

The l3draw package

Core drawing support

The L^AT_EX3 Project*

Released 2018/02/21

1 l3draw documentation

The l3draw package provides a set of tools for creating (vector) drawings in expl3. It is heavily inspired by the pgf layer of the TikZ system, with many of the interfaces having the same form. However, the code provided here is build entirely on core expl3 ideas and uses the L^AT_EX3 FPU for numerical support.

Numerical expressions in l3draw are handled as floating point expressions, unless otherwise noted. This means that they may contain or omit explicit units. Where units are omitted, they will automatically be taken as given in (T_EX) points.

The code here is *highly* experimental.

1.1 Drawings

```
\draw_begin: \draw_begin:
\draw_end:   ...
\draw_end:
```

Each drawing should be created within a `\draw_begin:/\draw_end:` function pair. The `begin` function sets up a number of key data structures for the rest of the functions here: unless otherwise specified, use of `\draw_...` functions outside of this “environment” is *not supported*.

The drawing created within the environment will be inserted into the typesetting stream by the `\draw_end:` function, which will switch out of vertical mode if required.

1.2 Graphics state

Within the drawing environment, a number of functions control how drawings will appear. Note that these all apply *globally*, though some are reset at the start of each drawing (`\draw_begin:`).

```
\g_draw_linewidth_default_dim
```

The default value of the linewidth for stokes, set at the start of every drawing (`\draw_begin:`).

*E-mail: latex-team@latex-project.org

<hr/> <code>\draw_linewidth:n</code> <hr/>	<code>\draw_linewidth:n {<width>}</code>
<code>\draw_inner_linewidth:n</code> <hr/>	Sets the width to be used for stroking to the <i><width></i> (an <i><fp expr></i>).
<hr/> <code>\draw_nonzero_rule:</code> <hr/>	<code>\draw_nonzero_rule:</code>
<code>\draw_evenodd_rule:</code> <hr/>	Active either the non-zero winding number or the even-odd rule, respectively, for determining what is inside a fill or clip area. For technical reasons, these command are not influenced by scoping and apply on an ongoing basis.
<hr/> <code>\draw_cap_but:</code> <hr/>	<code>\draw_cap_but:</code>
<code>\draw_cap_rectangle:</code> <hr/>	Sets the style of terminal stroke position to one of butt, rectangle or round.
<code>\draw_cap_round:</code> <hr/>	
<hr/> <code>\draw_join_bevel:</code> <hr/>	<code>\draw_cap_but:</code>
<code>\draw_join_miter:</code> <hr/>	Sets the style of stroke joins to one of bevel, miter or round.
<code>\draw_join_round:</code> <hr/>	
<hr/> <code>\draw_miterlimit:n</code> <hr/>	<code>\draw_miterlimit:n {<limit>}</code>
	Sets the miter <i><limit></i> of lines joined as a miter, as described in the PDF and PostScript manuals. The <i><limit></i> is an <i><fp expr></i> .

1.3 Points

Functions supporting the calculation of points (co-ordinates) are expandable and may be used outside of the drawing environment. When used in this way, they all yield a co-ordinate tuple, for example

```
\tl_set:Nx \l_tmpa_tl { \draw_point:nn { 1 } { 2 } }
\tl_show:N \l_tmpa_tl
```

gives

```
> \l_tmpa_tl=1pt,2pt.
<recently read> }
```

This output form is then suitable as *input* for subsequent point calculations, *i.e.* where a *<point>* is required it may be given as a tuple. This *may* include units and surrounding parentheses, for example

```
1,2
(1,2)
1cm,3pt
(1pt,2cm)
2 * sind(30), 2^4in
```

are all valid input forms. Notice that each part of the tuple may itself be a float point expression.

Point co-ordinates are relative to the canvas axes, but can be transformed by `\draw_point_transform:n`. These manipulation is applied by many higher-level functions, for example path construction, and allows parts of a drawing to be rotated, scaled or skewed. This occurs before writing any data to the driver, and so such manipulations are tracked by the drawing mechanisms. See `\driver_draw_transformcm:nnnnnn` for driver-level manipulation of the canvas axes themselves.

Notice that in contrast to `pgf` it is possible to give the positions of points *directly*.

1.3.1 Basic point functions

<code>\draw_point:nn</code>	★	<code>\draw_point:nn {⟨x⟩} {⟨y⟩}</code>
-----------------------------	---	---

Gives the co-ordinates of the point at $\langle x \rangle$ and $\langle y \rangle$, both of which are $\langle fp\ expr \rangle$.

<code>\draw_point_polar:nn</code>	★	<code>\draw_point_polar:nn {⟨angle⟩} {⟨radius⟩}</code>
<code>\draw_point_polar:nnn</code>	★	<code>\draw_point_polar:nnn {⟨angle⟩} {⟨radius-a⟩} {⟨radius-b⟩}</code>

Gives the co-ordinates of the point at $\langle angle \rangle$ (an $\langle fp\ expr \rangle$ in *degrees*) and $\langle radius \rangle$. The three-argument version accepts two radii of different lengths.

Note the interface here is somewhat different from that in `pgf`: the one- and two-radii versions in `l3draw` use separate functions, whilst in `pgf` they use the same function and a keyword.

<code>\draw_point_add:nn</code>	★	<code>\draw_point_add:nn {⟨point1⟩} {⟨point2⟩}</code>
---------------------------------	---	---

Adds $\langle point1 \rangle$ to $\langle point2 \rangle$.

<code>\draw_point_diff:nn</code>	★	<code>\draw_point_diff:nn {⟨point1⟩} {⟨point2⟩}</code>
----------------------------------	---	--

Subtracts $\langle point1 \rangle$ from $\langle point2 \rangle$.

<code>\draw_point_scale:nn</code>	★	<code>\draw_point_scale:nn {⟨scale⟩} {⟨point⟩}</code>
-----------------------------------	---	---

Scales the $\langle point \rangle$ by the $\langle scale \rangle$ (an $\langle fp\ expr \rangle$).

<code>\draw_point_unit_vector:n</code>	★	<code>\draw_point_unit_vector:n {⟨point⟩}</code>
--	---	--

Expands to the co-ordinates of a unit vector joining the $\langle point \rangle$ with the origin.

<code>\draw_point_transform:n</code>	★	<code>\draw_point_transform:n {⟨point⟩}</code>
--------------------------------------	---	--

Evaluates the position of the $\langle point \rangle$ subject to the current transformation matrix. This operation is applied automatically by most higher-level functions (*e.g.* path manipulations).

1.3.2 Points on a vector basis

As well as giving explicit values, it is possible to describe points in terms of underlying direction vectors. The latter are initially co-incident with the standard Cartesian axes, but may be altered by the user.

<code>\draw_xvec_set:n</code>	<code>\draw_xvec_set:n {⟨point⟩}</code>
<code>\draw_yvec_set:n</code>	
<code>\draw_zvec_set:n</code>	

Defines the appropriate base vector to point toward the $\langle point \rangle$ on the canvas. The standard settings for the x - and y -vectors are 1 cm along the relevant canvas axis, whilst for the z -vector an appropriate direction is taken.

<code>\draw_point_vec:nn</code>	★	<code>\draw_point_vec:nn {⟨xscale⟩} {⟨yscale⟩}</code>
<code>\draw_point_vec:nnn</code>	★	<code>\draw_point_vec:nnn {⟨xscale⟩} {⟨yscale⟩} {⟨zscale⟩}</code>

Expands to the co-ordinate of the point at $\langle xscale \rangle$ times the x -vector and $\langle yscale \rangle$ times the y -vector. The three-argument version extends this to include the z -vector.

<code>\draw_point_vec_polar:nn</code>	★	<code>\draw_point_vec_polar:nn {⟨angle⟩} {⟨radius⟩}</code>
<code>\draw_point_vec_polar:nnn</code>	★	<code>\draw_point_vec_polar:nnn {⟨angle⟩} {⟨radius-a⟩} {⟨radius-b⟩}</code>

Gives the co-ordinates of the point at $\langle angle \rangle$ (an $\langle fp\ expr \rangle$ in *degrees*) and $\langle radius \rangle$, relative to the prevailing x - and y -vectors. The three-argument version accepts two radii of different lengths.

Note the interface here is somewhat different from that in `pgf`: the one- and two-radii versions in `l3draw` use separate functions, whilst in `pgf` they use the same function and a keyword.

1.3.3 Intersections

<code>\draw_point_intersect_lines:nnnn</code>	★	<code>\draw_point_intersect_lines:nnnn {⟨point1⟩} {⟨point2⟩} {⟨point3⟩} {⟨point4⟩}</code>
---	---	---

Evaluates the point at the intersection of one line, joining $\langle point1 \rangle$ and $\langle point2 \rangle$, and a second line joining $\langle point3 \rangle$ and $\langle point4 \rangle$. If the lines do not intersect, or are coincident, and error will occur.

<code>\draw_point_intersect_circles:nnnn</code>	★	<code>\draw_point_intersect_circles:nnnn {⟨center1⟩} {⟨radius1⟩} {⟨center2⟩} {⟨radius2⟩} {⟨root⟩}</code>
---	---	--

Evaluates the point at the intersection of one circle with $\langle center1 \rangle$ and $\langle radius1 \rangle$, and a second circle with $\langle center2 \rangle$ and $\langle radius2 \rangle$. If the circles do not intersect, or are coincident, and error will occur.

Note the interface here has a different argument ordering from that in `pgf`, which has the two centers then the two radii.

1.3.4 Interpolations

<code>\draw_point_interpolate_line:nnn</code>	★	<code>\draw_point_interpolate_line:nnn {⟨part⟩} {⟨point1⟩} {⟨point2⟩}</code>
---	---	--

Expands to the point which is $\langle part \rangle$ way along the line joining $\langle point1 \rangle$ and $\langle point2 \rangle$. The $\langle part \rangle$ may be an interpolation or an extrapolation, and is a floating point value expressing a percentage along the line, *e.g.* a value of 0.5 would be half-way between the two points.

<code>\draw_point_interpolate_distance:nnn</code>	★	<code>\draw_point_interpolate_distance:nnn {⟨distance⟩} {⟨point expr1⟩} {⟨point expr2⟩}</code>
---	---	--

Expands to the point which is $\langle distance \rangle$ way along the line joining $\langle point1 \rangle$ and $\langle point2 \rangle$. The $\langle distance \rangle$ may be an interpolation or an extrapolation.

<code>\draw_point_interpolate_curve:nnnnnn</code>	★	<code>\draw_point_interpolate_curve:nnnnnn {⟨part⟩} {⟨start⟩} {⟨control1⟩} {⟨control2⟩} {⟨end⟩}</code>
---	---	--

Expands to the point which is $\langle part \rangle$ way along the curve between $\langle start \rangle$ and $\langle end \rangle$ and defined by $\langle control1 \rangle$ and $\langle control2 \rangle$. The $\langle part \rangle$ may be an interpolation or an extrapolation, and is a floating point value expressing a percentage along the curve, *e.g.* a value of 0.5 would be half-way along the curve.

1.4 Paths

Paths are constructed by combining one or more operations before applying one or more actions. Thus until a path is “used”, it may be manipulated or indeed discarded entirely. Only one path is active at any one time, and the path is *not* affected by T_EX grouping.

<hr/> <hr/> <code>\draw_path_corner_arc:n</code>	<code>\draw_path_corner_arc:n {⟨length⟩}</code>
	Sets the degree of rounding applied to corners in a path: if the <i>⟨length⟩</i> is 0pt then no rounding applies. The value of the <i>⟨length⟩</i> is local to the current T _E X group. <i>At present, corner arcs are not activated in the code.</i>
<hr/> <hr/> <code>\draw_path_moveto:n</code>	<code>\draw_path_moveto:n {⟨point⟩}</code>
	Moves the reference point of the path to the <i>⟨point⟩</i> , but will not join this to any previous point.
<hr/> <hr/> <code>\draw_path_lineto:n</code>	<code>\draw_path_lineto:n {⟨point⟩}</code>
	Joins the current path to the <i>⟨point⟩</i> with a straight line.
<hr/> <hr/> <code>\draw_path_curveto:nnn</code>	<code>\draw_path_curveto:nnn {⟨control1⟩} {⟨control2⟩} {⟨end⟩}</code>
	Joins the current path to the <i>⟨end⟩</i> with a curved line defined by cubic B��zier points <i>⟨control1⟩</i> and <i>⟨control2⟩</i> .
<hr/> <hr/> <code>\draw_path_curveto:nn</code>	<code>\draw_path_curveto:nn {⟨control⟩} {⟨end⟩}</code>
	Joins the current path to the <i>⟨end⟩</i> with a curved line defined by quadratic B��zier point <i>⟨control⟩</i> .
<hr/> <hr/> <code>\draw_path_arc:nnn</code> <code>\draw_path_arc:nnnn</code>	<code>\draw_path_arc:nnn {⟨angle1⟩} {⟨angle2⟩} {⟨radius⟩}</code> <code>\draw_path_arc:nnnn {⟨angle1⟩} {⟨angle2⟩} {⟨radius-a⟩} {⟨radius-b⟩}</code>
	Joins the current path with an arc between <i>⟨angle1⟩</i> and <i>⟨angle2⟩</i> and of <i>⟨radius⟩</i> . The four-argument version accepts two radii of different lengths. Note the interface here has a different argument ordering from that in pgf, which has the two centers then the two radii.
<hr/> <hr/> <code>\draw_path_arc_axes:nnnn</code>	<code>\draw_path_arc_axes:nnn {⟨angle1⟩} {⟨angle2⟩} {⟨vector1⟩} {⟨vector2⟩}</code>
	Appends the portion of an ellipse from <i>⟨angle1⟩</i> to <i>⟨angle2⟩</i> of an ellipse with axes along <i>⟨vector1⟩</i> and <i>⟨vector2⟩</i> to the current path.
<hr/> <hr/> <code>\draw_path_ellipse:nnnn</code>	<code>\draw_path_ellipse:nnn {⟨center⟩} {⟨vector1⟩} {⟨vector2⟩}</code>
	Appends an ellipse at <i>⟨center⟩</i> with axes along <i>⟨vector1⟩</i> and <i>⟨vector2⟩</i> to the current path.
<hr/> <hr/> <code>\draw_path_circle:nn</code>	<code>\draw_path_circle:nn {⟨center⟩} {⟨radius⟩}</code>
	Appends a circle of <i>⟨radius⟩</i> at <i>⟨center⟩</i> to the current path.

<hr/> <code>\draw_path_rectangle:nn</code> <hr/>	<code>\draw_path_rectangle:nn {<lower-left>} {<displacement>}</code>
<code>\draw_path_rectangle_corners:nn</code>	<code>\draw_path_rectangle_corners:nn {<lower-left>} {<top-right>}</code>
	Appends a rectangle starting at <i><lower-left></i> to the current path, with the size of the rectangle determined either by a <i><displacement></i> or the position of the <i><top-right></i> .
<hr/> <code>\draw_path_grid:nnnn</code> <hr/>	<code>\draw_path_grid:nnnn {<xspace>} {<yspace>} {<lower-left>} {<upper-right>}</code>
	Constructs a grid of <i><xspace></i> and <i><yspace></i> from the <i><lower-left></i> to the <i><upper-right></i> , and appends this to the current path.
<hr/> <code>\draw_path_close:</code> <hr/>	<code>\draw_path_close:</code>
	Closes the current part of the path by appending a straight line from the current point to the starting point of the path.
<hr/> <code>\draw_path_use:n</code> <hr/>	<code>\draw_path_use:n {<action(s)>}</code>
<code>\draw_path_use_clear:n</code>	Inserts the current path, carrying out one ore more possible <i><actions></i> (a comma list):
	<ul style="list-style-type: none"> • <code>clear</code> Resets the path to empty • <code>clip</code> Clips any content outside of the path • <code>draw</code> • <code>fill</code> Fills the interior of the path with the current file color • <code>stroke</code> Draws a line along the current path

1.5 Color

<hr/> <code>\draw_color:n</code> <hr/>	<code>\draw_color:n {<color expression>}</code>
<code>\draw_fill:n</code>	Evaluates the <i><color expression></i> as described for <code>l3color</code> .
<code>\draw_stroke:n</code>	

1.6 Transformations

Points are normally used unchanged relative to the canvas axes. This can be modified by applying a transformation matrix. The canvas axes themselves may be adjusted using `\driver_draw_transformcm:nnnnnn`: note that this is transparent to the drawing code so is not tracked.

<hr/> <code>\draw_transform_reset:</code> <hr/>	<code>\draw_transform_reset:</code>
	Resets the matrix to the identity.

<hr/> <code>\draw_transform_concat:nnnnn</code> <hr/>	<code>\draw_transform_concat:nnnnn</code> <code>{<a>} {} {<c>} {<d>} {<vector>}</code>
	Appends the given transformation to the currently-active one. The transformation is made up of a matrix <i><a></i> , <i></i> , <i><c></i> and <i><d></i> , and a shift by the <i><vector></i> .

`\draw_transform:nnnnn`

`\draw_transform:nnnnn`
`{\langle a \rangle} {\langle b \rangle} {\langle c \rangle} {\langle d \rangle} {\langle vector \rangle}`

Applies the transformation matrix specified, over-writing any existing matrix. The transformation is made up of a matrix $\langle a \rangle$, $\langle b \rangle$, $\langle c \rangle$ and $\langle d \rangle$, and a shift by the $\langle vector \rangle$.

`\draw_transform_triangle:nnn`

`\draw_transform_triangle:nnn`
`{\langle origin \rangle} {\langle point1 \rangle} {\langle point2 \rangle}`

Applies a transformation such that the co-ordinates $(0,0)$, $(1,0)$ and $(0,1)$ are given by the $\langle origin \rangle$, $\langle point1 \rangle$ and $\langle point2 \rangle$, respectively.

