handlers/.update~value/.code={
    \tl_if_eq:nnF {#1} {\pgfkeysnovalue}
    {
      \pgfkeyssetvalue{\pgfkeyscurrentpath}{#1}
    }
  },
}
```

Some wrappers around the TikZ keys.

```
\NewDocumentCommand \pen { O{} }
{
  \path[define~ pen,every~ calligraphy~ pen/.try,#1]
}

\NewDocumentCommand \definepen { O{} }
{
  \tikz \path[define~ pen,every~ calligraphy~ pen/.try,#1]
}

```

```
\NewDocumentCommand \calligraphy { O{} }
{
  \path[use~ pen,every~ calligraphy/.try,#1]
}
```

5.3 The Path Creation

`\cal_path_create:NN`

This is the main command for creating the calligraphic paths. First argument is the given path Second argument is the pen path

```
\cs_new_protected_nopar:Npn \cal_path_create:NN #1#2
{
  \int_zero:N \l__cal_tmpa_int
  \seq_map_inline:Nn #1
  {
    \int_compare:nT {\tl_count:n {##1} > 3}
    {

      \int_incr:N \l__cal_tmpa_int
      \int_zero:N \l__cal_tmpb_int

      \tl_set:Nn \l__cal_tmp_path_tl {##1}
      \spath_open:N \l__cal_tmp_path_tl
      \spath_reverse:NV \l__cal_tmp_rpath_tl \l__cal_tmp_path_tl

      \seq_map_inline:Nn #2
      {
        \int_incr:N \l__cal_tmpb_int
        \group_begin:
        \pgfsys@beginscope
        \cal_apply_style:c {l__cal_stroke_style_ \int_use:N \l__cal_tmpa_int}
        \cal_apply_style:c {l__cal_stroke_inline_style_ \int_use:N \l__cal_tmpa_int}
        \cal_apply_style:c {l__cal_nib_style_ \int_use:N \l__cal_tmpb_int}

        \spath_initialpoint:Nn \l__cal_tmpa_tl {####1}
        \tl_set_eq:NN \l__cal_tmp_patha_tl \l__cal_tmp_path_tl
        \spath_translate:NV \l__cal_tmp_patha_tl \l__cal_tmpa_tl

        \int_compare:nTF {\tl_count:n {####1} == 3}
        {
          \cal_at_least_three:N \l__cal_tmp_patha_tl
          \spath_protocol_path:V \l__cal_tmp_patha_tl

          \tikz@options
          \dim_set:Nn \l__cal_line_width_dim {\pgflinewidth}
          \cal_maybe_taper:N \l__cal_tmp_patha_tl
        }
        {
          \spath_weld:Nn \l__cal_tmp_patha_tl {####1}
          \spath_weld:NV \l__cal_tmp_patha_tl \l__cal_tmp_rpath_tl
          \spath_reverse:Nn \l__cal_tmp_rpathb_tl {####1}
          \spath_weld:NV \l__cal_tmp_patha_tl \l__cal_tmp_rpathb_tl

          \tl_clear:N \l__cal_tmpa_tl
          \tl_set:Nn \l__cal_tmpa_tl {fill=\pgfkeysvalueof{/tikz/pen~colour},draw=none}
          \tl_if_exist:cT  {l__cal_stroke_style_ \int_use:N \l__cal_tmpa_int}
          {
            \tl_put_right:Nv \l__cal_tmpa_tl {l__cal_stroke_style_ \int_use:N \l__cal_tmpa_i
          }
          \tl_if_exist:cT  {l__cal_stroke_inline_style_ \int_use:N \l__cal_tmpa_int}
          {
            \tl_put_right:Nn \l__cal_tmpa_tl {,}
            \tl_put_right:Nv \l__cal_tmpa_tl {l__cal_stroke_inline_style_ \int_use:N \l__cal
          }
          \tl_if_exist:cT  {l__cal_nib_style_ \int_use:N \l__cal_tmpb_int}
          {
            \tl_put_right:Nn \l__cal_tmpa_tl {,}
            \tl_put_right:Nv \l__cal_tmpa_tl {l__cal_nib_style_ \int_use:N \l__cal_tmpb_int}
          }
          \spath_tikz_path:VV \l__cal_tmpa_tl \l__cal_tmp_patha_tl

        }
        \pgfsys@endscope
        \group_end:
      }

      \bool_if:NT \l__cal_annotate_bool
      {
        \seq_get_right:NN #2 \l__cal_tmpa_tl
        \spath_finalpoint:NV \l__cal_tmpa_tl \l__cal_tmpa_tl
        \spath_translate:NV \l__cal_tmp_path_tl \l__cal_tmpa_tl
        \tikz@scan@one@point\pgfutil@firstofone\pgfkeysvalueof{/tikz/annotation~shift}

        \spath_translate:Nnn \l__cal_tmp_path_tl {\pgf@x} {\pgf@y}

        \pgfkeysgetvalue{/tikz/annotation~style}{\l__cal_tmpa_tl}
        \spath_tikz_path:VV \l__cal_tmpa_tl \l__cal_tmp_path_tl

        \spath_finalpoint:NV \l__cal_tmpa_tl \l__cal_tmp_path_tl

        \exp_last_unbraced:NV \pgfqpoint \l__cal_tmpa_tl
        \begin{scope}[reset~ cm]
        \node[every~annotation~node/.try,tl~use:c =  {l__cal_annotation_style_ \int_use:N \l
        \end{scope}
      }
    }
  }
}
\cs_generate_variant:Nn \cal_path_create:NN {Nc}
```

(*End definition for* `\cal_path_create:NN`.)

`\cal_moveto:n`

When creating the path, we need to keep track of the number of components so that we can apply styles accordingly.

```
\cs_new_eq:NN \cal_orig_moveto:n \pgfpathmoveto
\cs_new_nopar:Npn \cal_moveto:n #1
{
  \int_gincr:N \g__cal_path_component_int
  \cal_orig_moveto:n {#1}
}
```

(*End definition for* `\cal_moveto:n`*.*)

`\cal_apply_style:N`

Interface for applying `\tikzset` to a token list.

```
\cs_new_nopar:Npn \cal_apply_style:N #1
{
  \tl_if_exist:NT #1 {
    \exp_args:NV \tikzset #1
  }
}
\cs_generate_variant:Nn \cal_apply_style:N {c}
```

(*End definition for* `\cal_apply_style:N`*.*)

`\cal_at_least_three:Nn`

A tapered path has to have at least three components. This figures out if it is necessary and sets up the splitting.

```
\cs_new_protected_nopar:Npn \cal_at_least_three:Nn #1#2
{
  \spath_reallength:Nn \l__cal_tmpa_int {#2}
  \tl_clear:N \l__cal_tmpb_tl
  \tl_set:Nn \l__cal_tmpb_tl {#2}
  \int_compare:nTF {\l__cal_tmpa_int = 1}
  {
    \spath_split_at:Nn \l__cal_tmpb_tl {2/3}
    \spath_split_at:Nn \l__cal_tmpb_tl {1/2}
  }
  {
    \int_compare:nT {\l__cal_tmpa_int = 2}
    {
      \spath_split_at:Nn \l__cal_tmpb_tl {1.5}
      \spath_split_at:Nn \l__cal_tmpb_tl {.5}
    }
  }
  \tl_set_eq:NN #1 \l__cal_tmpb_tl
}
\cs_generate_variant:Nn \cal_at_least_three:Nn {NV}
\cs_new_protected_nopar:Npn \cal_at_least_three:N #1
{
  \cal_at_least_three:NV #1#1
}
\cs_generate_variant:Nn \cal_at_least_three:N {c}
```

(*End definition for* `\cal_at_least_three:Nn`*.*)

`\cal_maybe_taper:N`

Possibly tapers the path, depending on the booleans.

```
\cs_new_protected_nopar:Npn \cal_maybe_taper:N #1
{
  \tl_set_eq:NN \l__cal_tmpa_tl #1

  \bool_if:NT \l__cal_taper_start_bool
  {

    \dim_set:Nn \l__cal_tmpa_dim {\tl_item:Nn \l__cal_tmpa_tl {2}}
    \dim_set:Nn \l__cal_tmpb_dim {\tl_item:Nn \l__cal_tmpa_tl {3}}
    \tl_set:Nx \l__cal_tmpb_tl {\tl_item:Nn \l__cal_tmpa_tl {4}}

    \tl_case:NnF \l__cal_tmpb_tl
    {
      \c_spath_lineto_tl
      {

        \bool_set_true:N \l__cal_taperable_bool
        \dim_set:Nn \l__cal_tmpg_dim {\tl_item:Nn \l__cal_tmpa_tl {5}}
        \dim_set:Nn \l__cal_tmph_dim {\tl_item:Nn \l__cal_tmpa_tl {6}}
        \dim_set:Nn \l__cal_tmpc_dim {(2\l__cal_tmpa_dim + \l__cal_tmpg_dim)/3}
        \dim_set:Nn \l__cal_tmpd_dim {(2\l__cal_tmpb_dim + \l__cal_tmph_dim)/3}
        \dim_set:Nn \l__cal_tmpe_dim {(\l__cal_tmpa_dim + 2\l__cal_tmpg_dim)/3}
        \dim_set:Nn \l__cal_tmpf_dim {(\l__cal_tmpb_dim + 2\l__cal_tmph_dim)/3}
        \prg_replicate:nn {4}
        {
          \tl_set:Nx \l__cal_tmpa_tl {\tl_tail:N \l__cal_tmpa_tl}
        }
        \tl_put_left:NV \l__cal_tmpa_tl \c_spath_moveto_tl
      }
      \c_spath_curvetoa_tl
      {
        \bool_set_true:N \l__cal_taperable_bool
        \dim_set:Nn \l__cal_tmpc_dim {\tl_item:Nn \l__cal_tmpa_tl {5}}
        \dim_set:Nn \l__cal_tmpd_dim {\tl_item:Nn \l__cal_tmpa_tl {6}}
        \dim_set:Nn \l__cal_tmpe_dim {\tl_item:Nn \l__cal_tmpa_tl {8}}
        \dim_set:Nn \l__cal_tmpf_dim {\tl_item:Nn \l__cal_tmpa_tl {9}}
        \dim_set:Nn \l__cal_tmpg_dim {\tl_item:Nn \l__cal_tmpa_tl {11}}
        \dim_set:Nn \l__cal_tmph_dim {\tl_item:Nn \l__cal_tmpa_tl {12}}
        \prg_replicate:nn {10}
        {
          \tl_set:Nx \l__cal_tmpa_tl {\tl_tail:N \l__cal_tmpa_tl}
        }
        \tl_put_left:NV \l__cal_tmpa_tl \c_spath_moveto_tl
      }
    }
    {
      \bool_set_false:N \l__cal_taperable_bool
    }

