

File I

Implementation

1 l3draw implementation

```

1 <*package>
2 <@@=draw>
3 \ProvidesExplPackage{l3draw}{2021-08-04}{ }
4 {L3 Experimental core drawing support}

```

1.1 Internal auxiliaries

`\s__draw_mark` Internal scan marks.

```

\s__draw_stop 5 \scan_new:N \s__draw_mark
6 \scan_new:N \s__draw_stop

```

(End definition for \s__draw_mark and \s__draw_stop.)

`\q__draw_recursion_tail` Internal recursion quarks.

```

\s__draw_recursion_stop 7 \quark_new:N \q__draw_recursion_tail
8 \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.

```

9 \__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!

```

10 </package>

```

2 l3draw-boxes implementation

```

11 <*package>

```

```

12 <@@=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`

```

13 \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-row matrix. The process is split into two so that coffins are also supported.

```

14 \cs_new_protected:Npn \draw_box_use:N #1
15 {

```

```

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

```

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

`\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.

```

60 \cs_new_protected:Npn \draw_coffin_use:Nnn #1#2#3
61 {
62     \group_begin:
63     \hbox_set:Nn \l__draw_tmp_box

```

```

64         { \coffin_typeset:Nnnnn #1 {#2} {#3} { Opt } { Opt } }
65     \__draw_box_use:Nnnnn \l__draw_tmp_box
66     { \box_wd:N \l__draw_tmp_box - \coffin_wd:N #1 }
67     { -\box_dp:N \l__draw_tmp_box }
68     { \box_wd:N \l__draw_tmp_box }
69     { \box_ht:N \l__draw_tmp_box }
70 \group_end:
71 }

```

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

```

72 \</package>

```

3 l3draw-layers implementation

```

73 \*package>

```

```

74 \@@=draw>

```

3.1 User interface

`\draw_layer_new:n`

```

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

```

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

`\l__draw_layer_tl` The name of the current layer: we start off with `main`.

```

84 \tl_new:N \l__draw_layer_tl
85 \tl_set:Nn \l__draw_layer_tl { main }

```

(End definition for `\l__draw_layer_tl`.)

`\l__draw_layer_close_bool` Used to track if a layer needs to be closed.

```

86 \bool_new:N \l__draw_layer_close_bool

```

(End definition for `\l__draw_layer_close_bool`.)

`\l_draw_layers_clist` The list of layers to use starts off with just the `main` one.

```

\g__draw_layers_clist
87 \clist_new:N \l_draw_layers_clist
88 \clist_set:Nn \l_draw_layers_clist { main }
89 \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 grouping to get everything in order. Layers have to be zero width, so they get set as we go along.

```

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

```

(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.

```

122 \cs_new_protected:Npn \__draw_layers_insert:
123 {
124   \clist_map_inline:Nn \l_draw_layers_clist
125   {
126     \str_if_eq:nnTF {##1} { main }
127     {
128       \box_set_wd:Nn \l__draw_layer_main_box { Opt }
129       \box_use_drop:N \l__draw_layer_main_box
130     }
131     {
132       \__draw_backend_scope_begin:
133       \box_gset_wd:cn { g__draw_layer_ ##1 _box } { Opt }

```

```

134         \box_use_drop:c { g__draw_layer_ ##1 _box }
135         \__draw_backend_scope_end:
136     }
137 }
138 }

```

(End definition for __draw_layers_insert:.)

__draw_layers_save: Simple save/restore functions.

```

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

```

(End definition for __draw_layers_save: and __draw_layers_restore:.)

```

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

```

4 l3draw-paths implementation

```

173 <*package>
174 <@@=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 175 \tl_new:N \l__draw_path_tmp_tl
\l__draw_path_tmpb_fp 176 \fp_new:N \l__draw_path_tmpa_fp
177 \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 178 \dim_new:N \g__draw_path_lastx_dim
179 \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 180 \dim_new:N \g__draw_path_xmax_dim
\g__draw_path_ymax_dim 181 \dim_new:N \g__draw_path_xmin_dim
\g__draw_path_ymin_dim 182 \dim_new:N \g__draw_path_ymax_dim
183 \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 is always true: that will change as more complex functionality is added.)

```

\__draw_path_reset_limits: 184 \cs_new_protected:Npn \__draw_path_update_limits:nn #1#2
185 {
186   \dim_gset:Nn \g__draw_path_xmax_dim
187   { \dim_max:nn \g__draw_path_xmax_dim {#1} }
188   \dim_gset:Nn \g__draw_path_xmin_dim
189   { \dim_min:nn \g__draw_path_xmin_dim {#1} }
190   \dim_gset:Nn \g__draw_path_ymax_dim
191   { \dim_max:nn \g__draw_path_ymax_dim {#2} }
192   \dim_gset:Nn \g__draw_path_ymin_dim
193   { \dim_min:nn \g__draw_path_ymin_dim {#2} }
194   \bool_if:NT \l_draw_bb_update_bool
195   {
196     \dim_gset:Nn \g__draw_xmax_dim
197     { \dim_max:nn \g__draw_xmax_dim {#1} }
198     \dim_gset:Nn \g__draw_xmin_dim
199     { \dim_min:nn \g__draw_xmin_dim {#1} }

```

```

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

```

(End definition for __draw_path_update_limits:nn and __draw_path_reset_limits:.)

__draw_path_update_last:nn A simple auxiliary to avoid repetition.

```

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

```

(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
218 \dim_new:N \l__draw_corner_xarc_dim
219 \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.

```

220 \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.

```

221 \cs_new_protected:Npn \draw_path_corner_arc:nn #1#2
222 {
223     \dim_set:Nn \l__draw_corner_xarc_dim {#1}
224     \dim_set:Nn \l__draw_corner_yarc_dim {#2}
225     \bool_lazy_and:nnTF
226     { \dim_compare_p:nNn \l__draw_corner_xarc_dim = { 0pt } }
227     { \dim_compare_p:nNn \l__draw_corner_yarc_dim = { 0pt } }
228     { \bool_set_false:N \l__draw_corner_arc_bool }
229     { \bool_set_true:N \l__draw_corner_arc_bool }
230 }

```

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

```

(End definition for __draw_path_mark_corner:.)

4.3 Basic path constructions

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

```

\__draw_path_moveto:nn 240 \cs_new_protected:Npn \draw_path_moveto:n #1
\__draw_path_lineto:nn 241 {
\draw_path_curveto:nnn 242   \__draw_point_process:nn
\__draw_path_curveto:nnnnnn 243   { \__draw_path_moveto:nn }
244   { \draw_point_transform:n {#1} }
245 }
246 \cs_new_protected:Npn \__draw_path_moveto:nn #1#2
247 {
248   \__draw_path_update_limits:nn {#1} {#2}
249   \__draw_softpath_moveto:nn {#1} {#2}
250   \__draw_path_update_last:nn {#1} {#2}
251 }
252 \cs_new_protected:Npn \draw_path_lineto:n #1
253 {
254   \__draw_point_process:nn
255   { \__draw_path_lineto:nn }
256   { \draw_point_transform:n {#1} }
257 }
258 \cs_new_protected:Npn \__draw_path_lineto:nn #1#2
259 {
260   \__draw_path_mark_corner:
261   \__draw_path_update_limits:nn {#1} {#2}
262   \__draw_softpath_lineto:nn {#1} {#2}
263   \__draw_path_update_last:nn {#1} {#2}
264 }
265 \cs_new_protected:Npn \draw_path_curveto:nnn #1#2#3
266 {
267   \__draw_point_process:nnnn
268   {
269     \__draw_path_mark_corner:
270     \__draw_path_curveto:nnnnnn
271   }
272   { \draw_point_transform:n {#1} }
273   { \draw_point_transform:n {#2} }
274   { \draw_point_transform:n {#3} }

```



```

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

```

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

`\draw_path_close:` A simple wrapper.

```

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

```

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

```

(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
\__draw_path_curveto:nnnn
\c__draw_path_curveto_a_fp
\c__draw_path_curveto_b_fp

```

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

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

and

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

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

```

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

<pre> \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 \l__draw_path_arc_start_fp \c__draw_path_arc_90_fp \c__draw_path_arc_60_fp </pre>	<p>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°.</p> <pre> 345 \cs_new_protected:Npn \draw_path_arc:nnn #1#2#3 346 { \draw_path_arc:nnnn {#1} {#2} {#3} {#3} } 347 \cs_new_protected:Npn \draw_path_arc:nnnn #1#2#3#4 348 { 349 \use:x </pre>
--	---

```

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

```

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

```

394 \cs_new_protected:Npn \__draw_path_arc_auxi:nnnnNnn #1#2#3#4#5#6#7
395 {
396     \use:x
397     {
398         \__draw_path_arc_auxii:nnnNnnnn

```

```

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

```

421 \cs_new_protected:Npn \__draw_path_arc_auxii:nnnnnnn #1#2#3#4#5#6#7#8
422 {
423     \tl_clear:N \l__draw_path_tmp_tl
424     \__draw_point_process:nn
425     { \__draw_path_arc_auxiii:nn }
426     {
427         \__draw_point_transform_noshift:n
428         { \draw_point_polar:nnn {#7} {#8} { #1 #4 90 } }
429     }
430     \__draw_point_process:nnn
431     { \__draw_path_arc_auxiv:nnnn }
432     {
433         \draw_point_transform:n
434         { \draw_point_polar:nnn {#5} {#6} {#1} }
435     }
436     {
437         \draw_point_transform:n
438         { \draw_point_polar:nnn {#5} {#6} {#2} }
439     }
440     \__draw_point_process:nn
441     { \__draw_path_arc_auxv:nn }
442     {
443         \__draw_point_transform_noshift:n
444         { \draw_point_polar:nnn {#7} {#8} { #2 #4 -90 } }
445     }
446     \exp_after:wN \__draw_path_curveto:nnnnnnn \l__draw_path_tmp_tl

```

```

447 \fp_set:Nn \l__draw_path_arc_delta_fp { abs ( #2 - #3 ) }
448 \fp_set:Nn \l__draw_path_arc_start_fp {#2}
449 }

```

The first control point.

```

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

```

The end point: simple arithmetic.

```

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

```

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

```

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

```

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

```

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

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

`__draw_path_ellipse:nnnnnn`

`__draw_path_ellipse_arci:nnnnnn`

`__draw_path_ellipse_arcii:nnnnnn`

`__draw_path_ellipse_arciiii:nnnnnn`

`__draw_path_ellipse_arciv:nnnnnn`

`\c__draw_path_ellipse_fp`

```

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

```

```

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

```

552   \cs_new_protected:Npn \draw_path_circle:nn #1#2
553   { \draw_path_ellipse:nnn {#1} { #2 , Opt } { Opt , #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-by-step construction.

__draw_path_rectangle:nnnn

_draw_path_rectangle_rounded:nnnn

```

554   \cs_new_protected:Npn \draw_path_rectangle:nn #1#2
555   {
556     \__draw_point_process:nnn
557     {
558       \bool_lazy_or:nnTF
559       { \l__draw_corner_arc_bool }
560       { \l__draw_matrix_active_bool }
561       { \__draw_path_rectangle_rounded:nnnn }
562       { \__draw_path_rectangle:nnnn }
563     }
564     { \draw_point_transform:n {#1} }
565     {#2}
566   }
567   \cs_new_protected:Npn \__draw_path_rectangle:nnnn #1#2#3#4
568   {
569     \__draw_path_update_limits:nn {#1} {#2}
570     \__draw_path_update_limits:nn { #1 + #3 } { #2 + #4 }
571     \__draw_softpath_rectangle:nnnn {#1} {#2} {#3} {#4}
572     \__draw_path_update_last:nn {#1} {#2}
573   }
574   \cs_new_protected:Npn \__draw_path_rectangle_rounded:nnnn #1#2#3#4
575   {
576     \draw_path_moveto:n { #1 + #3 , #2 + #4 }
577     \draw_path_lineto:n { #1 , #2 + #4 }
578     \draw_path_lineto:n { #1 , #2 }
579     \draw_path_lineto:n { #1 + #3 , #2 }
580     \draw_path_close:
581     \draw_path_moveto:n { #1 , #2 }
582   }

```

(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`
`__draw_path_rectangle_corners:nnnn`

Another shortcut wrapper.