`\draw_transform_invert:`

`\draw_transform_invert:`

Inverts the current transformation matrix and reverses the current shift vector.

File I

Implementation

1 l3draw implementation

```
1 \<initex | package>
2 \<@@=draw>
3 \<*package>
4 \ProvidesExplPackage{l3draw}{2018/02/21}{ }
5 {L3 Experimental core drawing support}
6 \</package>
7 \RequirePackage { l3color }
8 \</initex | package>
  Everything else is in the sub-files!
```

2 l3draw-paths implementation

```
9 \<initex | package>
10 \<@@=draw>
```

This sub-module covers more-or-less the same ideas as `pgfcorepathconstruct.code.tex`, though using the expandable FPU means that the implementation often varies. At present, equivalents of the following are currently absent:

- `\pgfpatharcto`, `\pgfpatharctoprecomputed`: These are extremely specialised and are very complex in implementation. If the functionality is required, it is likely that it will be set up from scratch here.
- `\pgfpathparabola`: Seems to be unused other than defining a *TikZ* interface, which itself is then not used further.
- `\pgfpathsine`, `\pgfpathcosine`: Need to see exactly how these need to work, in particular whether a wider input range is needed and what approximation to make.
- `\pgfpathcurvebetweentime`, `\pgfpathcurvebetweentimecontinue`: These don't seem to be used at all.

`\l__draw_path_tmp_tl` Scratch space.
`\l__draw_path_tmpa_fp` 11 `\tl_new:N \l__draw_path_tmp_tl`
`\l__draw_path_tmpb_fp` 12 `\fp_new:N \l__draw_path_tmpa_fp`
13 `\fp_new:N \l__draw_path_tmpb_fp`
(End definition for `\l__draw_path_tmp_tl`, `\l__draw_path_tmpa_fp`, and `\l__draw_path_tmpb_fp`.)

2.1 Tracking paths

`\g__draw_path_lastx_dim` The last point visited on a path.
`\g__draw_path_lasty_dim` 14 `\dim_new:N \g__draw_path_lastx_dim`
15 `\dim_new:N \g__draw_path_lasty_dim`
(End definition for `\g__draw_path_lastx_dim` and `\g__draw_path_lasty_dim`.)

`\g__draw_path_xmax_dim` The limiting size of a path.
`\g__draw_path_xmin_dim` 16 `\dim_new:N \g__draw_path_xmax_dim`
`\g__draw_path_ymax_dim` 17 `\dim_new:N \g__draw_path_xmin_dim`
`\g__draw_path_ymin_dim` 18 `\dim_new:N \g__draw_path_ymax_dim`
19 `\dim_new:N \g__draw_path_ymin_dim`
(End definition for `\g__draw_path_xmax_dim` and others.)

`__draw_path_update_limits:nn` Track the limits of a path and (perhaps) of the picture as a whole. (At present the latter
`__draw_path_reset_limits:` is always true: that will change as more complex functionality is added.)
20 `\cs_new_protected:Npn __draw_path_update_limits:nn #1#2`
21 `{`
22 `\dim_gset:Nn \g__draw_path_xmax_dim`
23 `{ \dim_max:nn \g__draw_path_xmax_dim {#1} }`
24 `\dim_gset:Nn \g__draw_path_xmin_dim`
25 `{ \dim_min:nn \g__draw_path_xmin_dim {#1} }`
26 `\dim_gset:Nn \g__draw_path_ymax_dim`
27 `{ \dim_max:nn \g__draw_path_ymax_dim {#2} }`
28 `\dim_gset:Nn \g__draw_path_ymin_dim`
29 `{ \dim_min:nn \g__draw_path_ymin_dim {#2} }`
30 `\bool_if:NT \l__draw_update_bb_bool`
31 `{`
32 `\dim_gset:Nn \g__draw_xmax_dim`
33 `{ \dim_max:nn \g__draw_xmax_dim {#1} }`
34 `\dim_gset:Nn \g__draw_xmin_dim`
35 `{ \dim_min:nn \g__draw_xmin_dim {#1} }`
36 `\dim_gset:Nn \g__draw_ymax_dim`
37 `{ \dim_max:nn \g__draw_ymax_dim {#2} }`
38 `\dim_gset:Nn \g__draw_ymin_dim`
39 `{ \dim_min:nn \g__draw_ymin_dim {#2} }`
40 `}`
41 `}`
42 `\cs_new_protected:Npn __draw_path_reset_limits:`
43 `{`
44 `\dim_gset:Nn \g__draw_path_xmax_dim { -\c_max_dim }`
45 `\dim_gset:Nn \g__draw_path_xmin_dim { \c_max_dim }`
46 `\dim_gset:Nn \g__draw_path_ymax_dim { -\c_max_dim }`
47 `\dim_gset:Nn \g__draw_path_ymin_dim { \c_max_dim }`
48 `}`

(End definition for `_draw_path_update_limits:nn` and `_draw_path_reset_limits:.`)

`_draw_path_update_last:nn` A simple auxiliary to avoid repetition.

```

49 \cs_new_protected:Npn \_draw_path_update_last:nn #1#2
50 {
51   \dim_gset:Nn \g__draw_path_lastx_dim {#1}
52   \dim_gset:Nn \g__draw_path_lasty_dim {#2}
53 }
```

(End definition for `_draw_path_update_last:nn`.)

2.2 Corner arcs

At the level of path *construction*, rounded corners are handled by inserting a marker into the path: that is then picked up once the full path is constructed. Thus we need to set up the appropriate data structures here, such that this can be applied every time it is relevant.

`\l__draw_corner_xarc_dim` The two arcs in use.

`\l__draw_corner_yarc_dim`

```

54 \dim_new:N \l__draw_corner_xarc_dim
55 \dim_new:N \l__draw_corner_yarc_dim
```

(End definition for `\l__draw_corner_xarc_dim` and `\l__draw_corner_yarc_dim`.)

`\l__draw_corner_arc_bool` A flag to speed up the repeated checks.

```

56 \bool_new:N \l__draw_corner_arc_bool
```

(End definition for `\l__draw_corner_arc_bool`.)

`\draw_path_corner_arc:n` Calculate the arcs, check they are non-zero.

`_draw_path_corner_arc:nn`

```

57 \cs_new_protected:Npn \draw_path_corner_arc:n #1
58 {
59   \_draw_point_process:nn { \_draw_path_corner_arc:nn } {#1}
60 }
61 \cs_new_protected:Npn \_draw_path_corner_arc:nn #1#2
62 {
63   \dim_set:Nn \l__draw_corner_xarc_dim {#1}
64   \dim_set:Nn \l__draw_corner_yarc_dim {#2}
65   \bool_lazy_and:nnTF
66     { \dim_compare_p:nNn \l__draw_corner_xarc_dim = { 0pt } }
67     { \dim_compare_p:nNn \l__draw_corner_yarc_dim = { 0pt } }
68     { \bool_set_false:N \l__draw_corner_arc_bool }
69     { \bool_set_true:N \l__draw_corner_arc_bool }
70 }
```

(End definition for `\draw_path_corner_arc:n` and `_draw_path_corner_arc:nn`. This function is documented on page 5.)

`_draw_path_mark_corner:` Mark up corners for arc post-processing.

```

71 \cs_new_protected:Npn \_draw_path_mark_corner:
72 {
73   \bool_if:NT \l__draw_corner_arc_bool
74   {
75     \_draw_softpath_roundpoint:VV
```

```

76         \l__draw_corner_xarc_dim
77         \l__draw_corner_yarc_dim
78     }
79 }

```

(End definition for __draw_path_mark_corner:.)

2.3 Basic path constructions

At present, stick to purely linear transformation support and skip the soft path business: that will likely need to be revisited later.

```

\draw_path_moveto:n
\draw_path_lineto:n
\__draw_path_moveto:nn
\__draw_path_lineto:nn
\draw_path_curveto:nnn
\__draw_path_curveto:nnnnnn

```

```

80 \cs_new_protected:Npn \draw_path_moveto:n #1
81 {
82     \__draw_point_process:nn
83     { \__draw_path_moveto:nn }
84     { \draw_point_transform:n {#1} }
85 }
86 \cs_new_protected:Npn \__draw_path_moveto:nn #1#2
87 {
88     \__draw_path_update_limits:nn {#1} {#2}
89     \__draw_softpath_moveto:nn {#1} {#2}
90     \__draw_path_update_last:nn {#1} {#2}
91 }
92 \cs_new_protected:Npn \draw_path_lineto:n #1
93 {
94     \__draw_point_process:nn
95     { \__draw_path_lineto:nn }
96     { \draw_point_transform:n {#1} }
97 }
98 \cs_new_protected:Npn \__draw_path_lineto:nn #1#2
99 {
100     \__draw_path_mark_corner:
101     \__draw_path_update_limits:nn {#1} {#2}
102     \__draw_softpath_lineto:nn {#1} {#2}
103     \__draw_path_update_last:nn {#1} {#2}
104 }
105 \cs_new_protected:Npn \draw_path_curveto:nnn #1#2#3
106 {
107     \__draw_point_process:nnn
108     {
109         \__draw_point_process:nn
110         {
111             \__draw_path_mark_corner:
112             \__draw_path_curveto:nnnnnn
113         }
114         { \draw_point_transform:n {#1} }
115     }
116     { \draw_point_transform:n {#2} }
117     { \draw_point_transform:n {#3} }
118 }
119 \cs_new_protected:Npn \__draw_path_curveto:nnnnnn #1#2#3#4#5#6
120 {
121     \__draw_path_update_limits:nn {#1} {#2}
122     \__draw_path_update_limits:nn {#3} {#4}

```

```

123     \draw_path_update_limits:nn {#5} {#6}
124     \draw_softpath_curveto:nnnnn {#1} {#2} {#3} {#4} {#5} {#6}
125     \draw_path_update_last:nn {#5} {#6}
126 }

```

(End definition for `\draw_path_moveto:n` and others. These functions are documented on page 5.)

`\draw_path_close:` A simple wrapper.

```

127 \cs_new_protected:Npn \draw_path_close:
128 {
129     \draw_path_mark_corner:
130     \draw_softpath_closepath:
131 }

```

(End definition for `\draw_path_close:`. This function is documented on page 6.)

2.4 Computed curves

More complex operations need some calculations. To assist with those, various constants are pre-defined.

`\draw_path_curveto:nn`
`\draw_path_curveto:nnnn`
`\c_draw_path_curveto_a_fp`
`\c_draw_path_curveto_b_fp`

A quadratic curve with one control point (x_c, y_c) . The two required control points are then

$$x_1 = \frac{1}{3}x_s + \frac{2}{3}x_c \quad y_1 = \frac{1}{3}y_s + \frac{2}{3}y_c$$

and

$$x_2 = \frac{1}{3}x_e + \frac{2}{3}x_c \quad y_2 = \frac{1}{3}y_e + \frac{2}{3}y_c$$

using the start (last) point (x_s, y_s) and the end point (x_e, y_e) .

```

132 \cs_new_protected:Npn \draw_path_curveto:nn #1#2
133 {
134     \draw_point_process:nnn
135     { \draw_path_curveto:nnnn }
136     { \draw_point_transform:n {#1} }
137     { \draw_point_transform:n {#2} }
138 }
139 \cs_new_protected:Npn \draw_path_curveto:nnnn #1#2#3#4
140 {
141     \fp_set:Nn \l__draw_path_tmpa_fp { \c_draw_path_curveto_b_fp * #1 }
142     \fp_set:Nn \l__draw_path_tmpb_fp { \c_draw_path_curveto_b_fp * #2 }
143     \use:x
144     {
145         \draw_path_mark_corner:
146         \draw_path_curveto:nnnnn
147         {
148             \fp_to_dim:n
149             {
150                 \c_draw_path_curveto_a_fp * \g__draw_path_lastx_dim
151                 + \l__draw_path_tmpa_fp
152             }
153         }
154         {
155             \fp_to_dim:n
156             {

```

```

157         \c__draw_path_curveto_a_fp * \g__draw_path_lasty_dim
158         + \l__draw_path_tmpb_fp
159     }
160 }
161 {
162     \fp_to_dim:n
163     { \c__draw_path_curveto_a_fp * #3 + \l__draw_path_tmpa_fp }
164 }
165 {
166     \fp_to_dim:n
167     { \c__draw_path_curveto_a_fp * #4 + \l__draw_path_tmpb_fp }
168 }
169 {#3}
170 {#4}
171 }
172 }
173 \fp_const:Nn \c__draw_path_curveto_a_fp { 1 / 3 }
174 \fp_const:Nn \c__draw_path_curveto_b_fp { 2 / 3 }

```

(End definition for `\draw_path_curveto:nn` and others. This function is documented on page 5.)

\draw_path_arc:nnn Drawing an arc means dividing the total curve required into sections: using B zier curves we can cover at most 90  at once. To allow for later manipulations, we aim to have roughly equal last segments to the line, with the split set at a final part of 115 .

\draw_path_arc:nnnn

```

\__draw_path_arc:nnnn
\__draw_path_arc:nnNnn
\__draw_path_arc_auxi:nnnnNnn
\__draw_path_arc_auxi:fnnnNnn
\__draw_path_arc_auxi:fnfnNnn
\__draw_path_arc_auxii:nnnNnnnn
\__draw_path_arc_auxiii:nn
\__draw_path_arc_auxiv:nnnn
\__draw_path_arc_auxv:nn
\__draw_path_arc_auxvi:nn
\__draw_path_arc_add:nnnn
\l__draw_path_arc_delta_fp
\l__draw_path_arc_start_fp
\c__draw_path_arc_90_fp
\c__draw_path_arc_60_fp

```

```

175 \cs_new_protected:Npn \draw_path_arc:nnn #1#2#3
176 { \draw_path_arc:nnnn {#1} {#2} {#3} {#3} }
177 \cs_new_protected:Npn \draw_path_arc:nnnn #1#2#3#4
178 {
179     \use:x
180     {
181         \__draw_path_arc:nnnn
182         { \fp_eval:n {#1} }
183         { \fp_eval:n {#2} }
184         { \fp_to_dim:n {#3} }
185         { \fp_to_dim:n {#4} }
186     }
187 }
188 \cs_new_protected:Npn \__draw_path_arc:nnnn #1#2#3#4
189 {
190     \fp_compare:nNnTF {#1} > {#2}
191     { \__draw_path_arc:nnNnn {#1} {#2} - {#3} {#4} }
192     { \__draw_path_arc:nnNnn {#1} {#2} + {#3} {#4} }
193 }
194 \cs_new_protected:Npn \__draw_path_arc:nnNnn #1#2#3#4#5
195 {
196     \fp_set:Nn \l__draw_path_arc_start_fp {#1}
197     \fp_set:Nn \l__draw_path_arc_delta_fp { abs( #1 - #2 ) }
198     \fp_while_do:nNnn { \l__draw_path_arc_delta_fp } > { 90 }
199     {
200         \fp_compare:nNnTF \l__draw_path_arc_delta_fp > { 115 }
201         {
202             \__draw_path_arc_auxi:fnnnNnn
203             { \fp_to_decimal:N \l__draw_path_arc_start_fp }
204             { \fp_eval:n { \l__draw_path_arc_start_fp #3 90 } }

```

```

205         { 90 } {#2}
206         #3 {#4} {#5}
207     }
208     {
209         \__draw_path_arc_auxi:ffnnNnn
210         { \fp_to_decimal:N \l__draw_path_arc_start_fp }
211         { \fp_eval:n { \l__draw_path_arc_start_fp #3 60 } }
212         { 60 } {#2}
213         #3 {#4} {#5}
214     }
215 }
216 \__draw_path_mark_corner:
217 \__draw_path_arc_auxi:fnfnNnn
218 { \fp_to_decimal:N \l__draw_path_arc_start_fp }
219 {#2}
220 { \fp_eval:n { abs( \l__draw_path_arc_start_fp - #2 ) } }
221 {#2}
222 #3 {#4} {#5}
223 }

```

The auxiliary is responsible for calculating the required points. The “magic” number required to determine the length of the control vectors is well-established for a right-angle: $\frac{4}{3}(\sqrt{2} - 1) = 0.55228475$. For other cases, we follow the calculation used by `pgf` but with the second common case of 60° pre-calculated for speed.

```

224 \cs_new_protected:Npn \__draw_path_arc_auxi:nnnnNnn #1#2#3#4#5#6#7
225 {
226     \use:x
227     {
228         \__draw_path_arc_auxii:nnnNnnnn
229         {#1} {#2} {#4} #5 {#6} {#7}
230         {
231             \fp_to_dim:n
232             {
233                 \cs_if_exist_use:cF
234                 { c__draw_path_arc_ #3 _fp }
235                 { 4/3 * tand( 0.25 * #3 ) }
236                 * #6
237             }
238         }
239         {
240             \fp_to_dim:n
241             {
242                 \cs_if_exist_use:cF
243                 { c__draw_path_arc_ #3 _fp }
244                 { 4/3 * tand( 0.25 * #3 ) }
245                 * #7
246             }
247         }
248     }
249 }
250 \cs_generate_variant:Nn \__draw_path_arc_auxi:nnnnNnn { fnf , ff }