    \bool_if:NT \l__cal_taperable_bool
    {
      \__cal_taper_aux:
    }

  }

  \bool_if:NT \l__cal_taper_end_bool
  {

    \dim_set:Nn \l__cal_tmpa_dim {\tl_item:Nn \l__cal_tmpa_tl {-2}}
    \dim_set:Nn \l__cal_tmpb_dim {\tl_item:Nn \l__cal_tmpa_tl {-1}}
    \tl_set:Nx \l__cal_tmpb_tl {\tl_item:Nn \l__cal_tmpa_tl {-3}}

    \tl_case:NnF \l__cal_tmpb_tl
    {
      \c_spath_lineto_tl
      {

        \bool_set_true:N \l__cal_taperable_bool
        \dim_set:Nn \l__cal_tmpg_dim {\tl_item:Nn \l__cal_tmpa_tl {-5}}
        \dim_set:Nn \l__cal_tmph_dim {\tl_item:Nn \l__cal_tmpa_tl {-4}}
        \dim_set:Nn \l__cal_tmpc_dim {(2\l__cal_tmpa_dim + \l__cal_tmpg_dim)/3}
        \dim_set:Nn \l__cal_tmpd_dim {(2\l__cal_tmpb_dim + \l__cal_tmph_dim)/3}
        \dim_set:Nn \l__cal_tmpe_dim {(\l__cal_tmpa_dim + 2\l__cal_tmpg_dim)/3}
        \dim_set:Nn \l__cal_tmpf_dim {(\l__cal_tmpb_dim + 2\l__cal_tmph_dim)/3}
        \tl_reverse:N \l__cal_tmpa_tl
        \prg_replicate:nn {3}
        {
          \tl_set:Nx \l__cal_tmpa_tl {\tl_tail:N \l__cal_tmpa_tl}
        }
        \tl_reverse:N \l__cal_tmpa_tl
      }
      \c_spath_curveto_tl
      {
        \bool_set_true:N \l__cal_taperable_bool
        \dim_set:Nn \l__cal_tmpc_dim {\tl_item:Nn \l__cal_tmpa_tl {-5}}
        \dim_set:Nn \l__cal_tmpd_dim {\tl_item:Nn \l__cal_tmpa_tl {-4}}
        \dim_set:Nn \l__cal_tmpe_dim {\tl_item:Nn \l__cal_tmpa_tl {-8}}
        \dim_set:Nn \l__cal_tmpf_dim {\tl_item:Nn \l__cal_tmpa_tl {-7}}
        \dim_set:Nn \l__cal_tmpg_dim {\tl_item:Nn \l__cal_tmpa_tl {-11}}
        \dim_set:Nn \l__cal_tmph_dim {\tl_item:Nn \l__cal_tmpa_tl {-10}}
        \tl_reverse:N \l__cal_tmpa_tl
        \prg_replicate:nn {9}
        {
          \tl_set:Nx \l__cal_tmpa_tl {\tl_tail:N \l__cal_tmpa_tl}
        }
        \tl_reverse:N \l__cal_tmpa_tl
      }
    }
    {
      \bool_set_false:N \l__cal_taperable_bool
    }

    \bool_if:NT \l__cal_taperable_bool
    {
      \__cal_taper_aux:
    }

  }

  \pgfsyssoftpath@setcurrentpath\l__cal_tmpa_tl
  \pgfsetstrokecolor{\pgfkeysvalueof{/tikz/pen~colour}}
  \pgfusepath{stroke}

}
```

(*End definition for* `\cal_maybe_taper:N`.)

`\__cal_taper_aux:` Auxiliary macro to avoid unnecessary code duplication.

```
\cs_new_protected_nopar:Npn \__cal_taper_aux:
{
  \tl_clear:N \l__cal_tmpb_tl
  \tl_put_right:NV \l__cal_tmpb_tl \c_spath_moveto_tl

  \fp_set:Nn \l__cal_tmpa_fp
  {
    \l__cal_tmpd_dim - \l__cal_tmpb_dim
  }
  \fp_set:Nn \l__cal_tmpb_fp
  {
    \l__cal_tmpa_dim - \l__cal_tmpc_dim
  }
  \fp_set:Nn \l__cal_tmpe_fp
  {
    (\l__cal_tmpa_fp^2 + \l__cal_tmpb_fp^2)^.5
  }

  \fp_set:Nn \l__cal_tmpa_fp {.5*\l__cal_taper_width_dim *     \l__cal_tmpa_fp / \l__cal_tmp
  \fp_set:Nn \l__cal_tmpb_fp {.5*\l__cal_taper_width_dim *     \l__cal_tmpb_fp / \l__cal_tmp

  \fp_set:Nn \l__cal_tmpc_fp
  {
    \l__cal_tmph_dim - \l__cal_tmpf_dim
  }
  \fp_set:Nn \l__cal_tmpd_fp
  {
    \l__cal_tmpe_dim - \l__cal_tmpg_dim
  }
  \fp_set:Nn \l__cal_tmpe_fp
  {
    (\l__cal_tmpc_fp^2 + \l__cal_tmpd_fp^2)^.5
  }

  \fp_set:Nn \l__cal_tmpc_fp {.5*\l__cal_line_width_dim * \l__cal_tmpc_fp / \l__cal_tmpe_fp}
  \fp_set:Nn \l__cal_tmpd_fp {.5*\l__cal_line_width_dim * \l__cal_tmpd_fp / \l__cal_tmpe_fp}

  \tl_put_right:Nx \l__cal_tmpb_tl
  {
    {\dim_eval:n { \fp_to_dim:N \l__cal_tmpa_fp + \l__cal_tmpa_dim}}
    {\dim_eval:n { \fp_to_dim:N \l__cal_tmpb_fp +             \l__cal_tmpb_dim}}
  }

  \tl_put_right:NV \l__cal_tmpb_tl \c_spath_curvetoa_tl

  \tl_put_right:Nx \l__cal_tmpb_tl
  {
    {\dim_eval:n { \fp_to_dim:N \l__cal_tmpa_fp + \l__cal_tmpc_dim}}
    {\dim_eval:n { \fp_to_dim:N \l__cal_tmpb_fp + \l__cal_tmpd_dim}}
  }

  \tl_put_right:NV \l__cal_tmpb_tl \c_spath_curvetob_tl

```

```
\tl_put_right:Nx \l__cal_tmpb_tl
{
  {\dim_eval:n { \fp_to_dim:N \l__cal_tmpc_fp + \l__cal_tmpe_dim}}
  {\dim_eval:n { \fp_to_dim:N \l__cal_tmpd_fp + \l__cal_tmpf_dim}}
}

\tl_put_right:NV \l__cal_tmpb_tl \c_spath_curveto_tl

\tl_put_right:Nx \l__cal_tmpb_tl
{
  {\dim_eval:n { \fp_to_dim:N \l__cal_tmpc_fp + \l__cal_tmpg_dim}}
  {\dim_eval:n { \fp_to_dim:N \l__cal_tmpd_fp + \l__cal_tmph_dim}}
}

\tl_put_right:NV \l__cal_tmpb_tl \c_spath_curvetoa_tl

\tl_put_right:Nx \l__cal_tmpb_tl
{
  {\dim_eval:n { \fp_to_dim:N \l__cal_tmpc_fp + \l__cal_tmpg_dim - \fp_to_dim:n{ 1.32 * \l
  {\dim_eval:n { \fp_to_dim:N \l__cal_tmpd_fp + \l__cal_tmph_dim + \fp_to_dim:n {1.32* \l_
}

\tl_put_right:NV \l__cal_tmpb_tl \c_spath_curvetob_tl

\tl_put_right:Nx \l__cal_tmpb_tl
{
  {\dim_eval:n { -\fp_to_dim:N \l__cal_tmpc_fp + \l__cal_tmpg_dim - \fp_to_dim:n {1.32 * \
  {\dim_eval:n { -\fp_to_dim:N \l__cal_tmpd_fp + \l__cal_tmph_dim + \fp_to_dim:n {1.32 * \
}

\tl_put_right:NV \l__cal_tmpb_tl \c_spath_curveto_tl

\tl_put_right:Nx \l__cal_tmpb_tl
{
  {\dim_eval:n { -\fp_to_dim:N \l__cal_tmpc_fp + \l__cal_tmpg_dim}}
  {\dim_eval:n { -\fp_to_dim:N \l__cal_tmpd_fp + \l__cal_tmph_dim}}
}

\tl_put_right:NV \l__cal_tmpb_tl \c_spath_curvetoa_tl

\tl_put_right:Nx \l__cal_tmpb_tl
{
  {\dim_eval:n { -\fp_to_dim:N \l__cal_tmpc_fp + \l__cal_tmpe_dim}}
  {\dim_eval:n { -\fp_to_dim:N \l__cal_tmpd_fp + \l__cal_tmpf_dim}}
}

\tl_put_right:NV \l__cal_tmpb_tl \c_spath_curvetob_tl

\tl_put_right:Nx \l__cal_tmpb_tl
{
  {\dim_eval:n { -\fp_to_dim:N \l__cal_tmpa_fp + \l__cal_tmpc_dim}}
  {\dim_eval:n { -\fp_to_dim:N \l__cal_tmpb_fp + \l__cal_tmpd_dim}}
}

```