```
583 \cs_new_protected:Npn \draw_path_rectangle_corners:nn #1#2
584 {
585   \__draw_point_process:nnn
586   { \__draw_path_rectangle_corners:nnnnn {#1} }
587   {#1} {#2}
588 }
589 \cs_new_protected:Npn \__draw_path_rectangle_corners:nnnnn #1#2#3#4#5
590 { \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`
`__draw_path_grid_auxi:nnnnnn`
`__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`

The main complexity here is lining up the grid correctly. To keep it simple, we tidy up the argument ordering first.

```
591 \cs_new_protected:Npn \draw_path_grid:nnnn #1#2#3#4
592 {
593   \__draw_point_process:nnn
594   {
595     \__draw_path_grid_auxi:ffnnnn
596     { \dim_eval:n { \dim_abs:n {#1} } } }
597     { \dim_eval:n { \dim_abs:n {#2} } } }
598   }
599   {#3} {#4}
600 }
601 \cs_new_protected:Npn \__draw_path_grid_auxi:nnnnnn #1#2#3#4#5#6
602 {
603   \dim_compare:nNnTF {#3} > {#5}
604   { \__draw_path_grid_auxii:nnnnnn {#1} {#2} {#5} {#4} {#3} {#6} }
605   { \__draw_path_grid_auxii:nnnnnn {#1} {#2} {#3} {#4} {#5} {#6} }
606 }
607 \cs_generate_variant:Nn \__draw_path_grid_auxi:nnnnnn { ff }
608 \cs_new_protected:Npn \__draw_path_grid_auxii:nnnnnn #1#2#3#4#5#6
609 {
610   \dim_compare:nNnTF {#4} > {#6}
611   { \__draw_path_grid_auxiii:nnnnnn {#1} {#2} {#3} {#6} {#5} {#4} }
612   { \__draw_path_grid_auxiii:nnnnnn {#1} {#2} {#3} {#4} {#5} {#6} }
613 }
614 \cs_new_protected:Npn \__draw_path_grid_auxiii:nnnnnn #1#2#3#4#5#6
615 {
616   \__draw_path_grid_auxiv:ffnnnnnn
617   { \fp_to_dim:n { #1 * trunc(#3/(#1)) } } }
618   { \fp_to_dim:n { #2 * trunc(#4/(#2)) } } }
619   {#1} {#2} {#3} {#4} {#5} {#6}
620 }
621 \cs_new_protected:Npn \__draw_path_grid_auxiv:nnnnnnnn #1#2#3#4#5#6#7#8
622 {
623   \dim_step_inline:nnnn
624   {#1}
```



```

625     {#3}
626     {#7}
627     {
628         \draw_path_moveto:n { ##1 , #6 }
629         \draw_path_lineto:n { ##1 , #8 }
630     }
631     \dim_step_inline:nnnn
632     {#2}
633     {#4}
634     {#8}
635     {
636         \draw_path_moveto:n { #5 , ##1 }
637         \draw_path_lineto:n { #7 , ##1 }
638     }
639 }
640 \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

Actions to pass to the driver.

```

641 \bool_new:N \l__draw_path_use_clip_bool
642 \bool_new:N \l__draw_path_use_fill_bool
643 \bool_new:N \l__draw_path_use_stroke_bool

```

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

Actions handled at the macro layer.

```

644 \bool_new:N \l__draw_path_use_bb_bool
645 \bool_new:N \l__draw_path_use_clear_bool

```

(End definition for \l__draw_path_use_bb_bool and \l__draw_path_use_clear_bool.)

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

```

646 \cs_new_protected:Npn \draw_path_use:n #1
647 {
648     \tl_if_blank:nF {#1}
649     { \__draw_path_use:n {#1} }
650 }
651 \cs_new_protected:Npn \draw_path_use_clear:n #1
652 {
653     \bool_lazy_or:nnTF
654     { \tl_if_blank_p:n {#1} }
655     { \str_if_eq_p:nn {#1} { clear } }
656     {
657         \__draw_softpath_clear:
658         \__draw_path_reset_limits:
659     }
660     { \__draw_path_use:n { #1 , clear } }
661 }

```

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).

```

662 \cs_new_protected:Npn \__draw_path_use:n #1
663 {
664   \bool_set_false:N \l__draw_path_use_clip_bool
665   \bool_set_false:N \l__draw_path_use_fill_bool
666   \bool_set_false:N \l__draw_path_use_stroke_bool
667   \clist_map_inline:nn {#1}
668   {
669     \cs_if_exist:cTF { l__draw_path_use_ ##1 _ bool }
670     { \bool_set_true:c { l__draw_path_use_ ##1 _ bool } }
671     {
672       \cs_if_exist_use:cF { __draw_path_use_action_ ##1 : }
673       { \msg_error:nnn { draw } { invalid-path-action } {##1} }
674     }
675   }
676   \__draw_softpath_round_corners:
677   \bool_lazy_and:nnT
678   { \l_draw_bb_update_bool }
679   { \l__draw_path_use_stroke_bool }
680   { \__draw_path_use_stroke_bb: }
681   \__draw_softpath_use:
682   \bool_if:NT \l__draw_path_use_clip_bool
683   {
684     \__draw_backend_clip:
685     \bool_set_false:N \l_draw_bb_update_bool
686     \bool_lazy_or:nnF
687     { \l__draw_path_use_fill_bool }
688     { \l__draw_path_use_stroke_bool }
689     { \__draw_backend_discardpath: }
690   }
691   \bool_lazy_or:nnT
692   { \l__draw_path_use_fill_bool }
693   { \l__draw_path_use_stroke_bool }
694   {
695     \use:c
696     {
697       __draw_backend_
698       \bool_if:NT \l__draw_path_use_fill_bool { fill }
699       \bool_if:NT \l__draw_path_use_stroke_bool { stroke }
700       :
701     }
702   }
703   \bool_if:NT \l__draw_path_use_clear_bool
704   { \__draw_softpath_clear: }
705 }
706 \cs_new_protected:Npn \__draw_path_use_action_draw:
707 {
708   \bool_set_true:N \l__draw_path_use_stroke_bool
709 }
710 \cs_new_protected:Npn \__draw_path_use_action_fillstroke:
711 {
712   \bool_set_true:N \l__draw_path_use_fill_bool

```

```

713 \bool_set_true:N \l__draw_path_use_stroke_bool
714 }

```

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.

```

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

```

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

4.9 Scoping paths

`\l__draw_path_lastx_dim` 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).

```

\l__draw_path_xmax_dim
\l__draw_path_xmin_dim
\l__draw_path_ymax_dim
\l__draw_path_ymin_dim
\l__draw_softpath_corners_bool
737 \dim_new:N \l__draw_path_lastx_dim
738 \dim_new:N \l__draw_path_lasty_dim
739 \dim_new:N \l__draw_path_xmax_dim
740 \dim_new:N \l__draw_path_xmin_dim
741 \dim_new:N \l__draw_path_ymax_dim
742 \dim_new:N \l__draw_path_ymin_dim
743 \dim_new:N \l__draw_softpath_lastx_dim
744 \dim_new:N \l__draw_softpath_lasty_dim
745 \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 hold of.

```

746 \cs_new_protected:Npn \draw_path_scope_begin:
747 {
748   \group_begin:
749   \dim_set_eq:NN \l__draw_path_lastx_dim \g__draw_path_lastx_dim
750   \dim_set_eq:NN \l__draw_path_lasty_dim \g__draw_path_lasty_dim

```

```

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

```

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

```

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

```

5 l3draw-points implementation

```

788 <*package>
789 <@@=draw>

```

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

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

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

```

\__draw_point_process:nn
  \__draw_point_process_auxi:nn
  \__draw_point_process_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
790 \cs_new:Npn \__draw_point_process:nn #1#2
791 {
792   \exp_args:Nf \__draw_point_process_auxi:nn
793   { \draw_point:n {#2} }
794   {#1}
795 }
796 \cs_new:Npn \__draw_point_process_auxi:nn #1#2
797 { \__draw_point_process_auxii:nw {#2} #1 \s__draw_stop }
798 \cs_new:Npn \__draw_point_process_auxii:nw #1 #2 , #3 \s__draw_stop
799 { #1 {#2} {#3} }
800 \cs_new:Npn \__draw_point_process:nnn #1#2#3
801 {
802   \exp_args:Nff \__draw_point_process_auxiii:nnn
803   { \draw_point:n {#2} }
804   { \draw_point:n {#3} }
805   {#1}
806 }
807 \cs_new:Npn \__draw_point_process_auxiii:nnn #1#2#3
808 { \__draw_point_process_auxiv:nw {#3} #1 \s__draw_mark #2 \s__draw_stop }
809 \cs_new:Npn \__draw_point_process_auxiv:nw #1 #2 , #3 \s__draw_mark #4 , #5 \s__draw_stop
810 { #1 {#2} {#3} {#4} {#5} }
811 \cs_new:Npn \__draw_point_process:nnnn #1#2#3#4
812 {
813   \exp_args:Nfff \__draw_point_process_auxv:nnnn
814   { \draw_point:n {#2} }
815   { \draw_point:n {#3} }
816   { \draw_point:n {#4} }
817   {#1}
818 }

```

```

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

```

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

5.2 Basic points

<code>\draw_point:n</code> <code>__draw_point_to_dim:n</code> <code>__draw_point_to_dim:f</code> <code>__draw_point_to_dim:w</code>	Co-ordinates are always returned as two dimensions. <pre> 841 \cs_new:Npn \draw_point:n #1 842 { __draw_point_to_dim:f { \fp_eval:n {#1} } } 843 \cs_new:Npn __draw_point_to_dim:n #1 844 { __draw_point_to_dim:w #1 } 845 \cs_generate_variant:Nn __draw_point_to_dim:n { f } 846 \cs_new:Npn __draw_point_to_dim:w (#1 , ~ #2) { #1pt , #2pt } </pre>
---	---

5.3 Polar co-ordinates

<code>\draw_point_polar:nn</code> <code>\draw_point_polar:nnn</code> <code>__draw_draw_polar:nnn</code> <code>__draw_draw_polar:fnn</code>	Polar co-ordinates may have either one or two lengths, so there is a need to do a simple split before the calculation. As the angle gets used twice, save on any expression evaluation there and force expansion. <pre> 847 \cs_new:Npn \draw_point_polar:nn #1#2 848 { \draw_point_polar:nnn {#1} {#1} {#2} } 849 \cs_new:Npn \draw_point_polar:nnn #1#2#3 850 { __draw_draw_polar:fnn { \fp_eval:n {#3} } {#1} {#2} } 851 \cs_new:Npn __draw_draw_polar:nnn #1#2#3 852 { \draw_point:n { cosd(#1) * (#2) , sind(#1) * (#3) } } 853 \cs_generate_variant:Nn __draw_draw_polar:nnn { f } </pre>
---	---

5.4 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.

```

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

```

5.5 Intersection calculations

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

$$P_x = \frac{(x_1y_2 - y_1x_2)(x_3 - x_4) - (x_3y_4 - y_3x_4)(x_1 - x_2)}{(x_1 - x_2)(y_3 - y_4) - (y_1 - y_2)(x_3 - x_4)}$$

and

$$P_y = \frac{(x_1y_2 - y_1x_2)(y_3 - y_4) - (x_3y_4 - y_3x_4)(y_1 - y_2)}{(x_1 - x_2)(y_3 - y_4) - (y_1 - y_2)(x_3 - x_4)}$$

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.)

```

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

```

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

```

#1 x1
#2 y1
#3 x2
#4 y2

```

#5 x_3

#6 y_3

#7 x_4

#8 y_4

so now just have to do all of the calculation.