```

We can now calculate the required points. As everything here is non-expandable, that is best done by using `x`-type expansion to build up the tokens. The three points are calculated out-of-order, since finding the second control point needs the position of the

end point. Once the points are found, fire-off the fundamental path operation and update the record of where we are up to. The final point has to be

```

251 \cs_new_protected:Npn \__draw_path_arc_auxii:nnnNnnnn #1#2#3#4#5#6#7#8
252 {
253   \tl_clear:N \l__draw_path_tmp_tl
254   \__draw_point_process:nn
255     { \__draw_path_arc_auxiii:nn }
256     {
257       \__draw_point_transform_noshift:n
258       { \draw_point_polar:nnn { #1 #4 90 } {#7} {#8} }
259     }
260   \__draw_point_process:nn
261     {
262       \__draw_point_process:nn
263       { \__draw_path_arc_auxiv:nnnn }
264       {
265         \draw_point_transform:n
266         { \draw_point_polar:nnn {#1} {#5} {#6} }
267       }
268     }
269     {
270       \draw_point_transform:n
271       { \draw_point_polar:nnn {#2} {#5} {#6} }
272     }
273   \__draw_point_process:nn
274     { \__draw_path_arc_auxv:nn }
275     {
276       \__draw_point_transform_noshift:n
277       { \draw_point_polar:nnn { #2 #4 -90 } {#7} {#8} }
278     }
279   \exp_after:wN \__draw_path_curveto:nnnnnn \l__draw_path_tmp_tl
280   \fp_set:Nn \l__draw_path_arc_delta_fp { abs ( #2 - #3 ) }
281   \fp_set:Nn \l__draw_path_arc_start_fp {#2}
282 }

```

The first control point.

```

283 \cs_new_protected:Npn \__draw_path_arc_auxiii:nn #1#2
284 {
285   \__draw_path_arc_aux_add:nn
286     { \g__draw_path_lastx_dim + #1 }
287     { \g__draw_path_lasty_dim + #2 }
288 }

```

The end point: simple arithmetic.

```

289 \cs_new_protected:Npn \__draw_path_arc_auxiv:nnnn #1#2#3#4
290 {
291   \__draw_path_arc_aux_add:nn
292     { \g__draw_path_lastx_dim - #1 + #3 }
293     { \g__draw_path_lasty_dim - #2 + #4 }
294 }

```

The second control point: extract the last point, do some rearrangement and record.

```

295 \cs_new_protected:Npn \__draw_path_arc_auxv:nn #1#2
296 {
297   \exp_after:wN \__draw_path_arc_auxvi:nn

```

```

298     \l__draw_path_tmp_tl {#1} {#2}
299   }
300   \cs_new_protected:Npn \__draw_path_arc_auxvi:nn #1#2#3#4#5#6
301   {
302     \tl_set:Nn \l__draw_path_tmp_tl { {#1} {#2} }
303     \__draw_path_arc_aux_add:nn
304       { #5 + #3 }
305       { #6 + #4 }
306     \tl_put_right:Nn \l__draw_path_tmp_tl { {#3} {#4} }
307   }
308   \cs_new_protected:Npn \__draw_path_arc_aux_add:nn #1#2
309   {
310     \tl_put_right:Nx \l__draw_path_tmp_tl
311       { { \fp_to_dim:n {#1} } { \fp_to_dim:n {#2} } }
312   }
313   \fp_new:N \l__draw_path_arc_delta_fp
314   \fp_new:N \l__draw_path_arc_start_fp
315   \fp_const:cn { c__draw_path_arc_90_fp } { 4/3 * (sqrt(2) - 1) }
316   \fp_const:cn { c__draw_path_arc_60_fp } { 4/3 * tand(15) }

```

(End definition for `\draw_path_arc:nnn` and others. These functions are documented on page 5.)

`\draw_path_arc_axes:nnnn` A simple wrapper.

```

317   \cs_new_protected:Npn \draw_path_arc_axes:nnnn #1#2#3#4
318   {
319     \draw_transform_triangle:nnn { 0cm , 0cm } {#3} {#4}
320     \draw_path_arc:nnn {#1} {#2} { 1pt }
321   }

```

(End definition for `\draw_path_arc_axes:nnnn`. This function is documented on page 5.)

`\draw_path_ellipse:nnn` Drawing an ellipse is an optimised version of drawing an arc, in particular reusing the same constant. We need to deal with the ellipse in four parts and also deal with moving to the right place, closing it and ending up back at the center. That is handled on a per-arc basis, each in a separate auxiliary for readability.

```

\__draw_path_ellipse:nnnnnn
  \__draw_path_ellipse_arci:nnnnnn
  \__draw_path_ellipse_arci:nnnnnn
  \__draw_path_ellipse_arci:nnnnnn
  \__draw_path_ellipse_arci:nnnnnn
  \c__draw_path_ellipse_fp
322   \cs_new_protected:Npn \draw_path_ellipse:nnn #1#2#3
323   {
324     \__draw_point_process:nnn
325     {
326       \__draw_point_process:nn
327       { \__draw_path_ellipse:nnnnnn }
328       { \draw_point_transform:n {#1} }
329     }
330     { \__draw_point_transform_noshift:n {#2} }
331     { \__draw_point_transform_noshift:n {#3} }
332   }
333   \cs_new_protected:Npn \__draw_path_ellipse:nnnnnn #1#2#3#4#5#6
334   {
335     \use:x
336     {
337       \__draw_path_moveto:nn
338       { \fp_to_dim:n { #1 + #3 } } { \fp_to_dim:n { #2 + #4 } }
339       \__draw_path_ellipse_arci:nnnnnn {#1} {#2} {#3} {#4} {#5} {#6}
340       \__draw_path_ellipse_arci:nnnnnn {#1} {#2} {#3} {#4} {#5} {#6}

```

```

341     \__draw_path_ellipse_arciiii:nnnnnn {#1} {#2} {#3} {#4} {#5} {#6}
342     \__draw_path_ellipse_arciv:nnnnnn {#1} {#2} {#3} {#4} {#5} {#6}
343   }
344   \__draw_softpath_closepath:
345   \__draw_path_moveto:nn {#1} {#2}
346 }
347 \cs_new:Npn \__draw_path_ellipse_arci:nnnnnn #1#2#3#4#5#6
348 {
349   \__draw_path_curveto:nnnnnn
350   { \fp_to_dim:n { #1 + #3 + #5 * \c__draw_path_ellipse_fp } }
351   { \fp_to_dim:n { #2 + #4 + #6 * \c__draw_path_ellipse_fp } }
352   { \fp_to_dim:n { #1 + #3 * \c__draw_path_ellipse_fp + #5 } }
353   { \fp_to_dim:n { #2 + #4 * \c__draw_path_ellipse_fp + #6 } }
354   { \fp_to_dim:n { #1 + #5 } }
355   { \fp_to_dim:n { #2 + #6 } }
356 }
357 \cs_new:Npn \__draw_path_ellipse_arci:nnnnnn #1#2#3#4#5#6
358 {
359   \__draw_path_curveto:nnnnnn
360   { \fp_to_dim:n { #1 - #3 * \c__draw_path_ellipse_fp + #5 } }
361   { \fp_to_dim:n { #2 - #4 * \c__draw_path_ellipse_fp + #6 } }
362   { \fp_to_dim:n { #1 - #3 + #5 * \c__draw_path_ellipse_fp } }
363   { \fp_to_dim:n { #2 - #4 + #6 * \c__draw_path_ellipse_fp } }
364   { \fp_to_dim:n { #1 - #3 } }
365   { \fp_to_dim:n { #2 - #4 } }
366 }
367 \cs_new:Npn \__draw_path_ellipse_arciiii:nnnnnn #1#2#3#4#5#6
368 {
369   \__draw_path_curveto:nnnnnn
370   { \fp_to_dim:n { #1 - #3 - #5 * \c__draw_path_ellipse_fp } }
371   { \fp_to_dim:n { #2 - #4 - #6 * \c__draw_path_ellipse_fp } }
372   { \fp_to_dim:n { #1 - #3 * \c__draw_path_ellipse_fp - #5 } }
373   { \fp_to_dim:n { #2 - #4 * \c__draw_path_ellipse_fp - #6 } }
374   { \fp_to_dim:n { #1 - #5 } }
375   { \fp_to_dim:n { #2 - #6 } }
376 }
377 \cs_new:Npn \__draw_path_ellipse_arciv:nnnnnn #1#2#3#4#5#6
378 {
379   \__draw_path_curveto:nnnnnn
380   { \fp_to_dim:n { #1 + #3 * \c__draw_path_ellipse_fp - #5 } }
381   { \fp_to_dim:n { #2 + #4 * \c__draw_path_ellipse_fp - #6 } }
382   { \fp_to_dim:n { #1 + #3 - #5 * \c__draw_path_ellipse_fp } }
383   { \fp_to_dim:n { #2 + #4 - #6 * \c__draw_path_ellipse_fp } }
384   { \fp_to_dim:n { #1 + #3 } }
385   { \fp_to_dim:n { #2 + #4 } }
386 }
387 \fp_const:Nn \c__draw_path_ellipse_fp { \fp_use:c { c__draw_path_arc_90_fp } }

```

(End definition for \draw_path_ellipse:nnn and others. This function is documented on page ??.)

\draw_path_circle:nn A shortcut.

```

388 \cs_new_protected:Npn \draw_path_circle:nn #1#2
389 { \draw_path_ellipse:nnn {#1} { #2 , Opt } { Opt , #2 } }

```

(End definition for \draw_path_circle:nn. This function is documented on page 5.)

2.5 Rectangles

`\draw_path_rectangle:nn`
`__draw_path_rectangle:nnnn`
`__draw_path_rectangle_rounded:nnnn`

Building a rectangle can be a single operation, or for rounded versions will involve step-by-step construction.

```

390 \cs_new_protected:Npn \draw_path_rectangle:nn #1#2
391 {
392   \__draw_point_process:nnn
393   {
394     \bool_if:NTF \l__draw_corner_arc_bool
395     { \__draw_path_rectangle_rounded:nnnn }
396     { \__draw_path_rectangle:nnnn }
397   }
398   { \draw_point_transform:n {#1} }
399   {#2}
400 }
401 \cs_new_protected:Npn \__draw_path_rectangle:nnnn #1#2#3#4
402 {
403   \__draw_path_update_limits:nn {#1} {#2}
404   \__draw_path_update_limits:nn { #1 + #3 } { #2 + #4 }
405   \__draw_softpath_rectangle:nnnn {#1} {#2} {#3} {#4}
406   \__draw_path_update_last:nn {#1} {#2}
407 }
408 \cs_new_protected:Npn \__draw_path_rectangle_rounded:nnnn #1#2#3#4
409 {
410   \draw_path_moveto:n { #1 + #3 , #2 + #4 }
411   \draw_path_lineto:n { #1 , #2 + #4 }
412   \draw_path_lineto:n { #1 , #2 }
413   \draw_path_lineto:n { #1 + #3 , #2 }
414   \draw_path_close:
415   \draw_path_moveto:n { #1 , #2 }
416 }
```

(End definition for `\draw_path_rectangle:nn`, `__draw_path_rectangle:nnnn`, and `__draw_path_rectangle_rounded:nnnn`. This function is documented on page 6.)

`\draw_path_rectangle_corners:nn`
`__draw_path_rectangle_corners:nnnn`

Another shortcut wrapper.

```

417 \cs_new_protected:Npn \draw_path_rectangle_corners:nn #1#2
418 {
419   \__draw_point_process:nnn
420   { \__draw_path_rectangle_corners:nnnnn {#1} }
421   {#1} {#2}
422 }
423 \cs_new_protected:Npn \__draw_path_rectangle_corners:nnnnn #1#2#3#4#5
424 { \draw_path_rectangle:nn {#1} { #4 - #2 , #5 - #3 } }
```

(End definition for `\draw_path_rectangle_corners:nn` and `__draw_path_rectangle_corners:nnnnn`. This function is documented on page 6.)

2.6 Grids

`\draw_path_grid:nnnn`
`__draw_path_grid:nnnnnn`

A simple set of loops.

```

425 \cs_new_protected:Npn \draw_path_grid:nnnn #1#2#3#4
426 {
427   \__draw_point_process:nnn
```

```

428     { \_draw_path_grid:nnnnnn {#1} {#2} }
429     {#3} {#4}
430   }
431 \cs_new_protected:Npn \_draw_path_grid:nnnnnn #1#2#3#4#5#6
432 {
433   \dim_step_inline:nnnn
434   {#3} { \dim_compare:nNnF {#3} < {#5} { - } \dim_abs:n {#1} } {#5}
435   {
436     \draw_path_moveto:n { ##1 , #4 }
437     \draw_path_lineto:n { ##1 , #6 }
438   }
439   \dim_step_inline:nnnn
440   {#4} { \dim_compare:nNnF {#4} < {#6} { - } \dim_abs:n {#2} } {#6}
441   {
442     \draw_path_moveto:n { #3 , ##1 }
443     \draw_path_lineto:n { #5 , ##1 }
444   }
445 }

```

(End definition for `\draw_path_grid:nnnn` and `_draw_path_grid:nnnnnn`. This function is documented on page 6.)

2.7 Using paths

Actions to pass to the driver.

```

\l__draw_path_use_clip_bool
\l__draw_path_use_fill_bool
\l__draw_path_use_stroke_bool
446 \bool_new:N \l__draw_path_use_clip_bool
447 \bool_new:N \l__draw_path_use_fill_bool
448 \bool_new:N \l__draw_path_use_stroke_bool

```

(End definition for `\l__draw_path_use_clip_bool`, `\l__draw_path_use_fill_bool`, and `\l__draw_path_use_stroke_bool`.)

Actions handled at the macro layer.

```

\l__draw_path_use_bb_bool
\l__draw_path_use_clear_bool
449 \bool_new:N \l__draw_path_use_bb_bool
450 \bool_new:N \l__draw_path_use_clear_bool

```

(End definition for `\l__draw_path_use_bb_bool` and `\l__draw_path_use_clear_bool`.)

There are a range of actions which can apply to a path: they are handled in a single function which can carry out several of them. The first step is to deal with the special case of clearing the path.

```

\draw_path_use:n
\draw_path_use_clear:n
\_draw_path_use:n
\_draw_path_use_action_draw:
\_draw_path_use_stroke_bb:
\_draw_path_use_stroke_bb_aux:NnN
451 \cs_new_protected:Npn \draw_path_use:n #1
452 {
453   \tl_if_blank:nF {#1}
454   { \_draw_path_use:n {#1} }
455 }
456 \cs_new_protected:Npn \draw_path_use_clear:n #1
457 {
458   \bool_lazy_or:nnTF
459   { \tl_if_blank_p:n {#1} }
460   { \str_if_eq_p:nn {#1} { clear } }
461   {
462     \_draw_softpath_clear:
463     \_draw_path_reset_limits:

```

```

464     }
465     { \__draw_path_use:n { #1 , clear } }
466 }

```

Map over the actions and set up the data: mainly just booleans, but with the possibility to cover more complex cases. The business end of the function is a series of checks on the various flags, then taking the appropriate action(s).

```

467 \cs_new_protected:Npn \__draw_path_use:n #1
468 {
469   \bool_set_false:N \l__draw_path_use_clip_bool
470   \bool_set_false:N \l__draw_path_use_fill_bool
471   \bool_set_false:N \l__draw_path_use_stroke_bool
472   \clist_map_inline:nn {#1}
473   {
474     \cs_if_exist:cTF { l__draw_path_use_ ##1 _ bool }
475     { \bool_set_true:c { l__draw_path_use_ ##1 _ bool } }
476     {
477       \cs_if_exist_use:cF { __draw_path_use_action_ ##1 : }
478       { \ERROR }
479     }
480   }
481   \bool_lazy_and:nnT
482   { \l__draw_update_bb_bool }
483   { \l__draw_path_use_stroke_bool }
484   { \__draw_path_use_stroke_bb: }
485   \bool_if:NTF \l__draw_path_use_clear_bool
486   { \__draw_softpath_use_clear: }
487   { \__draw_softpath_use: }
488   \bool_if:NT \l__draw_path_use_clip_bool
489   { \driver_draw_clip: }
490   \bool_lazy_or:nnT
491   { \l__draw_path_use_fill_bool }
492   { \l__draw_path_use_stroke_bool }
493   {
494     \use:c
495     {
496       driver_draw_
497       \bool_if:NT \l__draw_path_use_fill_bool { fill }
498       \bool_if:NT \l__draw_path_use_stroke_bool { stroke }
499       :
500     }
501   }
502 }
503 \cs_new_protected:Npn \__draw_path_use_action_draw:
504 {
505   \bool_set_true:N \l__draw_path_use_stroke_bool
506 }

```

Where the path is relevant to size and is stroked, we need to allow for the part which overlaps the edge of the bounding box.

```

507 \cs_new_protected:Npn \__draw_path_use_stroke_bb:
508 {
509   \__draw_path_use_stroke_bb_aux:NnN x { max } +
510   \__draw_path_use_stroke_bb_aux:NnN y { max } +
511   \__draw_path_use_stroke_bb_aux:NnN x { min } -

```

```

512   \__draw_path_use_stroke_bb_aux:NnN y { min } -
513   }
514   \cs_new_protected:Npn \__draw_path_use_stroke_bb_aux:NnN #1#2#3
515   {
516     \dim_compare:nNnF { \dim_use:c { g__draw_ #1#2 _dim } } = { #3 -\c_max_dim }
517     {
518       \dim_gset:cn { g__draw_ #1#2 _dim }
519       {
520         \use:c { dim_ #2 :nn }
521         { \dim_use:c { g__draw_ #1#2 _dim } }
522         {
523           \dim_use:c { g__draw_path_ #1#2 _dim }
524           #3 0.5 \g__draw_linewidth_dim
525         }
526       }
527     }
528   }

```

(End definition for `\draw_path_use:n` and others. These functions are documented on page 6.)

```

529 </initex | package>

```

3 l3draw-points implementation

```

530 <*initex | package>

```

```

531 <@@=draw>

```

This sub-module covers more-or-less the same ideas as `pgfcorepoints.code.tex`, though the approach taken to returning values is different: point expressions here are processed by expansion and return a co-ordinate pair in the form $\{\langle x \rangle\}\{\langle y \rangle\}$. Equivalents of following `pgf` functions are deliberately omitted:

- `\pgfpointorigin`: Can be given explicitly as $\{0pt\}\{0pt\}$.
- `\pgfextractx`, `\pgfextracty`: Available by applying `\use_i:nn/\use_ii:nn` or similar to the `x`-type expansion of a point expression.
- `\pgfgetlastxy`: Unused in the entire `pgf` core, may be emulated by `x`-type expansion of a point expression, then using the result.