```
  \tl_put_right:NV \l__cal_tmpb_tl \c_spath_curveto_tl

  \tl_put_right:Nx \l__cal_tmpb_tl
  {
    {\dim_eval:n { -\fp_to_dim:N \l__cal_tmpa_fp + \l__cal_tmpa_dim}}
    {\dim_eval:n { -\fp_to_dim:N \l__cal_tmpb_fp + \l__cal_tmpb_dim}}
  }

  \tl_put_right:NV \l__cal_tmpb_tl \c_spath_curvetoa_tl

  \tl_put_right:Nx \l__cal_tmpb_tl
  {
    {\dim_eval:n { -\fp_to_dim:N \l__cal_tmpa_fp + \l__cal_tmpa_dim + \fp_to_dim:n{ 1.32 * \
    {\dim_eval:n { -\fp_to_dim:N \l__cal_tmpb_fp + \l__cal_tmpb_dim - \fp_to_dim:n {1.32* \l
  }

  \tl_put_right:NV \l__cal_tmpb_tl \c_spath_curvetob_tl

  \tl_put_right:Nx \l__cal_tmpb_tl
  {
    {\dim_eval:n { \fp_to_dim:N \l__cal_tmpa_fp + \l__cal_tmpa_dim + \fp_to_dim:n {1.32 * \l
    {\dim_eval:n { \fp_to_dim:N \l__cal_tmpb_fp + \l__cal_tmpb_dim - \fp_to_dim:n {1.32 * \l
  }

  \tl_put_right:NV \l__cal_tmpb_tl \c_spath_curveto_tl

  \tl_put_right:Nx \l__cal_tmpb_tl
  {
    {\dim_eval:n { \fp_to_dim:N \l__cal_tmpa_fp + \l__cal_tmpa_dim}}
    {\dim_eval:n { \fp_to_dim:N \l__cal_tmpb_fp +             \l__cal_tmpb_dim}}
  }

  \pgfsyssoftpath@setcurrentpath\l__cal_tmpb_tl
  \pgfsetfillcolor{\pgfkeysvalueof{/tikz/pen~colour}}
  \pgfusepath{fill}
}
```

(*End definition for* `\__cal_taper_aux:`.)

Defines a copperplate pen.

```
\tl_set:Nn \l__cal_tmpa_tl {\pgfsyssoftpath@movetotoken{0pt}{0pt}}
\spath_components_to_seq:NV \l__cal_tmpa_seq \l__cal_tmpa_tl
\seq_gclear_new:N \g__cal_pen_copperplate_seq
\seq_gset_eq:NN \g__cal_pen_copperplate_seq \l__cal_tmpa_seq
```

`\CopperplatePath` This is used in the decorations section to convert a path to a copperplate path.

```
\DeclareDocumentCommand \CopperplatePath { m }
{
  \spath_components_to_seq:NV \l__cal_tmpa_seq #1
  \cal_path_create:NN \l__cal_tmpa_seq \g__cal_pen_copperplate_seq
}
```

(*End definition for* `\CopperplatePath`. *This function is documented on page* **??**.)

```
\ExplSyntaxOff
```

5.4 Decorations

If a decoration library is loaded we define some decorations that use the calligraphy library, specifically the copperplate pen with its tapering.

First, a brace decoration.

```
\expandafter\ifx\csname pgfdeclaredecoration\endcsname\relax
\else
\pgfdeclaredecoration{calligraphic brace}{brace}
{
  \state{brace}[width=+\pgfdecoratedremainingdistance,next state=final]
  {
    \pgfsyssoftpath@setcurrentpath{\pgfutil@empty}
    \pgfpathmoveto{\pgfpointorigin}
    \pgfpathcurveto
    {\pgfqpoint{.15\pgfdecorationsegmentamplitude}{.3\pgfdecorationsegmentamplitude}}
    {\pgfqpoint{.5\pgfdecorationsegmentamplitude}{.5\pgfdecorationsegmentamplitude}}
    {\pgfqpoint{\pgfdecorationsegmentamplitude}{.5\pgfdecorationsegmentamplitude}}
    {
      \pgftransformxshift{+\pgfdecorationsegmentaspect\pgfdecoratedremainingdistance}
      \pgfpathlineto{\pgfqpoint{-\pgfdecorationsegmentamplitude}{.5\pgfdecorationsegmentampl
      \pgfpathcurveto
      {\pgfqpoint{-.5\pgfdecorationsegmentamplitude}{.5\pgfdecorationsegmentamplitude}}
      {\pgfqpoint{-.15\pgfdecorationsegmentamplitude}{.7\pgfdecorationsegmentamplitude}}
      {\pgfqpoint{0\pgfdecorationsegmentamplitude}{1\pgfdecorationsegmentamplitude}}
      \pgfpathmoveto{\pgfqpoint{0\pgfdecorationsegmentamplitude}{1\pgfdecorationsegmentampli
      \pgfpathcurveto
      {\pgfqpoint{.15\pgfdecorationsegmentamplitude}{.7\pgfdecorationsegmentamplitude}}
      {\pgfqpoint{.5\pgfdecorationsegmentamplitude}{.5\pgfdecorationsegmentamplitude}}
      {\pgfqpoint{\pgfdecorationsegmentamplitude}{.5\pgfdecorationsegmentamplitude}}
    }
    {
      \pgftransformxshift{+\pgfdecoratedremainingdistance}
      \pgfpathlineto{\pgfqpoint{-\pgfdecorationsegmentamplitude}{.5\pgfdecorationsegmentampl
      \pgfpathcurveto
      {\pgfqpoint{-.5\pgfdecorationsegmentamplitude}{.5\pgfdecorationsegmentamplitude}}
      {\pgfqpoint{-.15\pgfdecorationsegmentamplitude}{.3\pgfdecorationsegmentamplitude}}
      {\pgfqpoint{0pt}{0pt}}
    }
    \tikzset{
      taper width=.5\pgflinewidth,
      taper
    }%
    \pgfsyssoftpath@getcurrentpath\cal@tmp@path
    \CopperplatePath{\cal@tmp@path}
  }
  \state{final}{}
}
```

The second is a straightened parenthesis (so that when very large it doesn't bow out too far).

```
\pgfdeclaredecoration{calligraphic straight parenthesis}{brace}
{
  \state{brace}[width=+\pgfdecoratedremainingdistance,next state=final]
  {
    \pgfsyssoftpath@setcurrentpath{\pgfutil@empty}
    \pgfpathmoveto{\pgfpointorigin}
    \pgfpathcurveto
    {\pgfqpoint{.76604\pgfdecorationsegmentamplitude}{.64279\pgfdecorationsegmentamplitude}}
    {\pgfqpoint{2.3333\pgfdecorationsegmentamplitude}{\pgfdecorationsegmentamplitude}}
    {\pgfqpoint{3.3333\pgfdecorationsegmentamplitude}{\pgfdecorationsegmentamplitude}}
    {
      \pgftransformxshift{+\pgfdecoratedremainingdistance}
      \pgfpathlineto{\pgfqpoint{-3.3333\pgfdecorationsegmentamplitude}{\pgfdecorationsegment
      \pgfpathcurveto
      {\pgfqpoint{-2.3333\pgfdecorationsegmentamplitude}{\pgfdecorationsegmentamplitude}}
      {\pgfqpoint{-.76604\pgfdecorationsegmentamplitude}{.64279\pgfdecorationsegmentamplitud
      {\pgfqpoint{0pt}{0pt}}
    }
    \tikzset{
      taper width=.5\pgflinewidth,
      taper
    }%
    \pgfsyssoftpath@getcurrentpath\cal@tmp@path
    \CopperplatePath{\cal@tmp@path}
  }
  \state{final}{}%
}
```

The third is a curved parenthesis.

```
\pgfdeclaredecoration{calligraphic curved parenthesis}{brace}
{
  \state{brace}[width=+\pgfdecoratedremainingdistance,next state=final]
  {
    \pgfsyssoftpath@setcurrentpath{\pgfutil@empty}
    \pgfpathmoveto{\pgfpointorigin}
    \pgf@xa=\pgfdecoratedremainingdistance\relax
    \advance\pgf@xa by -1.5890\pgfdecorationsegmentamplitude\relax
    \edef\cgrphy@xa{\the\pgf@xa}
    \pgfpathcurveto
    {\pgfqpoint{1.5890\pgfdecorationsegmentamplitude}{1.3333\pgfdecorationsegmentamplitude}}
    {\pgfqpoint{\cgrphy@xa}{1.3333\pgfdecorationsegmentamplitude}}
    {\pgfqpoint{\pgfdecoratedremainingdistance}{0pt}}
    \tikzset{
      taper width=.5\pgflinewidth,
      taper
    }%
    \pgfsyssoftpath@getcurrentpath\cal@tmp@path
    \CopperplatePath{\cal@tmp@path}
  }
  \state{final}{}%
}
```

End the conditional for if pgfdecoration module is loaded

```
\fi
```

6 Drawing Knots

```
⟨@@=knot⟩
```

6.1 Initialisation

We load the `spath3` library and the `intersections` TikZ library. Then we get going.

