File I
Implementation

1 l3draw implementation

1 ⟨*package⟩

2 ⟨@@=draw⟩

3 \ProvidesExplPackage{l3draw}{2020-01-29}{}
4   {L3 Experimental core drawing support}

5 \RequirePackage { l3color }

1.1 Internal auxiliaries

\s__draw_mark   Internal scan marks.
\s__draw_stop
6 \scan_new:N \s__draw_mark
7 \scan_new:N \s__draw_stop

(*End definition for* \s__draw_mark *and* \s__draw_stop.)

\q__draw_recursion_tail   Internal recursion quarks.
\q__draw_recursion_stop
8 \quark_new:N \q__draw_recursion_tail
9 \quark_new:N \q__draw_recursion_stop

(*End definition for* \q__draw_recursion_tail *and* \q__draw_recursion_stop.)

__draw_if_recursion_tail_stop_do:Nn   Functions to query recursion quarks.

10 __kernel_quark_new_test:N __draw_if_recursion_tail_stop_do:Nn

(*End definition for* __draw_if_recursion_tail_stop_do:Nn.)

Everything else is in the sub-files!

11 ⟨/package⟩

2 l3draw-boxes implementation

12 ⟨*package⟩

13 ⟨@@=draw⟩

Inserting boxes requires us to "interrupt" the drawing state, so is closely linked to scoping. At the same time, there are a few additional features required to make text work in a flexible way.

\l__draw_tmp_box

14 \box_new:N \l__draw_tmp_box

(*End definition for* \l__draw_tmp_box.)

\draw_box_use:N
__draw_box_use:Nnnnn

Before inserting a box, we need to make sure that the bounding box is being updated correctly. As drawings track transformations as a whole, rather than as separate operations, we do the insertion using an almost-raw matrix. The process is split into two so that coffins are also supported.

```
15 \cs_new_protected:Npn \draw_box_use:N #1
16   {
17     \__draw_box_use:Nnnnn #1
18       { 0pt } { -\box_dp:N #1 } { \box_wd:N #1 } { \box_ht:N #1 }
19   }
20 \cs_new_protected:Npn \__draw_box_use:Nnnnn #1#2#3#4#5
21   {
22     \bool_if:NT \l_draw_bb_update_bool
23       {
24         \__draw_point_process:nn
25           { \__draw_path_update_limits:nn }
26           { \draw_point_transform:n { #2 , #3 } }
27         \__draw_point_process:nn
28           { \__draw_path_update_limits:nn }
29           { \draw_point_transform:n { #4 , #3 } }
30         \__draw_point_process:nn
31           { \__draw_path_update_limits:nn }
32           { \draw_point_transform:n { #4 , #5 } }
33         \__draw_point_process:nn
34           { \__draw_path_update_limits:nn }
35           { \draw_point_transform:n { #2 , #5 } }
36       }
37     \group_begin:
38       \hbox_set:Nn \l__draw_tmp_box
39         {
40           \use:x
41             {
42               \__draw_backend_box_use:Nnnnn #1
43                 { \fp_use:N \l__draw_matrix_a_fp }
44                 { \fp_use:N \l__draw_matrix_b_fp }
45                 { \fp_use:N \l__draw_matrix_c_fp }
46                 { \fp_use:N \l__draw_matrix_d_fp }
47             }
48         }
49       \hbox_set:Nn \l__draw_tmp_box
50         {
51           \tex_kern:D \l__draw_xshift_dim
52           \box_move_up:nn { \l__draw_yshift_dim }
53             { \box_use_drop:N \l__draw_tmp_box }
54         }
55       \box_set_ht:Nn \l__draw_tmp_box { 0pt }
56       \box_set_dp:Nn \l__draw_tmp_box { 0pt }
57       \box_set_wd:Nn \l__draw_tmp_box { 0pt }
58       \box_use_drop:N \l__draw_tmp_box
59     \group_end:
60   }
```

(*End definition for* \draw_box_use:N *and* __draw_box_use:Nnnnn. *This function is documented on page* **??**.)

$\texttt{\textbackslash draw\_coffin\_use:Nnn}$ Slightly more than a shortcut: we have to allow for the fact that coffins have no apparent width before the reference point.

```
61 \cs_new_protected:Npn \draw_coffin_use:Nnn #1#2#3
62   {
63     \group_begin:
64       \hbox_set:Nn \l__draw_tmp_box
65         { \coffin_typeset:Nnnnn #1 {#2} {#3} { 0pt } { 0pt } }
66       \__draw_box_use:Nnnnn \l__draw_tmp_box
67         { \box_wd:N \l__draw_tmp_box - \coffin_wd:N #1 }
68         { -\box_dp:N \l__draw_tmp_box }
69         { \box_wd:N \l__draw_tmp_box }
70         { \box_ht:N \l__draw_tmp_box }
71     \group_end:
72   }
```

(*End definition for* $\texttt{\textbackslash draw\_coffin\_use:Nnn}$*. This function is documented on page* **??**.)

```
73 ⟨/package⟩
```

3   l3draw-layers implementation

```
74 ⟨*package⟩
```

```
75 ⟨@@=draw⟩
```

3.1   User interface

$\texttt{\textbackslash draw\_layer\_new:n}$

```
76 \cs_new_protected:Npn \draw_layer_new:n #1
77   {
78     \str_if_eq:nnTF {#1} { main }
79       { \msg_error:nnn { draw } { main-reserved } }
80       {
81         \box_new:c { g__draw_layer_ #1 _box }
82         \box_new:c { l__draw_layer_ #1 _box }
83       }
84   }
```

(*End definition for* $\texttt{\textbackslash draw\_layer\_new:n}$*. This function is documented on page* **??**.)

$\texttt{\textbackslash l\_\_draw\_layer\_tl}$ The name of the current layer: we start off with $\texttt{main}$.

```
85 \tl_new:N \l__draw_layer_tl
86 \tl_set:Nn \l__draw_layer_tl { main }
```

(*End definition for* $\texttt{\textbackslash l\_\_draw\_layer\_tl}$*.*)

$\texttt{\textbackslash l\_\_draw\_layer\_close\_bool}$ Used to track if a layer needs to be closed.

```
87 \bool_new:N \l__draw_layer_close_bool
```

(*End definition for* $\texttt{\textbackslash l\_\_draw\_layer\_close\_bool}$*.*)

$\texttt{\textbackslash l\_draw\_layers\_clist}$
$\texttt{\textbackslash g\_\_draw\_layers\_clist}$ The list of layers to use starts off with just the $\texttt{main}$ one.

```
88 \clist_new:N \l_draw_layers_clist
89 \clist_set:Nn \l_draw_layers_clist { main }
90 \clist_new:N \g__draw_layers_clist
```

(*End definition for* \l_draw_layers_clist *and* \g__draw_layers_clist. *This variable is documented on page* **??**.)

\draw_layer_begin:n Layers may be called multiple times and have to work when nested. That drives a bit of
\draw_layer_end: grouping to get everything in order. Layers have to be zero width, so they get set as we
go along.

```
91 \cs_new_protected:Npn \draw_layer_begin:n #1
92   {
93     \group_begin:
94       \box_if_exist:cTF { g__draw_layer_ #1 _box }
95         {
96           \str_if_eq:VnTF \l__draw_layer_tl {#1}
97             { \bool_set_false:N \l__draw_layer_close_bool }
98             {
99               \bool_set_true:N \l__draw_layer_close_bool
100              \tl_set:Nn \l__draw_layer_tl {#1}
101              \box_gset_wd:cn { g__draw_layer_ #1 _box } { 0pt }
102              \hbox_gset:cw { g__draw_layer_ #1 _box }
103                \box_use_drop:c { g__draw_layer_ #1 _box }
104                \group_begin:
105            }
106            \draw_linewidth:n { \l_draw_default_linewidth_dim }
107        }
108        {
109          \str_if_eq:nnTF {#1} { main }
110            { \msg_error:nnn { draw } { unknown-layer } {#1} }
111            { \msg_error:nnn { draw } { main-layer } }
112        }
113    }
114 \cs_new_protected:Npn \draw_layer_end:
115   {
116     \bool_if:NT \l__draw_layer_close_bool
117       {
118         \group_end:
119         \hbox_gset_end:
120       }
121     \group_end:
122   }
```

(*End definition for* \draw_layer_begin:n *and* \draw_layer_end:. *These functions are documented on page* **??**.)

3.2 Internal cross-links

__draw_layers_insert: The main layer is special, otherwise just dump the layer box inside a scope.

```
123 \cs_new_protected:Npn \__draw_layers_insert:
124   {
125     \clist_map_inline:Nn \l_draw_layers_clist
126       {
127         \str_if_eq:nnTF {##1} { main }
128           {
129             \box_set_wd:Nn \l__draw_layer_main_box { 0pt }
130             \box_use_drop:N \l__draw_layer_main_box
131           }
```

4

```
132              {
133                \__draw_backend_scope_begin:
134                \box_gset_wd:cn { g__draw_layer_ ##1 _box } { 0pt }
135                \box_use_drop:c { g__draw_layer_ ##1 _box }
136                \__draw_backend_scope_end:
137              }
138          }
139      }
```

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

`\__draw_layers_save:`   Simple save/restore functions.
`\__draw_layers_restore:`
```
140 \cs_new_protected:Npn \__draw_layers_save:
141   {
142     \clist_map_inline:Nn \l_draw_layers_clist
143       {
144         \str_if_eq:nnF {##1} { main }
145           {
146             \box_set_eq:cc { l__draw_layer_ ##1 _box }
147               { g__draw_layer_ ##1 _box }
148           }
149       }
150   }
151 \cs_new_protected:Npn \__draw_layers_restore:
152   {
153     \clist_map_inline:Nn \l_draw_layers_clist
154       {
155         \str_if_eq:nnF {##1} { main }
156           {
157             \box_gset_eq:cc { g__draw_layer_ ##1 _box }
158               { l__draw_layer_ ##1 _box }
159           }
160       }
161   }
```

(*End definition for* `\__draw_layers_save:` *and* `\__draw_layers_restore:.`)

```
162 \msg_new:nnnn { draw } { main-layer }
163   { Material~cannot~be~added~to~'main'~layer. }
164   { The~main~layer~may~only~be~accessed~at~the~top~level. }
165 \msg_new:nnn { draw } { main-reserved }
166   { The~'main'~layer~is~reserved. }
167 \msg_new:nnnn { draw } { unknown-layer }
168   { Layer~'#1'~has~not~been~created. }
169   { You~have~tried~to~use~layer~'#1',~but~it~was~never~set~up. }
170 % \end{macrocode}
171 %
172 %    \begin{macrocode}
173 ⟨/package⟩
```

4   l3draw-paths implementation

```
174 ⟨*package⟩
```

```
175 ⟨@@=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`
`\l__draw_path_tmpb_fp`

```
176 \tl_new:N \l__draw_path_tmp_tl
177 \fp_new:N \l__draw_path_tmpa_fp
178 \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`.)

4.1 Tracking paths

`\g__draw_path_lastx_dim` The last point visited on a path.
`\g__draw_path_lasty_dim`

```
179 \dim_new:N \g__draw_path_lastx_dim
180 \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`
`\g__draw_path_ymax_dim`
`\g__draw_path_ymin_dim`

```
181 \dim_new:N \g__draw_path_xmax_dim
182 \dim_new:N \g__draw_path_xmin_dim
183 \dim_new:N \g__draw_path_ymax_dim
184 \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.)

```
185 \cs_new_protected:Npn \__draw_path_update_limits:nn #1#2
186   {
187     \dim_gset:Nn \g__draw_path_xmax_dim
188       { \dim_max:nn \g__draw_path_xmax_dim {#1} }
189     \dim_gset:Nn \g__draw_path_xmin_dim
190       { \dim_min:nn \g__draw_path_xmin_dim {#1} }
191     \dim_gset:Nn \g__draw_path_ymax_dim
192       { \dim_max:nn \g__draw_path_ymax_dim {#2} }
193     \dim_gset:Nn \g__draw_path_ymin_dim
194       { \dim_min:nn \g__draw_path_ymin_dim {#2} }
195     \bool_if:NT \l_draw_bb_update_bool
196       {
197         \dim_gset:Nn \g__draw_xmax_dim
```

```
198              { \dim_max:nn \g__draw_xmax_dim {#1} }
199          \dim_gset:Nn \g__draw_xmin_dim
200              { \dim_min:nn \g__draw_xmin_dim {#1} }
201          \dim_gset:Nn \g__draw_ymax_dim
202              { \dim_max:nn \g__draw_ymax_dim {#2} }
203          \dim_gset:Nn \g__draw_ymin_dim
204              { \dim_min:nn \g__draw_ymin_dim {#2} }
205        }
206    }
207  \cs_new_protected:Npn \__draw_path_reset_limits:
208    {
209      \dim_gset:Nn \g__draw_path_xmax_dim { -\c_max_dim }
210      \dim_gset:Nn \g__draw_path_xmin_dim {  \c_max_dim }
211      \dim_gset:Nn \g__draw_path_ymax_dim { -\c_max_dim }
212      \dim_gset:Nn \g__draw_path_ymin_dim {  \c_max_dim }
213    }
```

(*End definition for* `\__draw_path_update_limits:nn` *and* `\__draw_path_reset_limits:`.)

`\__draw_path_update_last:nn`   A simple auxiliary to avoid repetition.

```
214  \cs_new_protected:Npn \__draw_path_update_last:nn #1#2
215    {
216      \dim_gset:Nn \g__draw_path_lastx_dim {#1}
217      \dim_gset:Nn \g__draw_path_lasty_dim {#2}
218    }
```

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

4.2   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`
```
219  \dim_new:N \l__draw_corner_xarc_dim
220  \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.
```
221  \bool_new:N \l__draw_corner_arc_bool
```

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

`\draw_path_corner_arc:nn`   Calculate the arcs, check they are non-zero.
```
222  \cs_new_protected:Npn \draw_path_corner_arc:nn #1#2
223    {
224      \dim_set:Nn \l__draw_corner_xarc_dim {#1}
225      \dim_set:Nn \l__draw_corner_yarc_dim {#2}
226      \bool_lazy_and:nnTF
227        { \dim_compare_p:nNn \l__draw_corner_xarc_dim = { 0pt } }
228        { \dim_compare_p:nNn \l__draw_corner_yarc_dim = { 0pt } }
```

7

```
229          { \bool_set_false:N \l__draw_corner_arc_bool }
230          { \bool_set_true:N \l__draw_corner_arc_bool }
231     }
```

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

`\__draw_path_mark_corner:`   Mark up corners for arc post-processing.