In addition, equivalents of the following *may* be added in future but are currently absent:

- `\pgfpointcylindrical`, `\pgfpointspherical`: The usefulness of these commands is not currently clear.
- `\pgfpointborderrectangle`, `\pgfpointborderellipse`: To be revisited once the semantics and use cases are clear.
- `\pgfqpoint`, `\pgfqpointscale`, `\pgfqpointpolar`, `\pgfqpointxy`, `\pgfqpointxyz`: The expandable approach taken in the code here, along with the absolute requirement for ε -TeX, means it is likely many use cases for these commands may be covered in other ways. This may be revisited as higher-level structures are constructed.

3.1 Support functions

Execute whatever code is passed to extract the x and y co-ordinates. The first argument here should itself absorb two arguments. There is also a version to deal with two co-ordinates: common enough to justify a separate function.

```

\__draw_point_process:nn
  \__draw_point_process_auxi:nn
  \__draw_point_process_auxi:fn
  \__draw_point_process_auxii:nw
\__draw_point_process:nnn
  \__draw_point_process_auxiii:nnn
  \__draw_point_process_auxiii:ffn
  \__draw_point_process_auxiv:nw
532 \cs_new:Npn \__draw_point_process:nn #1#2
533 {
534   \__draw_point_process_auxi:fn
535   { \__draw_point_to_dim:n {#2} }
536   {#1}
537 }
538 \cs_new:Npn \__draw_point_process_auxi:nn #1#2
539 { \__draw_point_process_auxii:nw {#2} #1 \q_stop }
540 \cs_generate_variant:Nn \__draw_point_process_auxi:nn { f }
541 \cs_new:Npn \__draw_point_process_auxii:nw #1 #2 , #3 \q_stop
542 { #1 {#2} {#3} }
543 \cs_new:Npn \__draw_point_process:nnn #1#2#3
544 {
545   \__draw_point_process_auxiii:ffn
546   { \__draw_point_to_dim:n {#2} }
547   { \__draw_point_to_dim:n {#3} }
548   {#1}
549 }
550 \cs_new:Npn \__draw_point_process_auxiii:nnn #1#2#3
551 { \__draw_point_process_auxiv:nw {#3} #1 \q_mark #2 \q_stop }
552 \cs_generate_variant:Nn \__draw_point_process_auxiii:nnn { ff }
553 \cs_new:Npn \__draw_point_process_auxiv:nw #1 #2 , #3 \q_mark #4 , #5 \q_stop
554 { #1 {#2} {#3} {#4} {#5} }

```

(End definition for `__draw_point_process:nn` and others.)

Co-ordinates are always returned as two dimensions.

```

\__draw_point_to_dim:n
\__draw_point_to_dim_aux:n
\__draw_point_to_dim_aux:f
\__draw_point_to_dim_aux:w
555 \cs_new:Npn \__draw_point_to_dim:n #1
556 { \__draw_point_to_dim_aux:f { \fp_eval:n {#1} } }
557 \cs_new:Npn \__draw_point_to_dim_aux:n #1
558 { \__draw_point_to_dim_aux:w #1 }
559 \cs_generate_variant:Nn \__draw_point_to_dim_aux:n { f }
560 \cs_new:Npn \__draw_point_to_dim_aux:w ( #1 , ~ #2 ) { #1pt , #2pt }

```

3.2 Co-ordinates

The most basic way of giving points is as simple (x, y) co-ordinates.

Simply turn the given values into dimensions.

```

\draw_point:nn
561 \cs_new:Npn \draw_point:nn #1#2
562 { \__draw_point_to_dim:n { #1 , #2 } }

```

3.3 Polar co-ordinates

Polar co-ordinates may have either one or two lengths, so there is a need to do a simple split before the calculation. As the angle gets used twice, save on any expression evaluation there and force expansion.

```

\draw_point_polar:nn
\draw_point_polar:nnn
\__draw_draw_polar:nnn
\__draw_draw_polar:fn
563 \cs_new:Npn \draw_point_polar:nn #1#2

```

```

564 { \draw_point_polar:nnn {#1} {#2} {#2} }
565 \cs_new:Npn \draw_point_polar:nnn #1#2#3
566 { \__draw_draw_polar:fnn { \fp_eval:n {#1} } {#2} {#3} }
567 \cs_new:Npn \__draw_draw_polar:nnn #1#2#3
568 { \__draw_point_to_dim:n { cosd(#1) * (#2) , sind(#1) * (#3) } }
569 \cs_generate_variant:Nn \__draw_draw_polar:nnn { f }

```

3.4 Point expression arithmetic

These functions all take point expressions as arguments.

Simple mathematics.

```

\draw_point_add:nn 570 \cs_new:Npn \draw_point_add:nn #1#2
\draw_point_diff:nn 571 { \__draw_point_to_dim:n { (#1) + (#2) } }
\draw_point_scale:nn 572 \cs_new:Npn \draw_point_diff:nn #1#2
573 { \__draw_point_to_dim:n { (#2) - (#1) } }
574 \cs_new:Npn \draw_point_scale:nn #1#2
575 { \__draw_point_to_dim:n { #1 * (#2) } }

```

Only a single point expression so the expansion is done here. The outcome is the normalised vector from (0,0) in the direction of the point, *i.e.*

$$P_x = \frac{x}{\sqrt{x^2 + y^2}} \quad P_y = \frac{y}{\sqrt{x^2 + y^2}}$$

```

576 \cs_new:Npn \draw_point_unit_vector:n #1
577 { \__draw_point_process:nn { \__draw_point_unit_vector:nn } {#1} }
578 \cs_new:Npn \__draw_point_unit_vector:nn #1#2
579 {
580   \__draw_point_to_dim:n
581     { ( #1 , #2 ) / (sqrt(#1 * #1 + #2 * #2)) }
582 }

```

3.5 Intersection calculations

The intersection point P between a line joining points (x_1, y_1) and (x_2, y_2) with a second line joining points (x_3, y_3) and (x_4, y_4) can be calculated using the formulae

$$P_x = \frac{(x_1 y_2 - y_1 x_2)(x_3 - x_4) - (x_3 y_4 - y_3 x_4)(x_1 - x_2)}{(x_1 - x_2)(y_3 - y_4) - (y_1 - y_2)(x_3 - x_4)}$$

and

$$P_y = \frac{(x_1 y_2 - y_1 x_2)(y_3 - y_4) - (x_3 y_4 - y_3 x_4)(y_1 - y_2)}{(x_1 - x_2)(y_3 - y_4) - (y_1 - y_2)(x_3 - x_4)}$$

The work therefore comes down to expanding the incoming data, then pre-calculating as many parts as possible before the final work to find the intersection. (Expansion and argument re-ordering is much less work than additional floating point calculations.)

```

583 \cs_new:Npn \draw_point_intersect_lines:nnnn #1#2#3#4
584 {
585   \__draw_point_process:nnn
586   {
587     \__draw_point_process:nnn
588     { \__draw_point_intersect_lines:nnnnnnnn } {#3} {#4}

```

```

589     }
590     {#1} {#2}
591 }

```

At this stage we have all of the information we need, fully expanded:

```

#1 x3
#2 y3
#3 x4
#4 y4
#5 x1
#6 y1
#7 x2
#8 y2

```

so now just have to do all of the calculation.

```

592 \cs_new:Npn \__draw_point_intersect_lines:nnnnnnnn #1#2#3#4#5#6#7#8
593 {
594   \__draw_point_intersect_lines_aux:ffffff
595   { \fp_eval:n { #1 * #4 - #2 * #3 } }
596   { \fp_eval:n { #5 * #8 - #6 * #7 } }
597   { \fp_eval:n { #1 - #3 } }
598   { \fp_eval:n { #5 - #7 } }
599   { \fp_eval:n { #2 - #4 } }
600   { \fp_eval:n { #6 - #8 } }
601 }
602 \cs_new:Npn \__draw_point_intersect_lines_aux:nnnnnn #1#2#3#4#5#6
603 {
604   \__draw_point_to_dim:n
605   {
606     ( #2 * #3 - #1 * #4 , #2 * #5 - #1 * #6 )
607     / ( #4 * #5 - #6 * #3 )
608   }
609 }
610 \cs_generate_variant:Nn \__draw_point_intersect_lines_aux:nnnnnn { fffffff }

```

Another long expansion chain to get the values in the right places. We have two circles, the first with center (a, b) and radius r , the second with center (c, d) and radius s . We use the intermediate values

```

\draw_point_intersect_circles:nnnnn
\__draw_point_intersect_circles_auxi:nnnnnnnn
\__draw_point_intersect_circles_auxii:nnnnnnnn
\__draw_point_intersect_circles_auxiii:ffnnnnnn
\__draw_point_intersect_circles_auxiiii:ffnnnnnn
\__draw_point_intersect_circles_auxiv:nnnnnnnn
\__draw_point_intersect_circles_auxiv:fnnnnnnn
\__draw_point_intersect_circles_auxv:nnnnnnnnnn
\__draw_point_intersect_circles_auxv:ffnnnnnnnn
\__draw_point_intersect_circles_auxvi:nnnnnnnnnn
\__draw_point_intersect_circles_auxvi:fnnnnnnnnn
\__draw_point_intersect_circles_auxvii:nnnnnnnn
\__draw_point_intersect_circles_auxvii:ffnnnnnn

```

$$\begin{aligned}
e &= c - a \\
f &= d - b \\
p &= \sqrt{e^2 + f^2} \\
k &= \frac{p^2 + r^2 - s^2}{2p}
\end{aligned}$$

in either

$$P_x = a + \frac{ek}{p} + \frac{f}{p}\sqrt{r^2 - k^2}$$

$$P_y = b + \frac{fk}{p} - \frac{e}{p}\sqrt{r^2 - k^2}$$

or

$$P_x = a + \frac{ek}{p} - \frac{f}{p}\sqrt{r^2 - k^2}$$

$$P_y = b + \frac{fk}{p} + \frac{e}{p}\sqrt{r^2 - k^2}$$

depending on which solution is required. The rest of the work is simply forcing the appropriate expansion and shuffling arguments.

```

611 \cs_new:Npn \draw_point_intersect_circles:nnnnn #1#2#3#4#5
612 {
613   \__draw_point_process:nnn
614   { \__draw_point_intersect_circles_auxi:nnnnnnn {#2} {#4} {#5} }
615   {#1} {#3}
616 }
617 \cs_new:Npn \__draw_point_intersect_circles_auxi:nnnnnnn #1#2#3#4#5#6#7
618 {
619   \__draw_point_intersect_circles_auxii:ffnnnnnn
620   { \fp_eval:n {#1} } { \fp_eval:n {#2} } {#4} {#5} {#6} {#7} {#3}
621 }

```

At this stage we have all of the information we need, fully expanded:

```

#1 r
#2 s
#3 a
#4 b
#5 c
#6 d
#7 n

```

Once we evaluate e and f , the co-ordinate (c, d) is no longer required: handy as we will need various intermediate values in the following.

```

622 \cs_new:Npn \__draw_point_intersect_circles_auxii:nnnnnnn #1#2#3#4#5#6#7
623 {
624   \__draw_point_intersect_circles_auxiii:ffnnnnnn
625   { \fp_eval:n { #5 - #3 } }
626   { \fp_eval:n { #6 - #4 } }
627   {#1} {#2} {#3} {#4} {#7}
628 }
629 \cs_generate_variant:Nn \__draw_point_intersect_circles_auxii:nnnnnnn { ff }
630 \cs_new:Npn \__draw_point_intersect_circles_auxiii:nnnnnnn #1#2#3#4#5#6#7
631 {

```



```

632   \_draw_point_intersect_circles_auxiv:fnnnnnnnn
633   { \fp_eval:n { sqrt( #1 * #1 + #2 * #2 ) } }
634   {#1} {#2} {#3} {#4} {#5} {#6} {#7}
635 }

```

```

636 \cs_generate_variant:Nn \_draw_point_intersect_circles_auxiii:nnnnnnnn { ff }

```

We now have p : we pre-calculate $1/p$ as it is needed a few times and is relatively expensive. We also need r^2 twice so deal with that here too.

```

637 \cs_new:Npn \_draw_point_intersect_circles_auxiv:nnnnnnnn #1#2#3#4#5#6#7#8
638 {
639   \_draw_point_intersect_circles_auxv:ffnnnnnnnn
640   { \fp_eval:n { 1 / #1 } }
641   { \fp_eval:n { #4 * #4 } }
642   {#1} {#2} {#3} {#5} {#6} {#7} {#8}
643 }
644 \cs_generate_variant:Nn \_draw_point_intersect_circles_auxiv:nnnnnnnn { f }
645 \cs_new:Npn \_draw_point_intersect_circles_auxv:nnnnnnnnnn #1#2#3#4#5#6#7#8#9
646 {
647   \_draw_point_intersect_circles_auxvi:fnnnnnnnn
648   { \fp_eval:n { 0.5 * #1 * ( #2 + #3 * #3 - #6 * #6 ) } }
649   {#1} {#2} {#4} {#5} {#7} {#8} {#9}
650 }
651 \cs_generate_variant:Nn \_draw_point_intersect_circles_auxv:nnnnnnnnnn { ff }

```

We now have all of the intermediate values we require, with one division carried out up-front to avoid doing this expensive step twice:

```

#1  $k$ 
#2  $1/p$ 
#3  $r^2$ 
#4  $e$ 
#5  $f$ 
#6  $a$ 
#7  $b$ 
#8  $n$ 

```

There are some final pre-calculations, k/p , $\frac{\sqrt{r^2-k^2}}{p}$ and the usage of n , then we can yield a result.

```

652 \cs_new:Npn \_draw_point_intersect_circles_auxvi:nnnnnnnn #1#2#3#4#5#6#7#8
653 {
654   \_draw_point_intersect_circles_auxvii:ffnnnnnn
655   { \fp_eval:n { #1 * #2 } }
656   { \int_if_odd:nTF {#8} { 1 } { -1 } }
657   { \fp_eval:n { sqrt ( #3 - #1 * #1 ) * #2 } }
658   {#4} {#5} {#6} {#7}
659 }
660 \cs_generate_variant:Nn \_draw_point_intersect_circles_auxvi:nnnnnnnn { f }
661 \cs_new:Npn \_draw_point_intersect_circles_auxvii:nnnnnnnn #1#2#3#4#5#6#7
662 {

```

```

663 \__draw_point_to_dim:n
664 { #6 + #4 * #1 + #2 * #3 * #5 , #7 + #5 * #1 + -1 * #2 * #3 * #4 }
665 }
666 \cs_generate_variant:Nn \__draw_point_intersect_circles_auxvii:nnnnnnn { fff }

```

3.6 Interpolation on a line (vector) or arc

Simple maths after expansion.

```

\draw_point_interpolate_line:nnn
\__draw_point_interpolate_line_aux:nnnnn
\__draw_point_interpolate_line_aux:fnnnn
\__draw_point_interpolate_line_aux:nnnnnnn
\__draw_point_interpolate_line_aux:fnnnnnnn
667 \cs_new:Npn \draw_point_interpolate_line:nnn #1#2#3
668 {
669 \__draw_point_process:nnn
670 { \__draw_point_interpolate_line_aux:fnnnn { \fp_eval:n {#1} } }
671 {#2} {#3}
672 }
673 \cs_new:Npn \__draw_point_interpolate_line_aux:nnnnn #1#2#3#4#5
674 {
675 \__draw_point_interpolate_line_aux:fnnnnn { \fp_eval:n { 1 - #1 } }
676 {#1} {#2} {#3} {#4} {#5}
677 }
678 \cs_generate_variant:Nn \__draw_point_interpolate_line_aux:nnnnn { f }
679 \cs_new:Npn \__draw_point_interpolate_line_aux:nnnnnn #1#2#3#4#5#6
680 { \__draw_point_to_dim:n { #2 * #3 + #1 * #5 , #2 * #4 + #1 * #6 } }
681 \cs_generate_variant:Nn \__draw_point_interpolate_line_aux:nnnnnnn { f }

```

Same idea but using the normalised length to obtain the scale factor.

```

\draw_point_interpolate_distance:nnn
\__draw_point_interpolate_distance:nnnnn
\__draw_point_interpolate_distance_aux:nnnnnnn
\__draw_point_interpolate_distance_aux:nnnnnnnn
\__draw_point_interpolate_distance_aux:nnnnnnnnn
\__draw_point_interpolate_distance_aux:fnnnnnnnn
682 \cs_new:Npn \draw_point_interpolate_distance:nnn #1#2#3
683 {
684 \__draw_point_process:nnn
685 { \__draw_point_interpolate_distance:nnnnn {#1} }
686 {#2} {#3}
687 }
688 \cs_new:Npn \__draw_point_interpolate_distance:nnnnn #1#2#3#4#5
689 {
690 \__draw_point_interpolate_distance_aux:nnnnnnn
691 { \fp_eval:n { #4 - #2 } }
692 { \fp_eval:n { #5 - #3 } }
693 {#2} {#3} {#4} {#5} {#1}
694 }
695 \cs_new:Npn \__draw_point_interpolate_distance_aux:nnnnnnnn #1#2#3#4#5#6#7
696 {
697 \__draw_point_interpolate_distance_aux:fnnnnn
698 { \fp_eval:n { (#7) / (sqrt ( #1 * #1 + #2 * #2 )) } }
699 {#3} {#4} {#5} {#6}
700 }
701 \cs_generate_variant:Nn \__draw_point_interpolate_distance_aux:nnnnnnnn { ff }
702 \cs_new:Npn \__draw_point_interpolate_distance_aux:nnnnnn #1#2#3#4#5
703 { \__draw_point_to_dim:n { #2 + #1 * #4 , #3 + #1 * #5 } }
704 \cs_generate_variant:Nn \__draw_point_interpolate_distance_aux:nnnnnnn { f }