```
\RequirePackage{spath3}
\usetikzlibrary{intersections,spath3}

\ExplSyntaxOn

\tl_new:N \l__knot_tmpa_tl
\tl_new:N \l__knot_tmpb_tl
\tl_new:N \l__knot_tmpc_tl
\tl_new:N \l__knot_tmpd_tl
\tl_new:N \l__knot_tmpg_tl
\tl_new:N \l__knot_redraws_tl
\tl_new:N \l__knot_clip_width_tl
\tl_new:N \l__knot_name_tl
\tl_new:N \l__knot_node_tl
\tl_new:N \l__knot_aux_tl
\tl_new:N \l__knot_auxa_tl
\tl_new:N \l__knot_prefix_tl

\seq_new:N \l__knot_segments_seq

\int_new:N \l__knot_tmpa_int
\int_new:N \l__knot_strands_int
\int_new:N \g__knot_intersections_int
\int_new:N \g__knot_filaments_int
\int_new:N \l__knot_component_start_int

\fp_new:N \l__knot_tmpa_fp
\fp_new:N \l__knot_tmpb_fp

\dim_new:N \l__knot_tmpa_dim
\dim_new:N \l__knot_tmpb_dim
\dim_new:N \l__knot_tolerance_dim
\dim_new:N \l__knot_clip_bg_radius_dim
\dim_new:N \l__knot_clip_draw_radius_dim

\bool_new:N \l__knot_draft_bool
\bool_new:N \l__knot_ignore_ends_bool
\bool_new:N \l__knot_self_intersections_bool
\bool_new:N \l__knot_splits_bool
\bool_new:N \l__knot_super_draft_bool

\bool_new:N \l__knot_prepend_prev_bool
\bool_new:N \l__knot_append_next_bool
\bool_new:N \l__knot_skip_bool
\bool_new:N \l__knot_save_bool

\seq_new:N \g__knot_nodes_seq

\bool_set_true:N \l__knot_ignore_ends_bool
```

Configuration is via TikZ keys and styles.

```
\tikzset{
  spath/prefix/knot/.style={
    spath/set~ prefix=knot strand,
  },
  spath/suffix/knot/.style={
    spath/set~ suffix={},
  },
  knot/.code={
    \tl_if_eq:nnTF {#1} {none}
    {
      \tikz@addmode{\tikz@mode@doublefalse}
    }
    {
      \tikz@addmode{\tikz@mode@doubletrue}
      \tl_if_eq:nnTF {\pgfkeysnovalue} {#1}
      {
        \tikz@addoption{\pgfsetinnerstrokecolor{.}}
      }
      {
        \pgfsetinnerstrokecolor{#1}
      }
      \tikz@addoption{
        \pgfsetstrokecolor{knotbg}
      }
      \tl_set:Nn \tikz@double@setup{
        \pgfsetinnerlinewidth{\pgflinewidth}
        \pgfsetlinewidth{\dim_eval:n {\tl_use:N \l__knot_gap_tl \pgflinewidth}}
      }
    }
  },
  knot~ gap/.store~ in=\l__knot_gap_tl,
  knot~ gap=3,
  knot~ diagram/.is~family,
  knot~ diagram/.unknown/.code={
    \tl_set_eq:NN \l__knot_tmpa_tl \pgfkeyscurrentname
    \pgfkeysalso{
      /tikz/\l__knot_tmpa_tl=#1
    }
  },
  background~ colour/.code={%
    \colorlet{knotbg}{#1}%
  },
  background~ color/.code={%
    \colorlet{knotbg}{#1}%
  },
  background~ colour=white,
  knot~ diagram,
  name/.store~ in=\l__knot_name_tl,
  name={knot},
  save~ intersections/.is~ choice,
  save~ intersections/.default=true,
  save~ intersections/true/.code={
    \bool_set_true:N \l__knot_save_bool
  },
  save~ intersections/false/.code={
    \bool_set_false:N \l__knot_save_bool
  },
  every~ strand/.style={draw},
  ignore~ endpoint~ intersections/.code={
    \tl_if_eq:nnTF {#1} {true}
    {
      \bool_set_true:N \l__knot_ignore_ends_bool
    }
    {
      \bool_set_false:N \l__knot_ignore_ends_bool
    }
  },
  ignore~ endpoint~ intersections/.default=true,
  consider~ self~ intersections/.is~choice,
  consider~ self~ intersections/true/.code={
    \bool_set_true:N \l__knot_self_intersections_bool
    \bool_set_true:N \l__knot_splits_bool
  },
  consider~ self~ intersections/false/.code={
    \bool_set_false:N \l__knot_self_intersections_bool
    \bool_set_false:N \l__knot_splits_bool
  },
  consider~ self~ intersections/no~ splits/.code={
    \bool_set_true:N \l__knot_self_intersections_bool
    \bool_set_false:N \l__knot_splits_bool
  },
  consider~ self~ intersections/.default={true},
  clip~ radius/.code={
    \dim_set:Nn \l__knot_clip_bg_radius_dim {#1}
    \dim_set:Nn \l__knot_clip_draw_radius_dim {#1+2pt}
  },
  clip~ draw~ radius/.code={
    \dim_set:Nn \l__knot_clip_draw_radius_dim {#1}
  },
  clip~ background~ radius/.code={
    \dim_set:Nn \l__knot_clip_bg_radius_dim {#1}
  },
  clip~ radius=10pt,
  end~ tolerance/.code={
    \dim_set:Nn \l__knot_tolerance_dim {#1}
  },
  end~ tolerance=14pt,
  clip/.style={
    clip
  },
  background~ clip/.style={
    clip
  },
  clip~ width/.code={
    \tl_set:Nn \l__knot_clip_width_tl {#1}
  },
  clip~ width=3,
  flip~ crossing/.code={%
```

```
    \tl_clear_new:c {l__knot_crossing_#1}
    \tl_set:cn {l__knot_crossing_#1} {x}
  },
  ignore~ crossing/.code={%
    \tl_clear_new:c {l__knot_ignore_crossing_#1}
    \tl_set:cn {l__knot_ignore_crossing_#1} {x}
  },
  draft~ mode/.is~ choice,
  draft~ mode/off/.code={%
    \bool_set_false:N \l__knot_draft_bool
    \bool_set_false:N \l__knot_super_draft_bool
  },
  draft~ mode/crossings/.code={%
    \bool_set_true:N \l__knot_draft_bool
    \bool_set_false:N \l__knot_super_draft_bool
  },
  draft~ mode/strands/.code={%
    \bool_set_true:N \l__knot_draft_bool
    \bool_set_true:N \l__knot_super_draft_bool
  },
  draft/.is~ family,
  draft,
  crossing~ label/.style={
    overlay,
    fill=white,
    fill~ opacity=.5,
    text~ opacity=1,
    text=blue,
    pin~ edge={blue,<-}
  },
  strand~ label/.style={
    overlay,
    circle,
    draw=purple,
    fill=white,
    fill~ opacity=.5,
    text~ opacity=1,
    text=purple,
    inner~ sep=0pt
  },
}
```

Wrapper around `\tikzset` for applying keys from a token list, checking for if the given token list exists.

```
\cs_new_nopar:Npn \knot_apply_style:N #1
{
  \tl_if_exist:NT #1 {
    \exp_args:NV \tikzset #1
  }
}
\cs_generate_variant:Nn \knot_apply_style:N {c}
```

`\flipcrossings` The user can specify a comma separated list of crossings to flip.

```
\NewDocumentCommand \flipcrossings {m}
```

```
{
  \tikzset{knot~ diagram/flip~ crossing/.list={#1}}%
}
```

(*End definition for* `\flipcrossings`.)

`\strand` This is how the user specifies a strand of the knot.

```
\NewDocumentCommand \strand { O{} }
{
  \int_incr:N \l__knot_strands_int
  \tl_clear_new:c {l__knot_options_strand \int_use:N \l__knot_strands_int}
  \tl_set:cn {l__knot_options_strand \int_use:N \l__knot_strands_int} {#1}
  \path[#1,spath/set~ name=knot,spath/save=\int_use:N \l__knot_strands_int]
}
```

(*End definition for* `\strand`.)

`knot` This is the wrapper environment that calls the knot generation code.

```
\NewDocumentEnvironment{knot} { O{} }
{
  \knot_initialise:n {#1}
}
{
  \knot_render:
}
```

(*End definition for* `knot`.)

`\knot_initialise:n` Set up some stuff before loading in the strands.

```
\cs_new_protected_nopar:Npn \knot_initialise:n #1
{
  \tikzset{knot~ diagram/.cd,every~ knot~ diagram/.try,#1}
  \int_zero:N \l__knot_strands_int
  \tl_clear:N \l__knot_redraws_tl
  \seq_gclear:N \g__knot_nodes_seq
}
```

(*End definition for* `\knot_initialise:n`.)

`\knot_render:` This is the code that starts the work of rendering the knot.

```
\cs_new_protected_nopar:Npn \knot_render:
{
```

Start a scope and reset the transformation (since all transformations have already been taken into account when defining the strands).

```
  \pgfscope
  \pgftransformreset
```

Loop through the strands drawing each one for the first time.

```
  \int_step_function:nnnN {1} {1} {\l__knot_strands_int} \knot_draw_strand:n
```

In super draft mode we don't do anything else.

```
  \bool_if:NF \l__knot_super_draft_bool
  {
```

In draft mode we draw labels at the ends of the strands; this also handles splitting curves to avoid self-intersections of Bezier curves if that's requested.

```
    \int_step_function:nnnN {1} {1} {\l__knot_strands_int} \knot_draw_labels:n
```

If we're considering self intersections we need to split the strands into filaments.

```
    \bool_if:NTF \l__knot_self_intersections_bool
    {
      \knot_split_strands:
      \int_set_eq:NN \l__knot_tmpa_int \g__knot_filaments_int
      \tl_set:Nn \l__knot_prefix_tl {filament}
    }
    {
      \int_set_eq:NN \l__knot_tmpa_int \l__knot_strands_int
      \tl_set:Nn \l__knot_prefix_tl {strand}
    }
```

Initialise the intersection count.

```
    \int_gzero:N \g__knot_intersections_int
```

If in draft mode we label the intersections, otherwise we just stick a coordinate at each one.

```
    \tl_clear:N \l__knot_node_tl
    \bool_if:NT \l__knot_draft_bool
    {
      \tl_set:Nn \l__knot_node_tl {
        \exp_not:N \node[coordinate,
          pin={[node~ contents={\int_use:N \g__knot_intersections_int},knot~ diagram/draft/c
            }]
      }
    }
```

This double loop steps through the pieces (strands or filaments) and computes the intersections and does stuff with those.

```
    \int_step_variable:nnnNn {1} {1} {\l__knot_tmpa_int - 1} \l__knot_tmpa_tl
    {
      \int_step_variable:nnnNn {\tl_use:N \l__knot_tmpa_tl + 1} {1}     {\l__knot_tmpa_int}
      {
        \knot_intersections:VV \l__knot_tmpa_tl \l__knot_tmpb_tl
      }
    }
```

If any redraws were requested, do them here.