```
232 \cs_new_protected:Npn \__draw_path_mark_corner:
233     {
234        \bool_if:NT \l__draw_corner_arc_bool
235           {
236              \__draw_softpath_roundpoint:VV
237                 \l__draw_corner_xarc_dim
238                 \l__draw_corner_yarc_dim
239           }
240     }
```

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

4.3   Basic path constructions

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

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

```
241 \cs_new_protected:Npn \draw_path_moveto:n #1
242     {
243        \__draw_point_process:nn
244           { \__draw_path_moveto:nn }
245           { \draw_point_transform:n {#1} }
246     }
247 \cs_new_protected:Npn \__draw_path_moveto:nn #1#2
248     {
249        \__draw_path_update_limits:nn {#1} {#2}
250        \__draw_softpath_moveto:nn {#1} {#2}
251        \__draw_path_update_last:nn {#1} {#2}
252     }
253 \cs_new_protected:Npn \draw_path_lineto:n #1
254     {
255        \__draw_point_process:nn
256           { \__draw_path_lineto:nn }
257           { \draw_point_transform:n {#1} }
258     }
259 \cs_new_protected:Npn \__draw_path_lineto:nn #1#2
260     {
261        \__draw_path_mark_corner:
262        \__draw_path_update_limits:nn {#1} {#2}
263        \__draw_softpath_lineto:nn {#1} {#2}
264        \__draw_path_update_last:nn {#1} {#2}
265     }
266 \cs_new_protected:Npn \draw_path_curveto:nnn #1#2#3
267     {
268        \__draw_point_process:nnnn
269           {
270              \__draw_path_mark_corner:
271              \__draw_path_curveto:nnnnnn
```

```
272        }
273        { \draw_point_transform:n {#1} }
274        { \draw_point_transform:n {#2} }
275        { \draw_point_transform:n {#3} }
276     }
277  \cs_new_protected:Npn \__draw_path_curveto:nnnnnn #1#2#3#4#5#6
278     {
279        \__draw_path_update_limits:nn {#1} {#2}
280        \__draw_path_update_limits:nn {#3} {#4}
281        \__draw_path_update_limits:nn {#5} {#6}
282        \__draw_softpath_curveto:nnnnnn {#1} {#2} {#3} {#4} {#5} {#6}
283        \__draw_path_update_last:nn {#5} {#6}
284     }
```

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

\draw_path_close: A simple wrapper.

```
285  \cs_new_protected:Npn \draw_path_close:
286     {
287        \__draw_path_mark_corner:
288        \__draw_softpath_closepath:
289     }
```

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

4.4   Canvas path constructions

\draw_path_canvas_moveto:n   Operations with no application of the transformation matrix.
\draw_path_canvas_lineto:n
\draw_path_canvas_curveto:nnn

```
290  \cs_new_protected:Npn \draw_path_canvas_moveto:n #1
291     { \__draw_point_process:nn { \__draw_path_moveto:nn } {#1} }
292  \cs_new_protected:Npn \draw_path_canvas_lineto:n #1
293     { \__draw_point_process:nn { \__draw_path_lineto:nn } {#1} }
294  \cs_new_protected:Npn \draw_path_canvas_curveto:nnn #1#2#3
295     {
296        \__draw_point_process:nnnn
297           {
298              \__draw_path_mark_corner:
299              \__draw_path_curveto:nnnnnn
300           }
301        {#1} {#2} {#3}
302     }
```

(*End definition for* `\draw_path_canvas_moveto:n`, `\draw_path_canvas_lineto:n`, *and* `\draw_path_-
canvas_curveto:nnn`. *These functions are documented on page* **??**.)

4.5   Computed curves

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

\draw_path_curveto:nn   A quadratic curve with one control point (x_c, y_c). The two required control points are
__draw_path_curveto:nnnn   then
\c__draw_path_curveto_a_fp
\c__draw_path_curveto_b_fp
$$x_1 = \frac{1}{3}x_s + \frac{2}{3}x_c \quad y_1 = \frac{1}{3}y_s + \frac{2}{3}y_c$$

9

and

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

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

```
303 \cs_new_protected:Npn \draw_path_curveto:nn #1#2
304   {
305     \__draw_point_process:nnn
306       { \__draw_path_curveto:nnnn }
307       { \draw_point_transform:n {#1} }
308       { \draw_point_transform:n {#2} }
309   }
310 \cs_new_protected:Npn \__draw_path_curveto:nnnn #1#2#3#4
311   {
312     \fp_set:Nn \l__draw_path_tmpa_fp { \c__draw_path_curveto_b_fp * #1 }
313     \fp_set:Nn \l__draw_path_tmpb_fp { \c__draw_path_curveto_b_fp * #2 }
314     \use:x
315       {
316         \__draw_path_mark_corner:
317         \__draw_path_curveto:nnnnnn
318           {
319             \fp_to_dim:n
320               {
321                 \c__draw_path_curveto_a_fp * \g__draw_path_lastx_dim
322                 + \l__draw_path_tmpa_fp
323               }
324           }
325           {
326             \fp_to_dim:n
327               {
328                 \c__draw_path_curveto_a_fp * \g__draw_path_lasty_dim
329                 + \l__draw_path_tmpb_fp
330               }
331           }
332           {
333             \fp_to_dim:n
334               { \c__draw_path_curveto_a_fp * #3 + \l__draw_path_tmpa_fp }
335           }
336           {
337             \fp_to_dim:n
338               { \c__draw_path_curveto_a_fp * #4 + \l__draw_path_tmpb_fp }
339           }
340           {#3}
341           {#4}
342       }
343   }
344 \fp_const:Nn \c__draw_path_curveto_a_fp { 1 / 3 }
345 \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* **??**.)

`\draw_path_arc:nnn`
`\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`

Drawing an arc means dividing the total curve required into sections: using Bézier 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°.

```
346 \cs_new_protected:Npn \draw_path_arc:nnn #1#2#3
```

10

```
347    { \draw_path_arc:nnnn {#1} {#2} {#3} {#3} }
348  \cs_new_protected:Npn \draw_path_arc:nnnn #1#2#3#4
349    {
350      \use:x
351        {
352          \__draw_path_arc:nnnn
353            { \fp_eval:n {#1} }
354            { \fp_eval:n {#2} }
355            { \fp_to_dim:n {#3} }
356            { \fp_to_dim:n {#4} }
357        }
358    }
359  \cs_new_protected:Npn \__draw_path_arc:nnnn #1#2#3#4
360    {
361      \fp_compare:nNnTF {#1} > {#2}
362        { \__draw_path_arc:nnNnn {#1} {#2} - {#3} {#4} }
363        { \__draw_path_arc:nnNnn {#1} {#2} + {#3} {#4} }
364    }
365  \cs_new_protected:Npn \__draw_path_arc:nnNnn #1#2#3#4#5
366    {
367      \fp_set:Nn \l__draw_path_arc_start_fp {#1}
368      \fp_set:Nn \l__draw_path_arc_delta_fp { abs( #1 - #2 ) }
369      \fp_while_do:nNnn { \l__draw_path_arc_delta_fp } > { 90 }
370        {
371          \fp_compare:nNnTF \l__draw_path_arc_delta_fp > { 115 }
372            {
373              \__draw_path_arc_auxi:ffnnNnn
374                { \fp_to_decimal:N \l__draw_path_arc_start_fp }
375                { \fp_eval:n { \l__draw_path_arc_start_fp #3 90 } }
376                { 90 } {#2}
377                #3 {#4} {#5}
378            }
379            {
380              \__draw_path_arc_auxi:ffnnNnn
381                { \fp_to_decimal:N \l__draw_path_arc_start_fp }
382                { \fp_eval:n { \l__draw_path_arc_start_fp #3 60 } }
383                { 60 } {#2}
384                #3 {#4} {#5}
385            }
386        }
387      \__draw_path_mark_corner:
388      \__draw_path_arc_auxi:fnfnNnn
389        { \fp_to_decimal:N \l__draw_path_arc_start_fp }
390        {#2}
391        { \fp_eval:n { abs( \l__draw_path_arc_start_fp - #2 ) } }
392        {#2}
393        #3 {#4} {#5}
394    }
```

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: $\frac{4}{3}(\sqrt{2} - 1) = 0.552\,284\,75$. For other cases, we follow the calculation used by pgf but with the second common case of 60° pre-calculated for speed.

```
395  \cs_new_protected:Npn \__draw_path_arc_auxi:nnnnNnn #1#2#3#4#5#6#7
```

```
396    {
397      \use:x
398        {
399          \__draw_path_arc_auxii:nnnNnnnn
400            {#1} {#2} {#4} #5 {#6} {#7}
401            {
402              \fp_to_dim:n
403                {
404                  \cs_if_exist_use:cF
405                    { c__draw_path_arc_ #3 _fp }
406                    { 4/3 * tand( 0.25 * #3 ) }
407                    * #6
408                }
409            }
410            {
411              \fp_to_dim:n
412                {
413                  \cs_if_exist_use:cF
414                    { c__draw_path_arc_ #3 _fp }
415                    { 4/3 * tand( 0.25 * #3 ) }
416                    * #7
417                }
418            }
419        }
420    }
421  \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

```
422  \cs_new_protected:Npn \__draw_path_arc_auxii:nnnNnnnn #1#2#3#4#5#6#7#8
423    {
424      \tl_clear:N \l__draw_path_tmp_tl
425      \__draw_point_process:nn
426        { \__draw_path_arc_auxiii:nn }
427        {
428          \__draw_point_transform_noshift:n
429            { \draw_point_polar:nnn {#7} {#8} { #1 #4 90 } }
430        }
431      \__draw_point_process:nnn
432        { \__draw_path_arc_auxiv:nnnn }
433        {
434          \draw_point_transform:n
435            { \draw_point_polar:nnn {#5} {#6} {#1} }
436        }
437        {
438          \draw_point_transform:n
439            { \draw_point_polar:nnn {#5} {#6} {#2} }
440        }
441      \__draw_point_process:nn
442        { \__draw_path_arc_auxv:nn }
443        {
```

```
444        \__draw_point_transform_noshift:n
445          { \draw_point_polar:nnn {#7} {#8} { #2 #4 -90 } }
446        }
447      \exp_after:wN \__draw_path_curveto:nnnnnn \l__draw_path_tmp_tl
448      \fp_set:Nn \l__draw_path_arc_delta_fp { abs ( #2 - #3 ) }
449      \fp_set:Nn \l__draw_path_arc_start_fp {#2}
450    }
```

The first control point.

```
451  \cs_new_protected:Npn \__draw_path_arc_auxiii:nn #1#2
452    {
453      \__draw_path_arc_aux_add:nn
454        { \g__draw_path_lastx_dim + #1 }
455        { \g__draw_path_lasty_dim + #2 }
456    }
```

The end point: simple arithmetic.

```
457  \cs_new_protected:Npn \__draw_path_arc_auxiv:nnnn #1#2#3#4
458    {
459      \__draw_path_arc_aux_add:nn
460        { \g__draw_path_lastx_dim - #1 + #3 }
461        { \g__draw_path_lasty_dim - #2 + #4 }
462    }
```

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

```
463  \cs_new_protected:Npn \__draw_path_arc_auxv:nn #1#2
464    {
465      \exp_after:wN \__draw_path_arc_auxvi:nn
466        \l__draw_path_tmp_tl {#1} {#2}
467    }
468  \cs_new_protected:Npn \__draw_path_arc_auxvi:nn #1#2#3#4#5#6
469    {
470      \tl_set:Nn \l__draw_path_tmp_tl { {#1} {#2} }
471      \__draw_path_arc_aux_add:nn
472        { #5 + #3 }
473        { #6 + #4 }
474      \tl_put_right:Nn \l__draw_path_tmp_tl { {#3} {#4} }
475    }
476  \cs_new_protected:Npn \__draw_path_arc_aux_add:nn #1#2
477    {
478      \tl_put_right:Nx \l__draw_path_tmp_tl
479        { { \fp_to_dim:n {#1} } { \fp_to_dim:n {#2} } }
480    }
481  \fp_new:N \l__draw_path_arc_delta_fp
482  \fp_new:N \l__draw_path_arc_start_fp
483  \fp_const:cn { c__draw_path_arc_90_fp } { 4/3 * (sqrt(2) - 1) }
484  \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* **??**.)

\draw_path_arc_axes:nnnn   A simple wrapper.

```
485  \cs_new_protected:Npn \draw_path_arc_axes:nnnn #1#2#3#4
486    {
487      \draw_transform_triangle:nnn { 0cm , 0cm } {#3} {#4}
488      \draw_path_arc:nnn {#1} {#2} { 1pt }
489    }
```

13

\draw_path_ellipse:nnn
__draw_path_ellipse:nnnnnn
__draw_path_ellipse_arci:nnnnnn
__draw_path_ellipse_arcii:nnnnnn
__draw_path_ellipse_arciii:nnnnnn
__draw_path_ellipse_arciv:nnnnnn
\c__draw_path_ellipse_fp

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.

```
490 \cs_new_protected:Npn \draw_path_ellipse:nnn #1#2#3
491   {
492     \__draw_point_process:nnnn
493       { \__draw_path_ellipse:nnnnnn }
494       { \draw_point_transform:n {#1} }
495       { \__draw_point_transform_noshift:n {#2} }
496       { \__draw_point_transform_noshift:n {#3} }
497   }
498 \cs_new_protected:Npn \__draw_path_ellipse:nnnnnn #1#2#3#4#5#6
499   {
500     \use:x
501       {
502         \__draw_path_moveto:nn
503           { \fp_to_dim:n { #1 + #3 } } { \fp_to_dim:n { #2 + #4 } }
504         \__draw_path_ellipse_arci:nnnnnn   {#1} {#2} {#3} {#4} {#5} {#6}
505         \__draw_path_ellipse_arcii:nnnnnn  {#1} {#2} {#3} {#4} {#5} {#6}
506         \__draw_path_ellipse_arciii:nnnnnn {#1} {#2} {#3} {#4} {#5} {#6}
507         \__draw_path_ellipse_arciv:nnnnnn  {#1} {#2} {#3} {#4} {#5} {#6}
508       }
509     \__draw_softpath_closepath:
510     \__draw_path_moveto:nn {#1} {#2}
511   }
512 \cs_new:Npn \__draw_path_ellipse_arci:nnnnnn #1#2#3#4#5#6
513   {
514     \__draw_path_curveto:nnnnnn
515       { \fp_to_dim:n { #1 + #3 + #5 * \c__draw_path_ellipse_fp } }
516       { \fp_to_dim:n { #2 + #4 + #6 * \c__draw_path_ellipse_fp } }
517       { \fp_to_dim:n { #1 + #3 * \c__draw_path_ellipse_fp + #5 } }
518       { \fp_to_dim:n { #2 + #4 * \c__draw_path_ellipse_fp + #6 } }
519       { \fp_to_dim:n { #1 + #5 } }
520       { \fp_to_dim:n { #2 + #6 } }
521   }
522 \cs_new:Npn \__draw_path_ellipse_arcii:nnnnnn #1#2#3#4#5#6
523   {
524     \__draw_path_curveto:nnnnnn
525       { \fp_to_dim:n { #1 - #3 * \c__draw_path_ellipse_fp + #5 } }
526       { \fp_to_dim:n { #2 - #4 * \c__draw_path_ellipse_fp + #6 } }
527       { \fp_to_dim:n { #1 - #3 + #5 * \c__draw_path_ellipse_fp } }
528       { \fp_to_dim:n { #2 - #4 + #6 * \c__draw_path_ellipse_fp } }
529       { \fp_to_dim:n { #1 - #3 } }
530       { \fp_to_dim:n { #2 - #4 } }
531   }
532 \cs_new:Npn \__draw_path_ellipse_arciii:nnnnnn #1#2#3#4#5#6
533   {
534     \__draw_path_curveto:nnnnnn
535       { \fp_to_dim:n { #1 - #3 - #5 * \c__draw_path_ellipse_fp } }
536       { \fp_to_dim:n { #2 - #4 - #6 * \c__draw_path_ellipse_fp } }
537       { \fp_to_dim:n { #1 - #3 * \c__draw_path_ellipse_fp - #5 } }
```

```
538        { \fp_to_dim:n { #2 - #4 * \c__draw_path_ellipse_fp - #6 } }
539        { \fp_to_dim:n { #1 - #5 } }
540        { \fp_to_dim:n { #2 - #6 } }
541    }
542 \cs_new:Npn \__draw_path_ellipse_arciv:nnnnnn #1#2#3#4#5#6
543    {
544      \__draw_path_curveto:nnnnnn
545        { \fp_to_dim:n { #1 + #3 * \c__draw_path_ellipse_fp - #5 } }
546        { \fp_to_dim:n { #2 + #4 * \c__draw_path_ellipse_fp - #6 } }
547        { \fp_to_dim:n { #1 + #3 - #5 * \c__draw_path_ellipse_fp } }
548        { \fp_to_dim:n { #2 + #4 - #6 * \c__draw_path_ellipse_fp } }
549        { \fp_to_dim:n { #1 + #3 } }
550        { \fp_to_dim:n { #2 + #4 } }
551    }
552 \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.