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

```
\draw_point_intersect_circles:nnnnn
__draw_point_intersect_circles_auxi:nnnnnnnn
__draw_point_intersect_circles_auxii:nnnnnnnn
__draw_point_intersect_circles_auxiii:ffnnnnnn
draw_point_intersect_circles_auxiii:ffnnnnnn
draw_point_intersect_circles_auxiv:nnnnnnnnnn
draw_point_intersect_circles_auxiv:fnnnnnnnnn
draw_point_intersect_circles_auxv:nnnnnnnnnnnn
draw_point_intersect_circles_auxv:ffnnnnnnnnnn
draw_point_intersect_circles_auxvi:nnnnnnnnnn
draw_point_intersect_circles_auxvi:fnnnnnnnnn
draw_point_intersect_circles_auxvii:nnnnnnnn
draw_point_intersect_circles_auxvii:ffnnnnnn
```

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

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

in either

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

or

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

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

```

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

```

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.

```

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

```

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

```

```

928 }
929 \cs_generate_variant:Nn \__draw_point_intersect_circles_auxiv:nnnnnnnn { f }
930 \cs_new:Npn \__draw_point_intersect_circles_auxv:nnnnnnnn #1#2#3#4#5#6#7#8#9
931 {
932   \__draw_point_intersect_circles_auxvi:fnnnnnnnn
933   { \fp_eval:n { 0.5 * #1 * ( #2 + #3 * #3 - #6 * #6 ) } }
934   {#1} {#2} {#4} {#5} {#7} {#8} {#9}
935 }
936 \cs_generate_variant:Nn \__draw_point_intersect_circles_auxv:nnnnnnnn { 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.

```

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

```

5.6 Interpolation on a line (vector) or arc

Simple maths after expansion.

```

\draw_point_interpolate_line:nnn
\__draw_point_interpolate_line_aux:nnnnn
\__draw_point_interpolate_line_aux:fnnnn
\__draw_point_interpolate_line_aux:nnnnnn
\__draw_point_interpolate_line_aux:fnnnnn
952 \cs_new:Npn \draw_point_interpolate_line:nnn #1#2#3
953 {
954   \__draw_point_process:nnn
955   { \__draw_point_interpolate_line_aux:fnnnn { \fp_eval:n {#1} } }
956   {#2} {#3}
957 }

```

```

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

```

\draw_point_interpolate_distance:nnn
\_draw_point_interpolate_distance:nnnnn
\_draw_point_interpolate_distance:nnnnnn
\_draw_point_interpolate_distance:fnnnnn

```

```

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

```

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

```

\draw_point_interpolate_arcaxes:nnnnnn
aw_point_interpolate_arcaxes_auxi:nnnnnnnnnn
w_point_interpolate_arcaxes_auxii:nnnnnnnnnn
w_point_interpolate_arcaxes_auxii:fnnnnnnnnnn
aw_point_interpolate_arcaxes_auxiii:nnnnnnnnnn
aw_point_interpolate_arcaxes_auxiii:fnnnnnnnnnn
aw_point_interpolate_arcaxes_auxiv:nnnnnnnnnn
aw_point_interpolate_arcaxes_auxiv:ffnnnnnnnn

```

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.

```

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

```

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

```

#1 p
#2  $\theta_1$ 
#3  $\theta_2$ 

```

#4 x_c

#5 y_c

#6 x_{a1}

#7 y_{a1}

#8 x_{a2}

#9 y_{a2}

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

```

996 \cs_new:Npn \__draw_point_interpolate_arcaxes_auxii:nnnnnnnnn #1#2#3#4#5#6#7#8#9
997 {
998   \__draw_point_interpolate_arcaxes_auxiii:fnnnnnnn
999   { \fp_eval:n { #1 * (#3) + ( 1 - #1 ) * (#2) } }
1000   {#4} {#5} {#6} {#7} {#8} {#9}
1001 }
1002 \cs_generate_variant:Nn \__draw_point_interpolate_arcaxes_auxii:nnnnnnnnn { f }
1003 \cs_new:Npn \__draw_point_interpolate_arcaxes_auxiii:nnnnnnn #1#2#3#4#5#6#7
1004 {
1005   \__draw_point_interpolate_arcaxes_auxiv:ffnnnnnnn
1006   { \fp_eval:n { cosd (#1) } }
1007   { \fp_eval:n { sind (#1) } }
1008   {#2} {#3} {#4} {#5} {#6} {#7}
1009 }
1010 \cs_generate_variant:Nn \__draw_point_interpolate_arcaxes_auxiii:nnnnnnn { f }
1011 \cs_new:Npn \__draw_point_interpolate_arcaxes_auxiv:nnnnnnnnn #1#2#3#4#5#6#7#8
1012 {
1013   \draw_point:n
1014   { #3 + #1 * #5 + #2 * #7 , #4 + #1 * #6 + #2 * #8 }
1015 }
1016 \cs_generate_variant:Nn \__draw_point_interpolate_arcaxes_auxiv:nnnnnnnnn { 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:nnnnnnnnn
draw_point_interpolate_curve_auxii:nnnnnnnnn
draw_point_interpolate_curve_auxiii:fnnnnnnnnn
\draw_point_interpolate_curve_auxiiii:nnnnnnn
\draw_point_interpolate_curve_auxiii:fnnnnnnn
\draw_point_interpolate_curve_auxiv:nnnnnnn
\draw_point_interpolate_curve_auxv:nnw
\draw_point_interpolate_curve_auxv:ffw
\draw_point_interpolate_curve_auxvi:n
draw_point_interpolate_curve_auxvii:nnnnnnnnn
draw_point_interpolate_curve_auxviii:nnnnnnn
draw_point_interpolate_curve_auxviii:ffnnnnn

```

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.

```

1017 \cs_new:Npn \draw_point_interpolate_curve:nnnnnn #1#2#3#4#5
1018 {
1019   \__draw_point_process:nnnnn
1020   { \__draw_point_interpolate_curve_auxi:nnnnnnnnn {#1} }
1021   {#2} {#3} {#4} {#5}
1022 }

```

```

1023 \cs_new:Npn \__draw_point_interpolate_curve_auxi:nnnnnnnnn #1#2#3#4#5#6#7#8#9
1024 {
1025   \__draw_point_interpolate_curve_auxii:fnnnnnnnnn
1026   { \fp_eval:n {#1} }
1027   {#2} {#3} {#4} {#5} {#6} {#7} {#8} {#9}
1028 }

```

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

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

In the second stage, we can drop the final point

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

and for the final stage only need one set of calculations

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

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

```

1029 \cs_new:Npn \__draw_point_interpolate_curve_auxii:nnnnnnnnn
1030   #1#2#3#4#5#6#7#8#9
1031 {
1032   \__draw_point_interpolate_curve_auxiii:fnnnnnnn
1033   { \fp_eval:n { 1 - #1 } }
1034   {#1}
1035   { {#2} {#3} } { {#4} {#5} } { {#6} {#7} } { {#8} {#9} }
1036 }
1037 \cs_generate_variant:Nn \__draw_point_interpolate_curve_auxii:nnnnnnnnn { f }
1038 % \begin{macrocode}
1039 % We need to do the first cycle, but haven't got enough arguments to keep
1040 % everything in play at once. So here we use a but of argument re-ordering
1041 % and a single auxiliary to get the job done.
1042 % \begin{macrocode}
1043 \cs_new:Npn \__draw_point_interpolate_curve_auxiii:nnnnnn #1#2#3#4#5#6
1044 {
1045   \__draw_point_interpolate_curve_auxiv:nnnnnn {#1} {#2} #3 #4
1046   \__draw_point_interpolate_curve_auxiv:nnnnnn {#1} {#2} #4 #5
1047   \__draw_point_interpolate_curve_auxiv:nnnnnn {#1} {#2} #5 #6
1048   \prg_do_nothing:
1049   \__draw_point_interpolate_curve_auxvi:n { {#1} {#2} }

```

```

1050 }
1051 \cs_generate_variant:Nn \__draw_point_interpolate_curve_auxiii:nnnnnn { f }
1052 \cs_new:Npn \__draw_point_interpolate_curve_auxiv:nnnnnn #1#2#3#4#5#6
1053 {
1054   \__draw_point_interpolate_curve_auxv:ffw
1055   { \fp_eval:n { #1 * #3 + #2 * #5 } }
1056   { \fp_eval:n { #1 * #4 + #2 * #6 } }
1057 }
1058 \cs_new:Npn \__draw_point_interpolate_curve_auxv:nnw
1059 #1#2#3 \prg_do_nothing: #4#5
1060 {
1061   #3
1062   \prg_do_nothing:
1063   #4 { #5 {#1} {#2} }
1064 }
1065 \cs_generate_variant:Nn \__draw_point_interpolate_curve_auxv:nnw { ff }
1066 % \begin{macrocode}
1067 % Get the arguments back into the right places and to the second and
1068 % third cycles directly.
1069 % \begin{macrocode}
1070 \cs_new:Npn \__draw_point_interpolate_curve_auxvi:n #1
1071 { \__draw_point_interpolate_curve_auxviii:nnnnnnnn #1 }
1072 \cs_new:Npn \__draw_point_interpolate_curve_auxvii:nnnnnnnn #1#2#3#4#5#6#7#8
1073 {
1074   \__draw_point_interpolate_curve_auxviii:ffffnn
1075   { \fp_eval:n { #1 * #5 + #2 * #3 } }
1076   { \fp_eval:n { #1 * #6 + #2 * #4 } }
1077   { \fp_eval:n { #1 * #7 + #2 * #5 } }
1078   { \fp_eval:n { #1 * #8 + #2 * #6 } }
1079   {#1} {#2}
1080 }
1081 \cs_new:Npn \__draw_point_interpolate_curve_auxviii:nnnnnn #1#2#3#4#5#6
1082 {
1083   \draw_point:n
1084   { #5 * #3 + #6 * #1 , #5 * #4 + #6 * #2 }
1085 }
1086 \cs_generate_variant:Nn \__draw_point_interpolate_curve_auxviii:nnnnnn { ffff }

```

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

5.7 Vector support

As well as co-ordinates relative to the drawing

```

\l__draw_xvec_x_dim Base vectors to map to the underlying two-dimensional drawing space.
\l__draw_xvec_y_dim
\l__draw_yvec_x_dim 1087 \dim_new:N \l__draw_xvec_x_dim
\l__draw_yvec_y_dim 1088 \dim_new:N \l__draw_xvec_y_dim
\l__draw_zvec_x_dim 1089 \dim_new:N \l__draw_yvec_x_dim
\l__draw_zvec_y_dim 1090 \dim_new:N \l__draw_yvec_y_dim
1091 \dim_new:N \l__draw_zvec_x_dim
1092 \dim_new:N \l__draw_zvec_y_dim

```

(End definition for \l__draw_xvec_x_dim and others.)

```

\draw_xvec:n Calculate the underlying position and store it.
\draw_yvec:n 1093 \cs_new_protected:Npn \draw_xvec:n #1
\draw_zvec:n 1094 { \__draw_vec:nn { x } {#1} }
\__draw_vec:nn 1095 \cs_new_protected:Npn \draw_yvec:n #1
\__draw_vec:nnn 1096 { \__draw_vec:nn { y } {#1} }
1097 \cs_new_protected:Npn \draw_zvec:n #1
1098 { \__draw_vec:nn { z } {#1} }
1099 \cs_new_protected:Npn \__draw_vec:nn #1#2
1100 {
1101   \__draw_point_process:nn { \__draw_vec:nnn {#1} } {#2}
1102 }
1103 \cs_new_protected:Npn \__draw_vec:nnn #1#2#3
1104 {
1105   \dim_set:cn { l__draw_ #1 vec_x_dim } {#2}
1106   \dim_set:cn { l__draw_ #1 vec_y_dim } {#3}
1107 }

```

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

Initialise the vectors.