```
    \tl_use:N \l__knot_redraws_tl
```

Draw the crossing nodes

```
    \seq_use:Nn \g__knot_nodes_seq {}
  }
```

Close the scope

```
  \endpgfscope
}
```

(*End definition for* `\knot_render:`.)

`\knot_draw_strand:n`

This renders a strand using the options originally specified.

```
\cs_new_protected_nopar:Npn \knot_draw_strand:n #1
{
  \pgfscope
  \group_begin:
  \spath_bake_round:c {knot strand #1}
  \tl_set:Nn \l__knot_tmpa_tl {knot~ diagram/every~ strand/.try,}
  \tl_put_right:Nv \l__knot_tmpa_tl {l__knot_options_strand #1}
  \tl_put_right:Nn \l__knot_tmpa_tl {,knot~ diagram/only~ when~ rendering/.try,only~ when~ r
  \spath_tikz_path:Vv \l__knot_tmpa_tl {knot strand #1}
  \group_end:
  \endpgfscope
}
\cs_generate_variant:Nn \tl_put_right:Nn {Nv}
```

(*End definition for* `\knot_draw_strand:n`.)

`\knot_draw_labels:n`

Draw a label at each end of each strand, if in draft mode. Also, if requested, split potentially self intersecting Bezier curves.

```
\cs_new_protected_nopar:Npn \knot_draw_labels:n #1
{
  \bool_if:NT \l__knot_draft_bool
  {
    \spath_finalpoint:Nv \l__knot_tmpb_tl {knot strand #1}
    \dim_set:Nn \l__knot_tmpa_dim {\tl_item:Nn \l__knot_tmpb_tl {1}}
    \dim_set:Nn \l__knot_tmpb_dim {\tl_item:Nn \l__knot_tmpb_tl {2}}
    \node[knot~ diagram/draft/strand~label] at (\l__knot_tmpa_dim,\l__knot_tmpb_dim) {#1};
    \spath_initialpoint:Nv \l__knot_tmpb_tl {knot strand #1}
    \dim_set:Nn \l__knot_tmpa_dim {\tl_item:Nn \l__knot_tmpb_tl {1}}
    \dim_set:Nn \l__knot_tmpb_dim {\tl_item:Nn \l__knot_tmpb_tl {2}}
    \node[knot~ diagram/draft/strand~label] at (\l__knot_tmpa_dim,\l__knot_tmpb_dim) {#1};
  }
  \bool_if:nT {
    \l__knot_self_intersections_bool
    &&
    \l__knot_splits_bool
  }
  {
    \tl_clear:N \l__knot_tmpa_tl
    \spath_segments_to_seq:Nv \l__knot_segments_seq {knot strand #1}
    \seq_map_function:NN \l__knot_segments_seq \knot_split_self_intersects:N
    \tl_set_eq:cN {knot strand #1} \l__knot_tmpa_tl
  }
}
```

(*End definition for* `\knot_draw_labels:n`.)

`\knot_split_self_intersects:N`

This is the macro that does the split. Figuring out whether a Bezier cubic self intersects is apparently a difficult problem so we don't bother. We compute a point such that if there is an intersection then it lies on either side of the point. I don't recall where the formula came from!

```
\cs_new_protected_nopar:Npn \knot_split_self_intersects:N #1
{
  \tl_set:Nx \l__knot_tmpc_tl {\tl_item:nn {#1} {4}}
```

```
\tl_case:NnF \l__knot_tmpc_tl
{
  \c_spath_curvetoa_tl
  {
    \fp_set:Nn \l__knot_tmpa_fp
    {
      (\tl_item:nn {#1} {3} - 3 * \tl_item:nn {#1} {6} + 3 * \tl_item:nn {#1} {9} - \tl_it
      *
      (3 * \tl_item:nn {#1} {8} - 3 * \tl_item:nn {#1} {11})
      -
      (\tl_item:nn {#1} {2} - 3 * \tl_item:nn {#1} {5} + 3 * \tl_item:nn {#1} {8} - \tl_it
      *
      (3 * \tl_item:nn {#1} {9} - 3 * \tl_item:nn {#1} {12})
    }
    \fp_set:Nn \l__knot_tmpb_fp
    {
      (\tl_item:nn {#1} {2} - 3 * \tl_item:nn {#1} {5} + 3 * \tl_item:nn {#1} {8} - \tl_it
      *
      (3 * \tl_item:nn {#1} {6} - 6 * \tl_item:nn {#1} {9} + 3 * \tl_item:nn {#1} {12})
      -
      (\tl_item:nn {#1} {3} - 3 * \tl_item:nn {#1} {6} + 3 * \tl_item:nn {#1} {9} - \tl_it
      *
      (3 * \tl_item:nn {#1} {5} - 6 * \tl_item:nn {#1} {8} + 3 * \tl_item:nn {#1} {11})
    }
    \fp_compare:nTF
    {
      \l__knot_tmpb_fp != 0
    }
    {
      \fp_set:Nn \l__knot_tmpa_fp {.5 * \l__knot_tmpa_fp / \l__knot_tmpb_fp}
      \fp_compare:nTF
      {
        0 < \l__knot_tmpa_fp && \l__knot_tmpa_fp < 1
      }
      {
        \spath_split_curve:NNnV \l__knot_tmpc_tl \l__knot_tmpd_tl {#1} \l__knot_tmpa_fp
        \tl_set:Nx \l__knot_tmpc_tl {\tl_tail:N \l__knot_tmpc_tl}
        \tl_set:Nx \l__knot_tmpc_tl {\tl_tail:N \l__knot_tmpc_tl}
        \tl_set:Nx \l__knot_tmpc_tl {\tl_tail:N \l__knot_tmpc_tl}
        \tl_set:Nx \l__knot_tmpd_tl {\tl_tail:N \l__knot_tmpd_tl}
        \tl_set:Nx \l__knot_tmpd_tl {\tl_tail:N \l__knot_tmpd_tl}
        \tl_set:Nx \l__knot_tmpd_tl {\tl_tail:N \l__knot_tmpd_tl}
        \tl_put_right:NV \l__knot_tmpa_tl \l__knot_tmpc_tl
        \tl_put_right:NV \l__knot_tmpa_tl \l__knot_tmpd_tl
      }
      {
        \tl_set:Nn \l__knot_tmpc_tl {#1}
        \tl_set:Nx \l__knot_tmpc_tl {\tl_tail:N \l__knot_tmpc_tl}
        \tl_set:Nx \l__knot_tmpc_tl {\tl_tail:N \l__knot_tmpc_tl}
        \tl_set:Nx \l__knot_tmpc_tl {\tl_tail:N \l__knot_tmpc_tl}
        \tl_put_right:NV \l__knot_tmpa_tl \l__knot_tmpc_tl
      }
    }
    {
        \tl_set:Nn \l__knot_tmpc_tl {#1}
        \tl_set:Nx \l__knot_tmpc_tl {\tl_tail:N \l__knot_tmpc_tl}
        \tl_set:Nx \l__knot_tmpc_tl {\tl_tail:N \l__knot_tmpc_tl}
        \tl_set:Nx \l__knot_tmpc_tl {\tl_tail:N \l__knot_tmpc_tl}
        \tl_put_right:NV \l__knot_tmpa_tl \l__knot_tmpc_tl
      }
    }
    \c_spath_lineto_tl
    {
      \tl_set:Nn \l__knot_tmpc_tl {#1}
      \tl_set:Nx \l__knot_tmpc_tl {\tl_tail:N \l__knot_tmpc_tl}
      \tl_set:Nx \l__knot_tmpc_tl {\tl_tail:N \l__knot_tmpc_tl}
      \tl_set:Nx \l__knot_tmpc_tl {\tl_tail:N \l__knot_tmpc_tl}
      \tl_put_right:NV \l__knot_tmpa_tl \l__knot_tmpc_tl
    }
  }
  {
    \tl_put_right:Nn \l__knot_tmpa_tl {#1}
  }
}
```

(*End definition for* `\knot_split_self_intersects:N`.)

`\knot_intersections:nn`

This computes the intersections of two pieces and steps through them.

```
\cs_new_protected_nopar:Npn \knot_intersections:nn #1#2
{
  \group_begin:
  \tl_set_eq:NN \l__knot_tmpa_tl \l__knot_prefix_tl
  \tl_put_right:Nn \l__knot_tmpa_tl {#1}
  \tl_set_eq:NN \l__knot_tmpb_tl \l__knot_prefix_tl
  \tl_put_right:Nn \l__knot_tmpb_tl {#2}
  \tl_set_eq:Nc \l__knot_tmpc_tl {knot \tl_use:N \l__knot_tmpa_tl}
  \tl_set_eq:Nc \l__knot_tmpd_tl {knot \tl_use:N \l__knot_tmpb_tl}

  \bool_if:nTF {
    \l__knot_save_bool
    &&
    \tl_if_exist_p:c {knot~ intersections~ \tl_use:N \l__knot_name_tl - \tl_use:N \l__knot_t
  }
  {
    \tl_use:c {knot~ intersections~ \tl_use:N \l__knot_name_tl - \tl_use:N \l__knot_tmpa_tl
  }
  {
\pgfintersectionofpaths{\pgfsetpath\l__knot_tmpc_tl}{\pgfsetpath\l__knot_tmpd_tl}

  }

  \int_compare:nT {\pgfintersectionsolutions > 0}
  {
    \int_step_function:nnnN {1} {1} {\pgfintersectionsolutions} \knot_do_intersection:n
  }

  \knot_save_intersections:VV \l__knot_tmpa_tl \l__knot_tmpb_tl
  \group_end:
}
```

(*End definition for* `\knot_intersections:nn`.)

\knot_save_intersections:nn