```

```

\draw_point_interpolate_arcaxes:nnnnnnn
\__draw_point_interpolate_arcaxes_auxi:nnnnnnnnnn
\__draw_point_interpolate_arcaxes_auxii:nnnnnnnnnnn
\__draw_point_interpolate_arcaxes_auxiii:fnnnnnnnnnn
\__draw_point_interpolate_arcaxes_auxiiii:nnnnnnnnn
\__draw_point_interpolate_arcaxes_auxiii:fnnnnnnnn
\__draw_point_interpolate_arcaxes_auxiv:nnnnnnnnnn
\__draw_point_interpolate_arcaxes_auxiv:fnnnnnnnnn

```

Finding a point on an ellipse arc is relatively easy: find the correct angle between the two given, use the sine and cosine of that angle, apply to the axes. We just have to work a bit with the co-ordinate expansion.

```

705 \cs_new:Npn \draw_point_interpolate_arcaxes:nnnnnn #1#2#3#4#5#6

```

```

706 {
707   \_draw_point_process:nnn
708   {
709     \_draw_point_process:nn
710     { \_draw_point_interpolate_arcaxes_auxi:nnnnnnnnn {#1} {#5} {#6} }
711     {#4}
712   }
713   {#2} {#3}
714 }
715 \cs_new:Npn \_draw_point_interpolate_arcaxes_auxi:nnnnnnnnn #1#2#3#4#5#6#7#8#9
716 {
717   \_draw_point_interpolate_arcaxes_auxii:fnnnnnnnn
718   { \fp_eval:n {#1} } {#2} {#3} {#6} {#7} {#8} {#9} {#4} {#5}
719 }

```

At this stage, the three co-ordinate pairs are fully expanded but somewhat re-ordered:

```

#1 p
#2  $\theta_1$ 
#3  $\theta_2$ 
#4  $x_c$ 
#5  $y_c$ 
#6  $x_{a1}$ 
#7  $y_{a1}$ 
#8  $x_{a2}$ 
#9  $y_{a2}$ 

```

We are now in a position to find the target angle, and from that the sine and cosine required.

```

720 \cs_new:Npn \_draw_point_interpolate_arcaxes_auxii:nnnnnnnnn #1#2#3#4#5#6#7#8#9
721 {
722   \_draw_point_interpolate_arcaxes_auxiii:fnnnnnnn
723   { \fp_eval:n { #1 * (#3) + ( 1 - #1 ) * (#2) } }
724   {#4} {#5} {#6} {#7} {#8} {#9}
725 }
726 \cs_generate_variant:Nn \_draw_point_interpolate_arcaxes_auxii:nnnnnnnnn { f }
727 \cs_new:Npn \_draw_point_interpolate_arcaxes_auxiii:nnnnnnn #1#2#3#4#5#6#7
728 {
729   \_draw_point_interpolate_arcaxes_auxiv:ffnnnnnnn
730   { \fp_eval:n { cosd (#1) } }
731   { \fp_eval:n { sind (#1) } }
732   {#2} {#3} {#4} {#5} {#6} {#7}
733 }
734 \cs_generate_variant:Nn \_draw_point_interpolate_arcaxes_auxiii:nnnnnnn { f }
735 \cs_new:Npn \_draw_point_interpolate_arcaxes_auxiv:nnnnnnnnn #1#2#3#4#5#6#7#8
736 {
737   \_draw_point_to_dim:n
738   { #3 + #1 * #5 + #2 * #7 , #4 + #1 * #6 + #2 * #8 }
739 }
740 \cs_generate_variant:Nn \_draw_point_interpolate_arcaxes_auxiv:nnnnnnnnn { ff }

```

(End definition for `_draw_point_to_dim:n` and others. These functions are documented on page 3.)

```
\draw_point_interpolate_curve:nnnnn
\draw_point_interpolate_curve_auxi:nnnnnnnnn
\draw_point_interpolate_curve_auxii:nnnnnnnnn
\draw_point_interpolate_curve_auxiii:nnnnnnnnn
\draw_point_interpolate_curve_auxiiii:nnnnnnnnn
\draw_point_interpolate_curve_auxv:nnnnnnnnn
\draw_point_interpolate_curve_auxvi:nnnnnnnnn
\draw_point_interpolate_curve_auxvii:nnnnnnnnn
\draw_point_interpolate_curve_auxviii:nnnnnnnnn
\draw_point_interpolate_curve_auxviiii:nnnnnnnnn
```

Here we start with a proportion of the curve (p) and four points

1. The initial point (x_1, y_1)
2. The first control point (x_2, y_2)
3. The second control point (x_3, y_3)
4. The final point (x_4, y_4)

The first phase is to expand out all of these values.

```
741 \cs_new:Npn \draw_point_interpolate_curve:nnnnnn #1#2#3#4#5
742 {
743   \__draw_point_process:nnn
744   {
745     \__draw_point_process:nnn
746     { \__draw_point_interpolate_curve_auxi:nnnnnnnnn {#1} }
747     {#4} {#5}
748   }
749   {#2} {#3}
750 }
751 \cs_new:Npn \__draw_point_interpolate_curve_auxi:nnnnnnnnn #1#2#3#4#5#6#7#8#9
752 {
753   \__draw_point_interpolate_curve_auxii:fnnnnnnnnn
754   { \fp_eval:n {#1} }
755   {#6} {#7} {#8} {#9} {#2} {#3} {#4} {#5}
756 }
```

At this stage, everything is fully expanded and back in the input order. The approach to finding the required point is iterative. We carry out three phases. In phase one, we need all of the input co-ordinates

$$\begin{aligned}
 x'_1 &= (1-p)x_1 + px_2 \\
 y'_1 &= (1-p)y_1 + py_2 \\
 x'_2 &= (1-p)x_2 + px_3 \\
 y'_2 &= (1-p)y_2 + py_3 \\
 x'_3 &= (1-p)x_3 + px_4 \\
 y'_3 &= (1-p)y_3 + py_4
 \end{aligned}$$

In the second stage, we can drop the final point

$$\begin{aligned}
 x''_1 &= (1-p)x'_1 + px'_2 \\
 y''_1 &= (1-p)y'_1 + py'_2 \\
 x''_2 &= (1-p)x'_2 + px'_3 \\
 y''_2 &= (1-p)y'_2 + py'_3
 \end{aligned}$$

and for the final stage only need one set of calculations

$$\begin{aligned}
 P_x &= (1-p)x''_1 + px''_2 \\
 P_y &= (1-p)y''_1 + py''_2
 \end{aligned}$$

Of course, this does mean a lot of calculations and expansion!

```

757 \cs_new:Npn \__draw_point_interpolate_curve_auxii:nnnnnnnnn
758   #1#2#3#4#5#6#7#8#9
759   {
760     \__draw_point_interpolate_curve_auxiii:fnnnnnn
761     { \fp_eval:n { 1 - #1 } }
762     {#1}
763     { {#2} {#3} } { {#4} {#5} } { {#6} {#7} } { {#8} {#9} }
764   }
765 \cs_generate_variant:Nn \__draw_point_interpolate_curve_auxii:nnnnnnnnn { f }
766 % \begin{macrocode}
767 % We need to do the first cycle, but haven't got enough arguments to keep
768 % everything in play at once. So here we use a but of argument re-ordering
769 % and a single auxiliary to get the job done.
770 % \begin{macrocode}
771 \cs_new:Npn \__draw_point_interpolate_curve_auxiii:nnnnnn #1#2#3#4#5#6
772   {
773     \__draw_point_interpolate_curve_auxiv:nnnnnn {#1} {#2} #3 #4
774     \__draw_point_interpolate_curve_auxiv:nnnnnn {#1} {#2} #4 #5
775     \__draw_point_interpolate_curve_auxiv:nnnnnn {#1} {#2} #5 #6
776     \prg_do_nothing:
777     \__draw_point_interpolate_curve_auxvi:n { {#1} {#2} }
778   }
779 \cs_generate_variant:Nn \__draw_point_interpolate_curve_auxiii:nnnnnn { f }
780 \cs_new:Npn \__draw_point_interpolate_curve_auxiv:nnnnnn #1#2#3#4#5#6
781   {
782     \__draw_point_interpolate_curve_auxv:ffw
783     { \fp_eval:n { #1 * #3 + #2 * #5 } }
784     { \fp_eval:n { #1 * #4 + #2 * #6 } }
785   }
786 \cs_new:Npn \__draw_point_interpolate_curve_auxv:nnw
787   #1#2#3 \prg_do_nothing: #4#5
788   {
789     #3
790     \prg_do_nothing:
791     #4 { #5 {#1} {#2} }
792   }
793 \cs_generate_variant:Nn \__draw_point_interpolate_curve_auxv:nnw { ff }
794 % \begin{macrocode}
795 % Get the arguments back into the right places and to the second and
796 % third cycles directly.
797 % \begin{macrocode}
798 \cs_new:Npn \__draw_point_interpolate_curve_auxvi:n #1
799   { \__draw_point_interpolate_curve_auxvii:nnnnnnnn #1 }
800 \cs_new:Npn \__draw_point_interpolate_curve_auxvii:nnnnnnnn #1#2#3#4#5#6#7#8
801   {
802     \__draw_point_interpolate_curve_auxviii:ffffnn
803     { \fp_eval:n { #1 * #5 + #2 * #3 } }
804     { \fp_eval:n { #1 * #6 + #2 * #4 } }
805     { \fp_eval:n { #1 * #7 + #2 * #5 } }
806     { \fp_eval:n { #1 * #8 + #2 * #6 } }
807     {#1} {#2}
808   }
809 \cs_new:Npn \__draw_point_interpolate_curve_auxviii:nnnnnn #1#2#3#4#5#6

```

```

810 {
811   \__draw_point_to_dim:n
812   { #5 * #3 + #6 * #1 , #5 * #4 + #6 * #2 }
813 }
814 \cs_generate_variant:Nn \__draw_point_interpolate_curve_auxviii:nnnnnn { ffff }

```

(End definition for \draw_point_interpolate_curve:nnnnn and others. These functions are documented on page ??.)

3.7 Vector support

As well as co-ordinates relative to the drawing

\l__draw_xvec_x_dim Base vectors to map to the underlying two-dimensional drawing space.

```

\l__draw_xvec_y_dim 815 \dim_new:N \l__draw_xvec_x_dim
\l__draw_yvec_x_dim 816 \dim_new:N \l__draw_xvec_y_dim
\l__draw_yvec_y_dim 817 \dim_new:N \l__draw_yvec_x_dim
\l__draw_zvec_x_dim 818 \dim_new:N \l__draw_yvec_y_dim
\l__draw_zvec_y_dim 819 \dim_new:N \l__draw_zvec_x_dim
820 \dim_new:N \l__draw_zvec_y_dim

```

(End definition for \l__draw_xvec_x_dim and others.)

```

\draw_xvec:n Calculate the underlying position and store it.
\draw_yvec:n 821 \cs_new_protected:Npn \draw_xvec:n #1
\draw_zvec:n 822 { \__draw_vec:nn { x } {#1} }
\__draw_vec:nn 823 \cs_new_protected:Npn \draw_yvec:n #1
\__draw_vec:nnn 824 { \__draw_vec:nn { y } {#1} }
825 \cs_new_protected:Npn \draw_zvec:n #1
826 { \__draw_vec:nn { z } {#1} }
827 \cs_new_protected:Npn \__draw_vec:nn #1#2
828 {
829   \__draw_point_process:nn { \__draw_vec:nnn {#1} } {#2}
830 }
831 \cs_new_protected:Npn \__draw_vec:nnn #1#2#3
832 {
833   \dim_set:cn { l__draw_ #1 vec_x_dim } {#2}
834   \dim_set:cn { l__draw_ #1 vec_y_dim } {#3}
835 }

```

(End definition for \draw_xvec:n and others. These functions are documented on page ??.)

Initialise the vectors.

```

836 \draw_xvec:n { 1cm , 0cm }
837 \draw_yvec:n { 0cm , 1cm }
838 \draw_zvec:n { -0.385cm , -0.385cm }

```

```

\draw_point_vec:nn Force a single evaluation of each factor, then use these to work out the underlying point.
\__draw_point_vec:nn 839 \cs_new:Npn \draw_point_vec:nn #1#2
\__draw_point_vec:ff 840 { \__draw_point_vec:ff { \fp_eval:n {#1} } { \fp_eval:n {#2} } }
\draw_point_vec:nnn 841 \cs_new:Npn \__draw_point_vec:nn #1#2
\__draw_point_vec:nnn 842 {
\__draw_point_vec:fff 843   \__draw_point_to_dim:n
844   {
845     #1 * \l__draw_xvec_x_dim + #2 * \l__draw_yvec_x_dim ,

```

```

846         #1 * \l__draw_xvec_y_dim + #2 * \l__draw_yvec_y_dim
847     }
848 }
849 \cs_generate_variant:Nn \__draw_point_vec:nn { ff }
850 \cs_new:Npn \draw_point_vec:nnn #1#2#3
851 {
852     \__draw_point_vec:fff
853     { \fp_eval:n {#1} } { \fp_eval:n {#2} } { \fp_eval:n {#3} }
854 }
855 \cs_new:Npn \__draw_point_vec:nnn #1#2#3
856 {
857     \__draw_point_to_dim:n
858     {
859         #1 * \l__draw_xvec_x_dim
860         + #2 * \l__draw_yvec_x_dim
861         + #3 * \l__draw_zvec_x_dim
862     ,
863         #1 * \l__draw_xvec_y_dim
864         + #2 * \l__draw_yvec_y_dim
865         + #3 * \l__draw_zvec_y_dim
866     }
867 }
868 \cs_generate_variant:Nn \__draw_point_vec:nnn { fff }

```

(End definition for `\draw_point_vec:nn` and others. These functions are documented on page 3.)

```

\draw_point_vec_polar:nn Much the same as the core polar approach.
\draw_point_vec_polar:nnn
\__draw_point_vec_polar:nnn
\__draw_point_vec_polar:fnn
869 \cs_new:Npn \draw_point_vec_polar:nn #1#2
870 { \draw_point_vec_polar:nnn {#1} {#2} {#2} }
871 \cs_new:Npn \draw_point_vec_polar:nnn #1#2#3
872 { \__draw_draw_vec_polar:fnn { \fp_eval:n {#1} } {#2} {#3} }
873 \cs_new:Npn \__draw_draw_vec_polar:nnn #1#2#3
874 {
875     \__draw_point_to_dim:n
876     {
877         cosd(#1) * (#2) * \l__draw_xvec_x_dim ,
878         sind(#1) * (#3) * \l__draw_yvec_y_dim
879     }
880 }
881 \cs_generate_variant:Nn \__draw_draw_vec_polar:nnn { f }

```

(End definition for `\draw_point_vec_polar:nn`, `\draw_point_vec_polar:nnn`, and `__draw_point_vec_polar:nnn`. These functions are documented on page 4.)

3.8 Transformations

`\draw_point_transform:nn` Applies a transformation matrix to a point: see `l3draw-transforms` for the business end. Where possible, we avoid the relatively expensive multiplication step.

```

882 \cs_new:Npn \draw_point_transform:n #1
883 {
884     \__draw_point_process:nn
885     { \__draw_point_transform:nn } {#1}
886 }
887 \cs_new:Npn \__draw_point_transform:nn #1#2

```

```

888 {
889   \bool_if:NTF \l__draw_transformmcm_active_bool
890   {
891     \__draw_point_to_dim:n
892     {
893       (
894         \l__draw_transformmcm_aa_fp * #1
895         + \l__draw_transformmcm_ba_fp * #2
896         + \l__draw_transformmcm_xshift_dim
897       )
898       ,
899       (
900         \l__draw_transformmcm_ab_fp * #1
901         + \l__draw_transformmcm_bb_fp * #2
902         + \l__draw_transformmcm_yshift_dim
903       )
904     }
905   }
906   {
907     \__draw_point_to_dim:n
908     {
909       (#1, #2)
910       + ( \l__draw_transformmcm_xshift_dim ,
911          \l__draw_transformmcm_yshift_dim )
912     }
913   }
914 }

```

(End definition for `\draw_point_transform:n` and `__draw_point_transform:nn`. This function is documented on page 3.)

```

\__draw_point_transform_noshift:n A version with no shift: used for internal purposes.
\__draw_point_transform_noshift:nn
915 \cs_new:Npn \__draw_point_transform_noshift:n #1
916 {
917   \__draw_point_process:nn
918   { \__draw_point_transform_noshift:nn } {#1}
919 }
920 \cs_new:Npn \__draw_point_transform_noshift:nn #1#2
921 {
922   \bool_if:NTF \l__draw_transformmcm_active_bool
923   {
924     \__draw_point_to_dim:n
925     {
926       (
927         \l__draw_transformmcm_aa_fp * #1
928         + \l__draw_transformmcm_ba_fp * #2
929       )
930       ,
931       (
932         \l__draw_transformmcm_ab_fp * #1
933         + \l__draw_transformmcm_bb_fp * #2
934       )
935     }
936   }

```



```

937     { \__draw_point_to_dim:n { (#1, #2) } }
938   }

```

(End definition for `__draw_point_transform_noshift:n` and `__draw_point_transform_noshift:nn`.)

```

939 \</initex | package>

```

4 l3draw-scopes implementation

```

940 \*initex | package>

```

```

941 \@@=draw>

```

4.1 Drawing environment

`\g__draw_xmax_dim` Used to track the overall (official) size of the image created: may not actually be the natural size of the content.

```

\g__draw_xmin_dim
\g__draw_ymax_dim
\g__draw_ymin_dim
942 \dim_new:N \g__draw_xmax_dim
943 \dim_new:N \g__draw_xmin_dim
944 \dim_new:N \g__draw_ymax_dim
945 \dim_new:N \g__draw_ymin_dim

```

(End definition for `\g__draw_xmax_dim` and others.)