```

1108 \draw_xvec:n { 1cm , 0cm }
1109 \draw_yvec:n { 0cm , 1cm }
1110 \draw_zvec:n { -0.385cm , -0.385cm }

```

```

\draw_point_vec:nn Force a single evaluation of each factor, then use these to work out the underlying point.
\__draw_point_vec:nn 1111 \cs_new:Npn \draw_point_vec:nn #1#2
\__draw_point_vec:ff 1112 { \__draw_point_vec:ff { \fp_eval:n {#1} } { \fp_eval:n {#2} } }
\draw_point_vec:nnn 1113 \cs_new:Npn \__draw_point_vec:nn #1#2
\__draw_point_vec:nnn 1114 {
\__draw_point_vec:fff 1115   \draw_point:n
1116   {
1117     #1 * \l__draw_xvec_x_dim + #2 * \l__draw_yvec_x_dim ,
1118     #1 * \l__draw_xvec_y_dim + #2 * \l__draw_yvec_y_dim
1119   }
1120 }
1121 \cs_generate_variant:Nn \__draw_point_vec:nn { ff }
1122 \cs_new:Npn \draw_point_vec:nnn #1#2#3
1123 {
1124   \__draw_point_vec:fff
1125   { \fp_eval:n {#1} } { \fp_eval:n {#2} } { \fp_eval:n {#3} }
1126 }
1127 \cs_new:Npn \__draw_point_vec:nnn #1#2#3
1128 {
1129   \draw_point:n
1130   {
1131     #1 * \l__draw_xvec_x_dim
1132     + #2 * \l__draw_yvec_x_dim
1133     + #3 * \l__draw_zvec_x_dim
1134     ,
1135     #1 * \l__draw_xvec_y_dim
1136     + #2 * \l__draw_yvec_y_dim
1137     + #3 * \l__draw_zvec_y_dim
1138   }
1139 }
1140 \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 Much the same as the core polar approach.
\draw_point_vec_polar:nnn 1141 \cs_new:Npn \draw_point_vec_polar:nn #1#2
\__draw_point_vec_polar:nnn 1142 { \draw_point_vec_polar:nnn {#1} {#1} {#2} }
\__draw_point_vec_polar:fnn 1143 \cs_new:Npn \draw_point_vec_polar:nnn #1#2#3
1144 { \__draw_draw_vec_polar:fnn { \fp_eval:n {#3} } {#1} {#2} }
1145 \cs_new:Npn \__draw_draw_vec_polar:nnn #1#2#3
1146 {
1147   \draw_point:n
1148   {
1149     cosd(#1) * (#2) * \l__draw_xvec_x_dim ,
1150     sind(#1) * (#3) * \l__draw_yvec_y_dim
1151   }
1152 }
1153 \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 ??.)

5.8 Transformations

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

```

1154 \cs_new:Npn \draw_point_transform:n #1
1155 {
1156   \__draw_point_process:nn
1157   { \__draw_point_transform:nn } {#1}
1158 }
1159 \cs_new:Npn \__draw_point_transform:nn #1#2
1160 {
1161   \bool_if:NTF \l__draw_matrix_active_bool
1162   {
1163     \draw_point:n
1164     {
1165       (
1166         \l__draw_matrix_a_fp * #1
1167         + \l__draw_matrix_c_fp * #2
1168         + \l__draw_xshift_dim
1169       )
1170       ,
1171       (
1172         \l__draw_matrix_b_fp * #1
1173         + \l__draw_matrix_d_fp * #2
1174         + \l__draw_yshift_dim
1175       )
1176     }
1177   }
1178   {
1179     \draw_point:n
1180     {
1181       (#1, #2)
1182       + ( \l__draw_xshift_dim , \l__draw_yshift_dim )
1183     }

```



```

1184     }
1185 }

```

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

```

\__draw_point_transform_noshift:n
\__draw_point_transform_noshift:nn

```

A version with no shift: used for internal purposes.

```

1186 \cs_new:Npn \__draw_point_transform_noshift:n #1
1187 {
1188   \__draw_point_process:nn
1189   { \__draw_point_transform_noshift:nn } {#1}
1190 }
1191 \cs_new:Npn \__draw_point_transform_noshift:nn #1#2
1192 {
1193   \bool_if:NTF \l__draw_matrix_active_bool
1194   {
1195     \draw_point:n
1196     {
1197       (
1198         \l__draw_matrix_a_fp * #1
1199         + \l__draw_matrix_c_fp * #2
1200       )
1201       ,
1202       (
1203         \l__draw_matrix_b_fp * #1
1204         + \l__draw_matrix_d_fp * #2
1205       )
1206     }
1207   }
1208   { \draw_point:n { (#1, #2) } }
1209 }

```

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

```

1210 \endpackage

```

6 l3draw-scopes implementation

```

1211 \begin{package}

```

```

1212 \begin{draw}

```

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

- `\pgftext`: This is covered at this level by the coffin-based interface `\draw_coffin_use:Nnn`

6.1 Drawing environment

```

\g__draw_xmax_dim
\g__draw_xmin_dim
\g__draw_ymax_dim
\g__draw_ymin_dim

```

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

```

1213 \dim_new:N \g__draw_xmax_dim
1214 \dim_new:N \g__draw_xmin_dim
1215 \dim_new:N \g__draw_ymax_dim
1216 \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.

1217 `\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.

1218 `\box_new:N \l__draw_main_box`

1219 `\box_new:N \l__draw_layer_main_box`

(End definition for `\l__draw_layer_main_box`.)

`\g__draw_id_int` The drawing number.

1220 `\int_new:N \g__draw_id_int`

(End definition for `\g__draw_id_int`.)

`__draw_reset_bb:` A simple auxiliary.

1221 `\cs_new_protected:Npn __draw_reset_bb:`

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

(End definition for `__draw_reset_bb:.`)

`\draw_begin:` Drawings are created by setting them into a box, then adjusting the box before inserting 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

`\draw_end:`

1228 `\cs_new_protected:Npn \draw_begin:`

```
1229 {
1230   \group_begin:
1231   \int_gincr:N \g__draw_id_int
1232   \hbox_set:Nw \l__draw_main_box
1233   \__draw_backend_begin:
1234   \__draw_reset_bb:
1235   \__draw_path_reset_limits:
1236   \bool_set_true:N \l_draw_bb_update_bool
1237   \draw_transform_matrix_reset:
1238   \draw_transform_shift_reset:
1239   \__draw_softpath_clear:
1240   \draw_linewidth:n { \l_draw_default_linewidth_dim }
1241   \color_select:n { . }
1242   \draw_nonzero_rule:
1243   \draw_cap_but:
1244   \draw_join_miter:
1245   \draw_miterlimit:n { 10 }
1246   \draw_dash_pattern:nn { } { 0cm }
```

```

1247         \hbox_set:Nw \l__draw_layer_main_box
1248     }
1249 \cs_new_protected:Npn \draw_end:
1250 {
1251     \__draw_baseline_finalise:w
1252     \exp_args:NNNV \hbox_set_end:
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 \__draw_finalise:
1265 \box_set_wd:Nn \l__draw_main_box
1266 { \g__draw_xmax_dim - \g__draw_xmin_dim }
1267 \mode_leave_vertical:
1268 \box_use_drop:N \l__draw_main_box
1269 \group_end:
1270 }

```

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

`__draw_finalise:` Finalising the (vertical) size of the output depends on whether we have an explicit baseline or not. To allow for that, we have two functions, and the one that's used depends on whether the user has set a baseline. Notice that in contrast to `pgf` we *do* allow for a non-zero depth if the explicit baseline is above the lowest edge of the initial bounding box.

```

1271 \cs_new_protected:Npn \__draw_finalise:
1272 {
1273     \hbox_set:Nn \l__draw_main_box
1274     {
1275         \skip_horizontal:n { -\g__draw_xmin_dim }
1276         \box_move_down:nn
1277         { \g__draw_ymin_dim }
1278         { \box_use_drop:N \l__draw_main_box }
1279     }
1280     \box_set_dp:Nn \l__draw_main_box { Opt }
1281     \box_set_ht:Nn \l__draw_main_box
1282     { \g__draw_ymax_dim - \g__draw_ymin_dim }
1283 }
1284 \cs_new_protected:Npn \__draw_finalise_baseline:n #1
1285 {
1286     \hbox_set:Nn \l__draw_main_box
1287     {
1288         \skip_horizontal:n { -\g__draw_xmin_dim }
1289         \box_move_down:nn
1290         { #1 }
1291         { \box_use_drop:N \l__draw_main_box }
1292     }

```

```

1293 \box_set_dp:Nn \l__draw_main_box
1294 {
1295   \dim_max:nn
1296   { #1 - \g__draw_ymin_dim }
1297   { 0pt }
1298 }
1299 \box_set_ht:Nn \l__draw_main_box
1300 { \g__draw_ymax_dim + #1 }
1301 }

```

(End definition for `__draw_finalise:` and `__draw_finalise_baseline:n`.)

6.2 Baseline position

`\l__draw_baseline_bool` For tracking the explicit baseline and whether it is active.

```

\l__draw_baseline_dim 1302 \bool_new:N \l__draw_baseline_bool
1303 \dim_new:N \l__draw_baseline_dim

```

(End definition for `\l__draw_baseline_bool` and `\l__draw_baseline_dim`.)

`\draw_baseline:n` A simple setting of the baseline along with the flag we need to know that it is active.

```

1304 \cs_new_protected:Npn \draw_baseline:n #1
1305 {
1306   \bool_set_true:N \l__draw_baseline_bool
1307   \dim_set:Nn \l__draw_baseline_dim { \fp_to_dim:n {#1} }
1308 }

```

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

`__draw_baseline_finalise:w` Rather than use a global data structure, we can arrange to put the baseline value at the right group level with a small amount of shuffling. That happens here.

```

1309 \cs_new_protected:Npn \__draw_baseline_finalise:w #1 \__draw_finalise:
1310 {
1311   \bool_if:NTF \l__draw_baseline_bool
1312   {
1313     \use:x
1314     {
1315       \exp_not:n {#1}
1316       \__draw_finalise_baseline:n { \dim_use:N \l__draw_baseline_dim }
1317     }
1318   }
1319   { #1 \__draw_finalise: }
1320 }

```

(End definition for `__draw_baseline_finalise:w`.)

6.3 Scopes

`\l__draw_linewidth_dim` Storage for local variables.

```

\l__draw_fill_color_tl 1321 \dim_new:N \l__draw_linewidth_dim
\l__draw_stroke_color_tl 1322 \tl_new:N \l__draw_fill_color_tl
1323 \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: 1324 \cs_new_protected:Npn \draw_scope_begin:
1325 {
1326   \__draw_backend_scope_begin:
1327   \group_begin:
1328     \dim_set_eq:NN \l__draw_linewidth_dim \g__draw_linewidth_dim
1329     \draw_path_scope_begin:
1330   }
1331 \cs_new_protected:Npn \draw_scope_end:
1332 {
1333   \draw_path_scope_end:
1334   \dim_gset_eq:NN \g__draw_linewidth_dim \l__draw_linewidth_dim
1335   \group_end:
1336   \__draw_backend_scope_end:
1337 }

```

(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 1338 \dim_new:N \l__draw_xmax_dim
\l__draw_ymax_dim 1339 \dim_new:N \l__draw_xmin_dim
\l__draw_ymin_dim 1340 \dim_new:N \l__draw_ymax_dim
1341 \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: 1342 \cs_new_protected:Npn \__draw_scope_bb_begin:
1343 {
1344   \group_begin:
1345     \dim_set_eq:NN \l__draw_xmax_dim \g__draw_xmax_dim
1346     \dim_set_eq:NN \l__draw_xmin_dim \g__draw_xmin_dim
1347     \dim_set_eq:NN \l__draw_ymax_dim \g__draw_ymax_dim
1348     \dim_set_eq:NN \l__draw_ymin_dim \g__draw_ymin_dim
1349     \__draw_reset_bb:
1350   }
1351 \cs_new_protected:Npn \__draw_scope_bb_end:
1352 {
1353   \dim_gset_eq:NN \g__draw_xmax_dim \l__draw_xmax_dim
1354   \dim_gset_eq:NN \g__draw_xmin_dim \l__draw_xmin_dim
1355   \dim_gset_eq:NN \g__draw_ymax_dim \l__draw_ymax_dim
1356   \dim_gset_eq:NN \g__draw_ymin_dim \l__draw_ymin_dim
1357   \group_end:
1358 }

```

(End definition for __draw_scope_bb_begin: and __draw_scope_bb_end:.)

`\draw_suspend_begin:` Suspend all parts of a drawing.

```

\draw_suspend_end: 1359 \cs_new_protected:Npn \draw_suspend_begin:
1360 {
1361   \__draw_scope_bb_begin:
1362   \draw_path_scope_begin:
1363   \draw_transform_matrix_reset:
1364   \draw_transform_shift_reset:

```

```

1365     \__draw_layers_save:
1366   }
1367   \cs_new_protected:Npn \draw_suspend_end:
1368   {
1369     \__draw_layers_restore:
1370     \draw_path_scope_end:
1371     \__draw_scope_bb_end:
1372   }

```

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

```

1373 \</package>

```

7 l3draw-softpath implementation

```

1374 \*package>

```

```

1375 \@@=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.