```
553 \cs_new_protected:Npn \draw_path_circle:nn #1#2
554    { \draw_path_ellipse:nnn {#1} { #2 , 0pt } { 0pt , #2 } }
```

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

4.6 Rectangles

\draw_path_rectangle:nn   Building a rectangle can be a single operation, or for rounded versions will involve step-
__draw_path_rectangle:nnnn   by-step construction.
__draw_path_rectangle_rounded:nnnn

```
555 \cs_new_protected:Npn \draw_path_rectangle:nn #1#2
556    {
557      \__draw_point_process:nnn
558        {
559          \bool_lazy_or:nnTF
560            { \l__draw_corner_arc_bool }
561            { \l__draw_matrix_active_bool }
562            { \__draw_path_rectangle_rounded:nnnn }
563            { \__draw_path_rectangle:nnnn }
564        }
565        { \draw_point_transform:n {#1} }
566        {#2}
567    }
568 \cs_new_protected:Npn \__draw_path_rectangle:nnnn #1#2#3#4
569    {
570      \__draw_path_update_limits:nn {#1} {#2}
571      \__draw_path_update_limits:nn { #1 + #3 } { #2 + #4 }
572      \__draw_softpath_rectangle:nnnn {#1} {#2} {#3} {#4}
573      \__draw_path_update_last:nn {#1} {#2}
574    }
575 \cs_new_protected:Npn \__draw_path_rectangle_rounded:nnnn #1#2#3#4
576    {
577      \draw_path_moveto:n { #1 + #3 , #2 + #4 }
578      \draw_path_lineto:n { #1 , #2 + #4 }
579      \draw_path_lineto:n { #1 , #2 }
580      \draw_path_lineto:n { #1 + #3 , #2 }
```

```
581        \draw_path_close:
582        \draw_path_moveto:n { #1 , #2 }
583    }
```

(End definition for `\draw_path_rectangle:nn`, `\__draw_path_rectangle:nnnn`, *and* `\__draw_path_-` *`rectangle_rounded:nnnn`. This function is documented on page* **??**.*)*

`\draw_path_rectangle_corners:nn`   Another shortcut wrapper.

`\__draw_path_rectangle_corners:nnnn`

```
584 \cs_new_protected:Npn \draw_path_rectangle_corners:nn #1#2
585    {
586        \__draw_point_process:nnn
587            { \__draw_path_rectangle_corners:nnnnn {#1} }
588            {#1} {#2}
589    }
590 \cs_new_protected:Npn \__draw_path_rectangle_corners:nnnnn #1#2#3#4#5
591    { \draw_path_rectangle:nn {#1} { #4 - #2 , #5 - #3 } }
```

(End definition for `\draw_path_rectangle_corners:nn` *and* `\__draw_path_rectangle_corners:nnnn`. *This function is documented on page* **??**.*)*

4.7   Grids

`\draw_path_grid:nnnn`   The main complexity here is lining up the grid correctly. To keep it simple, we tidy up

`\__draw_path_grid_auxi:nnnnnn`   the argument ordering first.

`\__draw_path_grid_auxi:ffnnnn`

`\__draw_path_grid_auxii:nnnnnn`

`\__draw_path_grid_auxiii:nnnnnn`

`\__draw_path_grid_auxiiii:ffnnnn`

`\__draw_path_grid_auxiv:nnnnnnnn`

`\__draw_path_grid_auxiv:ffnnnnnn`

```
592 \cs_new_protected:Npn \draw_path_grid:nnnn #1#2#3#4
593    {
594        \__draw_point_process:nnn
595            {
596                \__draw_path_grid_auxi:ffnnnn
597                    { \dim_eval:n { \dim_abs:n {#1} } }
598                    { \dim_eval:n { \dim_abs:n {#2} } }
599            }
600            {#3} {#4}
601    }
602 \cs_new_protected:Npn \__draw_path_grid_auxi:nnnnnn #1#2#3#4#5#6
603    {
604        \dim_compare:nNnTF {#3} > {#5}
605            { \__draw_path_grid_auxii:nnnnnn {#1} {#2} {#5} {#4} {#3} {#6} }
606            { \__draw_path_grid_auxii:nnnnnn {#1} {#2} {#3} {#4} {#5} {#6} }
607    }
608 \cs_generate_variant:Nn \__draw_path_grid_auxi:nnnnnn { ff }
609 \cs_new_protected:Npn \__draw_path_grid_auxii:nnnnnn #1#2#3#4#5#6
610    {
611        \dim_compare:nNnTF {#4} > {#6}
612            { \__draw_path_grid_auxiii:nnnnnn {#1} {#2} {#3} {#6} {#5} {#4} }
613            { \__draw_path_grid_auxiii:nnnnnn {#1} {#2} {#3} {#4} {#5} {#6} }
614    }
615 \cs_new_protected:Npn \__draw_path_grid_auxiii:nnnnnn #1#2#3#4#5#6
616    {
617        \__draw_path_grid_auxiv:ffnnnnnn
618            { \fp_to_dim:n { #1 * trunc(#3/(#1)) } }
619            { \fp_to_dim:n { #2 * trunc(#4/(#2)) } }
620            {#1} {#2} {#3} {#4} {#5} {#6}
621    }
```

```
622 \cs_new_protected:Npn \__draw_path_grid_auxiv:nnnnnnnn #1#2#3#4#5#6#7#8
623   {
624     \dim_step_inline:nnnn
625       {#1}
626       {#3}
627       {#7}
628       {
629         \draw_path_moveto:n { ##1 , #6 }
630         \draw_path_lineto:n { ##1 , #8 }
631       }
632     \dim_step_inline:nnnn
633       {#2}
634       {#4}
635       {#8}
636       {
637         \draw_path_moveto:n { #5 , ##1 }
638         \draw_path_lineto:n { #7 , ##1 }
639       }
640   }
641 \cs_generate_variant:Nn \__draw_path_grid_auxiv:nnnnnnnn { ff }
```

(*End definition for* `\draw_path_grid:nnnn` *and others. This function is documented on page* **??**.)

4.8 Using paths

`\l__draw_path_use_clip_bool`
`\l__draw_path_use_fill_bool`
`\l__draw_path_use_stroke_bool`

Actions to pass to the driver.

```
642 \bool_new:N \l__draw_path_use_clip_bool
643 \bool_new:N \l__draw_path_use_fill_bool
644 \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`.)

`\l__draw_path_use_bb_bool`
`\l__draw_path_use_clear_bool`

Actions handled at the macro layer.

```
645 \bool_new:N \l__draw_path_use_bb_bool
646 \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`.)

`\draw_path_use:n`
`\draw_path_use_clear:n`
`\__draw_path_use:n`
`\__draw_path_use_action_draw:`
`\__draw_path_use_action_fillstroke:`
`\__draw_path_use_stroke_bb:`
`\__draw_path_use_stroke_bb_aux:NnN`

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.

```
647 \cs_new_protected:Npn \draw_path_use:n #1
648   {
649     \tl_if_blank:nF {#1}
650       { \__draw_path_use:n {#1} }
651   }
652 \cs_new_protected:Npn \draw_path_use_clear:n #1
653   {
654     \bool_lazy_or:nnTF
655       { \tl_if_blank_p:n {#1} }
656       { \str_if_eq_p:nn {#1} { clear } }
657       {
658         \__draw_softpath_clear:
```

17

```
659        \__draw_path_reset_limits:
660      }
661      { \__draw_path_use:n { #1 , clear } }
662    }
```

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).

```
663  \cs_new_protected:Npn \__draw_path_use:n #1
664    {
665      \bool_set_false:N \l__draw_path_use_clip_bool
666      \bool_set_false:N \l__draw_path_use_fill_bool
667      \bool_set_false:N \l__draw_path_use_stroke_bool
668      \clist_map_inline:nn {#1}
669        {
670          \cs_if_exist:cTF { l__draw_path_use_ ##1 _ bool }
671            { \bool_set_true:c { l__draw_path_use_ ##1 _ bool } }
672            {
673              \cs_if_exist_use:cF { __draw_path_use_action_ ##1 : }
674                { \msg_error:nnn { draw } { invalid-path-action } {##1} }
675            }
676        }
677      \__draw_softpath_round_corners:
678      \bool_lazy_and:nnT
679        { \l_draw_bb_update_bool }
680        { \l__draw_path_use_stroke_bool }
681        { \__draw_path_use_stroke_bb: }
682      \__draw_softpath_use:
683      \bool_if:NT \l__draw_path_use_clip_bool
684        {
685          \__draw_backend_clip:
686          \bool_set_false:N \l_draw_bb_update_bool
687          \bool_lazy_or:nnF
688            { \l__draw_path_use_fill_bool }
689            { \l__draw_path_use_stroke_bool }
690            { \__draw_backend_discardpath: }
691        }
692      \bool_lazy_or:nnT
693        { \l__draw_path_use_fill_bool }
694        { \l__draw_path_use_stroke_bool }
695        {
696          \use:c
697            {
698              __draw_backend_
699              \bool_if:NT \l__draw_path_use_fill_bool { fill }
700              \bool_if:NT \l__draw_path_use_stroke_bool { stroke }
701              :
702            }
703        }
704      \bool_if:NT \l__draw_path_use_clear_bool
705        { \__draw_softpath_clear: }
706    }
707  \cs_new_protected:Npn \__draw_path_use_action_draw:
708    {
```

```
709        \bool_set_true:N \l__draw_path_use_stroke_bool
710     }
711 \cs_new_protected:Npn \__draw_path_use_action_fillstroke:
712     {
713        \bool_set_true:N \l__draw_path_use_fill_bool
714        \bool_set_true:N \l__draw_path_use_stroke_bool
715     }
```

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.

```
716 \cs_new_protected:Npn \__draw_path_use_stroke_bb:
717     {
718        \__draw_path_use_stroke_bb_aux:NnN x { max } +
719        \__draw_path_use_stroke_bb_aux:NnN y { max } +
720        \__draw_path_use_stroke_bb_aux:NnN x { min } -
721        \__draw_path_use_stroke_bb_aux:NnN y { min } -
722     }
723 \cs_new_protected:Npn \__draw_path_use_stroke_bb_aux:NnN #1#2#3
724     {
725        \dim_compare:nNnF { \dim_use:c { g__draw_ #1#2 _dim } } = { #3 -\c_max_dim }
726          {
727             \dim_gset:cn { g__draw_ #1#2 _dim }
728               {
729                  \use:c { dim_ #2 :nn }
730                    { \dim_use:c { g__draw_ #1#2 _dim } }
731                    {
732                       \dim_use:c { g__draw_path_ #1#2 _dim }
733                      #3 0.5 \g__draw_linewidth_dim
734                    }
735               }
736          }
737     }
```

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

4.9   Scoping paths

Local storage for global data. There is already a \l__draw_softpath_main_tl for path manipulation, so we can reuse that (it is always grouped when the path is being reconstructed).

```
738 \dim_new:N \l__draw_path_lastx_dim
739 \dim_new:N \l__draw_path_lasty_dim
740 \dim_new:N \l__draw_path_xmax_dim
741 \dim_new:N \l__draw_path_xmin_dim
742 \dim_new:N \l__draw_path_ymax_dim
743 \dim_new:N \l__draw_path_ymin_dim
744 \dim_new:N \l__draw_softpath_lastx_dim
745 \dim_new:N \l__draw_softpath_lasty_dim
746 \bool_new:N \l__draw_softpath_corners_bool
```

(*End definition for* \l__draw_path_lastx_dim *and others.*)

\draw_path_scope_begin: Scoping a path is a bit more involved, largely as there are a number of variables to keep
\draw_path_scope_end: hold of.

```
747  \cs_new_protected:Npn \draw_path_scope_begin:
748    {
749      \group_begin:
750        \dim_set_eq:NN \l__draw_path_lastx_dim \g__draw_path_lastx_dim
751        \dim_set_eq:NN \l__draw_path_lasty_dim \g__draw_path_lasty_dim
752        \dim_set_eq:NN \l__draw_path_xmax_dim \g__draw_path_xmax_dim
753        \dim_set_eq:NN \l__draw_path_xmin_dim \g__draw_path_xmin_dim
754        \dim_set_eq:NN \l__draw_path_ymax_dim \g__draw_path_ymax_dim
755        \dim_set_eq:NN \l__draw_path_ymin_dim \g__draw_path_ymin_dim
756        \dim_set_eq:NN \l__draw_softpath_lastx_dim \g__draw_softpath_lastx_dim
757        \dim_set_eq:NN \l__draw_softpath_lasty_dim \g__draw_softpath_lasty_dim
758        \__draw_path_reset_limits:
759        \tl_build_get:NN \g__draw_softpath_main_tl \l__draw_softpath_main_tl
760        \bool_set_eq:NN
761          \l__draw_softpath_corners_bool
762          \g__draw_softpath_corners_bool
763        \__draw_softpath_clear:
764    }
765  \cs_new_protected:Npn \draw_path_scope_end:
766    {
767        \__draw_softpath_clear:
768        \bool_gset_eq:NN
769          \g__draw_softpath_corners_bool
770          \l__draw_softpath_corners_bool
771        \__draw_softpath_add:o \l__draw_softpath_main_tl
772        \dim_gset_eq:NN \g__draw_softpath_lastx_dim \l__draw_softpath_lastx_dim
773        \dim_gset_eq:NN \g__draw_softpath_lasty_dim \l__draw_softpath_lasty_dim
774        \dim_gset_eq:NN \g__draw_path_xmax_dim \l__draw_path_xmax_dim
775        \dim_gset_eq:NN \g__draw_path_xmin_dim \l__draw_path_xmin_dim
776        \dim_gset_eq:NN \g__draw_path_ymax_dim \l__draw_path_ymax_dim
777        \dim_gset_eq:NN \g__draw_path_ymin_dim \l__draw_path_ymin_dim
778        \dim_gset_eq:NN \g__draw_path_lastx_dim \l__draw_path_lastx_dim
779        \dim_gset_eq:NN \g__draw_path_lasty_dim \l__draw_path_lasty_dim
780      \group_end:
781    }
```

(*End definition for* `\draw_path_scope_begin:` *and* `\draw_path_scope_end:`. *These functions are documented on page* **??**.)