`\l__draw_update_bb_bool` Flag to indicate that a path (or similar) should update the bounding box of the drawing.

```

946 \bool_new:N \l__draw_update_bb_bool

```

(End definition for `\l__draw_update_bb_bool`.)

`\l__draw_main_box` Box for setting the drawing.

```

947 \box_new:N \l__draw_main_box

```

(End definition for `\l__draw_main_box`.)

`\draw_begin:` Drawings are created by setting them into a box, then adjusting the box before inserting into the surroundings. At present the content is simply collected then dumped: work will be required to manipulate the size as this data becomes more defined. It may be that a coffin construct is better here in the longer term: that may become clearer as the code is completed. Another obvious question is whether/where vertical mode should be ended (*i.e.* should this behave like a raw `\vbox` or like a coffin). In contrast to `pgf`, we use a vertical box here: material between explicit instructions should not be present anyway. (Consider adding an `\everypar` hook as done for the $\text{\LaTeX} 2_{\epsilon}$ preamble.)

```

948 \cs_new_protected:Npn \draw_begin:
949   {
950     \vbox_set:Nw \l__draw_main_box
951     \driver_draw_begin:
952     \dim_gset:Nn \g__draw_xmax_dim { -\c_max_dim }
953     \dim_gset:Nn \g__draw_xmin_dim { \c_max_dim }
954     \dim_gset:Nn \g__draw_ymax_dim { -\c_max_dim }
955     \dim_gset:Nn \g__draw_ymin_dim { \c_max_dim }
956     \bool_set_true:N \l__draw_update_bb_bool
957     \draw_transform_reset:
958     \draw_linewidth:n { \l__draw_default_linewidth_dim }
959   }
960 \cs_new_protected:Npn \draw_end:

```

```

961 {
962   \driver_draw_end:
963   \vbox_set_end:
964   \hbox_set:Nn \l__draw_main_box
965   {
966     \skip_horizontal:n { -\g__draw_xmin_dim }
967     \box_move_down:nn { \g__draw_ymin_dim }
968     { \box_use_drop:N \l__draw_main_box }
969   }
970   \box_set_ht:Nn \l__draw_main_box
971   { \g__draw_ymax_dim - \g__draw_ymin_dim }
972   \box_set_dp:Nn \l__draw_main_box { Opt }
973   \box_set_wd:Nn \l__draw_main_box
974   { \g__draw_xmax_dim - \g__draw_xmin_dim }
975   \mode_leave_vertical:
976   \box_use_drop:N \l__draw_main_box
977 }

```

(End definition for `\draw_begin:` and `\draw_end:`. These functions are documented on page 1.)

```

978 \</initex | package>

```

5 l3draw-softpath implementation

```

979 \<*initex | package>

```

```

980 \<@@=draw>

```

There are two linked aims in the code here. The most significant is to provide a way to modify paths, for example to shorten the ends or round the corners. This means that the path cannot be written piecemeal as specials, but rather needs to be held in macros. The second aspect that follows from this is performance: simply adding to a single macro a piece at a time will have poor performance as the list gets long. Paths need to be global (as specials are), so we cannot use `l3tl-build` or a similar approach. Instead, we use the same idea as `pgf`: use a series of buffer macros such that in most cases we don't add tokens to the main list. This will get slow only for *enormous* paths.

Each marker (operation) token takes two arguments, which makes processing more straight-forward. As such, some operations have dummy arguments, whilst others have to be split over several tokens. As the code here is at a low level, all dimension arguments are assumed to be explicit and fully-expanded.

The soft path itself.

```

\g__draw_softpath_main_tl
\g__draw_softpath_buffer_a_tl
\g__draw_softpath_buffer_b_tl
981 \tl_new:N \g__draw_softpath_main_tl
982 \tl_new:N \g__draw_softpath_buffer_a_tl
983 \tl_new:N \g__draw_softpath_buffer_b_tl

```

(End definition for `\g__draw_softpath_main_tl`, `\g__draw_softpath_buffer_a_tl`, and `\g__draw_softpath_buffer_b_tl`.)

Tracking data.

```

\g__draw_softpath_buffer_a_int
\g__draw_softpath_buffer_b_int
984 \int_new:N \g__draw_softpath_buffer_a_int
985 \int_new:N \g__draw_softpath_buffer_b_int

```

(End definition for `\g__draw_softpath_buffer_a_int` and `\g__draw_softpath_buffer_b_int`.)

`_draw_softpath_add:n` The softpath itself is quite simple. We use three token lists to hold the data: two buffers of limited length, and the main list of arbitrary size. Most of the time this will mean that we don't add to the full list, so performance will be acceptable.
`_draw_softpath_add:x`
`_draw_softpath_concat:n`
`_draw_softpath_reset_buffers:`

```

986 \cs_new_protected:Npn \\_draw_softpath_add:n #1
987 {
988   \int_compare:nNnTF \g__draw_softpath_buffer_a_int < { 40 }
989   {
990     \int_gincr:N \g__draw_softpath_buffer_a_int
991     \tl_gput_right:Nn \g__draw_softpath_buffer_a_tl {#1}
992   }
993   {
994     \int_compare:nNnTF \g__draw_softpath_buffer_b_int < { 40 }
995     {
996       \int_gincr:N \g__draw_softpath_buffer_b_int
997       \tl_gset:Nx \g__draw_softpath_buffer_b_tl
998       {
999         \exp_not:V \g__draw_softpath_buffer_b_tl
1000        \exp_not:V \g__draw_softpath_buffer_a_tl
1001        \exp_not:n {#1}
1002      }
1003      \int_gzero:N \g__draw_softpath_buffer_a_int
1004      \tl_gclear:N \g__draw_softpath_buffer_a_tl
1005    }
1006    { \\_draw_softpath_concat:n {#1} }
1007  }
1008 }
1009 \cs_generate_variant:Nn \\_draw_softpath_add:n { x }
1010 \cs_new_protected:Npn \\_draw_softpath_concat:n #1
1011 {
1012   \tl_gset:Nx \g__draw_softpath_main_tl
1013   {
1014     \exp_not:V \g__draw_softpath_main_tl
1015     \exp_not:V \g__draw_softpath_buffer_b_tl
1016     \exp_not:V \g__draw_softpath_buffer_a_tl
1017     \exp_not:n {#1}
1018   }
1019   \\_draw_softpath_reset_buffers:
1020 }
1021 \cs_new_protected:Npn \\_draw_softpath_reset_buffers:
1022 {
1023   \int_gzero:N \g__draw_softpath_buffer_a_int
1024   \tl_gclear:N \g__draw_softpath_buffer_a_tl
1025   \int_gzero:N \g__draw_softpath_buffer_b_int
1026   \tl_gclear:N \g__draw_softpath_buffer_b_tl
1027 }

```

(End definition for _draw_softpath_add:n, _draw_softpath_concat:n, and _draw_softpath_reset_buffers:.)

`_draw_softpath_get:N` Save and restore functions.
`_draw_softpath_set_eq:N`

```

1028 \cs_new_protected:Npn \\_draw_softpath_get:N #1
1029 {
1030   \\_draw_softpath_concat:n { }
1031   \tl_set_eq:NN #1 \g__draw_softpath_main_tl

```

```

1032 }
1033 \cs_new_protected:Npn \__draw_softpath_set_eq:N #1
1034 {
1035   \tl_gset_eq:NN \g__draw_softpath_main_tl #1
1036   \__draw_softpath_reset_buffers:
1037 }

```

(End definition for __draw_softpath_get:N and __draw_softpath_set_eq:N.)

```

\__draw_softpath_use: Using and clearing is trivial.
\__draw_softpath_clear:
\__draw_softpath_use_clear:
1038 \cs_new_protected:Npn \__draw_softpath_use:
1039 {
1040   \g__draw_softpath_main_tl
1041   \g__draw_softpath_buffer_b_tl
1042   \g__draw_softpath_buffer_a_tl
1043 }
1044 \cs_new_protected:Npn \__draw_softpath_clear:
1045 {
1046   \tl_gclear:N \g__draw_softpath_main_tl
1047   \tl_gclear:N \g__draw_softpath_buffer_a_tl
1048   \tl_gclear:N \g__draw_softpath_buffer_b_tl
1049 }
1050 \cs_new_protected:Npn \__draw_softpath_use_clear:
1051 {
1052   \__draw_softpath_use:
1053   \__draw_softpath_clear:
1054 }

```

(End definition for __draw_softpath_use:, __draw_softpath_clear:, and __draw_softpath_use_clear:.)

```

\g__draw_softpath_lastx_dim For tracking the end of the path (to close it).
\g__draw_softpath_lasty_dim
1055 \dim_new:N \g__draw_softpath_lastx_dim
1056 \dim_new:N \g__draw_softpath_lasty_dim

```

(End definition for \g__draw_softpath_lastx_dim and \g__draw_softpath_lasty_dim.)

```

\g__draw_softpath_move_bool Track if moving a point should update the close position.
1057 \bool_new:N \g__draw_softpath_move_bool
1058 \bool_gset_true:N \g__draw_softpath_move_bool

```

(End definition for \g__draw_softpath_move_bool.)

```

\__draw_softpath_curveto:nnnnnn The various parts of a path expressed as the appropriate soft path functions.
\__draw_softpath_lineto:nn
\__draw_softpath_moveto:nn
\__draw_softpath_rectangle:nnnn
\__draw_softpath_roundpoint:nn
\__draw_softpath_roundpoint:VV
1059 \cs_new_protected:Npn \__draw_softpath_closepath:
1060 {
1061   \__draw_softpath_add:x
1062   {
1063     \__draw_softpath_close_op:nn
1064     { \dim_use:N \g__draw_softpath_lastx_dim }
1065     { \dim_use:N \g__draw_softpath_lasty_dim }
1066   }
1067 }
1068 \cs_new_protected:Npn \__draw_softpath_curveto:nnnnnn #1#2#3#4#5#6
1069 {

```

```

1070     \__draw_softpath_add:n
1071     {
1072         \__draw_softpath_curveto_opi:nn {#1} {#2}
1073         \__draw_softpath_curveto_opii:nn {#3} {#4}
1074         \__draw_softpath_curveto_opiii:nn {#5} {#6}
1075     }
1076 }
1077 \cs_new_protected:Npn \__draw_softpath_lineto:nn #1#2
1078 {
1079     \__draw_softpath_add:n
1080     { \__draw_softpath_lineto_op:nn {#1} {#2} }
1081 }
1082 \cs_new_protected:Npn \__draw_softpath_moveto:nn #1#2
1083 {
1084     \__draw_softpath_add:n
1085     { \__draw_softpath_moveto_op:nn {#1} {#2} }
1086     \bool_if:NT \g__draw_softpath_move_bool
1087     {
1088         \dim_gset:Nn \g__draw_softpath_lastx_dim {#1}
1089         \dim_gset:Nn \g__draw_softpath_lasty_dim {#2}
1090     }
1091 }
1092 \cs_new_protected:Npn \__draw_softpath_rectangle:nnnn #1#2#3#4
1093 {
1094     \__draw_softpath_add:n
1095     {
1096         \__draw_softpath_rectangle_opi:nn {#1} {#2}
1097         \__draw_softpath_rectangle_opii:nn {#3} {#4}
1098     }
1099 }
1100 \cs_new_protected:Npn \__draw_softpath_roundpoint:nn #1#2
1101 {
1102     \__draw_softpath_add:n
1103     { \__draw_softpath_roundpoint_op:nn {#1} {#2} }
1104 }
1105 \cs_generate_variant:Nn \__draw_softpath_roundpoint:nn { VV }

```

(End definition for __draw_softpath_curveto:nnnnnn and others.)

__draw_softpath_close_op:nn The markers for operations: all the top-level ones take two arguments.

```

1106 \cs_new_protected:Npn \__draw_softpath_close_op:nn #1#2
1107 { \driver_draw_closepath: }
1108 \cs_new_protected:Npn \__draw_softpath_curveto_opi:nn #1#2
1109 { \__draw_softpath_curveto_opi:nnNnnNnn {#1} {#2} }
1110 \cs_new_protected:Npn \__draw_softpath_curveto_opi:nnNnnNnn #1#2#3#4#5#6#7#8
1111 { \driver_draw_curveto:nnnnnn {#1} {#2} {#4} {#5} {#7} {#8} }
1112 \cs_new_protected:Npn \__draw_softpath_curveto_opii:nn #1#2 { }
1113 \cs_new_protected:Npn \__draw_softpath_curveto_opiii:nn #1#2 { }
1114 \cs_new_protected:Npn \__draw_softpath_lineto_op:nn #1#2
1115 { \driver_draw_lineto:nn {#1} {#2} }
1116 \cs_new_protected:Npn \__draw_softpath_moveto_op:nn #1#2
1117 { \driver_draw_moveto:nn {#1} {#2} }
1118 \cs_new_protected:Npn \__draw_softpath_roundpoint_op:nn #1#2 { }
1119 \cs_new_protected:Npn \__draw_softpath_rectangle_opi:nn #1#2

```

```

1120 { \_draw_softpath_rectangle_opi:nnNnn {#1} {#2} }
1121 \cs_new_protected:Npn \_draw_softpath_rectangle_opi:nnNnn #1#2#3#4#5
1122 { \driver_draw_rectangle:nnnn {#1} {#2} {#4} {#5} }
1123 \cs_new_protected:Npn \_draw_softpath_rectangle_opii:nn #1#2 { }

(End definition for \_draw_softpath_close_op:nn and others.)

1124 \</initex | package>

```

6 l3draw-state implementation

```

1125 <*initex | package>
1126 <@@=draw>

```

`\g__draw_linewidth_dim` Linewidth for strokes: global as the scope for this relies on the graphics state. The inner
`\g__draw_inner_linewidth_dim` line width is used for places where two lines are used.

```

1127 \dim_new:N \g__draw_linewidth_dim
1128 \dim_new:N \g__draw_inner_linewidth_dim

```

(End definition for `\g__draw_linewidth_dim` and `\g__draw_inner_linewidth_dim`.)

`\l_draw_default_linewidth_dim` A default: this is used at the start of every drawing.

```

1129 \dim_new:N \l_draw_default_linewidth_dim
1130 \dim_set:Nn \l_draw_default_linewidth_dim { 0.4pt }

```

(End definition for `\l_draw_default_linewidth_dim`. This variable is documented on page ??.)

`\draw_linewidth:n` Set the linewidth: we need a wrapper as this has to pass to the driver layer. The inner
`\draw_inner_linewidth:n` version is handled at the macro layer but is given a consistent interface here.

```

1131 \cs_new_protected:Npn \draw_linewidth:n #1
1132 {
1133   \dim_gset:Nn \g__draw_linewidth_dim { \fp_to_dim:n {#1} }
1134   \driver_draw_linewidth:n \g__draw_linewidth_dim
1135 }
1136 \cs_new_protected:Npn \draw_inner_linewidth:n #1
1137 { \dim_gset:Nn \g__draw_inner_linewidth_dim { \fp_to_dim:n {#1} } }

```

(End definition for `\draw_linewidth:n` and `\draw_inner_linewidth:n`. These functions are documented on page 2.)

`\draw_miterlimit:n` Pass through to the driver layer.

```

1138 \cs_new_protected:Npn \draw_miterlimit:n #1
1139 { \driver_draw_miterlimit:n { \fp_to_dim:n {#1} } }

```

(End definition for `\draw_miterlimit:n`. This function is documented on page 2.)