```

1376 \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.

```

1377 \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.

```

1378 \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

```

1379 \cs_new_protected:Npn \__draw_softpath_add:n
1380 { \tl_build_gput_right:Nn \g__draw_softpath_main_tl }
1381 \cs_generate_variant:Nn \__draw_softpath_add:n { o, x }

```

(End definition for __draw_softpath_add:n.)

```

\__draw_softpath_use: Using and clearing is trivial.
\__draw_softpath_clear:
1382 \cs_new_protected:Npn \__draw_softpath_use:
1383 {
1384   \tl_build_get:NN \g__draw_softpath_main_tl \l__draw_softpath_internal_tl
1385   \l__draw_softpath_internal_tl
1386 }
1387 \cs_new_protected:Npn \__draw_softpath_clear:
1388 {
1389   \tl_build_gclear:N \g__draw_softpath_main_tl
1390   \bool_gset_false:N \g__draw_softpath_corners_bool
1391 }

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

```

```

\g__draw_softpath_lastx_dim For tracking the end of the path (to close it).
\g__draw_softpath_lasty_dim
1392 \dim_new:N \g__draw_softpath_lastx_dim
1393 \dim_new:N \g__draw_softpath_lasty_dim

(End definition for \g__draw_softpath_lastx_dim and \g__draw_softpath_lasty_dim.)

```

```

\g__draw_softpath_move_bool Track if moving a point should update the close position.
1394 \bool_new:N \g__draw_softpath_move_bool
1395 \bool_gset_true:N \g__draw_softpath_move_bool

(End definition for \g__draw_softpath_move_bool.)

```

```

\__draw_softpath_curveto:nnnnnn The various parts of a path expressed as the appropriate soft path functions.
\__draw_softpath_lineto:nn
\__draw_softpath_moveto:nn
\__draw_softpath_rectangle:nnnn
\__draw_softpath_roundpoint:nn
\__draw_softpath_roundpoint:VV
1396 \cs_new_protected:Npn \__draw_softpath_closepath:
1397 {
1398   \__draw_softpath_add:x
1399   {
1400     \__draw_softpath_close_op:nn
1401     { \dim_use:N \g__draw_softpath_lastx_dim }
1402     { \dim_use:N \g__draw_softpath_lasty_dim }
1403   }
1404 }
1405 \cs_new_protected:Npn \__draw_softpath_curveto:nnnnnn #1#2#3#4#5#6
1406 {
1407   \__draw_softpath_add:n
1408   {
1409     \__draw_softpath_curveto_opi:nn {#1} {#2}
1410     \__draw_softpath_curveto_opii:nn {#3} {#4}
1411     \__draw_softpath_curveto_opiii:nn {#5} {#6}
1412   }
1413 }
1414 \cs_new_protected:Npn \__draw_softpath_lineto:nn #1#2
1415 {
1416   \__draw_softpath_add:n
1417   { \__draw_softpath_lineto_op:nn {#1} {#2} }
1418 }
1419 \cs_new_protected:Npn \__draw_softpath_moveto:nn #1#2
1420 {
1421   \__draw_softpath_add:n
1422   { \__draw_softpath_moveto_op:nn {#1} {#2} }

```

```

1423     \bool_if:NT \g__draw_softpath_move_bool
1424     {
1425         \dim_gset:Nn \g__draw_softpath_lastx_dim {#1}
1426         \dim_gset:Nn \g__draw_softpath_lasty_dim {#2}
1427     }
1428 }
1429 \cs_new_protected:Npn \__draw_softpath_rectangle:nnnn #1#2#3#4
1430 {
1431     \__draw_softpath_add:n
1432     {
1433         \__draw_softpath_rectangle_opi:nn {#1} {#2}
1434         \__draw_softpath_rectangle_opii:nn {#3} {#4}
1435     }
1436 }
1437 \cs_new_protected:Npn \__draw_softpath_roundpoint:nn #1#2
1438 {
1439     \__draw_softpath_add:n
1440     { \__draw_softpath_roundpoint_op:nn {#1} {#2} }
1441     \bool_gset_true:N \g__draw_softpath_corners_bool
1442 }
1443 \cs_generate_variant:Nn \__draw_softpath_roundpoint:nn { VV }

```

(End definition for __draw_softpath_curveto:nnnnnn and others.)

```

\__draw_softpath_close_op:nn
  \__draw_softpath_curveto_opi:nn
  \__draw_softpath_curveto_opii:nn
  \__draw_softpath_curveto_opiii:nn
  \__draw_softpath_lineto_op:nn
  \__draw_softpath_moveto_op:nn
  \__draw_softpath_roundpoint_op:nn
  \__draw_softpath_rectangle_opi:nn
  \__draw_softpath_rectangle_opii:nn
\__draw_softpath_curveto_opi:nnNnnNnn
\__draw_softpath_rectangle_opi:nnNnn

```

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.

```

1444 \cs_new_protected:Npn \__draw_softpath_close_op:nn #1#2
1445 { \__draw_backend_closepath: }
1446 \cs_new_protected:Npn \__draw_softpath_curveto_opi:nn #1#2
1447 { \__draw_softpath_curveto_opi:nnNnnNnn {#1} {#2} }
1448 \cs_new_protected:Npn \__draw_softpath_curveto_opi:nnNnnNnn #1#2#3#4#5#6#7#8
1449 { \__draw_backend_curveto:nnnnnn {#1} {#2} {#4} {#5} {#7} {#8} }
1450 \cs_new_protected:Npn \__draw_softpath_curveto_opii:nn #1#2
1451 { \__draw_softpath_curveto_opii:nn }
1452 \cs_new_protected:Npn \__draw_softpath_curveto_opiii:nn #1#2
1453 { \__draw_softpath_curveto_opiii:nn }
1454 \cs_new_protected:Npn \__draw_softpath_lineto_op:nn #1#2
1455 { \__draw_backend_lineto:nn {#1} {#2} }
1456 \cs_new_protected:Npn \__draw_softpath_moveto_op:nn #1#2
1457 { \__draw_backend_moveto:nn {#1} {#2} }
1458 \cs_new_protected:Npn \__draw_softpath_roundpoint_op:nn #1#2 { }
1459 \cs_new_protected:Npn \__draw_softpath_rectangle_opi:nn #1#2
1460 { \__draw_softpath_rectangle_opi:nnNnn {#1} {#2} }
1461 \cs_new_protected:Npn \__draw_softpath_rectangle_opi:nnNnn #1#2#3#4#5
1462 { \__draw_backend_rectangle:nnnn {#1} {#2} {#4} {#5} }
1463 \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. 1464 \tl_new:N \l__draw_softpath_main_tl (End definition for \l__draw_softpath_main_tl.)
\l__draw_softpath_part_tl	Data structures. 1465 \tl_new:N \l__draw_softpath_part_tl 1466 \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. 1467 \fp_new:N \l__draw_softpath_lastx_fp 1468 \fp_new:N \l__draw_softpath_lasty_fp 1469 \dim_new:N \l__draw_softpath_corneri_dim 1470 \dim_new:N \l__draw_softpath_cornerii_dim 1471 \tl_new:N \l__draw_softpath_first_tl 1472 \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. 1473 \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. 1474 \cs_new_protected:Npn __draw_softpath_round_corners: 1475 { 1476 \bool_if:NT \g__draw_softpath_corners_bool 1477 { 1478 \group_begin: 1479 \tl_clear:N \l__draw_softpath_main_tl 1480 \tl_clear:N \l__draw_softpath_part_tl 1481 \fp_zero:N \l__draw_softpath_lastx_fp 1482 \fp_zero:N \l__draw_softpath_lasty_fp 1483 \tl_clear:N \l__draw_softpath_first_tl 1484 \tl_clear:N \l__draw_softpath_move_tl 1485 \tl_build_get:NN \g__draw_softpath_main_tl \l__draw_softpath_internal_tl 1486 \exp_after:wN __draw_softpath_round_loop:Nnn 1487 \l__draw_softpath_internal_tl 1488 \q__draw_recursion_tail ? ? 1489 \q__draw_recursion_stop 1490 \group_end: 1491 } 1492 \bool_gset_false:N \g__draw_softpath_corners_bool 1493 }

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.

```

1494 \cs_new_protected:Npn \__draw_softpath_round_loop:Nnn #1#2#3
1495 {
1496   \__draw_if_recursion_tail_stop_do:Nn #1 { \__draw_softpath_round_end: }
1497   \token_if_eq_meaning:NNTF #1 \__draw_softpath_roundpoint_op:nn
1498   { \__draw_softpath_round_action:nn {#2} {#3} }
1499   {
1500     \tl_if_empty:NT \l__draw_softpath_first_tl
1501     { \tl_set:Nn \l__draw_softpath_first_tl { {#2} {#3} } }
1502     \fp_set:Nn \l__draw_softpath_lastx_fp {#2}
1503     \fp_set:Nn \l__draw_softpath_lasty_fp {#3}
1504     \token_if_eq_meaning:NNTF #1 \__draw_softpath_moveto_op:nn
1505     {
1506       \tl_put_right:No \l__draw_softpath_main_tl
1507       \l__draw_softpath_move_tl
1508       \tl_put_right:No \l__draw_softpath_main_tl
1509       \l__draw_softpath_part_tl
1510       \tl_set:Nn \l__draw_softpath_move_tl { #1 {#2} {#3} }
1511       \tl_clear:N \l__draw_softpath_first_tl
1512       \tl_clear:N \l__draw_softpath_part_tl
1513     }
1514     { \tl_put_right:Nn \l__draw_softpath_part_tl { #1 {#2} {#3} } }
1515     \__draw_softpath_round_loop:Nnn
1516   }
1517 }
1518 \cs_new_protected:Npn \__draw_softpath_round_action:nn #1#2
1519 {
1520   \dim_set:Nn \l__draw_softpath_corneri_dim {#1}
1521   \dim_set:Nn \l__draw_softpath_cornerii_dim {#2}
1522   \bool_lazy_and:nnTF
1523   { \dim_compare_p:nNn \l__draw_softpath_corneri_dim = { Opt } }
1524   { \dim_compare_p:nNn \l__draw_softpath_cornerii_dim = { Opt } }
1525   { \__draw_softpath_round_loop:Nnn }
1526   { \__draw_softpath_round_action:Nnn }
1527 }

```

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.

```

1528 \cs_new_protected:Npn \__draw_softpath_round_action:Nnn #1#2#3
1529 {
1530   \tl_if_empty:NT \l__draw_softpath_first_tl
1531   { \tl_set:Nn \l__draw_softpath_first_tl { {#2} {#3} } }
1532   \token_if_eq_meaning:NNTF #1 \__draw_softpath_curveto_opi:nn
1533   { \__draw_softpath_round_action_curveto:NnnNnn }
1534   {
1535     \token_if_eq_meaning:NNTF #1 \__draw_softpath_close_op:nn
1536     { \__draw_softpath_round_action_close: }
1537     {
1538       \token_if_eq_meaning:NNTF #1 \__draw_softpath_lineto_op:nn

```

```

1539         { \_draw_softpath_round_lookahead:NnnNnn }
1540         { \_draw_softpath_round_loop:Nnn }
1541     }
1542 }
1543 #1 {#2} {#3}
1544 }

```

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.

```

1545 \cs_new_protected:Npn \_draw_softpath_round_action_curveto:NnnNnn
1546   #1#2#3#4#5#6
1547   {
1548     \tl_put_right:Nn \l__draw_softpath_part_tl
1549     { #1 {#2} {#3} #4 {#5} {#6} }
1550     \fp_set:Nn \l__draw_softpath_lastx_fp {#5}
1551     \fp_set:Nn \l__draw_softpath_lasty_fp {#6}
1552     \_draw_softpath_round_lookahead:NnnNnn
1553   }
1554 \cs_new_protected:Npn \_draw_softpath_round_action_close:
1555   {
1556     \bool_lazy_and:nnTF
1557     { ! \tl_if_empty_p:N \l__draw_softpath_first_tl }
1558     { ! \tl_if_empty_p:N \l__draw_softpath_move_tl }
1559     {
1560       \exp_after:wN \_draw_softpath_round_close:nn
1561       \l__draw_softpath_first_tl
1562     }
1563     { \_draw_softpath_round_loop:Nnn }
1564   }

```

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.