```
782  \msg_new:nnnn { draw } { invalid-path-action }
783    { Invalid~action~'#1'~for~path. }
784    { Paths~can~be~used~with~actions~'draw',~'clip',~'fill'~or~'stroke'. }
785  % \end{macrocode}
786  %
787  %    \begin{macrocode}
788  ⟨/package⟩
```

5 **l3draw-points** implementation

```
789  ⟨*package⟩
```

```
790  ⟨@@=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 {⟨x⟩}{⟨y⟩}. Equivalents of following pgf functions are deliberately omitted:

- \pgfpointorigin: Can be given explicitly as 0pt,0pt.

- \pgfpointadd, \pgfpointdiff, \pgfpointscale: Can be given explicitly.

- \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.

5.1 Support functions

__draw_point_process:nn
__draw_point_process_auxi:nn
__draw_point_process_auxii:nw
__draw_point_process:nnn
__draw_point_process_auxiii:nnn
__draw_point_process_auxiv:nw
__draw_point_process:nnnn
__draw_point_process_auxv:nnnn
__draw_point_process_auxvi:nw
__draw_point_process:nnnnn
__draw_point_process_auxvii:nnnnn
__draw_point_process_auxviii:nw

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.

```
791 \cs_new:Npn \__draw_point_process:nn #1#2
792   {
793     \exp_args:Nf \__draw_point_process_auxi:nn
794       { \__draw_point_to_dim:n {#2} }
795       {#1}
796   }
797 \cs_new:Npn \__draw_point_process_auxi:nn #1#2
798   { \__draw_point_process_auxii:nw {#2} #1 \s__draw_stop }
799 \cs_new:Npn \__draw_point_process_auxii:nw #1 #2 , #3 \s__draw_stop
800   { #1 {#2} {#3} }
801 \cs_new:Npn \__draw_point_process:nnn #1#2#3
802   {
803     \exp_args:Nff \__draw_point_process_auxiii:nnn
804       { \__draw_point_to_dim:n {#2} }
805       { \__draw_point_to_dim:n {#3} }
806       {#1}
807   }
808 \cs_new:Npn \__draw_point_process_auxiii:nnn #1#2#3
809   { \__draw_point_process_auxiv:nw {#3} #1 \s__draw_mark #2 \s__draw_stop }
810 \cs_new:Npn \__draw_point_process_auxiv:nw #1 #2 , #3 \s__draw_mark #4 , #5 \s__draw_stop
811   { #1 {#2} {#3} {#4} {#5} }
812 \cs_new:Npn \__draw_point_process:nnnn #1#2#3#4
813   {
```

```
814    \exp_args:Nfff \__draw_point_process_auxv:nnnn
815      { \__draw_point_to_dim:n {#2} }
816      { \__draw_point_to_dim:n {#3} }
817      { \__draw_point_to_dim:n {#4} }
818      {#1}
819    }
820  \cs_new:Npn \__draw_point_process_auxv:nnnn #1#2#3#4
821    { \__draw_point_process_auxvi:nw {#4} #1 \s__draw_mark #2 \s__draw_mark #3 \s__draw_stop }
822  \cs_new:Npn \__draw_point_process_auxvi:nw
823    #1 #2 , #3 \s__draw_mark #4 , #5 \s__draw_mark #6 , #7 \s__draw_stop
824    { #1 {#2} {#3} {#4} {#5} {#6} {#7} }
825  \cs_new:Npn \__draw_point_process:nnnnn #1#2#3#4#5
826    {
827      \exp_args:Nffff \__draw_point_process_auxvii:nnnnn
828        { \__draw_point_to_dim:n {#2} }
829        { \__draw_point_to_dim:n {#3} }
830        { \__draw_point_to_dim:n {#4} }
831        { \__draw_point_to_dim:n {#5} }
832        {#1}
833    }
834  \cs_new:Npn \__draw_point_process_auxvii:nnnnn #1#2#3#4#5
835    {
836      \__draw_point_process_auxviii:nw
837        {#5} #1 \s__draw_mark #2 \s__draw_mark #3 \s__draw_mark #4 \s__draw_stop
838    }
839  \cs_new:Npn \__draw_point_process_auxviii:nw
840    #1 #2 , #3 \s__draw_mark #4 , #5 \s__draw_mark #6 , #7 \s__draw_mark #8 , #9 \s__draw_stop
841    { #1 {#2} {#3} {#4} {#5} {#6} {#7} {#8} {#9} }
```

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

`\__draw_point_to_dim:n`   Co-ordinates are always returned as two dimensions.
`\__draw_point_to_dim_aux:n`
`\__draw_point_to_dim_aux:f`
`\__draw_point_to_dim_aux:w`

```
842  \cs_new:Npn \__draw_point_to_dim:n #1
843    { \__draw_point_to_dim_aux:f { \fp_eval:n {#1} } }
844  \cs_new:Npn \__draw_point_to_dim_aux:n #1
845    { \__draw_point_to_dim_aux:w #1 }
846  \cs_generate_variant:Nn \__draw_point_to_dim_aux:n { f }
847  \cs_new:Npn \__draw_point_to_dim_aux:w ( #1 , ~ #2 ) { #1pt , #2pt }
```

5.2   Polar co-ordinates

Polar co-ordinates may have either one or two lengths, so there is a need to do a sim-
`\draw_point_polar:nn`   ple split before the calculation. As the angle gets used twice, save on any expression
`\draw_point_polar:nnn`   evaluation there and force expansion.
`\__draw_draw_polar:nnn`
`\__draw_draw_polar:fnn`

```
848  \cs_new:Npn \draw_point_polar:nn #1#2
849    { \draw_point_polar:nnn {#1} {#1} {#2} }
850  \cs_new:Npn \draw_point_polar:nnn #1#2#3
851    { \__draw_draw_polar:fnn { \fp_eval:n {#3} } {#1} {#2} }
852  \cs_new:Npn \__draw_draw_polar:nnn #1#2#3
853    { \__draw_point_to_dim:n { cosd(#1) * (#2) , sind(#1) * (#3) } }
854  \cs_generate_variant:Nn \__draw_draw_polar:nnn { f }
```

5.3 Point expression arithmetic

These functions all take point expressions as arguments.

The outcome is the normalised vector from $(0,0)$ in the direction of the point, *i.e.*

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

except where the length is zero, in which case a vertical vector is returned.

```
855 \cs_new:Npn \draw_point_unit_vector:n #1
856   { \__draw_point_process:nn { \__draw_point_unit_vector:nn } {#1} }
857 \cs_new:Npn \__draw_point_unit_vector:nn #1#2
858   {
859     \exp_args:Nf \__draw_point_unit_vector:nnn
860       { \fp_eval:n { (sqrt(#1 * #1 + #2 * #2)) } }
861       {#1} {#2}
862   }
863 \cs_new:Npn \__draw_point_unit_vector:nnn #1#2#3
864   {
865     \fp_compare:nNnTF {#1} = \c_zero_fp
866       { 0pt, 1pt }
867       {
868         \__draw_point_to_dim:n
869           { ( #2 , #3 ) / #1 }
870       }
871   }
```

5.4 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

$$P_y = \frac{(x_1 y_2 - y_1 x_2)(y_3 - y_5) - (x_3 y_4 - y_3 x_4)(y_1 - y_2)}{(x_1 - x_2)(y_3 - y_4) - (y_1 - y_2)(x_3 - x_4)}$$

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.)

```
872 \cs_new:Npn \draw_point_intersect_lines:nnnn #1#2#3#4
873   {
874     \__draw_point_process:nnnnn
875       { \__draw_point_intersect_lines:nnnnnnnn }
876       {#1} {#2} {#3} {#4}
877   }
```

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

#1 x_1

#2 y_1

#3 x_2

#4 y_2

#5 x_3

#6 y_3

#7 x_4

#8 y_4

so now just have to do all of the calculation.

```
878 \cs_new:Npn \__draw_point_intersect_lines:nnnnnnnn #1#2#3#4#5#6#7#8
879   {
880     \__draw_point_intersect_lines_aux:ffffff
881       { \fp_eval:n { #1 * #4 - #2 * #3 } }
882       { \fp_eval:n { #5 * #8 - #6 * #7 } }
883       { \fp_eval:n { #1 - #3 } }
884       { \fp_eval:n { #5 - #7 } }
885       { \fp_eval:n { #2 - #4 } }
886       { \fp_eval:n { #6 - #8 } }
887   }
888 \cs_new:Npn \__draw_point_intersect_lines_aux:nnnnnn #1#2#3#4#5#6
889   {
890     \__draw_point_to_dim:n
891       {
892         ( #2 * #3 - #1 * #4 , #2 * #5 - #1 * #6 )
893           / ( #4 * #5 - #6 * #3 )
894       }
895   }
896 \cs_generate_variant:Nn \__draw_point_intersect_lines_aux:nnnnnn { ffffff }
```

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

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

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

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

24

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

```
897 \cs_new:Npn \draw_point_intersect_circles:nnnnn #1#2#3#4#5
898   {
899     \__draw_point_process:nnn
900       { \__draw_point_intersect_circles_auxi:nnnnnnn {#2} {#4} {#5} }
901       {#1} {#3}
902   }
903 \cs_new:Npn \__draw_point_intersect_circles_auxi:nnnnnnn #1#2#3#4#5#6#7
904   {
905     \__draw_point_intersect_circles_auxii:ffnnnnn
906       { \fp_eval:n {#1} } { \fp_eval:n {#2} } {#4} {#5} {#6} {#7} {#3}
907   }
```

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.

```
908 \cs_new:Npn \__draw_point_intersect_circles_auxii:nnnnnnn #1#2#3#4#5#6#7
909   {
910     \__draw_point_intersect_circles_auxiii:ffnnnnn
911       { \fp_eval:n { #5 - #3 } }
912       { \fp_eval:n { #6 - #4 } }
913       {#1} {#2} {#3} {#4} {#7}
914   }
915 \cs_generate_variant:Nn \__draw_point_intersect_circles_auxii:nnnnnnn { ff }
916 \cs_new:Npn \__draw_point_intersect_circles_auxiii:nnnnnnn #1#2#3#4#5#6#7
917   {
918     \__draw_point_intersect_circles_auxiv:fnnnnnnn
919       { \fp_eval:n { sqrt( #1 * #1 + #2 * #2 ) } }
920       {#1} {#2} {#3} {#4} {#5} {#6} {#7}
921   }
922 \cs_generate_variant:Nn \__draw_point_intersect_circles_auxiii:nnnnnnn { 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.

```
923 \cs_new:Npn \__draw_point_intersect_circles_auxiv:nnnnnnnn #1#2#3#4#5#6#7#8
924   {
925     \__draw_point_intersect_circles_auxv:ffnnnnnnn
926       { \fp_eval:n { 1 / #1 } }
927       { \fp_eval:n { #4 * #4 } }
928       {#1} {#2} {#3} {#5} {#6} {#7} {#8}
```

```
929    }
930  \cs_generate_variant:Nn \__draw_point_intersect_circles_auxiv:nnnnnnnn { f }
931  \cs_new:Npn \__draw_point_intersect_circles_auxv:nnnnnnnnn #1#2#3#4#5#6#7#8#9
932    {
933      \__draw_point_intersect_circles_auxvi:fnnnnnnn
934        { \fp_eval:n { 0.5 * #1 * ( #2 + #3 * #3 - #6 * #6 ) } }
935        {#1} {#2} {#4} {#5} {#7} {#8} {#9}
936    }
937  \cs_generate_variant:Nn \__draw_point_intersect_circles_auxv:nnnnnnnnn { 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.

```
938  \cs_new:Npn \__draw_point_intersect_circles_auxvi:nnnnnnnn #1#2#3#4#5#6#7#8
939    {
940      \__draw_point_intersect_circles_auxvii:fffnnnn
941        { \fp_eval:n { #1 * #2 } }
942        { \int_if_odd:nTF {#8} { 1 } { -1 } }
943        { \fp_eval:n { sqrt ( #3 - #1 * #1 ) * #2 } }
944        {#4} {#5} {#6} {#7}
945    }
946  \cs_generate_variant:Nn \__draw_point_intersect_circles_auxvi:nnnnnnnn { f }
947  \cs_new:Npn \__draw_point_intersect_circles_auxvii:nnnnnnn #1#2#3#4#5#6#7
948    {
949      \__draw_point_to_dim:n
950        { #6 + #4 * #1 + #2 * #3 * #5 , #7 + #5 * #1 + -1 * #2 * #3 * #4 }
951    }
952  \cs_generate_variant:Nn \__draw_point_intersect_circles_auxvii:nnnnnnn { fff }
```

5.5 Interpolation on a line (vector) or arc

Simple maths after expansion.

```
953  \cs_new:Npn \draw_point_interpolate_line:nnn #1#2#3
954    {
955      \__draw_point_process:nnn
956        { \__draw_point_interpolate_line_aux:fnnnn { \fp_eval:n {#1} } }
957        {#2} {#3}
958    }
```

26

```
959 \cs_new:Npn \__draw_point_interpolate_line_aux:nnnnn #1#2#3#4#5
960   {
961     \__draw_point_interpolate_line_aux:fnnnnn { \fp_eval:n { 1 - #1 } }
962       {#1} {#2} {#3} {#4} {#5}
963   }
964 \cs_generate_variant:Nn \__draw_point_interpolate_line_aux:nnnnn { f }
965 \cs_new:Npn \__draw_point_interpolate_line_aux:nnnnnn #1#2#3#4#5#6
966   { \__draw_point_to_dim:n { #2 * #3 + #1 * #5 , #2 * #4 + #1 * #6 } }
967 \cs_generate_variant:Nn \__draw_point_interpolate_line_aux:nnnnnn { f }
```

Same idea but using the normalised length to obtain the scale factor. The start point is needed twice, so we force evaluation, but the end point is needed only the once.