```
\cs_new_protected_nopar:Npn \knot_save_intersections:nn #1#2
{
  \bool_if:NT \l__knot_save_bool
  {
    \tl_clear:N \l__knot_aux_tl
    \tl_put_right:Nn \l__knot_aux_tl
    {
      \def\pgfintersectionsolutions
    }
    \tl_put_right:Nx \l__knot_aux_tl
    {
      {\int_eval:n {\pgfintersectionsolutions}}
    }
    \int_compare:nT {\pgfintersectionsolutions > 0}
    {
      \int_step_inline:nnnn {1} {1} {\pgfintersectionsolutions}
      {
        \pgfpointintersectionsolution{##1}
        \dim_set:Nn \l__knot_tmpa_dim {\pgf@x}
        \dim_set:Nn \l__knot_tmpb_dim {\pgf@y}
        \tl_put_right:Nn \l__knot_aux_tl
        {
          \expandafter\def\csname pgfpoint@intersect@solution@##1\endcsname
        }
        \tl_put_right:Nx \l__knot_aux_tl
        {
          {\exp_not:N \pgf@x=\dim_use:N \l__knot_tmpa_dim\exp_not:N\relax\exp_not:N \pgf@y =
        }
      }
      \tl_set:Nn \l__knot_auxa_tl {\expandafter \gdef \csname knot~ intersections~}
      \tl_put_right:Nx \l__knot_auxa_tl {\tl_use:N \l__knot_name_tl - #1 - #2}
      \tl_put_right:Nn \l__knot_auxa_tl {\endcsname}
      \tl_put_right:Nx \l__knot_auxa_tl {{\tl_to_str:N \l__knot_aux_tl}}
      \protected@write\@auxout{}{\tl_to_str:N \l__knot_auxa_tl}
    }
  }
}
\cs_generate_variant:Nn \knot_save_intersections:nn {VV}
```

(*End definition for* `\knot_save_intersections:nn`.)

\knot_do_intersection:n

This handles a specific intersection.

```
\cs_new_protected_nopar:Npn \knot_do_intersection:n #1
{
```

Get the intersection coordinates.

```
  \pgfpointintersectionsolution{#1}
  \dim_set:Nn \l__knot_tmpa_dim {\pgf@x}
  \dim_set:Nn \l__knot_tmpb_dim {\pgf@y}
```

If we're dealing with filaments, we can get false positives from the end points.

```
\bool_set_false:N \l__knot_skip_bool
\bool_if:NT \l__knot_self_intersections_bool
{
```

If one filament preceded the other, test for the intersection being at the relevant end point.

```
  \tl_set:Nn \l__knot_tmpc_tl {knot previous}
  \tl_put_right:NV \l__knot_tmpc_tl \l__knot_tmpa_tl
  \tl_set:Nv \l__knot_tmpc_tl \l__knot_tmpc_tl
  \tl_if_eq:NNT \l__knot_tmpc_tl \l__knot_tmpb_tl
  {
    \knot_test_endpoint:VnT \l__knot_tmpb_tl {final point}
    {
      \bool_set_true:N \l__knot_skip_bool
    }
  }

  \tl_set:Nn \l__knot_tmpc_tl {knot previous}
  \tl_put_right:NV \l__knot_tmpc_tl \l__knot_tmpb_tl
  \tl_set:Nv \l__knot_tmpc_tl \l__knot_tmpc_tl
  \tl_if_eq:NNT \l__knot_tmpc_tl \l__knot_tmpa_tl
  {
    \knot_test_endpoint:VnT \l__knot_tmpa_tl {final point}
    {
      \bool_set_true:N \l__knot_skip_bool
    }
  }
}
```

The user can also say that end points of filaments (or strands) should simply be ignored anyway.

```
\bool_if:NT \l__knot_ignore_ends_bool
{
  \knot_test_endpoint:VnT \l__knot_tmpa_tl {initial point}
  {
    \bool_set_true:N \l__knot_skip_bool
  }
  \knot_test_endpoint:VnT \l__knot_tmpa_tl {final point}
  {
    \bool_set_true:N \l__knot_skip_bool
  }
  \knot_test_endpoint:VnT \l__knot_tmpb_tl {initial point}
  {
    \bool_set_true:N \l__knot_skip_bool
  }
  \knot_test_endpoint:VnT \l__knot_tmpb_tl {final point}
  {
    \bool_set_true:N \l__knot_skip_bool
  }
}
```

Assuming that we passed all the above tests, we render the crossing.

```
\bool_if:NF \l__knot_skip_bool
{

    \int_gincr:N \g__knot_intersections_int
```

This is the intersection test. If the intersection finder finds too many, it might be useful to ignore some.

```
    \bool_if:nF
    {
      \tl_if_exist_p:c {l__knot_ignore_crossing_ \int_use:N
        \g__knot_intersections_int}
      &&
      ! \tl_if_empty_p:c {l__knot_ignore_crossing_ \int_use:N
        \g__knot_intersections_int}
    }
    {
```

This is the flip test. We only render one of the paths. The “flip” swaps which one we render.

```
      \bool_if:nTF
      {
        \tl_if_exist_p:c {l__knot_crossing_ \int_use:N
          \g__knot_intersections_int}
        &&
        ! \tl_if_empty_p:c {l__knot_crossing_ \int_use:N
          \g__knot_intersections_int}
      }
      {
        \tl_set_eq:NN \l__knot_tmpg_tl \l__knot_tmpb_tl
      }
      {
        \tl_set_eq:NN \l__knot_tmpg_tl \l__knot_tmpa_tl
      }
```

Now we know which one we’re rendering, we test to see if we should also render its predecessor or successor to ensure that we render a path through the entire crossing region.

```
      \bool_if:NT \l__knot_self_intersections_bool
      {
        \knot_test_endpoint:VnT \l__knot_tmpg_tl {initial point}
        {
          \bool_set_true:N \l__knot_prepend_prev_bool
        }
        {
          \bool_set_false:N \l__knot_prepend_prev_bool
        }
        \knot_test_endpoint:VnT \l__knot_tmpg_tl {final point}
        {
          \bool_set_true:N \l__knot_append_next_bool
        }
        {
          \bool_set_false:N \l__knot_append_next_bool
        }
```

If either of those tests succeeded, do the appending or prepending.

```
        \bool_if:nT
        {
```

```
     \l__knot_prepend_prev_bool || \l__knot_append_next_bool
    }
    {
     \tl_clear_new:c {knot \tl_use:N \l__knot_prefix_tl -1}
     \tl_set_eq:cc {knot \tl_use:N \l__knot_prefix_tl -1} {knot \tl_use:N \l__knot_tmpg

     \tl_clear_new:c {l__knot_options_ \tl_use:N \l__knot_prefix_tl -1}
     \tl_set_eq:cc {l__knot_options_ \tl_use:N \l__knot_prefix_tl -1} {l__knot_options_

     \bool_if:nT
     {
      \l__knot_prepend_prev_bool
      &&
      \tl_if_exist_p:c {knot previous \tl_use:N \l__knot_tmpg_tl}
      &&
      !\tl_if_empty_p:c {knot previous \tl_use:N \l__knot_tmpg_tl}
     }
     {
      \spath_prepend_no_move:cv {knot \tl_use:N \l__knot_prefix_tl -1} {knot \tl_use:c
```

If we split potentially self intersecting curves, we test to see if we should prepend yet another segment.

```
      \bool_if:nT
      {
       \l__knot_splits_bool
       &&
       \tl_if_exist_p:c {knot previous \tl_use:N \l__knot_tmpg_tl}
       &&
       !\tl_if_empty_p:c {knot previous \tl_use:N \l__knot_tmpg_tl}
      }
      {
       \knot_test_endpoint:vnT {knot previous \tl_use:N \l__knot_tmpg_tl} {initial po
       {

        \spath_prepend_no_move:cv {knot \tl_use:N \l__knot_prefix_tl -
1} {knot \tl_use:c {knot previous \tl_use:c {knot previous \tl_use:N \l__knot_tmpg_tl}}}
        \tl_set_eq:Nc \l__knot_tmpa_tl {knot \tl_use:N \l__knot_prefix_tl -
1}
       }
      }
     }
```

Now the same for appending.

```
     \bool_if:nT
     {
      \l__knot_append_next_bool
      &&
      \tl_if_exist_p:c {knot next \tl_use:N \l__knot_tmpg_tl}
      &&
      !\tl_if_empty_p:c {knot previous \tl_use:N \l__knot_tmpg_tl}
     }
     {
      \spath_append_no_move:cv {knot \tl_use:N \l__knot_prefix_tl -1} {knot \tl_use:c
      \bool_if:nT
      {
              \l__knot_splits_bool
              &&
              \tl_if_exist_p:c {knot previous \tl_use:N
                \l__knot_tmpg_tl}
              &&
              !\tl_if_empty_p:c {knot previous \tl_use:N \l__knot_tmpg_tl}
            }
            {
              \knot_test_endpoint:vnT {knot previous \tl_use:N \l__knot_tmpg_tl} {final poin
              {
                \spath_append_no_move:cv {knot \tl_use:N \l__knot_prefix_tl -1} {knot \tl_us

              }
            }
          }

          \tl_set:Nn \l__knot_tmpg_tl {\tl_use:N \l__knot_prefix_tl -1}
        }
      }
```

Now we render the crossing.

```
      \pgfscope
      \group_begin:
      \tikzset{knot~ diagram/every~ intersection/.try, every~ intersection/.try, knot~ diagr
      \knot_draw_crossing:VVV \l__knot_tmpg_tl \l__knot_tmpa_dim \l__knot_tmpb_dim
      \coordinate (\l__knot_name_tl \c_space_tl \int_use:N \g__knot_intersections_int) at (\
      \group_end:
      \endpgfscope
```

This ends the boolean as to whether to consider the intersection at all

```
    }
```

And possibly stick a coordinate with a label at the crossing.

```
    \tl_if_empty:NF \l__knot_node_tl
    {
      \seq_gpush:Nx \g__knot_nodes_seq { \l__knot_node_tl at (\dim_use:N \l__knot_tmpa_dim,
    }
  }
}

\cs_generate_variant:Nn \knot_intersections:nn {VV}
```

(*End definition for* `\knot_do_intersection:n`.)

`\knot_test_endpoint:N` Test whether the point is near the intersection point.