```

1565 \cs_new_protected:Npn \_draw_softpath_round_lookahead:NnnNnn #1#2#3#4#5#6
1566   {
1567     \bool_lazy_any:nTF
1568     {
1569       { \token_if_eq_meaning_p:NN #4 \_draw_softpath_lineto_op:nn }
1570       { \token_if_eq_meaning_p:NN #4 \_draw_softpath_curveto_opi:nn }
1571       { \token_if_eq_meaning_p:NN #4 \_draw_softpath_close_op:nn }
1572     }
1573     {
1574       \_draw_softpath_round_calc:NnnNnn
1575       \_draw_softpath_round_loop:Nnn
1576       {#5} {#6}
1577     }
1578     {
1579       \token_if_eq_meaning:NNTF #4 \_draw_softpath_roundpoint_op:nn
1580       { \_draw_softpath_round_roundpoint:NnnNnnNnn }
1581       { \_draw_softpath_round_loop:Nnn }
1582     }
1583     #1 {#2} {#3}
1584     #4 {#5} {#6}
1585   }
1586 \cs_new_protected:Npn \_draw_softpath_round_roundpoint:NnnNnnNnn

```

```

1587 #1#2#3#4#5#6#7#8#9
1588 {
1589     \__draw_softpath_round_calc:NnnNnn
1590     \__draw_softpath_round_loop:Nnn
1591     {#8} {#9}
1592     #1 {#2} {#3}
1593     #4 {#5} {#6} #7 {#8} {#9}
1594 }

```

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.

```

1595 \cs_new_protected:Npn \__draw_softpath_round_calc:NnnNnn #1#2#3#4#5#6
1596 {
1597     \tl_set:Nx \l__draw_softpath_curve_end_tl
1598     {
1599         \draw_point_interpolate_distance:nnn
1600         \l__draw_softpath_cornerii_dim
1601         { #5 , #6 } { #2 , #3 }
1602     }
1603     \tl_put_right:Nx \l__draw_softpath_part_tl
1604     {
1605         \exp_not:N #4
1606         \__draw_softpath_round_calc:fVnnnn
1607         {
1608             \draw_point_interpolate_distance:nnn
1609             \l__draw_softpath_corneri_dim
1610             { #5 , #6 }
1611             {
1612                 \l__draw_softpath_lastx_fp ,
1613                 \l__draw_softpath_lasty_fp
1614             }
1615         }
1616         \l__draw_softpath_curve_end_tl
1617         {#5} {#6} {#2} {#3}
1618     }
1619     \fp_set:Nn \l__draw_softpath_lastx_fp {#5}
1620     \fp_set:Nn \l__draw_softpath_lasty_fp {#6}
1621     #1
1622 }

```

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).

```

1623 \cs_new:Npn \__draw_softpath_round_calc:nnnnnn #1#2#3#4#5#6
1624 {
1625     \__draw_softpath_round_calc:nnnnw {#3} {#4} {#5} {#6}
1626     #1 \s__draw_mark #2 \s__draw_stop
1627 }
1628 \cs_generate_variant:Nn \__draw_softpath_round_calc:nnnnnn { fV }

```

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

```

1629 \cs_new:Npn \__draw_softpath_round_calc:nnnnw
1630   #1#2#3#4 #5 , #6 \s__draw_mark #7 , #8 \s__draw_stop
1631   {
1632     {#5} {#6}
1633     \exp_not:N \__draw_softpath_curveto_opi:nn
1634     {
1635       \fp_to_dim:n
1636       { #5 + \c__draw_softpath_arc_fp * ( #1 - #5 ) }
1637     }
1638     {
1639       \fp_to_dim:n
1640       { #6 + \c__draw_softpath_arc_fp * ( #2 - #6 ) }
1641     }
1642     \exp_not:N \__draw_softpath_curveto_opii:nn
1643     {
1644       \fp_to_dim:n
1645       { #7 + \c__draw_softpath_arc_fp * ( #1 - #7 ) }
1646     }
1647     {
1648       \fp_to_dim:n
1649       { #8 + \c__draw_softpath_arc_fp * ( #2 - #8 ) }
1650     }
1651     \exp_not:N \__draw_softpath_curveto_opiii:nn
1652     {#7} {#8}
1653   }

```

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.

```

1654 \cs_new_protected:Npn \__draw_softpath_round_close:nn #1#2
1655   {
1656     \use:x
1657     {
1658       \__draw_softpath_round_calc:NnnNnn
1659       {
1660         \tl_set:Nx \exp_not:N \l__draw_softpath_move_tl
1661         {
1662           \__draw_softpath_moveto_op:nn
1663           \exp_not:N \exp_after:wN
1664           \exp_not:N \__draw_softpath_round_close:w
1665           \exp_not:N \l__draw_softpath_curve_end_tl
1666           \s__draw_stop
1667         }
1668       }
1669       \use:x
1670       {
1671         \exp_not:N \exp_not:N \exp_not:N \use_i:nnnn
1672         {
1673           \__draw_softpath_round_loop:Nnn
1674           \__draw_softpath_close_op:nn
1675           \exp_not:N \exp_after:wN
1676           \exp_not:N \__draw_softpath_round_close:w

```

```

1676 \exp_not:N \l__draw_softpath_curve_end_tl
1677 \s__draw_stop
1678 }
1679 }
1680 }
1681 {#1} {#2}
1682 \__draw_softpath_lineto_op:nn
1683 \exp_after:wN \use_none:n \l__draw_softpath_move_tl
1684 }
1685 }
1686 \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.

```

1687 \cs_new_protected:Npn \__draw_softpath_round_end:
1688 {
1689   \tl_put_right:No \l__draw_softpath_main_tl
1690   \l__draw_softpath_move_tl
1691   \tl_put_right:No \l__draw_softpath_main_tl
1692   \l__draw_softpath_part_tl
1693   \tl_build_gclear:N \g__draw_softpath_main_tl
1694   \__draw_softpath_add:o \l__draw_softpath_main_tl
1695 }

```

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

```

1696 </package>

```

8 l3draw-state implementation

```

1697 <*package>
1698 <@@=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`, `\pgfsetinnerstrokecolor`
- 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.

```

1699 \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.

```

1700 \dim_new:N \l_draw_default_linewidth_dim
1701 \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.

```

1702 \cs_new_protected:Npn \draw_linewidth:n #1
1703 {
1704   \dim_gset:Nn \g__draw_linewidth_dim { \fp_to_dim:n {#1} }
1705   \__draw_backend_linewidth:n \g__draw_linewidth_dim
1706 }

```

(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 1707 \cs_new_protected:Npn \draw_dash_pattern:nn #1#2
1708 {
1709   \group_begin:
1710     \seq_set_from_clist:Nn \l__draw_tmp_seq {#1}
1711     \seq_set_map:Nn \l__draw_tmp_seq \l__draw_tmp_seq
1712       { \fp_to_dim:n {##1} }
1713     \use:x
1714     {
1715       \__draw_backend_dash_pattern:nn
1716       { \seq_use:Nn \l__draw_tmp_seq { , } }
1717       { \fp_to_dim:n {#2} }
1718     }
1719   \group_end:
1720 }
1721 \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.

```

1722 \cs_new_protected:Npn \draw_miterlimit:n #1
1723 { \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_but`: All straight wrappers.

```

\draw_cap_rectangle: 1724 \cs_new_protected:Npn \draw_cap_but: { \__draw_backend_cap_but: }
\draw_cap_round:      1725 \cs_new_protected:Npn \draw_cap_rectangle: { \__draw_backend_cap_rectangle: }
\draw_evenodd_rule:   1726 \cs_new_protected:Npn \draw_cap_round: { \__draw_backend_cap_round: }
\draw_nonzero_rule:   1727 \cs_new_protected:Npn \draw_evenodd_rule: { \__draw_backend_evenodd_rule: }
\draw_join_bevel:     1728 \cs_new_protected:Npn \draw_nonzero_rule: { \__draw_backend_nonzero_rule: }
\draw_join_miter:     1729 \cs_new_protected:Npn \draw_join_bevel: { \__draw_backend_join_bevel: }
\draw_join_round:     1730 \cs_new_protected:Npn \draw_join_miter: { \__draw_backend_join_miter: }
1731 \cs_new_protected:Npn \draw_join_round: { \__draw_backend_join_round: }

```

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

```

1732 \</package>

```

9 l3draw-transforms implementation

```

1733 \*package>

```

```

1734 \@@=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.
- `\pgftransformlineat`, `\pgftransformarcaxesat`, `\pgftransformcurveat`: 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 CircuiTikZ although also for shapes, likely needs more use cases before addressing.
- `\pgflevelsynchronism`, `\pgflevel`: 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.

```

1735 \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 1736 \fp_new:N \l__draw_matrix_a_fp
\l__draw_matrix_c_fp 1737 \fp_new:N \l__draw_matrix_b_fp
\l__draw_xshift_dim 1738 \fp_new:N \l__draw_matrix_c_fp
\l__draw_yshift_dim 1739 \fp_new:N \l__draw_matrix_d_fp
1740 \dim_new:N \l__draw_xshift_dim
1741 \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:`

```

1742 \cs_new_protected:Npn \draw_transform_matrix_reset:
1743 {
1744   \fp_set:Nn \l__draw_matrix_a_fp { 1 }
1745   \fp_zero:N \l__draw_matrix_b_fp
1746   \fp_zero:N \l__draw_matrix_c_fp
1747   \fp_set:Nn \l__draw_matrix_d_fp { 1 }
1748 }
1749 \cs_new_protected:Npn \draw_transform_shift_reset:
1750 {
1751   \dim_zero:N \l__draw_xshift_dim
1752   \dim_zero:N \l__draw_yshift_dim
1753 }
1754 \draw_transform_matrix_reset:
1755 \draw_transform_shift_reset:

(End definition for \draw_transform_matrix_reset: and \draw_transform_shift_reset:. These func-
tions are documented on page ??.)

```

`\draw_transform_matrix_absolute:nmmn` 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`

```

1756 \cs_new_protected:Npn \draw_transform_matrix_absolute:nmmn #1#2#3#4
1757 {
1758   \fp_set:Nn \l__draw_matrix_a_fp {#1}
1759   \fp_set:Nn \l__draw_matrix_b_fp {#2}
1760   \fp_set:Nn \l__draw_matrix_c_fp {#3}
1761   \fp_set:Nn \l__draw_matrix_d_fp {#4}
1762   \bool_lazy_all:nTF
1763   {
1764     { \fp_compare_p:nNn \l__draw_matrix_a_fp = \c_one_fp }
1765     { \fp_compare_p:nNn \l__draw_matrix_b_fp = \c_zero_fp }

```



```

1766     { \fp_compare_p:nNn \l__draw_matrix_c_fp = \c_zero_fp }
1767     { \fp_compare_p:nNn \l__draw_matrix_d_fp = \c_one_fp }
1768   }
1769   { \bool_set_false:N \l__draw_matrix_active_bool }
1770   { \bool_set_true:N \l__draw_matrix_active_bool }
1771 }
1772 \cs_new_protected:Npn \draw_transform_shift_absolute:n #1
1773 {
1774   \__draw_point_process:nn
1775   { \__draw_transform_shift_absolute:nn } {#1}
1776 }
1777 \cs_new_protected:Npn \__draw_transform_shift_absolute:nn #1#2
1778 {
1779   \dim_set:Nn \l__draw_xshift_dim {#1}
1780   \dim_set:Nn \l__draw_yshift_dim {#2}
1781 }