<div style="float:left">\draw_point_interpolate_distance:nnn
_draw_point_interpolate_distance:nnnn
_draw_point_interpolate_distance:nnnnn
_draw_point_interpolate_distance:fnnnn</div>

```
968 \cs_new:Npn \draw_point_interpolate_distance:nnn #1#2#3
969   {
970     \__draw_point_process:nn
971       { \__draw_point_interpolate_distance:nnnn {#1} {#3} }
972       {#2}
973   }
974 \cs_new:Npn \__draw_point_interpolate_distance:nnnn #1#2#3#4
975   {
976     \__draw_point_process:nn
977       {
978         \__draw_point_interpolate_distance:fnnnn
979           { \fp_eval:n {#1} } {#3} {#4}
980       }
981       { \draw_point_unit_vector:n { ( #2 ) - ( #3 , #4 ) } }
982   }
983 \cs_new:Npn \__draw_point_interpolate_distance:nnnnn #1#2#3#4#5
984   { \__draw_point_to_dim:n { #2 + #1 * #4 , #3 + #1 * #5 } }
985 \cs_generate_variant:Nn \__draw_point_interpolate_distance:nnnnn { f }
```

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

<div style="float:left">\draw_point_interpolate_arcaxes:nnnnnn
aw_point_interpolate_arcaxes_auxi:nnnnnnnnn
w_point_interpolate_arcaxes_auxii:nnnnnnnnn
w_point_interpolate_arcaxes_auxii:fnnnnnnnn
aw_point_interpolate_arcaxes_auxiii:nnnnnnnn
aw_point_interpolate_arcaxes_auxiii:fnnnnnn
aw_point_interpolate_arcaxes_auxiv:nnnnnnnn
aw_point_interpolate_arcaxes_auxiv:ffnnnnnn</div>

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.

```
986 \cs_new:Npn \draw_point_interpolate_arcaxes:nnnnnn #1#2#3#4#5#6
987   {
988     \__draw_point_process:nnnn
989       { \__draw_point_interpolate_arcaxes_auxi:nnnnnnnnn {#1} {#5} {#6} }
990       {#2} {#3} {#4}
991   }
992 \cs_new:Npn \__draw_point_interpolate_arcaxes_auxi:nnnnnnnnn #1#2#3#4#5#6#7#8#9
993   {
994     \__draw_point_interpolate_arcaxes_auxii:fnnnnnnnn
995       { \fp_eval:n {#1} } {#2} {#3} {#4} {#5} {#6} {#7} {#8} {#9}
996   }
```

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

#1 p

#2 θ_1

#3 θ_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.

```
997 \cs_new:Npn \__draw_point_interpolate_arcaxes_auxii:nnnnnnnnn #1#2#3#4#5#6#7#8#9
998   {
999     \__draw_point_interpolate_arcaxes_auxiii:fnnnnnn
1000       { \fp_eval:n { #1 * (#3) + ( 1 - #1 ) * (#2) } }
1001       {#4} {#5} {#6} {#7} {#8} {#9}
1002   }
1003 \cs_generate_variant:Nn \__draw_point_interpolate_arcaxes_auxii:nnnnnnnnn { f }
1004 \cs_new:Npn \__draw_point_interpolate_arcaxes_auxiii:nnnnnnn #1#2#3#4#5#6#7
1005   {
1006     \__draw_point_interpolate_arcaxes_auxiv:ffnnnnnn
1007       { \fp_eval:n { cosd (#1) } }
1008       { \fp_eval:n { sind (#1) } }
1009       {#2} {#3} {#4} {#5} {#6} {#7}
1010   }
1011 \cs_generate_variant:Nn \__draw_point_interpolate_arcaxes_auxiii:nnnnnnn { f }
1012 \cs_new:Npn \__draw_point_interpolate_arcaxes_auxiv:nnnnnnnn #1#2#3#4#5#6#7#8
1013   {
1014     \__draw_point_to_dim:n
1015       { #3 + #1 * #5 + #2 * #7 , #4 + #1 * #6 + #2 * #8 }
1016   }
1017 \cs_generate_variant:Nn \__draw_point_interpolate_arcaxes_auxiv:nnnnnnnn { ff }
```

(*End definition for* `\draw_point_interpolate_arcaxes:nnnnnn` *and others. This function is documented on page* **??**.)

\draw_point_interpolate_curve:nnnnn
draw_point_interpolate_curve_auxi:nnnnnnnn
raw_point_interpolate_curve_auxii:nnnnnnnn
raw_point_interpolate_curve_auxii:fnnnnnnnn
\draw_point_interpolate_curve_auxiii:nnnnnn
\draw_point_interpolate_curve_auxiii:fnnnnn
\draw_point_interpolate_curve_auxiv:nnnnnn
\draw_point_interpolate_curve_auxv:nnw
\draw_point_interpolate_curve_auxv:ffw
\draw_point_interpolate_curve_auxvi:n
raw_point_interpolate_curve_auxvii:nnnnnnnn
draw_point_interpolate_curve_auxviii:nnnnn
draw_point_interpolate_curve_auxviii:ffnnnn

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.

```
1018 \cs_new:Npn \draw_point_interpolate_curve:nnnnnn #1#2#3#4#5
1019   {
1020     \__draw_point_process:nnnnn
1021       { \__draw_point_interpolate_curve_auxi:nnnnnnnn {#1} }
1022       {#2} {#3} {#4} {#5}
1023   }
```

```
1024 \cs_new:Npn \__draw_point_interpolate_curve_auxi:nnnnnnnnn #1#2#3#4#5#6#7#8#9
1025   {
1026     \__draw_point_interpolate_curve_auxii:fnnnnnnnn
1027       { \fp_eval:n {#1} }
1028       {#2} {#3} {#4} {#5} {#6} {#7} {#8} {#9}
1029   }
```

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

$$x_1' = (1 - p)x_1 + px_2$$
$$y_1' = (1 - p)y_1 + py_2$$
$$x_2' = (1 - p)x_2 + px_3$$
$$y_2' = (1 - p)y_2 + py_3$$
$$x_3' = (1 - p)x_3 + px_4$$
$$y_3' = (1 - p)y_3 + py_4$$

In the second stage, we can drop the final point

$$x_1'' = (1 - p)x_1' + px_2'$$
$$y_1'' = (1 - p)y_1' + py_2'$$
$$x_2'' = (1 - p)x_2' + px_3'$$
$$y_2'' = (1 - p)y_2' + py_3'$$

and for the final stage only need one set of calculations

$$P_x = (1 - p)x_1'' + px_2''$$
$$P_y = (1 - p)y_1'' + py_2''$$

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

```
1030 \cs_new:Npn \__draw_point_interpolate_curve_auxii:nnnnnnnnn
1031   #1#2#3#4#5#6#7#8#9
1032   {
1033     \__draw_point_interpolate_curve_auxiii:fnnnnn
1034       { \fp_eval:n { 1 - #1 } }
1035       {#1}
1036       { {#2} {#3} } { {#4} {#5} } { {#6} {#7} } { {#8} {#9} }
1037   }
1038 \cs_generate_variant:Nn \__draw_point_interpolate_curve_auxii:nnnnnnnnn { f }
1039 %     \begin{macrocode}
1040 %   We need to do the first cycle, but haven't got enough arguments to keep
1041 %   everything in play at once. So her ewe use a but of argument re-ordering
1042 %   and a single auxiliary to get the job done.
1043 %     \begin{macrocode}
1044 \cs_new:Npn \__draw_point_interpolate_curve_auxiii:nnnnn #1#2#3#4#5#6
1045   {
1046     \__draw_point_interpolate_curve_auxiv:nnnnnn {#1} {#2} #3 #4
1047     \__draw_point_interpolate_curve_auxiv:nnnnnn {#1} {#2} #4 #5
1048     \__draw_point_interpolate_curve_auxiv:nnnnnn {#1} {#2} #5 #6
1049     \prg_do_nothing:
1050     \__draw_point_interpolate_curve_auxvi:n { {#1} {#2} }
```

```
1051        }
1052  \cs_generate_variant:Nn \__draw_point_interpolate_curve_auxiii:nnnnnn { f }
1053  \cs_new:Npn \__draw_point_interpolate_curve_auxiv:nnnnnn #1#2#3#4#5#6
1054        {
1055          \__draw_point_interpolate_curve_auxv:ffw
1056            { \fp_eval:n { #1 * #3 + #2 * #5 } }
1057            { \fp_eval:n { #1 * #4 + #2 * #6 } }
1058        }
1059  \cs_new:Npn \__draw_point_interpolate_curve_auxv:nnw
1060        #1#2#3 \prg_do_nothing: #4#5
1061        {
1062          #3
1063          \prg_do_nothing:
1064          #4 { #5 {#1} {#2} }
1065        }
1066  \cs_generate_variant:Nn \__draw_point_interpolate_curve_auxv:nnw { ff }
1067  %      \begin{macrocode}
1068  %   Get the arguments back into the right places and to the second and
1069  %   third cycles directly.
1070  %      \begin{macrocode}
1071  \cs_new:Npn \__draw_point_interpolate_curve_auxvi:n #1
1072        { \__draw_point_interpolate_curve_auxvii:nnnnnnnn #1 }
1073  \cs_new:Npn \__draw_point_interpolate_curve_auxvii:nnnnnnnn #1#2#3#4#5#6#7#8
1074        {
1075          \__draw_point_interpolate_curve_auxviii:ffffnn
1076            { \fp_eval:n { #1 * #5 + #2 * #3 } }
1077            { \fp_eval:n { #1 * #6 + #2 * #4 } }
1078            { \fp_eval:n { #1 * #7 + #2 * #5 } }
1079            { \fp_eval:n { #1 * #8 + #2 * #6 } }
1080            {#1} {#2}
1081        }
1082  \cs_new:Npn \__draw_point_interpolate_curve_auxviii:nnnnnn #1#2#3#4#5#6
1083        {
1084          \__draw_point_to_dim:n
1085            { #5 * #3 + #6 * #1 , #5 * #4 + #6 * #2 }
1086        }
1087  \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* **??**.)

5.6 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`
`\l__draw_yvec_x_dim`
`\l__draw_yvec_y_dim`
`\l__draw_zvec_x_dim`
`\l__draw_zvec_y_dim`

```
1088  \dim_new:N \l__draw_xvec_x_dim
1089  \dim_new:N \l__draw_xvec_y_dim
1090  \dim_new:N \l__draw_yvec_x_dim
1091  \dim_new:N \l__draw_yvec_y_dim
1092  \dim_new:N \l__draw_zvec_x_dim
1093  \dim_new:N \l__draw_zvec_y_dim
```

(*End definition for* `\l__draw_xvec_x_dim` *and others.*)

`\draw_xvec:n`
`\draw_yvec:n`
`\draw_zvec:n`
`\__draw_vec:nn`
`\__draw_vec:nnn`

Calculate the underlying position and store it.

```
1094 \cs_new_protected:Npn \draw_xvec:n #1
1095   { \__draw_vec:nn { x } {#1} }
1096 \cs_new_protected:Npn \draw_yvec:n #1
1097   { \__draw_vec:nn { y } {#1} }
1098 \cs_new_protected:Npn \draw_zvec:n #1
1099   { \__draw_vec:nn { z } {#1} }
1100 \cs_new_protected:Npn \__draw_vec:nn #1#2
1101   {
1102     \__draw_point_process:nn { \__draw_vec:nnn {#1} } {#2}
1103   }
1104 \cs_new_protected:Npn \__draw_vec:nnn #1#2#3
1105   {
1106     \dim_set:cn { l__draw_ #1 vec_x_dim } {#2}
1107     \dim_set:cn { l__draw_ #1 vec_y_dim } {#3}
1108   }
```

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

Initialise the vectors.

```
1109 \draw_xvec:n { 1cm , 0cm }
1110 \draw_yvec:n { 0cm , 1cm }
1111 \draw_zvec:n { -0.385cm , -0.385cm }
```

`\draw_point_vec:nn`
`\__draw_point_vec:nn`
`\__draw_point_vec:ff`
`\draw_point_vec:nnn`
`\__draw_point_vec:nnn`
`\__draw_point_vec:fff`

Force a single evaluation of each factor, then use these to work out the underlying point.

```
1112 \cs_new:Npn \draw_point_vec:nn #1#2
1113   { \__draw_point_vec:ff { \fp_eval:n {#1} } { \fp_eval:n {#2} } }
1114 \cs_new:Npn \__draw_point_vec:nn #1#2
1115   {
1116     \__draw_point_to_dim:n
1117       {
1118         #1 * \l__draw_xvec_x_dim + #2 * \l__draw_yvec_x_dim ,
1119         #1 * \l__draw_xvec_y_dim + #2 * \l__draw_yvec_y_dim
1120       }
1121   }
1122 \cs_generate_variant:Nn \__draw_point_vec:nn { ff }
1123 \cs_new:Npn \draw_point_vec:nnn #1#2#3
1124   {
1125     \__draw_point_vec:fff
1126       { \fp_eval:n {#1} } { \fp_eval:n {#2} } { \fp_eval:n {#3} }
1127   }
1128 \cs_new:Npn \__draw_point_vec:nnn #1#2#3
1129   {
1130     \__draw_point_to_dim:n
1131       {
1132             #1 * \l__draw_xvec_x_dim
1133           + #2 * \l__draw_yvec_x_dim
1134           + #3 * \l__draw_zvec_x_dim
1135         ,
1136             #1 * \l__draw_xvec_y_dim
1137           + #2 * \l__draw_yvec_y_dim
1138           + #3 * \l__draw_zvec_y_dim
1139       }
1140   }
1141 \cs_generate_variant:Nn \__draw_point_vec:nnn { fff }
```

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

`\draw_point_vec_polar:nn`
`\draw_point_vec_polar:nnn`
`\__draw_point_vec_polar:nnn`
`\__draw_point_vec_polar:fnn`

Much the same as the core polar approach.

```
1142 \cs_new:Npn \draw_point_vec_polar:nn #1#2
1143   { \draw_point_vec_polar:nnn {#1} {#1} {#2} }
1144 \cs_new:Npn \draw_point_vec_polar:nnn #1#2#3
1145   { \__draw_draw_vec_polar:fnn { \fp_eval:n {#3} } {#1} {#2} }
1146 \cs_new:Npn \__draw_draw_vec_polar:nnn #1#2#3
1147   {
1148     \__draw_point_to_dim:n
1149       {
1150         cosd(#1) * (#2) * \l__draw_xvec_x_dim ,
1151         sind(#1) * (#3) * \l__draw_yvec_y_dim
1152       }
1153   }
1154 \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* **??**.)