`\draw_cap_but:` All straight wrappers.

```

\draw_cap_rectangle: 1140 \cs_new_protected:Npn \draw_cap_but: { \driver_draw_cap_but: }
\draw_cap_round:     1141 \cs_new_protected:Npn \draw_cap_rectangle: { \driver_draw_cap_rectangle: }
\draw_evenodd_rule:  1142 \cs_new_protected:Npn \draw_cap_round: { \driver_draw_cap_round: }
\draw_nonzero_rule:  1143 \cs_new_protected:Npn \draw_evenodd_rule: { \driver_draw_evenodd_rule: }
\draw_join_bevel:    1144 \cs_new_protected:Npn \draw_nonzero_rule: { \driver_draw_nonzero_rule: }
\draw_join_miter:    1145 \cs_new_protected:Npn \draw_join_bevel: { \driver_draw_join_bevel: }
\draw_join_round:    1146 \cs_new_protected:Npn \draw_join_miter: { \driver_draw_join_miter: }
                    1147 \cs_new_protected:Npn \draw_join_round: { \driver_draw_join_round: }

```

(End definition for `\draw_cap_but`: and others. These functions are documented on page 2.)

```

\l__draw_color_tmp_tl Scratch space.
1148 \tl_new:N \l__draw_color_tmp_tl
(End definition for \l__draw_color_tmp_tl.)

\g__draw_fill_color_tl For tracking.
\g__draw_stroke_color_tl 1149 \tl_new:N \g__draw_fill_color_tl
1150 \tl_new:N \g__draw_stroke_color_tl
(End definition for \g__draw_fill_color_tl and \g__draw_stroke_color_tl.)

\draw_color:n Much the same as for core color support but calling the relevant driver-level function.
\draw_color_fill:n 1151 \cs_new_protected:Npn \draw_color:n #1
\draw_color_stroke:n 1152 { \__draw_color:nn { } {#1} }
\__draw_color:nn 1153 \cs_new_protected:Npn \draw_color_fill:n #1
\__draw_color_aux:nn 1154 { \__draw_color:nn { fill } {#1} }
\__draw_color_aux:Vn 1155 \cs_new_protected:Npn \draw_color_stroke:n #1
\__draw_color:nw 1156 { \__draw_color:nn { stroke } {#1} }
\__draw_select_cmyk:nw 1157 \cs_new_protected:Npn \__draw_color:nn #1#2
\__draw_select_gray:nw 1158 {
\__draw_select_rgb:nw 1159 \color_parse:nn {#2} \l__draw_color_tmp_tl
\__draw_split_select:nw 1160 \tl_if_blank:nTF {#1}
1161 {
1162 \tl_gset_eq:NN \g__draw_fill_color_tl \l__draw_color_tmp_tl
1163 \tl_gset_eq:NN \g__draw_stroke_color_tl \l__draw_color_tmp_tl
1164 \__draw_color_aux:Vn \l__draw_color_tmp_tl { color }
1165 }
1166 {
1167 \tl_gset_eq:cN { g__draw_ #1 _color_tl } \l__draw_color_tmp_tl
1168 \__draw_color_aux:Vn \l__draw_color_tmp_tl { #1 }
1169 }
1170 }
1171 \cs_new_protected:Npn \__draw_color_aux:nn #1#2
1172 { \__draw_color:nw {#2} #1 \q_stop }
1173 \cs_generate_variant:Nn \__draw_color_aux:nn { V }
1174 \cs_new_protected:Npn \__draw_color:nw #1#2 ~ #3 \q_stop
1175 { \use:c { __draw_color_ #2 :nw } {#1} #3 \q_stop }
1176 \cs_new_protected:Npn \__draw_color_cmyk:nw #1#2 ~ #3 ~ #4 ~ #5 \q_stop
1177 { \use:c { driver_draw_ #1 _cmyk:nnnn } {#2} {#3} {#4} {#5} }
1178 \cs_new_protected:Npn \__draw_color_gray:nw #1#2 \q_stop
1179 { \use:c { driver_draw_ #1 _gray:n } {#2} }
1180 \cs_new_protected:Npn \__draw_color_rgb:nw #1#2 ~ #3 ~ #4 \q_stop
1181 { \use:c { driver_draw_ #1 _rgb:nnn } {#2} {#3} {#4} }
1182 \cs_new_protected:Npn \__draw_color_spot:nw #1#2 ~ #3 \q_stop
1183 { \use:c { driver_draw_ #1 _spot:nn } {#2} {#3} }
(End definition for \draw_color:n and others. These functions are documented on page 6.)
1184 </initex | package>

```

7 l3draw-transforms implementation

```

1185 <*initex | package>
1186 <@@=draw>

\l__draw_transformmcm_active_bool An internal flag to avoid redundant calculations.
1187 \bool_new:N \l__draw_transformmcm_active_bool

(End definition for \l__draw_transformmcm_active_bool.)

\l__draw_transformmcm_aa_fp The active matrix itself.
\l__draw_transformmcm_ab_fp 1188 \fp_new:N \l__draw_transformmcm_aa_fp
\l__draw_transformmcm_ba_fp 1189 \fp_new:N \l__draw_transformmcm_ab_fp
\l__draw_transformmcm_bb_fp 1190 \fp_new:N \l__draw_transformmcm_ba_fp
\l__draw_transformmcm_xshift_dim 1191 \fp_new:N \l__draw_transformmcm_bb_fp
\l__draw_transformmcm_yshift_dim 1192 \dim_new:N \l__draw_transformmcm_xshift_dim
1193 \dim_new:N \l__draw_transformmcm_yshift_dim

(End definition for \l__draw_transformmcm_aa_fp and others.)

\draw_transform_reset: Fast resetting.
1194 \cs_new_protected:Npn \draw_transform_reset:
1195 {
1196   \fp_set:Nn \l__draw_transformmcm_aa_fp { 1 }
1197   \fp_zero:N \l__draw_transformmcm_ab_fp
1198   \fp_zero:N \l__draw_transformmcm_ba_fp
1199   \fp_set:Nn \l__draw_transformmcm_bb_fp { 1 }
1200   \dim_zero:N \l__draw_transformmcm_xshift_dim
1201   \dim_zero:N \l__draw_transformmcm_yshift_dim
1202 }
1203 \draw_transform_reset:

(End definition for \draw_transform_reset:.. This function is documented on page 6.)

\draw_transform:nnnnn Setting the transform matrix is straight-forward, with just a bit of expansion to sort out.
\__draw_transform:nnnnnnn With the mechanism active, the identity matrix is set.
1204 \cs_new_protected:Npn \draw_transform:nnnnn #1#2#3#4#5
1205 {
1206   \__draw_point_process:nn
1207   { \__draw_transform:nnnnnnn {#1} {#2} {#3} {#4} }
1208   {#5}
1209 }
1210 \cs_new_protected:Npn \__draw_transform:nnnnnnn #1#2#3#4#5#6
1211 {
1212   \fp_set:Nn \l__draw_transformmcm_aa_fp {#1}
1213   \fp_set:Nn \l__draw_transformmcm_ab_fp {#2}
1214   \fp_set:Nn \l__draw_transformmcm_ba_fp {#3}
1215   \fp_set:Nn \l__draw_transformmcm_bb_fp {#4}
1216   \dim_set:Nn \l__draw_transformmcm_xshift_dim {#5}
1217   \dim_set:Nn \l__draw_transformmcm_yshift_dim {#6}
1218   \bool_lazy_all:nTF
1219   {
1220     { \fp_compare_p:nNn \l__draw_transformmcm_aa_fp = \c_one_fp }
1221     { \fp_compare_p:nNn \l__draw_transformmcm_ab_fp = \c_zero_fp }
1222     { \fp_compare_p:nNn \l__draw_transformmcm_ba_fp = \c_zero_fp }

```



```

1223     { \fp_compare_p:nNn \l__draw_transformmcm_bb_fp = \c_one_fp }
1224   }
1225   { \bool_set_false:N \l__draw_transformmcm_active_bool }
1226   { \bool_set_true:N \l__draw_transformmcm_active_bool }
1227 }

```

(End definition for `\draw_transform:nnnnn` and `__draw_transform:nnnnnnn`. This function is documented on page 7.)

\draw_transform_concat:nnnnn Much the same story for adding to an existing matrix. The part that is more complex is the calculations required: everything gets passed back to `__draw_transform_set:nnnnnnn`, with pre-expansion just in case there are *e.g.* random values. The final step is `x`-type expanded as otherwise later values affect earlier ones.

```

1228 \cs_new_protected:Npn \draw_transform_concat:nnnnn #1#2#3#4#5
1229 {
1230   \__draw_point_process:nn
1231   { \__draw_transform_concat:nnnnnn {#1} {#2} {#3} {#4} }
1232   {#5}
1233 }
1234 \cs_new_protected:Npn \__draw_transform_concat:nnnnnn #1#2#3#4#5#6
1235 {
1236   \use:x
1237   {
1238     \__draw_transform_concat_aux:nnnnnn
1239     { \fp_eval:n {#1} }
1240     { \fp_eval:n {#2} }
1241     { \fp_eval:n {#3} }
1242     { \fp_eval:n {#4} }
1243     {#5}
1244     {#6}
1245   }
1246 }
1247 \cs_new_protected:Npn \__draw_transform_concat_aux:nnnnnn #1#2#3#4#5#6
1248 {
1249   \use:x
1250   {
1251     \__draw_transform:nnnnnnn
1252     { #1 * \l__draw_transformmcm_aa_fp + #2 * \l__draw_transformmcm_ba_fp }
1253     { #1 * \l__draw_transformmcm_ab_fp + #2 * \l__draw_transformmcm_bb_fp }
1254     { #3 * \l__draw_transformmcm_aa_fp + #4 * \l__draw_transformmcm_ba_fp }
1255     { #3 * \l__draw_transformmcm_ab_fp + #4 * \l__draw_transformmcm_bb_fp }
1256     {
1257       \fp_to_dim:n
1258       {
1259         \l__draw_transformmcm_xshift_dim
1260         + \l__draw_transformmcm_aa_fp * #5
1261         + \l__draw_transformmcm_ba_fp * #6
1262       }
1263     }
1264     {
1265       \fp_to_dim:n
1266       {
1267         \l__draw_transformmcm_yshift_dim
1268         + \l__draw_transformmcm_ab_fp * #5

```

```

1269         + \l__draw_transformmcm_bb_fp * #6
1270     }
1271 }
1272 }
1273 }

```

(End definition for \draw_transform_concat:nnnnn, __draw_transform_concat:nnnnnn, and __draw_transform_concat_aux:nnnnnn. This function is documented on page 6.)

```

\draw_transform_invert: Standard mathematics: calculate the inverse matrix and use that, then undo the shifts.
\__draw_transform_invert:n 1274 \cs_new_protected:Npn \draw_transform_invert:
\__draw_transform_invert:f 1275 {
1276   \bool_if:NT \l__draw_transformmcm_active_bool
1277   {
1278     \__draw_transform_invert:f
1279     {
1280       \fp_eval:n
1281       {
1282         1 /
1283         (
1284           \l__draw_transformmcm_aa_fp * \l__draw_transformmcm_bb_fp
1285           - \l__draw_transformmcm_ab_fp * \l__draw_transformmcm_ba_fp
1286         )
1287       }
1288     }
1289   }
1290   \dim_set:Nn \l__draw_transformmcm_xshift_dim
1291   {
1292     \fp_to_dim:n
1293     {
1294       -\l__draw_transformmcm_xshift_dim * \l__draw_transformmcm_aa_fp
1295       -\l__draw_transformmcm_yshift_dim * \l__draw_transformmcm_ba_fp
1296     }
1297   }
1298   \dim_set:Nn \l__draw_transformmcm_yshift_dim
1299   {
1300     \fp_to_dim:n
1301     {
1302       -\l__draw_transformmcm_xshift_dim * \l__draw_transformmcm_ab_fp
1303       -\l__draw_transformmcm_yshift_dim * \l__draw_transformmcm_bb_fp
1304     }
1305   }
1306 }
1307 \cs_new_protected:Npn \__draw_transform_invert:n #1
1308 {
1309   \fp_set:Nn \l__draw_transformmcm_aa_fp
1310   { \l__draw_transformmcm_bb_fp * #1 }
1311   \fp_set:Nn \l__draw_transformmcm_ab_fp
1312   { -\l__draw_transformmcm_ab_fp * #1 }
1313   \fp_set:Nn \l__draw_transformmcm_ba_fp
1314   { -\l__draw_transformmcm_ba_fp * #1 }
1315   \fp_set:Nn \l__draw_transformmcm_bb_fp
1316   { \l__draw_transformmcm_aa_fp * #1 }
1317 }
1318 \cs_generate_variant:Nn \__draw_transform_invert:n { f }

```

(End definition for `\draw_transform_invert:` and `_draw_transform_invert:n`. This function is documented on page 7.)

`\draw_transform_triangle:nnn` Simple maths to move the canvas origin to #1 and the two axes to #2 and #3.

```

1319 \cs_new_protected:Npn \draw_transform_triangle:nnn #1#2#3
1320 {
1321   \_draw_point_process:nnn
1322   {
1323     \_draw_point_process:nn
1324     { \_draw_tranform_triangle:nnnnnn }
1325     {#1}
1326   }
1327   {#2} {#3}
1328 }
1329 \cs_new_protected:Npn \_draw_tranform_triangle:nnnnnn #1#2#3#4#5#6
1330 {
1331   \use:x
1332   {
1333     \_draw_transform:nnnnnnnn
1334     { #3 - #1 }
1335     { #4 - #2 }
1336     { #5 - #1 }
1337     { #6 - #2 }
1338     {#1}
1339     {#2}
1340   }
1341 }

```

(End definition for `\draw_transform_triangle:nnn`. This function is documented on page 7.)

1342 `\</initex | package>`

Index

The italic numbers denote the pages where the corresponding entry is described, numbers underlined point to the definition, all others indicate the places where it is used.

- B**
- `\begin` 766, 770, 794, 797
- bool commands:
- `\bool_gset_true:N` 1058
 - `\bool_if:NTF` 30, 73, 394, 485, 488, 497, 498, 889, 922, 1086, 1276
 - `\bool_lazy_all:nTF` 1218
 - `\bool_lazy_and:nnTF` 65, 481
 - `\bool_lazy_or:nnTF` 458, 490
 - `\bool_new:N` 56, 446, 447, 448, 449, 450, 946, 1057, 1187
 - `\bool_set_false:N` 68, 469, 470, 471, 1225
 - `\bool_set_true:N` 69, 475, 505, 956, 1226
- box commands:
- `\box_move_down:nn` 967
 - `\box_new:N` 947
 - `\box_set_dp:Nn` 972
 - `\box_set_ht:Nn` 970
 - `\box_set_wd:Nn` 973
 - `\box_use_drop:N` 968, 976
- C**
- clist commands:
- `\clist_map_inline:nn` 472
- color commands:
- `\color_parse:nN` 1159
- cs commands:
- `\cs_generate_variant:Nn` . 250, 540, 552, 559, 569, 610, 629, 636, 644, 651, 660, 666, 678, 681, 701, 704, 726, 734, 740, 765, 779, 793, 814, 849, 868, 881, 1009, 1105, 1173, 1318
 - `\cs_if_exist:NTF` 474
 - `\cs_if_exist_use:NTF` .. 233, 242, 477
 - `\cs_new:Npn` 347, 357, 367, 377, 532, 538, 541, 543, 550, 553, 555, 557, 560, 561, 563, 565, 567, 570, 572, 574, 576, 578, 583, 592, 602, 611, 617, 622, 630, 637, 645, 652, 661, 667, 673, 679, 682, 688, 695, 702, 705, 715, 720, 727, 735, 741, 751, 757, 771, 780, 786, 798, 800, 809, 839, 841, 850, 855, 869, 871, 873, 882, 887, 915, 920
 - `\cs_new_protected:Npn` 20, 42, 49, 57, 61, 71, 80, 86, 92, 98, 105, 119, 127, 132, 139, 175, 177, 188, 194, 224, 251, 283, 289, 295, 300, 308, 317, 322, 333, 388, 390, 401, 408, 417, 423, 425, 431, 451, 456, 467, 503, 507, 514, 821, 823, 825, 827, 831, 948, 960, 986, 1010, 1021, 1028, 1033, 1038, 1044, 1050, 1059, 1068, 1077, 1082, 1092, 1100, 1106, 1108, 1110, 1112, 1113, 1114, 1116, 1118, 1119, 1121, 1123, 1131, 1136, 1138, 1140, 1141, 1142, 1143, 1144, 1145, 1146, 1147, 1151, 1153, 1155, 1157, 1171, 1174, 1176, 1178, 1180, 1182, 1194, 1204, 1210, 1228, 1234, 1247, 1274, 1307, 1319, 1329
- D**
- dim commands:
- `\dim_abs:n` 434, 440
 - `\dim_compare:nNnTF` 434, 440, 516
 - `\dim_compare_p:nNn` 66, 67
 - `\dim_gset:Nn` 22, 24, 26, 28, 32, 34, 36, 38, 44, 45, 46, 47, 51, 52, 518, 952, 953, 954, 955, 1088, 1089, 1133, 1137
 - `\dim_max:nn` 23, 27, 33, 37
 - `\dim_min:nn` 25, 29, 35, 39
 - `\dim_new:N` 14, 15, 16, 17, 18, 19, 54, 55, 815, 816, 817, 818, 819, 820, 942, 943, 944, 945, 1055, 1056, 1127, 1128, 1129, 1192, 1193
 - `\dim_set:Nn` 63, 64, 833, 834, 1130, 1216, 1217, 1290, 1298
 - `\dim_step_inline:nnnn` 433, 439
 - `\dim_use:N` .. 516, 521, 523, 1064, 1065
 - `\dim_zero:N` 1200, 1201
 - `\c_max_dim` 44, 45, 46, 47, 516, 952, 953, 954, 955
- draw commands:
- `\draw_begin:` 1, 1, 948
 - `\draw_cap_but:` 2, 2, 1140
 - `\draw_cap_rectangle:` 2, 1140
 - `\draw_cap_round:` 2, 1140
 - `\draw_color:n` 6, 1151
 - `\draw_color_fill:n` 1151
 - `\draw_color_stroke:n` 1151
 - `\l_draw_default_linewidth_dim` .. 958, 1129
 - `\draw_end:` 1, 948
 - `\draw_evenodd_rule:` 2, 1140