```

(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 that the calculation uses “frozen” values.

```

\__draw_transform:nnnn
\draw_transform_shift:n
\__draw_transform_shift:nn
1782 \cs_new_protected:Npn \draw_transform_matrix:nnnn #1#2#3#4
1783 {
1784   \use:x
1785   {
1786     \__draw_transform:nnnn
1787     { \fp_eval:n {#1} }
1788     { \fp_eval:n {#2} }
1789     { \fp_eval:n {#3} }
1790     { \fp_eval:n {#4} }
1791   }
1792 }
1793 \cs_new_protected:Npn \__draw_transform:nnnn #1#2#3#4
1794 {
1795   \use:x
1796   {
1797     \draw_transform_matrix_absolute:nnnn
1798     { #1 * \l__draw_matrix_a_fp + #2 * \l__draw_matrix_c_fp }
1799     { #1 * \l__draw_matrix_b_fp + #2 * \l__draw_matrix_d_fp }
1800     { #3 * \l__draw_matrix_a_fp + #4 * \l__draw_matrix_c_fp }
1801     { #3 * \l__draw_matrix_b_fp + #4 * \l__draw_matrix_d_fp }
1802   }
1803 }
1804 \cs_new_protected:Npn \draw_transform_shift:n #1
1805 {
1806   \__draw_point_process:nn
1807   { \__draw_transform_shift:nn } {#1}
1808 }
1809 \cs_new_protected:Npn \__draw_transform_shift:nn #1#2
1810 {
1811   \dim_set:Nn \l__draw_xshift_dim { \l__draw_xshift_dim + #1 }
1812   \dim_set:Nn \l__draw_yshift_dim { \l__draw_yshift_dim + #2 }
1813 }

```

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

```

\draw_transform_matrix_invert:
\__draw_transform_invert:n
\__draw_transform_invert:f
\draw_transform_shift_invert:
Standard mathematics: calculate the inverse matrix and use that, then undo the shifts.
1814 \cs_new_protected:Npn \draw_transform_matrix_invert:
1815 {
1816   \bool_if:NT \l__draw_matrix_active_bool
1817   {
1818     \__draw_transform_invert:f
1819     {
1820       \fp_eval:n
1821       {
1822         1 /
1823         (
1824           \l__draw_matrix_a_fp * \l__draw_matrix_d_fp
1825           - \l__draw_matrix_b_fp * \l__draw_matrix_c_fp
1826         )
1827       }
1828     }
1829   }
1830 }
1831 \cs_new_protected:Npn \__draw_transform_invert:n #1
1832 {
1833   \fp_set:Nn \l__draw_matrix_a_fp
1834   { \l__draw_matrix_d_fp * #1 }
1835   \fp_set:Nn \l__draw_matrix_b_fp
1836   { -\l__draw_matrix_b_fp * #1 }
1837   \fp_set:Nn \l__draw_matrix_c_fp
1838   { -\l__draw_matrix_c_fp * #1 }
1839   \fp_set:Nn \l__draw_matrix_d_fp
1840   { \l__draw_matrix_a_fp * #1 }
1841 }
1842 \cs_generate_variant:Nn \__draw_transform_invert:n { f }
1843 \cs_new_protected:Npn \draw_transform_shift_invert:
1844 {
1845   \dim_set:Nn \l__draw_xshift_dim { -\l__draw_xshift_dim }
1846   \dim_set:Nn \l__draw_yshift_dim { -\l__draw_yshift_dim }
1847 }

```

(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.

```

1848 \cs_new_protected:Npn \draw_transform_triangle:nnn #1#2#3
1849 {
1850   \__draw_point_process:nnn
1851   {
1852     \__draw_point_process:nn
1853     { \__draw_tranform_triangle:nnnnnn }
1854     {#1}
1855   }
1856   {#2} {#3}
1857 }
1858 \cs_new_protected:Npn \__draw_tranform_triangle:nnnnnn #1#2#3#4#5#6

```

```

1859 {
1860   \use:x
1861   {
1862     \draw_transform_matrix_absolute:nnnn
1863     { #3 - #1 }
1864     { #4 - #2 }
1865     { #5 - #1 }
1866     { #6 - #2 }
1867     \draw_transform_shift_absolute:n { #1 , #2 }
1868   }
1869 }

```

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

```

\draw_transform_scale:n Lots of shortcuts.
\draw_transform_xscale:n 1870 \cs_new_protected:Npn \draw_transform_scale:n #1
\draw_transform_yscale:n 1871 { \draw_transform_matrix:nnnn { #1 } { 0 } { 0 } { #1 } }
\draw_transform_xshift:n 1872 \cs_new_protected:Npn \draw_transform_xscale:n #1
\draw_transform_yshift:n 1873 { \draw_transform_matrix:nnnn { #1 } { 0 } { 0 } { 1 } }
\draw_transform_xslant:n 1874 \cs_new_protected:Npn \draw_transform_yscale:n #1
\draw_transform_yslant:n 1875 { \draw_transform_matrix:nnnn { 1 } { 0 } { 0 } { #1 } }
1876 \cs_new_protected:Npn \draw_transform_xshift:n #1
1877 { \draw_transform_shift:n { #1 , Opt } }
1878 \cs_new_protected:Npn \draw_transform_yshift:n #1
1879 { \draw_transform_shift:n { Opt , #1 } }
1880 \cs_new_protected:Npn \draw_transform_xslant:n #1
1881 { \draw_transform_matrix:nnnn { 1 } { 0 } { #1 } { 1 } }
1882 \cs_new_protected:Npn \draw_transform_yslant:n #1
1883 { \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 1884 \cs_new_protected:Npn \draw_transform_rotate:n #1
\__draw_transform_rotate:f 1885 { \__draw_transform_rotate:f { \fp_eval:n {#1} } }
\__draw_transform_rotate:nn 1886 \cs_new_protected:Npn \__draw_transform_rotate:n #1
\__draw_transform_rotate:ff 1887 {
1888   \__draw_transform_rotate:ff
1889   { \fp_eval:n { cosd(#1) } }
1890   { \fp_eval:n { sind(#1) } }
1891 }
1892 \cs_generate_variant:Nn \__draw_transform_rotate:n { f }
1893 \cs_new_protected:Npn \__draw_transform_rotate:nn #1#2
1894 { \draw_transform_matrix:nnnn {#1} {#2} { -#2 } { #1 } }
1895 \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 ??.)