5.7 Transformations

`\draw_point_transform:n`
`\__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.

```
1155 \cs_new:Npn \draw_point_transform:n #1
1156   {
1157     \__draw_point_process:nn
1158       { \__draw_point_transform:nn } {#1}
1159   }
1160 \cs_new:Npn \__draw_point_transform:nn #1#2
1161   {
1162     \bool_if:NTF \l__draw_matrix_active_bool
1163       {
1164         \__draw_point_to_dim:n
1165           {
1166             (
1167                 \l__draw_matrix_a_fp * #1
1168               + \l__draw_matrix_c_fp * #2
1169               + \l__draw_xshift_dim
1170             )
1171             ,
1172             (
1173                 \l__draw_matrix_b_fp * #1
1174               + \l__draw_matrix_d_fp * #2
1175               + \l__draw_yshift_dim
1176             )
1177           }
1178       }
1179       {
1180         \__draw_point_to_dim:n
1181           {
1182             (#1, #2)
1183             + ( \l__draw_xshift_dim , \l__draw_yshift_dim )
1184           }
```

```
1185              }
1186          }
```

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

__draw_point_transform_noshift:n   A version with no shift: used for internal purposes.
__draw_point_transform_noshift:nn

```
1187 \cs_new:Npn \__draw_point_transform_noshift:n #1
1188    {
1189      \__draw_point_process:nn
1190        { \__draw_point_transform_noshift:nn } {#1}
1191    }
1192 \cs_new:Npn \__draw_point_transform_noshift:nn #1#2
1193    {
1194      \bool_if:NTF \l__draw_matrix_active_bool
1195        {
1196          \__draw_point_to_dim:n
1197            {
1198              (
1199                  \l__draw_matrix_a_fp * #1
1200                + \l__draw_matrix_c_fp * #2
1201              )
1202              ,
1203              (
1204                  \l__draw_matrix_b_fp * #1
1205                + \l__draw_matrix_d_fp * #2
1206              )
1207            }
1208        }
1209        { \__draw_point_to_dim:n { (#1, #2) } }
1210    }
```

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

```
1211 ⟨/package⟩
```

6 l3draw-scopes implementation

```
1212 ⟨*package⟩
```

```
1213 ⟨@@=draw⟩
```

6.1 Drawing environment

\g__draw_xmax_dim   Used to track the overall (official) size of the image created: may not actually be the
\g__draw_xmin_dim   natural size of the content.
\g__draw_ymax_dim
\g__draw_ymin_dim
```
1214 \dim_new:N \g__draw_xmax_dim
1215 \dim_new:N \g__draw_xmin_dim
1216 \dim_new:N \g__draw_ymax_dim
1217 \dim_new:N \g__draw_ymin_dim
```

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

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

```
1218 \bool_new:N \l_draw_bb_update_bool
```

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

`\l__draw_layer_main_box` Box for setting the drawing itself and the top-level layer.

```
1219 \box_new:N \l__draw_main_box
1220 \box_new:N \l__draw_layer_main_box
```

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

`\g__draw_id_int` The drawing number.

```
1221 \int_new:N \g__draw_id_int
```

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

`\__draw_reset_bb:` A simple auxiliary.

```
1222 \cs_new_protected:Npn \__draw_reset_bb:
1223   {
1224     \dim_gset:Nn \g__draw_xmax_dim { -\c_max_dim }
1225     \dim_gset:Nn \g__draw_xmin_dim {  \c_max_dim }
1226     \dim_gset:Nn \g__draw_ymax_dim { -\c_max_dim }
1227     \dim_gset:Nn \g__draw_ymin_dim {  \c_max_dim }
1228   }
```

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

`\draw_begin:` Drawings are created by setting them into a box, then adjusting the box before inserting
`\draw_end:` into the surroundings. Color is set here using the drawing mechanism largely as it then
sets up the internal data structures. It may be that a coffin construct is better here in
the longer term: that may become clearer as the code is completed. As we need to avoid
any insertion of baseline skips, the outer box here has to be an `hbox`. To allow for layers,
there is some box nesting: notice that we

```
1229 \cs_new_protected:Npn \draw_begin:
1230   {
1231     \group_begin:
1232       \int_gincr:N \g__draw_id_int
1233       \hbox_set:Nw \l__draw_main_box
1234         \__draw_backend_begin:
1235         \__draw_reset_bb:
1236         \__draw_path_reset_limits:
1237         \bool_set_true:N \l_draw_bb_update_bool
1238         \draw_transform_matrix_reset:
1239         \draw_transform_shift_reset:
1240         \__draw_softpath_clear:
1241         \draw_linewidth:n { \l_draw_default_linewidth_dim }
1242         \color_select:n { . }
1243         \draw_nonzero_rule:
1244         \draw_cap_butt:
1245         \draw_join_miter:
1246         \draw_miterlimit:n { 10 }
1247         \draw_dash_pattern:nn { } { 0cm }
1248         \hbox_set:Nw \l__draw_layer_main_box
1249   }
1250 \cs_new_protected:Npn \draw_end:
1251   {
1252         \exp_args:NNNV \hbox_set_end:
```

34

```
1253        \clist_set:Nn \l_draw_layers_clist \l_draw_layers_clist
1254        \__draw_layers_insert:
1255      \__draw_backend_end:
1256    \hbox_set_end:
1257    \dim_compare:nNnT \g__draw_xmin_dim = \c_max_dim
1258      {
1259        \dim_gzero:N \g__draw_xmax_dim
1260        \dim_gzero:N \g__draw_xmin_dim
1261        \dim_gzero:N \g__draw_ymax_dim
1262        \dim_gzero:N \g__draw_ymin_dim
1263      }
1264    \hbox_set:Nn \l__draw_main_box
1265      {
1266        \skip_horizontal:n { -\g__draw_xmin_dim }
1267        \box_move_down:nn { \g__draw_ymin_dim }
1268          { \box_use_drop:N \l__draw_main_box }
1269      }
1270    \box_set_ht:Nn \l__draw_main_box
1271      { \g__draw_ymax_dim - \g__draw_ymin_dim }
1272    \box_set_dp:Nn \l__draw_main_box { 0pt }
1273    \box_set_wd:Nn \l__draw_main_box
1274      { \g__draw_xmax_dim - \g__draw_xmin_dim }
1275    \mode_leave_vertical:
1276    \box_use_drop:N \l__draw_main_box
1277      \group_end:
1278  }
```

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

6.2 Scopes

\l__draw_linewidth_dim Storage for local variables.
\l__draw_fill_color_tl
\l__draw_stroke_color_tl

```
1279 \dim_new:N \l__draw_linewidth_dim
1280 \tl_new:N \l__draw_fill_color_tl
1281 \tl_new:N \l__draw_stroke_color_tl
```

(*End definition for* \l__draw_linewidth_dim, \l__draw_fill_color_tl, *and* \l__draw_stroke_color_-
tl.)

\draw_scope_begin: As well as the graphics (and TeX) scope, also deal with global data structures.
\draw_scope_begin:

```
1282 \cs_new_protected:Npn \draw_scope_begin:
1283   {
1284     \__draw_backend_scope_begin:
1285     \group_begin:
1286       \dim_set_eq:NN \l__draw_linewidth_dim \g__draw_linewidth_dim
1287       \draw_path_scope_begin:
1288   }
1289 \cs_new_protected:Npn \draw_scope_end:
1290   {
1291       \draw_path_scope_end:
1292       \dim_gset_eq:NN \g__draw_linewidth_dim \l__draw_linewidth_dim
1293     \group_end:
1294     \__draw_backend_scope_end:
1295   }
```

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

\l__draw_xmax_dim Storage for the bounding box.
\l__draw_xmin_dim
\l__draw_ymax_dim   1296 \dim_new:N \l__draw_xmax_dim
\l__draw_ymin_dim   1297 \dim_new:N \l__draw_xmin_dim
                     1298 \dim_new:N \l__draw_ymax_dim
                     1299 \dim_new:N \l__draw_ymin_dim

(*End definition for* \l__draw_xmax_dim *and others.*)

__draw_scope_bb_begin: The bounding box is simple: a straight group-based save and restore approach.
  __draw_scope_bb_end:
                     1300 \cs_new_protected:Npn __draw_scope_bb_begin:
                     1301   {
                     1302   \group_begin:
                     1303   \dim_set_eq:NN \l__draw_xmax_dim \g__draw_xmax_dim
                     1304   \dim_set_eq:NN \l__draw_xmin_dim \g__draw_xmin_dim
                     1305   \dim_set_eq:NN \l__draw_ymax_dim \g__draw_ymax_dim
                     1306   \dim_set_eq:NN \l__draw_ymin_dim \g__draw_ymin_dim
                     1307   __draw_reset_bb:
                     1308   }
                     1309 \cs_new_protected:Npn __draw_scope_bb_end:
                     1310   {
                     1311   \dim_gset_eq:NN \g__draw_xmax_dim \l__draw_xmax_dim
                     1312   \dim_gset_eq:NN \g__draw_xmin_dim \l__draw_xmin_dim
                     1313   \dim_gset_eq:NN \g__draw_ymax_dim \l__draw_ymax_dim
                     1314   \dim_gset_eq:NN \g__draw_ymin_dim \l__draw_ymin_dim
                     1315   \group_end:
                     1316   }

(*End definition for* __draw_scope_bb_begin: *and* __draw_scope_bb_end:.)

\draw_suspend_begin: Suspend all parts of a drawing.
  \draw_suspend_end:
                     1317 \cs_new_protected:Npn \draw_suspend_begin:
                     1318   {
                     1319   __draw_scope_bb_begin:
                     1320   \draw_path_scope_begin:
                     1321   \draw_transform_matrix_reset:
                     1322   \draw_transform_shift_reset:
                     1323   __draw_layers_save:
                     1324   }
                     1325 \cs_new_protected:Npn \draw_suspend_end:
                     1326   {
                     1327   __draw_layers_restore:
                     1328   \draw_path_scope_end:
                     1329   __draw_scope_bb_end:
                     1330   }

(*End definition for* \draw_suspend_begin: *and* \draw_suspend_end:. *These functions are documented on page* **??**.)

                     1331 ⟨/package⟩

7 **l3draw-softpath** implementation

1332 ⟨*package⟩

1333 ⟨@@=draw⟩

7.1 Managing soft paths

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 so we use `\tl_build_...` functions.

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.

`\g__draw_softpath_main_tl` The soft path itself.

1334 `\tl_new:N \g__draw_softpath_main_tl`

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

`\l__draw_softpath_internal_tl` The soft path itself.

1335 `\tl_new:N \l__draw_softpath_internal_tl`

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

`\g__draw_softpath_corners_bool` Allow for optimised path use.

1336 `\bool_new:N \g__draw_softpath_corners_bool`

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

`\__draw_softpath_add:n`
`\__draw_softpath_add:o`
`\__draw_softpath_add:x`

1337 `\cs_new_protected:Npn \__draw_softpath_add:n`
1338 `  { \tl_build_gput_right:Nn \g__draw_softpath_main_tl }`
1339 `\cs_generate_variant:Nn \__draw_softpath_add:n { o, x }`

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

`\__draw_softpath_use:`
`\__draw_softpath_clear:`

Using and clearing is trivial.

1340 `\cs_new_protected:Npn \__draw_softpath_use:`
1341 `  {`
1342 `    \tl_build_get:NN \g__draw_softpath_main_tl \l__draw_softpath_internal_tl`
1343 `    \l__draw_softpath_internal_tl`
1344 `  }`
1345 `\cs_new_protected:Npn \__draw_softpath_clear:`
1346 `  {`
1347 `    \tl_build_gclear:N \g__draw_softpath_main_tl`
1348 `    \bool_gset_false:N \g__draw_softpath_corners_bool`
1349 `  }`

(*End definition for* `\__draw_softpath_use:` *and* `\__draw_softpath_clear:`.)

For tracking the end of the path (to close it).

```
1350 \dim_new:N \g__draw_softpath_lastx_dim
1351 \dim_new:N \g__draw_softpath_lasty_dim
```

(*End definition for* \g__draw_softpath_lastx_dim *and* \g__draw_softpath_lasty_dim.)

Track if moving a point should update the close position.

```
1352 \bool_new:N \g__draw_softpath_move_bool
1353 \bool_gset_true:N \g__draw_softpath_move_bool
```

(*End definition for* \g__draw_softpath_move_bool.)

The various parts of a path expressed as the appropriate soft path functions.

```
1354 \cs_new_protected:Npn \__draw_softpath_closepath:
1355   {
1356     \__draw_softpath_add:x
1357       {
1358         \__draw_softpath_close_op:nn
1359           { \dim_use:N \g__draw_softpath_lastx_dim }
1360           { \dim_use:N \g__draw_softpath_lasty_dim }
1361       }
1362   }
1363 \cs_new_protected:Npn \__draw_softpath_curveto:nnnnnn #1#2#3#4#5#6
1364   {
1365     \__draw_softpath_add:n
1366       {
1367         \__draw_softpath_curveto_opi:nn {#1} {#2}
1368         \__draw_softpath_curveto_opii:nn {#3} {#4}
1369         \__draw_softpath_curveto_opiii:nn {#5} {#6}
1370       }
1371   }
1372 \cs_new_protected:Npn \__draw_softpath_lineto:nn #1#2
1373   {
1374     \__draw_softpath_add:n
1375       { \__draw_softpath_lineto_op:nn {#1} {#2} }
1376   }
1377 \cs_new_protected:Npn \__draw_softpath_moveto:nn #1#2
1378   {
1379     \__draw_softpath_add:n
1380       { \__draw_softpath_moveto_op:nn {#1} {#2} }
1381     \bool_if:NT \g__draw_softpath_move_bool
1382       {
1383         \dim_gset:Nn \g__draw_softpath_lastx_dim {#1}
1384         \dim_gset:Nn \g__draw_softpath_lasty_dim {#2}
1385       }
1386   }
1387 \cs_new_protected:Npn \__draw_softpath_rectangle:nnnn #1#2#3#4
1388   {
1389     \__draw_softpath_add:n
1390       {
1391         \__draw_softpath_rectangle_opi:nn {#1} {#2}
1392         \__draw_softpath_rectangle_opii:nn {#3} {#4}
1393       }
1394   }
```

```
1395 \cs_new_protected:Npn \__draw_softpath_roundpoint:nn #1#2
1396   {
1397     \__draw_softpath_add:n
1398       { \__draw_softpath_roundpoint_op:nn {#1} {#2} }
1399     \bool_gset_true:N \g__draw_softpath_corners_bool
1400   }
1401 \cs_generate_variant:Nn \__draw_softpath_roundpoint:nn { VV }
```

(*End definition for* __draw_softpath_curveto:nnnnnn *and others.*)

The markers for operations: all the top-level ones take two arguments. The support tokens for curves have to be different in meaning to a round point, hence being quark-like.

```
1402 \cs_new_protected:Npn \__draw_softpath_close_op:nn #1#2
1403   { \__draw_backend_closepath: }
1404 \cs_new_protected:Npn \__draw_softpath_curveto_opi:nn #1#2
1405   { \__draw_softpath_curveto_opi:nnNnnNnn {#1} {#2} }
1406 \cs_new_protected:Npn \__draw_softpath_curveto_opi:nnNnnNnn #1#2#3#4#5#6#7#8
1407   { \__draw_backend_curveto:nnnnnn {#1} {#2} {#4} {#5} {#7} {#8} }
1408 \cs_new_protected:Npn \__draw_softpath_curveto_opii:nn #1#2
1409   { \__draw_softpath_curveto_opii:nn }
1410 \cs_new_protected:Npn \__draw_softpath_curveto_opiii:nn #1#2
1411   { \__draw_softpath_curveto_opiii:nn }
1412 \cs_new_protected:Npn \__draw_softpath_lineto_op:nn #1#2
1413   { \__draw_backend_lineto:nn {#1} {#2} }
1414 \cs_new_protected:Npn \__draw_softpath_moveto_op:nn #1#2
1415   { \__draw_backend_moveto:nn {#1} {#2} }
1416 \cs_new_protected:Npn \__draw_softpath_roundpoint_op:nn #1#2 { }
1417 \cs_new_protected:Npn \__draw_softpath_rectangle_opi:nn #1#2
1418   { \__draw_softpath_rectangle_opi:nnNnn {#1} {#2} }
1419 \cs_new_protected:Npn \__draw_softpath_rectangle_opi:nnNnn #1#2#3#4#5
1420   { \__draw_backend_rectangle:nnnn {#1} {#2} {#4} {#5} }
1421   \cs_new_protected:Npn \__draw_softpath_rectangle_opii:nn #1#2 { }
```

(*End definition for* __draw_softpath_close_op:nn *and others.*)

7.2 Rounding soft path corners

The aim here is to find corner rounding points and to replace them with arcs of appropriate length. The approach is exactly that in pgf: step through, find the corners, find the supporting data, do the rounding.