```
\prg_new_conditional:Npnn \knot_test_endpoint:N #1 {p,T,F,TF}
{
  \dim_compare:nTF
  {
    \dim_abs:n { \l__knot_tmpa_dim - \tl_item:Nn #1 {1}}
    +
    \dim_abs:n { \l__knot_tmpb_dim - \tl_item:Nn #1 {2}}
    <
    \l__knot_tolerance_dim
  }
  {
    \prg_return_true:
  }
  {
    \prg_return_false:
  }
}
```

(*End definition for* `\knot_test_endpoint:N`.)

`\knot_test_endpoint:nn`

Wrapper around the above.

```
\prg_new_protected_conditional:Npnn \knot_test_endpoint:nn #1#2 {T,F,TF}
{
  \use:c {spath_#2:Nv} \l__knot_tmpd_tl {knot #1}
  \knot_test_endpoint:NTF \l__knot_tmpd_tl
  {
    \prg_return_true:
  }
  {
    \prg_return_false:
  }
}

\cs_generate_variant:Nn \knot_test_endpoint:nnT {VnT,vnT}
\cs_generate_variant:Nn \knot_test_endpoint:nnF {VnF,vnF}
\cs_generate_variant:Nn \knot_test_endpoint:nnTF {VnTF,vnTF}
```

(*End definition for* `\knot_test_endpoint:nn`.)

`\knot_draw_crossing:nnn`

This is the code that actually renders a crossing.

```
\cs_new_protected_nopar:Npn \knot_draw_crossing:nnn #1#2#3
{
  \group_begin:
  \pgfscope
  \path[knot~ diagram/background~ clip] (#2, #3) circle[radius=\l__knot_clip_bg_radius_dim];

  \tl_set:Nn \l__knot_tmpa_tl {knot~ diagram/every~ strand/.try,}
  \tl_if_exist:cT {l__knot_options_ #1}
  {
  \tl_put_right:Nv \l__knot_tmpa_tl {l__knot_options_ #1}
  }
  \tl_put_right:Nn \l__knot_tmpa_tl {,knotbg,line~ width= \tl_use:N \l__knot_clip_width_tl *
  \spath_tikz_path:Vv \l__knot_tmpa_tl {knot #1}

  \endpgfscope

  \pgfscope
  \path[knot~ diagram/clip] (#2, #3) circle[radius=\l__knot_clip_draw_radius_dim];

  \tl_set:Nn \l__knot_tmpa_tl {knot~ diagram/every~ strand/.try,}
  \tl_if_exist:cT {l__knot_options_ #1}
  {
  \tl_put_right:Nv \l__knot_tmpa_tl {l__knot_options_ #1}
  }
  \tl_put_right:Nn \l__knot_tmpa_tl {,knot~ diagram/only~ when~ rendering/.try,only~ when~ r
  \spath_tikz_path:Vv \l__knot_tmpa_tl {knot #1}

  \endpgfscope
  \group_end:
}

\cs_generate_variant:Nn \knot_draw_crossing:nnn {nVV, VVV}

\cs_new_protected_nopar:Npn \knot_draw_crossing:nn #1#2
{
  \tikz@scan@one@point\pgfutil@firstofone #2 \relax
  \knot_draw_crossing:nVV {#1} \pgf@x \pgf@y
}
```

(*End definition for* `\knot_draw_crossing:nnn`.)

`\knot_split_strands:` This, and the following macros, are for splitting strands into filaments.

```
\cs_new_protected_nopar:Npn \knot_split_strands:
{
  \int_gzero:N \g__knot_filaments_int
  \int_step_function:nnnN {1} {1} {\l__knot_strands_int} \knot_split_strand:n
  \int_step_function:nnnN {1} {1} {\g__knot_filaments_int} \knot_compute_nexts:n
}
```

(*End definition for* `\knot_split_strands:`.)

`\knot_compute_nexts:n` Each filament needs to know its predecessor and successor. We work out the predecessors as we go along, this fills in the successors.

```
\cs_new_protected_nopar:Npn \knot_compute_nexts:n #1
{
  \tl_clear_new:c {knot next \tl_use:c {knot previous filament #1}}
  \tl_set:cn {knot next \tl_use:c {knot previous filament #1}} {filament #1}
}
```

(*End definition for* `\knot_compute_nexts:n`.)

`\knot_split_strand:n` Sets up the split for a single strand.

```
\cs_new_protected_nopar:Npn \knot_split_strand:n #1
{
  \int_set_eq:NN \l__knot_component_start_int \g__knot_filaments_int
  \int_incr:N \l__knot_component_start_int
  \tl_set_eq:Nc \l__knot_tmpa_tl {l__knot_options_strand #1}
  \spath_segments_to_seq:Nv \l__knot_segments_seq {knot strand #1}
  \seq_map_function:NN \l__knot_segments_seq \knot_save_filament:N
}
```

(*End definition for* `\knot_split_strand:n`.)

`\knot_save_filament:N` Saves a filament as a new `spath` object.

```
\cs_new_protected_nopar:Npn \knot_save_filament:N #1
{
  \tl_set:Nx \l__knot_tmpb_tl {\tl_item:nn {#1} {4}}
  \tl_case:NnF \l__knot_tmpb_tl
  {
    \c_spath_moveto_tl
    {
      \int_compare:nT {\l__knot_component_start_int < \g__knot_filaments_int}
      {
        \int_set_eq:NN \l__knot_component_start_int \g__knot_filaments_int
      }
    }
    \c_spath_lineto_tl
    {
      \int_gincr:N \g__knot_filaments_int
      \tl_clear_new:c {knot filament \int_use:N \g__knot_filaments_int}
      \tl_set:cn {knot filament \int_use:N \g__knot_filaments_int} {#1}

      \tl_clear_new:c {l__knot_options_filament \int_use:N \g__knot_filaments_int}
      \tl_set_eq:cN {l__knot_options_filament \int_use:N \g__knot_filaments_int} \l__knot_tm

      \tl_clear_new:c {knot previous filament \int_use:N \g__knot_filaments_int}
      \int_compare:nF {\l__knot_component_start_int == \g__knot_filaments_int}
      {
        \tl_set:cx {knot previous filament \int_use:N \g__knot_filaments_int} {filament \int
      }
    }
    \c_spath_curvetoa_tl
    {
      \int_gincr:N \g__knot_filaments_int
      \tl_clear_new:c {knot filament \int_use:N \g__knot_filaments_int}
      \tl_set:cn {knot filament \int_use:N \g__knot_filaments_int} {#1}
      \tl_clear_new:c {l__knot_options_filament \int_use:N \g__knot_filaments_int}
      \tl_set_eq:cN {l__knot_options_filament \int_use:N \g__knot_filaments_int} \l__knot_tm

      \tl_clear_new:c {knot previous filament \int_use:N \g__knot_filaments_int}
      \int_compare:nF {\l__knot_component_start_int == \g__knot_filaments_int}
      {
        \tl_set:cx {knot previous filament \int_use:N \g__knot_filaments_int} {filament \int
      }
    }
    \c_spath_closepath_tl
    {
      \int_gincr:N \g__knot_filaments_int
      \tl_clear_new:c {knot filament \int_use:N \g__knot_filaments_int}
      \tl_clear:N \l__knot_tmpa_tl
      \tl_put_right:Nx {\tl_item:nn {#1} {1}\tl_item:nn {#1} {2}\tl_item:nn {#1} {3}}
      \tl_put_right:NV \l__knot_tmpa_tl \c_spath_lineto_tl
      \tl_put_right:Nx {\tl_item:nn {#1} {5}\tl_item:nn {#1} {6}}

      \tl_set:cV {knot filament \int_use:N \g__knot_filaments_int} \l__knot_tmpa_tl
      \tl_set_eq:cN {l__knot_options_filament \int_use:N \g__knot_filaments_int} \l__knot_tm
      \tl_clear_new:c {knot previous filament \int_use:N \g__knot_filaments_int}
      \int_compare:nF {\l__knot_component_start_int == \g__knot_filaments_int}
      {
        \tl_set:cx {knot previous filament \int_use:N \g__knot_filaments_int} {filament \int
      }
      \tl_set:cx {knot previous filament \int_use:N \l__knot_component_start_int} {filament
    }
  }
```

```
  {
  }
}
```

(*End definition for* `\knot_save_filament:N`.)

`\redraw` The user can redraw segments of the strands at specific locations.

```
\NewDocumentCommand \redraw { m m }
{
%  \tikz@scan@one@point\pgfutil@firstofone #2 \relax
  \tl_put_right:Nn \l__knot_redraws_tl {\knot_draw_crossing:nn}
  \tl_put_right:Nx \l__knot_redraws_tl {
    {strand #1} {#2}% {\dim_use:N \pgf@x} {\dim_use:N \pgf@y}
  }
}
```

(*End definition for* `\redraw`.)

```
\ExplSyntaxOff
```

<@@=>

`\pgf@sh__knotknotanchor` Add the extra anchors for the knot crossing nodes.