<code>\draw_fill:n</code>	6
<code>\draw_inner_linewidth:n</code>	2, 1131
<code>\draw_join_bevel:</code>	2, 1140
<code>\draw_join_miter:</code>	2, 1140
<code>\draw_join_round:</code>	2, 1140
<code>\draw_linewidth:n</code>	2, 958, 1131
<code>\g_draw_linewidth_default_dim</code>	1
<code>\draw_miterlimit:n</code>	2, 1138
<code>\draw_nonzero_rule:</code>	2, 1140
<code>\draw_path_arc:nnn</code>	5, 175, 320
<code>\draw_path_arc:nnnn</code>	5, 175
<code>\draw_path_arc_axes:nnn</code>	5
<code>\draw_path_arc_axes:nnnn</code>	5, 317
<code>\draw_path_circle:nn</code>	5, 388
<code>\draw_path_close:</code>	6, 127, 414
<code>\draw_path_corner_arc:n</code>	5, 57
<code>\draw_path_curveto:nn</code>	5, 132
<code>\draw_path_curveto:nnn</code>	5, 80
<code>\draw_path_ellipse:nnn</code> ...	5, 322, 389
<code>\draw_path_ellipse:nnnn</code>	5
<code>\draw_path_grid:nnnn</code>	6, 425
<code>\draw_path_lineto:n</code>	5, 80, 411, 412, 413, 437, 443
<code>\draw_path_moveto:n</code>	5, 80, 410, 415, 436, 442
<code>\draw_path_rectangle:nn</code> ..	6, 390, 424
<code>\draw_path_rectangle_corners:nn</code> .	6, 417
<code>\draw_path_use:n</code>	6, 451
<code>\draw_path_use_clear:n</code>	6, 451
<code>\draw_point:nn</code>	3, 561
<code>\draw_point_add:nn</code>	3, 570
<code>\draw_point_diff:nn</code>	3, 570
<code>\draw_point_interpolate_arcaxes:nnnnnn</code>	705
<code>\draw_point_interpolate_curve:nnnnnn</code>	741
<code>\draw_point_interpolate_curve:nnnnnnnn</code>	4, 741
<code>\draw_point_interpolate_curve_-</code> <code>auxi:nnnnnnnnnn</code>	741
<code>\draw_point_interpolate_curve_-</code> <code>auxii:nnnnnnnnnn</code>	741
<code>\draw_point_interpolate_curve_-</code> <code>auxiii:nnnnnnnn</code>	741
<code>\draw_point_interpolate_curve_-</code> <code>auxiv:nnnnnnnn</code>	741
<code>\draw_point_interpolate_curve_-</code> <code>auxv:nnw</code>	741
<code>\draw_point_interpolate_curve_-</code> <code>auxvi:n</code>	741
<code>\draw_point_interpolate_curve_-</code> <code>auxvii:nnnnnnnnnn</code>	741
<code>\draw_point_interpolate_curve_-</code> <code>auxviii:nnnnnnnn</code>	741
<code>\draw_point_interpolate_distance:nnn</code>	4, 682
<code>\draw_point_interpolate_line:nnn</code>	4, 667
<code>\draw_point_intersect_circles:nnnn</code>	4
<code>\draw_point_intersect_circles:nnnnnn</code>	4, 611
<code>\draw_point_intersect_lines:nnnn</code>	4, 583
<code>\draw_point_polar:nn</code>	3, 563
<code>\draw_point_polar:nnn</code>	3, 258, 266, 271, 277, 563
<code>\draw_point_scale:nn</code>	3, 570
<code>\draw_point_transform:n</code>	2, 3, 84, 96, 114, 116, 117, 136, 137, 265, 270, 328, 398, 882
<code>\draw_point_unit_vector:n</code> ...	3, 576
<code>\draw_point_vec:nn</code>	3, 839
<code>\draw_point_vec:nnn</code>	3, 839
<code>\draw_point_vec_polar:nn</code>	4, 869
<code>\draw_point_vec_polar:nnn</code> ...	4, 869
<code>\draw_stroke:n</code>	6
<code>\draw_transform:nnnnnn</code>	7, 1204
<code>\draw_transform_concat:nnnnnn</code>	6, 1228
<code>\draw_transform_invert:</code>	7, 1274
<code>\draw_transform_reset:</code> ..	6, 957, 1194
<code>\draw_transform_triangle:nnn</code> ...	7, 319, 1319
<code>\draw_xvec:n</code>	821, 836
<code>\draw_xvec_set:n</code>	3
<code>\draw_yvec:n</code>	821, 837
<code>\draw_yvec_set:n</code>	3
<code>\draw_zvec:n</code>	821, 838
<code>\draw_zvec_set:n</code>	3
draw internal commands:	
<code>_draw_color:nn</code>	1151
<code>_draw_color:nw</code>	1151
<code>_draw_color_aux:nn</code>	1151
<code>_draw_color_cmyk:nw</code>	1176
<code>_draw_color_gray:nw</code>	1178
<code>_draw_color_rgb:nw</code>	1180
<code>_draw_color_spot:nw</code>	1182
<code>\l_draw_color_tmp_tl</code>	1148, 1159, 1162, 1163, 1164, 1167, 1168
<code>\l_draw_corner_arc_bool</code>	56, 68, 69, 73, 394
<code>\l_draw_corner_xarc_dim</code>	54, 63, 66, 76
<code>\l_draw_corner_yarc_dim</code>	54, 64, 67, 77
<code>_draw_draw_polar:nnn</code>	563
<code>_draw_draw_vec_polar:nnn</code>	872, 873, 881

<code>\g__draw_fill_color_tl ...</code>	1149 , 1162	<code>__draw_path_reset_limits: ..</code>	20 , 463
<code>\g__draw_inner_linewidth_dim ...</code>		<code>\l__draw_path_tmp_tl</code>	
<code>.....</code>	1127 , 1137	<code>.....</code>	11 , 253 , 279 , 298 , 302 , 306 , 310
<code>\g__draw_linewidth_dim</code>		<code>\l__draw_path_tmpa_fp</code>	11 , 141 , 151 , 163
<code>.....</code>	524 , 1127 , 1133 , 1134	<code>\l__draw_path_tmpb_fp</code>	11 , 142 , 158 , 167
<code>\l__draw_main_box</code>		<code>__draw_path_update_last:nn</code>	49 , 90 , 103 , 125 , 406
<code>.....</code>	947 , 950 , 964 , 968 , 970 , 972 , 973 , 976	<code>__draw_path_update_limits:nn</code>	20 , 88 , 101 , 121 , 122 , 123 , 403 , 404
<code>__draw_path_arc:nnnn</code>	175	<code>__draw_path_use:n</code>	451
<code>__draw_path_arc:nnNnn</code>	175	<code>__draw_path_use_action_draw: ..</code>	451
<code>\c__draw_path_arc_60_fp</code>	175	<code>\l__draw_path_use_bb_bool</code>	449
<code>\c__draw_path_arc_90_fp</code>	175	<code>\l__draw_path_use_clear_bool</code>	449 , 485
<code>__draw_path_arc_add:nnnn</code>	175	<code>\l__draw_path_use_clip_bool</code>	446 , 469 , 488
<code>__draw_path_arc_aux_add:nn</code>	285 , 291 , 303 , 308	<code>\l__draw_path_use_fill_bool</code>	446 , 470 , 491 , 497
<code>__draw_path_arc_auxi:nnnnNnn</code>	175 , 202 , 209	<code>__draw_path_use_stroke_bb: ..</code>	451
<code>__draw_path_arc_auxii:nnnNnnnn</code>	175	<code>__draw_path_use_stroke_bb-</code>	
<code>__draw_path_arc_auxiii:nn</code>	175	<code>aux:NnN</code>	451
<code>__draw_path_arc_auxiv:nnnn</code>	175	<code>\l__draw_path_use_stroke_bool</code>	446 , 471 , 483 , 492 , 498 , 505
<code>__draw_path_arc_auxv:nn</code>	175	<code>\g__draw_path_xmax_dim</code>	16 , 22 , 23 , 44
<code>__draw_path_arc_auxvi:nn</code>	175	<code>\g__draw_path_xmin_dim</code>	16 , 24 , 25 , 45
<code>\l__draw_path_arc_delta_fp</code>	175	<code>\g__draw_path_ymax_dim</code>	16 , 26 , 27 , 46
<code>\l__draw_path_arc_start_fp</code>	175	<code>\g__draw_path_ymin_dim</code>	16 , 28 , 29 , 47
<code>__draw_path_corner_arc:nn</code>	57	<code>__draw_point_interpolate-</code>	
<code>__draw_path_curveto:nnnn</code>	132	<code>arcaxes_auxi:nnnnnnnnnn</code>	705
<code>__draw_path_curveto:nnnnnn</code>	80 , 146 , 279 , 349 , 359 , 369 , 379	<code>__draw_point_interpolate-</code>	
<code>\c__draw_path_curveto_a_fp</code>	132	<code>arcaxes_auxii:nnnnnnnnnn</code>	705
<code>\c__draw_path_curveto_b_fp</code>	132	<code>__draw_point_interpolate-</code>	
<code>__draw_path_ellipse:nnnnnn</code>	322	<code>arcaxes_auxiii:nnnnnnnn</code>	705
<code>__draw_path_ellipse_arci:nnnnnn</code>	322	<code>__draw_point_interpolate-</code>	
<code>__draw_path_ellipse_arcii:nnnnnn</code>	322	<code>arcaxes_auxiv:nnnnnnnn</code>	705
<code>__draw_path_ellipse_arciiii:nnnnnn</code>	322	<code>__draw_point_interpolate_curve-</code>	
<code>__draw_path_ellipse_arciv:nnnnnn</code>	322	<code>auxi:nnnnnnnnnn</code>	746 , 751
<code>\c__draw_path_ellipse_fp</code>	322	<code>__draw_point_interpolate_curve-</code>	
<code>__draw_path_grid:nnnnnn</code>	425	<code>auxii:nnnnnnnnnn</code>	753 , 757 , 765
<code>\g__draw_path_lastx_dim</code>	14 , 51 , 150 , 286 , 292	<code>__draw_point_interpolate_curve-</code>	
<code>\g__draw_path_lasty_dim</code>	14 , 52 , 157 , 287 , 293	<code>auxiii:nnnnnn</code>	760 , 771 , 779
<code>__draw_path_lineto:nn</code>	80	<code>__draw_point_interpolate_curve-</code>	
<code>__draw_path_mark_corner:</code>	71 , 100 , 111 , 129 , 145 , 216	<code>auxiv:nnnnnn</code>	773 , 774 , 775 , 780
<code>__draw_path_moveto:nn</code>	80 , 337 , 345	<code>__draw_point_interpolate_curve-</code>	
<code>__draw_path_rectangle:nnnn</code>	390	<code>auxv:nnw</code>	782 , 786 , 793
<code>__draw_path_rectangle_corners:nnnn</code>	417	<code>__draw_point_interpolate_curve-</code>	
<code>__draw_path_rectangle_corners:nnnnn</code>	420 , 423	<code>auxvi:n</code>	777 , 798
<code>__draw_path_rectangle_rounded:nnnn</code>	390	<code>__draw_point_interpolate_curve-</code>	
		<code>auxvii:nnnnnnnn</code>	799 , 800
		<code>__draw_point_interpolate_curve-</code>	
		<code>auxviii:nnnnnn</code>	802 , 809 , 814
		<code>__draw_point_interpolate-</code>	
		<code>distance:nnnnn</code>	682
		<code>__draw_point_interpolate-</code>	
		<code>distance_aux:nnnnn</code>	697 , 702 , 704

_draw_softpath_roundpoint:nn	75, 1059	driver commands:	
_draw_softpath_roundpoint_-op:nn	1103, 1106	\driver_draw_begin:	951
_draw_softpath_set_eq:N	1028	\driver_draw_cap_but:	1140
_draw_softpath_use:	487, 1038	\driver_draw_cap_rectangle:	1141
_draw_softpath_use_clear:	486, 1038	\driver_draw_cap_round:	1142
_draw_split_select:nw	1151	\driver_draw_clip:	489
\g__draw_stroke_color_tl	1149, 1163	\driver_draw_closepath:	1107
_draw_tranform_triangle:nnnnnn	1324, 1329	\driver_draw_curveto:nnnnnn	1111
_draw_transform:nnnnnn	1204, 1251, 1333	\driver_draw_end:	962
_draw_transform_concat:nnnnnn	1228	\driver_draw_evenodd_rule:	1143
_draw_transform_concat_-aux:nnnnnn	1228	\driver_draw_join_bevel:	1145
_draw_transform_invert:n	1274	\driver_draw_join_miter:	1146
_draw_transform_set:nnnnnn	41	\driver_draw_join_round:	1147
\l__draw_transformcm_aa_fp	894, 927, 1188, 1196, 1212, 1220, 1252, 1254, 1260, 1284, 1294, 1309, 1316	\driver_draw_lineto:nn	1115
\l__draw_transformcm_ab_fp	900, 932, 1188, 1197, 1213, 1221, 1253, 1255, 1268, 1285, 1302, 1311, 1312	\driver_draw_linewidth:n	1134
\l__draw_transformcm_active_bool	889, 922, 1187, 1225, 1226, 1276	\driver_draw_miterlimit:n	1139
\l__draw_transformcm_ba_fp	895, 928, 1188, 1198, 1214, 1222, 1252, 1254, 1261, 1285, 1295, 1313, 1314	\driver_draw_moveto:nn	1117
\l__draw_transformcm_bb_fp	901, 933, 1191, 1199, 1215, 1223, 1253, 1255, 1269, 1284, 1303, 1310, 1315	\driver_draw_nonzero_rule:	1144
\l__draw_transformcm_xshift_dim	896, 910, 1188, 1200, 1216, 1259, 1290, 1294, 1302	\driver_draw_rectangle:nnnn	1122
\l__draw_transformcm_yshift_dim	902, 911, 1188, 1201, 1217, 1267, 1295, 1298, 1303	\driver_draw_transformcm:nnnnnn	2, 6
\l__draw_update_bb_bool	30, 482, 946, 956		
_draw_vec:nn	821	E	
_draw_vec:nnn	821	\ERROR	478
\g__draw_xmax_dim	32, 33, 942, 952, 974	exp commands:	
\g__draw_xmin_dim	34, 35, 942, 953, 966, 974	\exp_after:wN	279, 297
\l__draw_xvec_x_dim	815, 845, 859, 877	\exp_not:n	999, 1000, 1001, 1014, 1015, 1016, 1017
\l__draw_xvec_y_dim	815, 846, 863		
\g__draw_ymax_dim	36, 37, 942, 954, 971	F	
\g__draw_ymin_dim	38, 39, 942, 955, 967, 971	fp commands:	
\l__draw_yvec_x_dim	815, 845, 860	\fp_compare:nNnTF	190, 200
\l__draw_yvec_y_dim	815, 846, 864, 878	\fp_compare_p:nNn	1220, 1221, 1222, 1223
\l__draw_zvec_x_dim	815, 861	\fp_const:Nn	173, 174, 315, 316, 387
\l__draw_zvec_y_dim	815, 865	\fp_eval:n	182, 183, 204, 211, 220, 556, 566, 595, 596, 597, 598, 599, 600, 620, 625, 626, 633, 640, 641, 648, 655, 657, 670, 675, 691, 692, 698, 718, 723, 730, 731, 754, 761, 783, 784, 803, 804, 805, 806, 840, 853, 872, 1239, 1240, 1241, 1242, 1280
		\fp_new:N	12, 13, 313, 314, 1188, 1189, 1190, 1191
		\fp_set:Nn	141, 142, 196, 197, 280, 281, 1196, 1199, 1212, 1213, 1214, 1215, 1309, 1311, 1313, 1315
		\fp_to_decimal:N	203, 210, 218
		\fp_to_dim:n	148, 155, 162, 166, 184, 185, 231, 240, 311, 338, 350, 351, 352, 353, 354, 355, 360, 361, 362, 363, 364, 365, 370, 371, 372, 373, 374, 375,

380, 381, 382, 383, 384, 385, 1133, 1137, 1139, 1257, 1265, 1292, 1300	\ProvidesExplPackage 4
\fp_use:N 387	Q
\fp_while_do:nNnn 198	quark commands:
\fp_zero:N 1197, 1198	\q_mark 551, 553
\c_one_fp 1220, 1223	\q_stop 539, 541, 551, 553, 1172, 1174, 1175, 1176, 1178, 1180, 1182
\c_zero_fp 1221, 1222	
H	R
hbox commands:	\RequirePackage 7
\hbox_set:Nn 964	
I	S
int commands:	skip commands:
\int_compare:nNnTF 988, 994	\skip_horizontal:n 966
\int_gincr:N 990, 996	str commands:
\int_gzero:N 1003, 1023, 1025	\str_if_eq_p:nn 460
\int_if_odd:nTF 656	
\int_new:N 984, 985	T
M	tl commands:
mode commands:	\tl_clear:N 253
\mode_leave_vertical: 975	\tl_gclear:N 1004, 1024, 1026, 1046, 1047, 1048
P	\tl_gput_right:Nn 991
\pgfextractx 20	\tl_gset:Nn 997, 1012
\pgfextracty 20	\tl_gset_eq:NN 1035, 1162, 1163, 1167
\pgfgetlastxy 20	\tl_if_blank:nTF 453, 1160
\pgfpatharcto 7	\tl_if_blank_p:n 459
\pgfpatharctoprecomputed 7	\tl_new:N 11, 981, 982, 983, 1148, 1149, 1150
\pgfpathcosine 7	\tl_put_right:Nn 306, 310
\pgfpathcurvebetweentime 7	\tl_set:Nn 302
\pgfpathcurvebetweentimecontinue 7	\tl_set_eq:NN 1031
\pgfpathparabola 7	
\pgfpathsine 7	U
\pgfpointborderellipse 20	use commands:
\pgfpointborderrectangle 20	\use:N 494, 520, 1175, 1177, 1179, 1181, 1183
\pgfpointcylindrical 20	\use:n 143, 179, 226, 335, 1236, 1249, 1331
\pgfpointorigin 20	\use_i:nn 20
\pgfpointspherical 20	\use_ii:nn 20
\pgfqpoint 20	
\pgfqpointpolar 20	V
\pgfqpointscale 20	vbox commands:
\pgfqpointxy 20	\vbox_set:Nw 950
\pgfqpointxyz 20	\vbox_set_end: 963
prg commands:	
\prg_do_nothing: 776, 787, 790	