\l__draw_softpath_main_tl   For constructing the updated path.

```
1422 \tl_new:N \l__draw_softpath_main_tl
```

(*End definition for* \l__draw_softpath_main_tl.)

\l__draw_softpath_part_tl   Data structures.

```
1423 \tl_new:N \l__draw_softpath_part_tl
1424 \tl_new:N \l__draw_softpath_curve_end_tl
```

(*End definition for* \l__draw_softpath_part_tl.)

\l__draw_softpath_lastx_fp
\l__draw_softpath_lasty_fp
\l__draw_softpath_corneri_dim
\l__draw_softpath_cornerii_dim
\l__draw_softpath_first_tl
\l__draw_softpath_move_tl

Position tracking: the token list data may be entirely empty or set to a co-ordinate.

```
1425 \fp_new:N \l__draw_softpath_lastx_fp
1426 \fp_new:N \l__draw_softpath_lasty_fp
1427 \dim_new:N \l__draw_softpath_corneri_dim
1428 \dim_new:N \l__draw_softpath_cornerii_dim
1429 \tl_new:N \l__draw_softpath_first_tl
1430 \tl_new:N \l__draw_softpath_move_tl
```

(*End definition for* \l__draw_softpath_lastx_fp *and others.*)

\c__draw_softpath_arc_fp   The magic constant.

```
1431 \fp_const:Nn \c__draw_softpath_arc_fp { 4/3 * (sqrt(2) - 1) }
```

(*End definition for* \c__draw_softpath_arc_fp.)

__draw_softpath_round_corners:
__draw_softpath_round_loop:Nnn
__draw_softpath_round_action:nn
__draw_softpath_round_action:Nnn
__draw_softpath_round_action_curveto:NnnNnn
__draw_softpath_round_action_close:
__draw_softpath_round_lookahead:NnnNnn
__draw_softpath_round_roundpoint:NnnNnnNnn
__draw_softpath_round_calc:NnnNnn
__draw_softpath_round_calc:nnnnnn
__draw_softpath_round_calc:fVnnnn
__draw_softpath_round_calc:nnnnw
__draw_softpath_round_close:nn
__draw_softpath_round_close:w
__draw_softpath_round_end:

Rounding corners on a path means going through the entire path and adjusting it. As such, we avoid this entirely if we know there are no corners to deal with. Assuming there is work to do, we recover the existing path and start a loop.

```
1432 \cs_new_protected:Npn \__draw_softpath_round_corners:
1433   {
1434     \bool_if:NT \g__draw_softpath_corners_bool
1435       {
1436         \group_begin:
1437           \tl_clear:N \l__draw_softpath_main_tl
1438           \tl_clear:N \l__draw_softpath_part_tl
1439           \fp_zero:N \l__draw_softpath_lastx_fp
1440           \fp_zero:N \l__draw_softpath_lasty_fp
1441           \tl_clear:N \l__draw_softpath_first_tl
1442           \tl_clear:N \l__draw_softpath_move_tl
1443           \tl_build_get:NN \g__draw_softpath_main_tl \l__draw_softpath_internal_tl
1444           \exp_after:wN \__draw_softpath_round_loop:Nnn
1445             \l__draw_softpath_internal_tl
1446             \q__draw_recursion_tail ? ?
1447             \q__draw_recursion_stop
1448         \group_end:
1449       }
1450     \bool_gset_false:N \g__draw_softpath_corners_bool
1451   }
```

The loop can take advantage of the fact that all soft path operations are made up of a token followed by two arguments. At this stage, there is a simple split: have we round a round point. If so, is there any actual rounding to be done: if the arcs have come through zero, just ignore it. In cases where we are not at a corner, we simply move along the path, allowing for any new part starting due to a `moveto`.

```
1452 \cs_new_protected:Npn \__draw_softpath_round_loop:Nnn #1#2#3
1453   {
1454     \__draw_if_recursion_tail_stop_do:Nn #1 { \__draw_softpath_round_end: }
1455     \token_if_eq_meaning:NNTF #1 \__draw_softpath_roundpoint_op:nn
1456       { \__draw_softpath_round_action:nn {#2} {#3} }
1457       {
1458         \tl_if_empty:NT \l__draw_softpath_first_tl
1459           { \tl_set:Nn \l__draw_softpath_first_tl { {#2} {#3} } }
1460         \fp_set:Nn \l__draw_softpath_lastx_fp {#2}
1461         \fp_set:Nn \l__draw_softpath_lasty_fp {#3}
```

```
1462        \token_if_eq_meaning:NNTF #1 \__draw_softpath_moveto_op:nn
1463          {
1464            \tl_put_right:No \l__draw_softpath_main_tl
1465              \l__draw_softpath_move_tl
1466            \tl_put_right:No \l__draw_softpath_main_tl
1467              \l__draw_softpath_part_tl
1468            \tl_set:Nn \l__draw_softpath_move_tl { #1 {#2} {#3} }
1469            \tl_clear:N \l__draw_softpath_first_tl
1470            \tl_clear:N \l__draw_softpath_part_tl
1471          }
1472          { \tl_put_right:Nn \l__draw_softpath_part_tl { #1 {#2} {#3} } }
1473        \__draw_softpath_round_loop:Nnn
1474      }
1475  }
1476 \cs_new_protected:Npn \__draw_softpath_round_action:nn #1#2
1477  {
1478    \dim_set:Nn \l__draw_softpath_corneri_dim {#1}
1479    \dim_set:Nn \l__draw_softpath_cornerii_dim {#2}
1480    \bool_lazy_and:nnTF
1481      { \dim_compare_p:nNn \l__draw_softpath_corneri_dim = { 0pt } }
1482      { \dim_compare_p:nNn \l__draw_softpath_cornerii_dim = { 0pt } }
1483      { \__draw_softpath_round_loop:Nnn }
1484      { \__draw_softpath_round_action:Nnn }
1485  }
```

We now have a round point to work on and have grabbed the next item in the path. There are only a few cases where we have to do anything. Each of them is picked up by looking for the appropriate action.

```
1486 \cs_new_protected:Npn \__draw_softpath_round_action:Nnn #1#2#3
1487  {
1488    \tl_if_empty:NT \l__draw_softpath_first_tl
1489      { \tl_set:Nn \l__draw_softpath_first_tl { {#2} {#3} } }
1490    \token_if_eq_meaning:NNTF #1 \__draw_softpath_curveto_opi:nn
1491      { \__draw_softpath_round_action_curveto:NnnNnn }
1492      {
1493        \token_if_eq_meaning:NNTF #1 \__draw_softpath_close_op:nn
1494          { \__draw_softpath_round_action_close: }
1495          {
1496            \token_if_eq_meaning:NNTF #1 \__draw_softpath_lineto_op:nn
1497              { \__draw_softpath_round_lookahead:NnnNnn }
1498              { \__draw_softpath_round_loop:Nnn }
1499          }
1500      }
1501      #1 {#2} {#3}
1502  }
```

For a curve, we collect the two control points then move on to grab the end point and add the curve there: the second control point becomes our starter.

```
1503 \cs_new_protected:Npn \__draw_softpath_round_action_curveto:NnnNnn
1504   #1#2#3#4#5#6
1505  {
1506    \tl_put_right:Nn \l__draw_softpath_part_tl
1507      { #1 {#2} {#3} #4 {#5} {#6} }
1508    \fp_set:Nn \l__draw_softpath_lastx_fp {#5}
1509    \fp_set:Nn \l__draw_softpath_lasty_fp {#6}
```

```
1510        \__draw_softpath_round_lookahead:NnnNnn
1511      }
1512    \cs_new_protected:Npn \__draw_softpath_round_action_close:
1513      {
1514        \bool_lazy_and:nnTF
1515          { ! \tl_if_empty_p:N \l__draw_softpath_first_tl }
1516          { ! \tl_if_empty_p:N \l__draw_softpath_move_tl }
1517          {
1518            \exp_after:wN \__draw_softpath_round_close:nn
1519              \l__draw_softpath_first_tl
1520          }
1521          { \__draw_softpath_round_loop:Nnn }
1522      }
```

At this stage we have a current (sub)operation (`#1`) and the next operation (`#4`), and can therefore decide whether to round or not. In the case of yet another rounding marker, we have to look a bit further ahead.

```
1523    \cs_new_protected:Npn \__draw_softpath_round_lookahead:NnnNnn #1#2#3#4#5#6
1524      {
1525        \bool_lazy_any:nTF
1526          {
1527            { \token_if_eq_meaning_p:NN #4 \__draw_softpath_lineto_op:nn }
1528            { \token_if_eq_meaning_p:NN #4 \__draw_softpath_curveto_opi:nn }
1529            { \token_if_eq_meaning_p:NN #4 \__draw_softpath_close_op:nn }
1530          }
1531          {
1532            \__draw_softpath_round_calc:NnnNnn
1533              \__draw_softpath_round_loop:Nnn
1534              {#5} {#6}
1535          }
1536          {
1537            \token_if_eq_meaning:NNTF #4 \__draw_softpath_roundpoint_op:nn
1538              { \__draw_softpath_round_roundpoint:NnnNnnNnn }
1539              { \__draw_softpath_round_loop:Nnn }
1540          }
1541        #1 {#2} {#3}
1542        #4 {#5} {#6}
1543      }
1544    \cs_new_protected:Npn \__draw_softpath_round_roundpoint:NnnNnnNnn
1545      #1#2#3#4#5#6#7#8#9
1546      {
1547        \__draw_softpath_round_calc:NnnNnn
1548          \__draw_softpath_round_loop:Nnn
1549          {#8} {#9}
1550          #1 {#2} {#3}
1551        #4 {#5} {#6} #7 {#8} {#9}
1552      }
```

We now have all of the data needed to construct a rounded corner: all that is left to do is to work out the detail! At this stage, we have details of where the corner itself is (`#5`, `#6`), and where the next point is (`#2`, `#3`). There are two types of calculations to do. First, we need to interpolate from those two points in the direction of the corner, in order to work out where the curve we are adding will start and end. From those, plus the points we already have, we work out where the control points will lie. All of this is done

in an expansion to avoid multiple calls to `\tl_put_right:Nx`. The end point of the line is worked out up-front and saved: we need that if dealing with a close-path operation.

```
1553 \cs_new_protected:Npn \__draw_softpath_round_calc:NnnNnn #1#2#3#4#5#6
1554   {
1555     \tl_set:Nx \l__draw_softpath_curve_end_tl
1556       {
1557         \draw_point_interpolate_distance:nnn
1558           \l__draw_softpath_cornerii_dim
1559           { #5 , #6 } { #2 , #3 }
1560       }
1561     \tl_put_right:Nx \l__draw_softpath_part_tl
1562       {
1563         \exp_not:N #4
1564         \__draw_softpath_round_calc:fVnnnn
1565           {
1566             \draw_point_interpolate_distance:nnn
1567               \l__draw_softpath_corneri_dim
1568               { #5 , #6 }
1569               {
1570                 \l__draw_softpath_lastx_fp ,
1571                 \l__draw_softpath_lasty_fp
1572               }
1573           }
1574           \l__draw_softpath_curve_end_tl
1575           {#5} {#6} {#2} {#3}
1576       }
1577     \fp_set:Nn \l__draw_softpath_lastx_fp {#5}
1578     \fp_set:Nn \l__draw_softpath_lasty_fp {#6}
1579     #1
1580   }
```

At this stage we have the two curve end points, but they are in co-ordinate form. So we split them up (with some more reordering).

```
1581 \cs_new:Npn \__draw_softpath_round_calc:nnnnnn #1#2#3#4#5#6
1582   {
1583     \__draw_softpath_round_calc:nnnnw {#3} {#4} {#5} {#6}
1584       #1 \s__draw_mark #2 \s__draw_stop
1585   }
1586 \cs_generate_variant:Nn \__draw_softpath_round_calc:nnnnnn { fV }
```

The calculations themselves are relatively straight-forward, as we use a quadratic Bézier curve.

```
1587 \cs_new:Npn \__draw_softpath_round_calc:nnnnw
1588   #1#2#3#4 #5 , #6 \s__draw_mark #7 , #8 \s__draw_stop
1589   {
1590     {#5} {#6}
1591     \exp_not:N \__draw_softpath_curveto_opi:nn
1592       {
1593         \fp_to_dim:n
1594           { #5 + \c__draw_softpath_arc_fp * ( #1 - #5 ) }
1595       }
1596       {
1597         \fp_to_dim:n
1598           { #6 + \c__draw_softpath_arc_fp * ( #2 - #6 ) }
```

```
1599          }
1600      \exp_not:N \__draw_softpath_curveto_opii:nn
1601        {
1602          \fp_to_dim:n
1603            { #7 + \c__draw_softpath_arc_fp * ( #1 - #7 ) }
1604        }
1605        {
1606          \fp_to_dim:n
1607            { #8 + \c__draw_softpath_arc_fp* ( #2 - #8 ) }
1608        }
1609      \exp_not:N \__draw_softpath_curveto_opiii:nn
1610          {#7} {#8}
1611    }
```

To deal with a close-path operation, we need to do some manipulation. It needs to be treated as a line operation for rounding, and then have the close path operation re-added at the point where the curve ends. That means saving the end point in the calculation step (see earlier), and shuffling a lot.

```
1612 \cs_new_protected:Npn \__draw_softpath_round_close:nn #1#2
1613    {
1614      \use:x
1615        {
1616          \__draw_softpath_round_calc:NnnNnn
1617            {
1618              \tl_set:Nx \exp_not:N \l__draw_softpath_move_tl
1619                {
1620                  \__draw_softpath_moveto_op:nn
1621                  \exp_not:N \exp_after:wN
1622                    \exp_not:N \__draw_softpath_round_close:w
1623                    \exp_not:N \l__draw_softpath_curve_end_tl
1624                      \s__draw_stop
1625                }
1626              \use:x
1627                {
1628                  \exp_not:N \exp_not:N \exp_not:N \use_i:nnnn
1629                    {
1630                      \__draw_softpath_round_loop:Nnn
1631                        \__draw_softpath_close_op:nn
1632                        \exp_not:N \exp_after:wN
1633                          \exp_not:N \__draw_softpath_round_close:w
1634                          \exp_not:N \l__draw_softpath_curve_end_tl
1635                            \s__draw_stop
1636                    }
1637                }
1638            }
1639          {#1} {#2}
1640          \__draw_softpath_lineto_op:nn
1641          \exp_after:wN \use_none:n \l__draw_softpath_move_tl
1642        }
1643    }
1644 \cs_new:Npn \__draw_softpath_round_close:w #1 , #2 \s__draw_stop { {#1} {#2} }
```

Tidy up the parts of the path, complete the built token list and put it back into action.

```
1645 \cs_new_protected:Npn \__draw_softpath_round_end:
1646    {
```

44

```
1647      \tl_put_right:No \l__draw_softpath_main_tl
1648        \l__draw_softpath_move_tl
1649      \tl_put_right:No \l__draw_softpath_main_tl
1650        \l__draw_softpath_part_tl
1651      \tl_build_gclear:N \g__draw_softpath_main_tl
1652      \__draw_softpath_add:o \l__draw_softpath_main_tl
1653    }
```

(*End definition for* __draw_softpath_round_corners: *and others.*)

```
1654 ⟨/package⟩
```

8   l3draw-state implementation

```
1655 ⟨*package⟩
```

```
1656 ⟨@@=draw⟩
```

This sub-module covers more-or-less the same ideas as `pgfcoregraphicstate.code.tex`.
At present, equivalents of the following are currently absent:

- \pgfsetinnerlinewidth, \pgfinnerlinewidth, \pgfsetinnerstrokecolor, \pgfsetinnerstro
  Likely to be added on further work is done on paths/stroking.