```
\def\pgf@sh__knotknotanchor#1#2{%
  \anchor{#2 north west}{%
    \csname pgf@anchor@knot #1@north west\endcsname%
    \pgf@x=#2\pgf@x%
    \pgf@y=#2\pgf@y%
  }%
  \anchor{#2 north east}{%
    \csname pgf@anchor@knot #1@north east\endcsname%
    \pgf@x=#2\pgf@x%
    \pgf@y=#2\pgf@y%
  }%
  \anchor{#2 south west}{%
    \csname pgf@anchor@knot #1@south west\endcsname%
    \pgf@x=#2\pgf@x%
    \pgf@y=#2\pgf@y%
  }%
  \anchor{#2 south east}{%
    \csname pgf@anchor@knot #1@south east\endcsname%
    \pgf@x=#2\pgf@x%
    \pgf@y=#2\pgf@y%
  }%
  \anchor{#2 north}{%
    \csname pgf@anchor@knot #1@north\endcsname%
    \pgf@x=#2\pgf@x%
    \pgf@y=#2\pgf@y%
  }%
  \anchor{#2 east}{%
    \csname pgf@anchor@knot #1@east\endcsname%
    \pgf@x=#2\pgf@x%
    \pgf@y=#2\pgf@y%
  }%
  \anchor{#2 west}{%
    \csname pgf@anchor@knot #1@west\endcsname%
    \pgf@x=#2\pgf@x%
    \pgf@y=#2\pgf@y%
  }%
  \anchor{#2 south}{%
    \csname pgf@anchor@knot #1@south\endcsname%
    \pgf@x=#2\pgf@x%
    \pgf@y=#2\pgf@y%
  }%
}
```

(*End definition for* `\pgf@sh__knotknotanchor`.)

`knot␣crossing`

```
\pgfdeclareshape{knot crossing}
{
  \inheritsavedanchors[from=circle] % this is nearly a circle
  \inheritanchorborder[from=circle]
  \inheritanchor[from=circle]{north}
  \inheritanchor[from=circle]{north west}
  \inheritanchor[from=circle]{north east}
  \inheritanchor[from=circle]{center}
  \inheritanchor[from=circle]{west}
  \inheritanchor[from=circle]{east}
  \inheritanchor[from=circle]{mid}
  \inheritanchor[from=circle]{mid west}
  \inheritanchor[from=circle]{mid east}
  \inheritanchor[from=circle]{base}
  \inheritanchor[from=circle]{base west}
  \inheritanchor[from=circle]{base east}
  \inheritanchor[from=circle]{south}
  \inheritanchor[from=circle]{south west}
  \inheritanchor[from=circle]{south east}
  \inheritanchorborder[from=circle]
  \pgf@sh__knotknotanchor{crossing}{2}
  \pgf@sh__knotknotanchor{crossing}{3}
  \pgf@sh__knotknotanchor{crossing}{4}
  \pgf@sh__knotknotanchor{crossing}{8}
  \pgf@sh__knotknotanchor{crossing}{16}
  \pgf@sh__knotknotanchor{crossing}{32}
  \backgroundpath{
    \pgfutil@tempdima=\radius%
    \pgfmathsetlength{\pgf@xb}{\pgfkeysvalueof{/pgf/outer xsep}}%
    \pgfmathsetlength{\pgf@yb}{\pgfkeysvalueof{/pgf/outer ysep}}%
    \ifdim\pgf@xb<\pgf@yb%
      \advance\pgfutil@tempdima by-\pgf@yb%
    \else%
      \advance\pgfutil@tempdima by-\pgf@xb%
    \fi%
  }
}
```

(*End definition for* `knot crossing`.)

`knot␣over␣cross`

```
\pgfdeclareshape{knot over cross}
{
  \inheritsavedanchors[from=rectangle] % this is nearly a circle
  \inheritanchorborder[from=rectangle]
  \inheritanchor[from=rectangle]{north}
  \inheritanchor[from=rectangle]{north west}
  \inheritanchor[from=rectangle]{north east}
  \inheritanchor[from=rectangle]{center}
  \inheritanchor[from=rectangle]{west}
  \inheritanchor[from=rectangle]{east}
  \inheritanchor[from=rectangle]{mid}
  \inheritanchor[from=rectangle]{mid west}
  \inheritanchor[from=rectangle]{mid east}
  \inheritanchor[from=rectangle]{base}
  \inheritanchor[from=rectangle]{base west}
  \inheritanchor[from=rectangle]{base east}
  \inheritanchor[from=rectangle]{south}
  \inheritanchor[from=rectangle]{south west}
  \inheritanchor[from=rectangle]{south east}
  \inheritanchorborder[from=rectangle]
  \backgroundpath{
    \southwest \pgf@xa=\pgf@x \pgf@ya=\pgf@y
    \northeast \pgf@xb=\pgf@x \pgf@yb=\pgf@y
    \pgfpathmoveto{\pgfqpoint{\pgf@xa}{\pgf@ya}}
    \pgfpathlineto{\pgfqpoint{\pgf@xb}{\pgf@yb}}
  }
  \foregroundpath{
% store lower right in xa/ya and upper right in xb/yb
    \southwest \pgf@xa=\pgf@x \pgf@ya=\pgf@y
    \northeast \pgf@xb=\pgf@x \pgf@yb=\pgf@y
    \pgfpathmoveto{\pgfqpoint{\pgf@xa}{\pgf@yb}}
    \pgfpathlineto{\pgfqpoint{\pgf@xb}{\pgf@ya}}
 }
}
```

(*End definition for* `knot over cross`*.*)

`knot␣under␣cross`

```
\pgfdeclareshape{knot under cross}
{
  \inheritsavedanchors[from=rectangle] % this is nearly a circle
  \inheritanchorborder[from=rectangle]
  \inheritanchor[from=rectangle]{north}
  \inheritanchor[from=rectangle]{north west}
  \inheritanchor[from=rectangle]{north east}
  \inheritanchor[from=rectangle]{center}
  \inheritanchor[from=rectangle]{west}
  \inheritanchor[from=rectangle]{east}
  \inheritanchor[from=rectangle]{mid}
  \inheritanchor[from=rectangle]{mid west}
  \inheritanchor[from=rectangle]{mid east}
  \inheritanchor[from=rectangle]{base}
  \inheritanchor[from=rectangle]{base west}
  \inheritanchor[from=rectangle]{base east}
  \inheritanchor[from=rectangle]{south}
  \inheritanchor[from=rectangle]{south west}
  \inheritanchor[from=rectangle]{south east}
  \inheritanchorborder[from=rectangle]
  \backgroundpath{
    \southwest \pgf@xa=\pgf@x \pgf@ya=\pgf@y
    \northeast \pgf@xb=\pgf@x \pgf@yb=\pgf@y
    \pgfpathmoveto{\pgfqpoint{\pgf@xa}{\pgf@yb}}
    \pgfpathlineto{\pgfqpoint{\pgf@xb}{\pgf@ya}}
  }
  \foregroundpath{
% store lower right in xa/ya and upper right in xb/yb
    \southwest \pgf@xa=\pgf@x \pgf@ya=\pgf@y
    \northeast \pgf@xb=\pgf@x \pgf@yb=\pgf@y
    \pgfpathmoveto{\pgfqpoint{\pgf@xa}{\pgf@ya}}
    \pgfpathlineto{\pgfqpoint{\pgf@xb}{\pgf@yb}}
 }
}
```

(*End definition for* `knot under cross`.)

`knot␣vert`

```
\pgfdeclareshape{knot vert}
{
  \inheritsavedanchors[from=rectangle] % this is nearly a circle
  \inheritanchorborder[from=rectangle]
  \inheritanchor[from=rectangle]{north}
  \inheritanchor[from=rectangle]{north west}
  \inheritanchor[from=rectangle]{north east}
  \inheritanchor[from=rectangle]{center}
  \inheritanchor[from=rectangle]{west}
  \inheritanchor[from=rectangle]{east}
  \inheritanchor[from=rectangle]{mid}
  \inheritanchor[from=rectangle]{mid west}
  \inheritanchor[from=rectangle]{mid east}
  \inheritanchor[from=rectangle]{base}
  \inheritanchor[from=rectangle]{base west}
  \inheritanchor[from=rectangle]{base east}
  \inheritanchor[from=rectangle]{south}
  \inheritanchor[from=rectangle]{south west}
  \inheritanchor[from=rectangle]{south east}
  \inheritanchorborder[from=rectangle]
  \backgroundpath{
% store lower right in xa/ya and upper right in xb/yb
    \southwest \pgf@xa=\pgf@x \pgf@ya=\pgf@y
    \northeast \pgf@xb=\pgf@x \pgf@yb=\pgf@y
    \pgfpathmoveto{\pgfqpoint{\pgf@xa}{\pgf@ya}}
    \pgfpathlineto{\pgfqpoint{\pgf@xa}{\pgf@yb}}
    \pgfpathmoveto{\pgfqpoint{\pgf@xb}{\pgf@yb}}
    \pgfpathlineto{\pgfqpoint{\pgf@xb}{\pgf@ya}}
 }
}
```

(*End definition for* `knot vert`.)

knot␣horiz

```
\pgfdeclareshape{knot horiz}
{
  \inheritsavedanchors[from=rectangle] % this is nearly a circle
  \inheritanchorborder[from=rectangle]
  \inheritanchor[from=rectangle]{north}
  \inheritanchor[from=rectangle]{north west}
  \inheritanchor[from=rectangle]{north east}
  \inheritanchor[from=rectangle]{center}
  \inheritanchor[from=rectangle]{west}
  \inheritanchor[from=rectangle]{east}
  \inheritanchor[from=rectangle]{mid}
  \inheritanchor[from=rectangle]{mid west}
  \inheritanchor[from=rectangle]{mid east}
  \inheritanchor[from=rectangle]{base}
  \inheritanchor[from=rectangle]{base west}
  \inheritanchor[from=rectangle]{base east}
  \inheritanchor[from=rectangle]{south}
  \inheritanchor[from=rectangle]{south west}
  \inheritanchor[from=rectangle]{south east}
  \inheritanchorborder[from=rectangle]
  \foregroundpath{
% store lower right in xa/ya and upper right in xb/yb
    \southwest \pgf@xa=\pgf@x \pgf@ya=\pgf@y
    \northeast \pgf@xb=\pgf@x \pgf@yb=\pgf@y
    \pgfpathmoveto{\pgfqpoint{\pgf@xa}{\pgf@ya}}
    \pgfpathlineto{\pgfqpoint{\pgf@xb}{\pgf@ya}}
    \pgfpathmoveto{\pgfqpoint{\pgf@xa}{\pgf@yb}}
    \pgfpathlineto{\pgfqpoint{\pgf@xb}{\pgf@yb}}
 }
}
```

(*End definition for* `knot horiz`*.*)