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

```
1657 \dim_new:N \g__draw_linewidth_dim
```

(*End definition for* \g__draw_linewidth_dim.)

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

```
1658 \dim_new:N \l_draw_default_linewidth_dim
1659 \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.

```
1660 \cs_new_protected:Npn \draw_linewidth:n #1
1661   {
1662     \dim_gset:Nn \g__draw_linewidth_dim { \fp_to_dim:n {#1} }
1663     \__draw_backend_linewidth:n \g__draw_linewidth_dim
1664   }
```

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

\draw_dash_pattern:nn   Evaluated all of the list and pass it to the driver layer.
\l__draw_tmp_seq
```
1665 \cs_new_protected:Npn \draw_dash_pattern:nn #1#2
1666   {
1667     \group_begin:
1668       \seq_set_from_clist:Nn \l__draw_tmp_seq {#1}
1669       \seq_set_map:NNn \l__draw_tmp_seq \l__draw_tmp_seq
1670         { \fp_to_dim:n {##1} }
1671       \use:x
1672         {
1673           \__draw_backend_dash_pattern:nn
1674             { \seq_use:Nn \l__draw_tmp_seq { , } }
```

```
1675                    { \fp_to_dim:n {#2} }
1676              }
1677        \group_end:
1678    }
1679 \seq_new:N \l__draw_tmp_seq
```

(*End definition for* \draw_dash_pattern:nn *and* \l__draw_tmp_seq. *This function is documented on page* **??**.)

\draw_miterlimit:n   Pass through to the driver layer.

```
1680 \cs_new_protected:Npn \draw_miterlimit:n #1
1681    { \exp_args:Nx \__draw_backend_miterlimit:n { \fp_eval:n {#1} } }
```

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

\draw_cap_butt: All straight wrappers.
\draw_cap_rectangle:
\draw_cap_round:
\draw_evenodd_rule:
\draw_nonzero_rule:
\draw_join_bevel:
\draw_join_miter:
\draw_join_round:

```
1682 \cs_new_protected:Npn \draw_cap_butt: { \__draw_backend_cap_butt: }
1683 \cs_new_protected:Npn \draw_cap_rectangle: { \__draw_backend_cap_rectangle: }
1684 \cs_new_protected:Npn \draw_cap_round: { \__draw_backend_cap_round: }
1685 \cs_new_protected:Npn \draw_evenodd_rule: { \__draw_backend_evenodd_rule: }
1686 \cs_new_protected:Npn \draw_nonzero_rule: { \__draw_backend_nonzero_rule: }
1687 \cs_new_protected:Npn \draw_join_bevel: { \__draw_backend_join_bevel: }
1688 \cs_new_protected:Npn \draw_join_miter: { \__draw_backend_join_miter: }
1689 \cs_new_protected:Npn \draw_join_round: { \__draw_backend_join_round: }
```

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

```
1690 ⟨/package⟩
```

9   l3draw-transforms implementation

```
1691 ⟨*package⟩
```

```
1692 ⟨@@=draw⟩
```

This sub-module covers more-or-less the same ideas as pgfcoretransformations.code.tex. At present, equivalents of the following are currently absent:

- \pgfgettransform, \pgfgettransformentries: Awaiting use cases.

- \pgftransformlineattime, \pgftransformarcaxesattime, \pgftransformcurveattime: Need to look at the use cases for these to fully understand them.

- \pgftransformarrow: Likely to be done when other arrow functions are added.

- \pgftransformationadjustments: Used mainly by CircuiTi*k*Z although also for shapes, likely needs more use cases before addressing.

- \pgflowlevelsynccm, \pgflowlevel: Likely to be added when use cases are encountered in other parts of the code.

- \pgfviewboxscope: Seems very speicalied, need to understand the requirements here.

\l__draw_matrix_active_bool   An internal flag to avoid redundant calculations.

```
1693 \bool_new:N \l__draw_matrix_active_bool
```

(*End definition for* \l__draw_matrix_active_bool.)

\l__draw_matrix_a_fp   The active matrix and shifts.
\l__draw_matrix_b_fp
\l__draw_matrix_c_fp   1694 \fp_new:N \l__draw_matrix_a_fp
\l__draw_xshift_dim   1695 \fp_new:N \l__draw_matrix_b_fp
\l__draw_yshift_dim   1696 \fp_new:N \l__draw_matrix_c_fp
                       1697 \fp_new:N \l__draw_matrix_d_fp
                       1698 \dim_new:N \l__draw_xshift_dim
                       1699 \dim_new:N \l__draw_yshift_dim

(*End definition for* \l__draw_matrix_a_fp *and others.*)

\draw_transform_matrix_reset: Fast resetting.
\draw_transform_shift_reset:
                       1700 \cs_new_protected:Npn \draw_transform_matrix_reset:
                       1701   {
                       1702   \fp_set:Nn \l__draw_matrix_a_fp { 1 }
                       1703   \fp_zero:N \l__draw_matrix_b_fp
                       1704   \fp_zero:N \l__draw_matrix_c_fp
                       1705   \fp_set:Nn \l__draw_matrix_d_fp { 1 }
                       1706   }
                       1707 \cs_new_protected:Npn \draw_transform_shift_reset:
                       1708   {
                       1709   \dim_zero:N \l__draw_xshift_dim
                       1710   \dim_zero:N \l__draw_yshift_dim
                       1711   }
                       1712 \draw_transform_matrix_reset:
                       1713 \draw_transform_shift_reset:

(*End definition for* \draw_transform_matrix_reset: *and* \draw_transform_shift_reset:. *These functions are documented on page* **??**.)

\draw_transform_matrix_absolute:nnnn   Setting the transform matrix is straight-forward, with just a bit of expansion to sort out.
\draw_transform_shift_absolute:n   With the mechanism active, the identity matrix is set.
__draw_transform_shift_absolute:nn
                       1714 \cs_new_protected:Npn \draw_transform_matrix_absolute:nnnn #1#2#3#4
                       1715   {
                       1716   \fp_set:Nn \l__draw_matrix_a_fp {#1}
                       1717   \fp_set:Nn \l__draw_matrix_b_fp {#2}
                       1718   \fp_set:Nn \l__draw_matrix_c_fp {#3}
                       1719   \fp_set:Nn \l__draw_matrix_d_fp {#4}
                       1720   \bool_lazy_all:nTF
                       1721   {
                       1722   { \fp_compare_p:nNn \l__draw_matrix_a_fp = \c_one_fp }
                       1723   { \fp_compare_p:nNn \l__draw_matrix_b_fp = \c_zero_fp }
                       1724   { \fp_compare_p:nNn \l__draw_matrix_c_fp = \c_zero_fp }
                       1725   { \fp_compare_p:nNn \l__draw_matrix_d_fp = \c_one_fp }
                       1726   }
                       1727   { \bool_set_false:N \l__draw_matrix_active_bool }
                       1728   { \bool_set_true:N \l__draw_matrix_active_bool }
                       1729   }
                       1730 \cs_new_protected:Npn \draw_transform_shift_absolute:n #1
                       1731   {
                       1732   __draw_point_process:nn
                       1733   { __draw_transform_shift_absolute:nn } {#1}
                       1734   }

```
1735 \cs_new_protected:Npn \__draw_transform_shift_absolute:nn #1#2
1736   {
1737     \dim_set:Nn \l__draw_xshift_dim {#1}
1738     \dim_set:Nn \l__draw_yshift_dim {#2}
1739   }
```

(*End definition for* \draw_transform_matrix_absolute:nnnn *,* \draw_transform_shift_absolute:n *, and*
__draw_transform_shift_absolute:nn*. These functions are documented on page* **??**.*)

\draw_transform_matrix:nnnn   Much the same story for adding to an existing matrix, with a bit of pre-expansion so
__draw_transform:nnnn   that the calculation uses "frozen" values.
\draw_transform_shift:n
__draw_transform_shift:nn

```
1740 \cs_new_protected:Npn \draw_transform_matrix:nnnn #1#2#3#4
1741   {
1742     \use:x
1743       {
1744         \__draw_transform:nnnn
1745           { \fp_eval:n {#1} }
1746           { \fp_eval:n {#2} }
1747           { \fp_eval:n {#3} }
1748           { \fp_eval:n {#4} }
1749       }
1750   }
1751 \cs_new_protected:Npn \__draw_transform:nnnn #1#2#3#4
1752   {
1753     \use:x
1754       {
1755         \draw_transform_matrix_absolute:nnnn
1756           { #1 * \l__draw_matrix_a_fp + #2 * \l__draw_matrix_c_fp }
1757           { #1 * \l__draw_matrix_b_fp + #2 * \l__draw_matrix_d_fp }
1758           { #3 * \l__draw_matrix_a_fp + #4 * \l__draw_matrix_c_fp }
1759           { #3 * \l__draw_matrix_b_fp + #4 * \l__draw_matrix_d_fp }
1760       }
1761   }
1762 \cs_new_protected:Npn \draw_transform_shift:n #1
1763   {
1764     \__draw_point_process:nn
1765       { \__draw_transform_shift:nn } {#1}
1766   }
1767 \cs_new_protected:Npn \__draw_transform_shift:nn #1#2
1768   {
1769     \dim_set:Nn \l__draw_xshift_dim { \l__draw_xshift_dim + #1 }
1770     \dim_set:Nn \l__draw_yshift_dim { \l__draw_yshift_dim + #2 }
1771   }
```

(*End definition for* \draw_transform_matrix:nnnn *and others. These functions are documented on page*
??.*)

\draw_transform_matrix_invert: Standard mathematics: calculate the inverse matrix and use that, then undo the shifts.
__draw_transform_invert:n
__draw_transform_invert:f
\draw_transform_shift_invert:

```
1772 \cs_new_protected:Npn \draw_transform_matrix_invert:
1773   {
1774     \bool_if:NT \l__draw_matrix_active_bool
1775       {
1776         \__draw_transform_invert:f
1777           {
1778             \fp_eval:n
```

48

```
1779                    {
1780                       1 /
1781                          (
1782                               \l__draw_matrix_a_fp * \l__draw_matrix_d_fp
1783                            - \l__draw_matrix_b_fp * \l__draw_matrix_c_fp
1784                          )
1785                    }
1786                 }
1787          }
1788    }
1789 \cs_new_protected:Npn \__draw_transform_invert:n #1
1790    {
1791      \fp_set:Nn \l__draw_matrix_a_fp
1792        { \l__draw_matrix_d_fp * #1 }
1793      \fp_set:Nn \l__draw_matrix_b_fp
1794        { -\l__draw_matrix_b_fp * #1 }
1795      \fp_set:Nn \l__draw_matrix_c_fp
1796        { -\l__draw_matrix_c_fp * #1 }
1797      \fp_set:Nn \l__draw_matrix_d_fp
1798        { \l__draw_matrix_a_fp * #1 }
1799    }
1800 \cs_generate_variant:Nn \__draw_transform_invert:n { f }
1801 \cs_new_protected:Npn \draw_transform_shift_invert:
1802    {
1803      \dim_set:Nn \l__draw_xshift_dim { -\l__draw_xshift_dim }
1804      \dim_set:Nn \l__draw_yshift_dim { -\l__draw_yshift_dim }
1805    }
```

(*End definition for* \draw_transform_matrix_invert:, __draw_transform_invert:n, *and* \draw_-
transform_shift_invert:. *These functions are documented on page* **??**.)

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

```
1806 \cs_new_protected:Npn \draw_transform_triangle:nnn #1#2#3
1807    {
1808      \__draw_point_process:nnn
1809        {
1810          \__draw_point_process:nn
1811            { \__draw_tranform_triangle:nnnnnn }
1812            {#1}
1813        }
1814        {#2} {#3}
1815    }
1816 \cs_new_protected:Npn \__draw_tranform_triangle:nnnnnn #1#2#3#4#5#6
1817    {
1818      \use:x
1819        {
1820          \draw_transform_matrix_absolute:nnnn
1821            { #3 - #1 }
1822            { #4 - #2 }
1823            { #5 - #1 }
1824            { #6 - #2 }
1825          \draw_transform_shift_absolute:n { #1 , #2 }
1826        }
1827    }
```

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

`\draw_transform_scale:n` Lots of shortcuts.
`\draw_transform_xscale:n`
`\draw_transform_yscale:n`
`\draw_transform_xshift:n`
`\draw_transform_yshift:n`
`\draw_transform_xslant:n`
`\draw_transform_yslant:n`

```
1828 \cs_new_protected:Npn \draw_transform_scale:n #1
1829   { \draw_transform_matrix:nnnn { #1 } { 0 } { 0 } { #1 } }
1830 \cs_new_protected:Npn \draw_transform_xscale:n #1
1831   { \draw_transform_matrix:nnnn { #1 } { 0 } { 0 } { 1 } }
1832 \cs_new_protected:Npn \draw_transform_yscale:n #1
1833   { \draw_transform_matrix:nnnn { 1 } { 0 } { 0 } { #1 } }
1834 \cs_new_protected:Npn \draw_transform_xshift:n #1
1835   { \draw_transform_shift:n { #1 , 0pt } }
1836 \cs_new_protected:Npn \draw_transform_yshift:n #1
1837   { \draw_transform_shift:n { 0pt , #1 } }
1838 \cs_new_protected:Npn \draw_transform_xslant:n #1
1839   { \draw_transform_matrix:nnnn { 1 } { 0 } { #1 } { 1 } }
1840 \cs_new_protected:Npn \draw_transform_yslant:n #1
1841   { \draw_transform_matrix:nnnn { 1 } { #1 } { 0 } { 1 } }
```

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

`\draw_transform_rotate:n` Slightly more involved: evaluate the angle only once, and the sine and cosine only once.
`\__draw_transform_rotate:n`
`\__draw_transform_rotate:f`
`\__draw_transform_rotate:nn`
`\__draw_transform_rotate:ff`

```
1842 \cs_new_protected:Npn \draw_transform_rotate:n #1
1843   { \__draw_transform_rotate:f { \fp_eval:n {#1} } }
1844 \cs_new_protected:Npn \__draw_transform_rotate:n #1
1845   {
1846     \__draw_transform_rotate:ff
1847       { \fp_eval:n { cosd(#1) } }
1848       { \fp_eval:n { sind(#1) } }
1849   }
1850 \cs_generate_variant:Nn \__draw_transform_rotate:n { f }
1851 \cs_new_protected:Npn \__draw_transform_rotate:nn #1#2
1852   { \draw_transform_matrix:nnnn {#1} {#2} { -#2 } { #1 } }
1853 \cs_generate_variant:Nn \__draw_transform_rotate:nn { ff }
```

(*End definition for* `\draw_transform_rotate:n`*,* `\__draw_transform_rotate:n`*, and* `\__draw_transform_-rotate:nn`*. This function is documented on page* **??**.)

```
1854 ⟨/package⟩
```

50

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.

53

58