```

1896 </package>

```

Index

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

B

`\begin` . . . 171, 786, 1038, 1042, 1066, 1069

bool commands:

`\bool_gset_eq:NN` 767

`\bool_gset_false:N` 1390, 1492

`\bool_gset_true:N` 1395, 1441

`\bool_if:NTF`

. 21, 115, 194, 233, 682, 698, 699,

703, 1161, 1193, 1311, 1423, 1476, 1816

`\bool_lazy_all:nTF` 1762

`\bool_lazy_and:nnTF`

. 225, 677, 1522, 1556

`\bool_lazy_any:nTF` 1567

`\bool_lazy_or:nnTF` . 558, 653, 686, 691

`\bool_new:N`

. 86, 220, 641, 642, 643, 644,

645, 745, 1217, 1302, 1378, 1394, 1735

`\bool_set_eq:NN` 759

`\bool_set_false:N`

. 96, 228, 664, 665, 666, 685, 1769

`\bool_set_true:N` 98, 229,

670, 708, 712, 713, 1236, 1306, 1770

box commands:

`\box_dp:N` 17, 67

`\box_gset_eq:NN` 156

`\box_gset_wd:Nn` 100, 133

`\box_ht:N` 17, 69

`\box_if_exist:NTF` 93

`\box_move_down:nn` 1276, 1289

`\box_move_up:nn` 51

`\box_new:N` 13, 80, 81, 1218, 1219

`\box_set_dp:Nn` 55, 1280, 1293

`\box_set_eq:NN` 145

`\box_set_ht:Nn` 54, 1281, 1299

`\box_set_wd:Nn` 56, 128, 1265

`\box_use_drop:N` 52,

57, 102, 129, 134, 1268, 1278, 1291

`\box_wd:N` 17, 66, 68

C

clist commands:

`\clist_map_inline:Nn` . . 124, 141, 152

`\clist_map_inline:nn` 667

`\clist_new:N` 87, 89

`\clist_set:Nn` 88, 1253

coffin commands:

`\coffin_typeset:Nnnnn` 64

`\coffin_wd:N` 66

color commands:

`\color_select:n` 1241

cs commands:

`\cs_generate_variant:Nn`

. 420, 607, 640, 845, 853, 895,

914, 921, 929, 936, 945, 951, 963,

966, 984, 1002, 1010, 1016, 1037,

1051, 1065, 1086, 1121, 1140, 1153,

1381, 1443, 1628, 1842, 1892, 1895

`\cs_if_exist:NTF` 669

`\cs_if_exist_use:NTF` . . 403, 412, 672

`\cs_new:Npn` 511, 521, 531, 541,

790, 796, 798, 800, 807, 809, 811,

819, 821, 824, 833, 838, 841, 843,

846, 847, 849, 851, 854, 856, 862,

871, 877, 887, 896, 902, 907, 915,

922, 930, 937, 946, 952, 958, 964,

967, 973, 982, 985, 991, 996, 1003,

1011, 1017, 1023, 1029, 1043, 1052,

1058, 1070, 1072, 1081, 1111, 1113,

1122, 1127, 1141, 1143, 1145, 1154,

1159, 1186, 1191, 1623, 1629, 1686

`\cs_new_protected:Npn` . 14, 19, 60,

75, 90, 113, 122, 139, 150, 184, 206,

213, 221, 231, 240, 246, 252, 258,

265, 276, 284, 289, 291, 293, 302,

309, 345, 347, 358, 364, 394, 421,

450, 456, 462, 467, 475, 484, 489,

497, 552, 554, 567, 574, 583, 589,

591, 601, 608, 614, 621, 646, 651,

662, 706, 710, 715, 722, 746, 764,

1093, 1095, 1097, 1099, 1103, 1221,

1228, 1249, 1271, 1284, 1304, 1309,

1324, 1331, 1342, 1351, 1359, 1367,

1379, 1382, 1387, 1396, 1405, 1414,

1419, 1429, 1437, 1444, 1446, 1448,

1450, 1452, 1454, 1456, 1458, 1459,

1461, 1463, 1474, 1494, 1518, 1528,

1545, 1554, 1565, 1586, 1595, 1654,

1687, 1702, 1707, 1722, 1724, 1725,

1726, 1727, 1728, 1729, 1730, 1731,

1742, 1749, 1756, 1772, 1777, 1782,

1793, 1804, 1809, 1814, 1831, 1843,

1848, 1858, 1870, 1872, 1874, 1876,

1878, 1880, 1882, 1884, 1886, 1893

D

dim commands:

<code>\dim_abs:n</code>	596 , 597
<code>\dim_compare:nNnTF</code>	603 , 610 , 724 , 1257
<code>\dim_compare_p:nNn</code>	226 , 227 , 1523 , 1524
<code>\dim_eval:n</code>	596 , 597
<code>\dim_gset:Nn</code>	186 , 188 , 190 , 192 , 196 , 198 , 200 , 202 , 208 , 209 , 210 , 211 , 215 , 216 , 726 , 1223 , 1224 , 1225 , 1226 , 1425 , 1426 , 1704
<code>\dim_gset_eq:NN</code>	771 , 772 , 773 , 774 , 775 , 776 , 777 , 778 , 1334 , 1353 , 1354 , 1355 , 1356
<code>\dim_gzero:N</code> ..	1259 , 1260 , 1261 , 1262
<code>\dim_max:nn</code> ..	187 , 191 , 197 , 201 , 1295
<code>\dim_min:nn</code>	189 , 193 , 199 , 203
<code>\dim_new:N</code>	178 , 179 , 180 , 181 , 182 , 183 , 218 , 219 , 737 , 738 , 739 , 740 , 741 , 742 , 743 , 744 , 1087 , 1088 , 1089 , 1090 , 1091 , 1092 , 1213 , 1214 , 1215 , 1216 , 1303 , 1321 , 1338 , 1339 , 1340 , 1341 , 1392 , 1393 , 1469 , 1470 , 1699 , 1700 , 1740 , 1741
<code>\dim_set:Nn</code>	223 , 224 , 1105 , 1106 , 1307 , 1520 , 1521 , 1701 , 1779 , 1780 , 1811 , 1812 , 1845 , 1846
<code>\dim_set_eq:NN</code>	749 , 750 , 751 , 752 , 753 , 754 , 755 , 756 , 1328 , 1345 , 1346 , 1347 , 1348
<code>\dim_step_inline:nnnn</code>	623 , 631
<code>\dim_use:N</code>	724 , 729 , 731 , 1316 , 1401 , 1402
<code>\dim_zero:N</code>	1751 , 1752
<code>\c_max_dim</code>	208 , 209 , 210 , 211 , 724 , 1223 , 1224 , 1225 , 1226 , 1257

draw commands:

<code>\draw_baseline:n</code>	1304
<code>\l_draw_bb_update_bool</code>	21 , 194 , 678 , 685 , 1217 , 1236
<code>\draw_begin:</code>	1228
<code>\draw_box_use:N</code>	14
<code>\draw_cap_but:</code>	1243 , 1724
<code>\draw_cap_rectangle:</code>	1724
<code>\draw_cap_round:</code>	1724
<code>\draw_coffin_use:Nnn</code>	33 , 60
<code>\draw_dash_pattern:nn</code> ...	1246 , 1707
<code>\l_draw_default_linewidth_dim</code> ...	105 , 1240 , 1700
<code>\draw_end:</code>	1228
<code>\draw_evenodd_rule:</code>	1724
<code>\draw_join_bevel:</code>	1724
<code>\draw_join_miter:</code>	1244 , 1724
<code>\draw_join_round:</code>	1724
<code>\draw_layer_begin:n</code>	90

<code>\draw_layer_end:</code>	90
<code>\draw_layer_new:n</code>	75
<code>\l_draw_layers_clist</code>	87 , 124 , 141 , 152 , 1253
<code>\draw_linewidth:n</code>	105 , 1240 , 1702
<code>\draw_miterlimit:n</code>	1245 , 1722
<code>\draw_nonzero_rule:</code>	1242 , 1724
<code>\draw_path_arc:nnn</code>	345 , 487
<code>\draw_path_arc:nnnn</code>	345
<code>\draw_path_arc_axes:nnnn</code>	484
<code>\draw_path_canvas_curteto:nnn</code> ..	289
<code>\draw_path_canvas_lineto:n</code>	289
<code>\draw_path_canvas_moveto:n</code>	289
<code>\draw_path_circle:nn</code>	552
<code>\draw_path_close:</code>	284 , 580
<code>\draw_path_corner_arc:nn</code>	221
<code>\draw_path_curteto:nn</code>	302
<code>\draw_path_curteto:nnn</code>	240
<code>\draw_path_ellipse:nnn</code>	489 , 553
<code>\draw_path_grid:nnnn</code>	591
<code>\draw_path_lineto:n</code>	240 , 577 , 578 , 579 , 629 , 637
<code>\draw_path_moveto:n</code>	240 , 576 , 581 , 628 , 636
<code>\draw_path_rectangle:nn</code>	554 , 590
<code>\draw_path_rectangle_corners:nn</code>	583
<code>\draw_path_scope_begin:</code>	746 , 1329 , 1362
<code>\draw_path_scope_end:</code>	746 , 1333 , 1370
<code>\draw_path_use:n</code>	646
<code>\draw_path_use_clear:n</code>	646
<code>\draw_point:n</code>	793 , 803 , 804 , 814 , 815 , 816 , 827 , 828 , 829 , 830 , 841 , 1013 , 1083 , 1115 , 1129 , 1147 , 1163 , 1179 , 1195 , 1208
<code>\draw_point_interpolate_arcaxes:nnnnnn</code>	985
<code>\draw_point_interpolate_curve:nnnnn</code>	1017
<code>\draw_point_interpolate_curve:nnnnnn</code>	1017
<code>\draw_point_interpolate_curve_</code> <code>auxi:nnnnnnnn</code>	1017
<code>\draw_point_interpolate_curve_</code> <code>auxii:nnnnnnnn</code>	1017
<code>\draw_point_interpolate_curve_</code> <code>auxiii:nnnnnn</code>	1017
<code>\draw_point_interpolate_curve_</code> <code>auxiv:nnnnnn</code>	1017
<code>\draw_point_interpolate_curve_</code> <code>auxv:nnw</code>	1017
<code>\draw_point_interpolate_curve_</code> <code>auxvi:n</code>	1017

\draw_point_interpolate_curve- auxvii:nnnnnnnn	1017	__draw_backend_box_use:Nnnnn . . .	41
\draw_point_interpolate_curve- auxviii:nnnnnn	1017	__draw_backend_cap_but:	1724
\draw_point_interpolate_distance:nnn	967, 1599, 1608	__draw_backend_cap_rectangle: . . .	1725
\draw_point_interpolate_line:nnn	952	__draw_backend_cap_round: . . .	1726
\draw_point_intersect_circles:nnnn	896	__draw_backend_clip:	684
\draw_point_intersect_lines:nnnn	871	__draw_backend_closepath: . . .	1445
\draw_point_polar:nn	847	__draw_backend_curveto:nnnnnn	1449
\draw_point_polar:nnn	428, 434, 438, 444, 847	__draw_backend_dash_pattern:nn	1715
\draw_point_transform:n	25, 28, 31, 34, 244, 256, 272, 273, 274, 306, 307, 433, 437, 493, 564, 1154	__draw_backend_discardpath: . .	689
\draw_point_unit_vector:n . .	854, 980	__draw_backend_end:	1255
\draw_point_vec:nn	1111	__draw_backend_evenodd_rule: .	1727
\draw_point_vec:nnn	1111	__draw_backend_join_bevel: . .	1729
\draw_point_vec_polar:nn	1141	__draw_backend_join_miter: . .	1730
\draw_point_vec_polar:nnn	1141	__draw_backend_join_round: . .	1731
\draw_scope_begin:	1324	__draw_backend_lineto:nn	1455
\draw_scope_end:	1331	__draw_backend_linewidth:n . .	1705
\draw_suspend_begin:	1359	__draw_backend_miterlimit:n .	1723
\draw_suspend_end:	1359	__draw_backend_moveto:nn	1457
\draw_transform_matrix:nnnn	1782, 1871, 1873, 1875, 1881, 1883, 1894	__draw_backend_nonzero_rule: .	1728
\draw_transform_matrix_absolute:nnnn	1756, 1797, 1862	__draw_backend_rectangle:nnnn	1462
\draw_transform_matrix_invert: .	1814	__draw_backend_scope_begin: . . .	132, 1326
\draw_transform_matrix_reset: . . .	1237, 1363, 1742	__draw_backend_scope_end: . . .	135, 1336
\draw_transform_rotate:n	1884	\l__draw_baseline_bool	1302, 1306, 1311
\draw_transform_scale:n	1870	\l__draw_baseline_dim	1302, 1307, 1316
\draw_transform_shift:n	1782, 1877, 1879	__draw_baseline_finalise:w	1251, 1309
\draw_transform_shift_absolute:n	1756, 1867	__draw_box_use:Nnnnn	14, 65
\draw_transform_shift_invert: .	1814	\l__draw_corner_arc_bool	220, 228, 229, 233, 559
\draw_transform_shift_reset: . . .	1238, 1364, 1742	\l__draw_corner_xarc_dim	218, 223, 226, 236
\draw_transform_triangle:nnn . . .	486, 1848	\l__draw_corner_yarc_dim	218, 224, 227, 237
\draw_transform_xscale:n	1870	__draw_draw_polar:nnn	847
\draw_transform_xshift:n	1870	__draw_draw_vec_polar:nnn	1144, 1145, 1153
\draw_transform_xslant:n	1870	\l__draw_fill_color_tl	1321
\draw_transform_yscale:n	1870	__draw_finalise:	1264, 1271, 1309, 1319
\draw_transform_yshift:n	1870	__draw_finalise_baseline:n	1271, 1316
\draw_transform_yslant:n	1870	\g__draw_id_int	1220, 1231
\draw_xvec:n	1093, 1108	__draw_if_recursion_tail_stop- do:Nn	9, 1496
\draw_yvec:n	1093, 1109	\l__draw_layer_close_bool	86, 96, 98, 115
\draw_zvec:n	1093, 1110	\l__draw_layer_main_box	128, 129, 1218, 1247
draw internal commands:		\l__draw_layer_tl	84, 95, 99
__draw_backend_begin:	1233	\g__draw_layers_clist	87
		__draw_layers_insert:	122, 1254
		__draw_layers_restore: . . .	139, 1369

__draw_layers_save:	139 , 1365	__draw_path_grid_auxiii:nnnnnn	591
\g__draw_linewidth_dim 732 , 1328 , 1334 , 1699 , 1704 , 1705	__draw_path_grid_auxiiii:nnnnnn	591
\l__draw_linewidth_dim 1321 , 1328 , 1334	__draw_path_grid_auxiv:nnnnnnnn	591
\l__draw_main_box	1218 , 1232 , 1265 , 1268 , 1273 , 1278 , 1280 , 1281 , 1286 , 1291 , 1293 , 1299	\g__draw_path_lastx_dim 178 , 215 , 320 , 453 , 459 , 749 , 777
\l__draw_matrix_a_fp 42 , 1166 , 1198 , 1736 , 1744 , 1758 , 1764 , 1798 , 1800 , 1824 , 1833 , 1840	\l__draw_path_lastx_dim	737 , 749 , 777
\l__draw_matrix_active_bool	560 , 1161 , 1193 , 1735 , 1769 , 1770 , 1816	\g__draw_path_lasty_dim 178 , 216 , 327 , 454 , 460 , 750 , 778
\l__draw_matrix_b_fp 43 , 1172 , 1203 , 1736 , 1745 , 1759 , 1765 , 1799 , 1801 , 1825 , 1835 , 1836	\l__draw_path_lasty_dim	737 , 750 , 778
\l__draw_matrix_c_fp 44 , 1167 , 1199 , 1736 , 1746 , 1760 , 1766 , 1798 , 1800 , 1825 , 1837 , 1838	__draw_path_lineto:nn	240 , 292
\l__draw_matrix_d_fp 45 , 1173 , 1204 , 1739 , 1747 , 1761 , 1767 , 1799 , 1801 , 1824 , 1834 , 1839	__draw_path_mark_corner: 231 , 260 , 269 , 286 , 297 , 315 , 386
__draw_path_arc:nnnn	345	__draw_path_moveto:nn 240 , 290 , 501 , 509
__draw_path_arc:nnNnn	345	__draw_path_rectangle:nnnn	554
\c__draw_path_arc_60_fp	345	__draw_path_rectangle_corners:nnnn 583
\c__draw_path_arc_90_fp	345	__draw_path_rectangle_corners:nnnnn 586 , 589
__draw_path_arc_add:nnnn	345	__draw_path_rectangle_rounded:nnnn 554
__draw_path_arc_aux_add:nn 452 , 458 , 470 , 475	__draw_path_reset_limits: 184 , 658 , 757 , 1235
__draw_path_arc_auxi:nnnnNnn 345 , 372 , 379	\l__draw_path_tmp_tl 175 , 423 , 446 , 465 , 469 , 473 , 477
__draw_path_arc_auxii:nnnNnnnn	345	\l__draw_path_tmpa_fp 175 , 311 , 321 , 333
__draw_path_arc_auxiii:nn	345	\l__draw_path_tmpb_fp 175 , 312 , 328 , 337
__draw_path_arc_auxiv:nnnn	345	__draw_path_update_last:nn 213 , 250 , 263 , 282 , 572
__draw_path_arc_auxv:nn	345	__draw_path_update_limits:nn 24 , 27 , 30 , 33 , 184 , 248 , 261 , 278 , 279 , 280 , 569 , 570
__draw_path_arc_auxvi:nn	345	__draw_path_use:n	646
\l__draw_path_arc_delta_fp	345	__draw_path_use_action_draw: . .	646
\l__draw_path_arc_start_fp	345	__draw_path_use_action_fillstroke: 646
__draw_path_curveto:nnnn	302	\l__draw_path_use_bb_bool	644
__draw_path_curveto:nnnnnn	240 , 298 , 316 , 446 , 513 , 523 , 533 , 543	\l__draw_path_use_clear_bool	644 , 703
\c__draw_path_curveto_a_fp	302	\l__draw_path_use_clip_bool 641 , 664 , 682
\c__draw_path_curveto_b_fp	302	\l__draw_path_use_fill_bool 641 , 665 , 687 , 692 , 698 , 712
__draw_path_ellipse:nnnnnn	489	__draw_path_use_stroke_bb:	646
__draw_path_ellipse_arci:nnnnnn	489	__draw_path_use_stroke_bb-	aux:NnN 646
__draw_path_ellipse_arcii:nnnnnn 489	\l__draw_path_use_stroke_bool 641 , 666 , 679 , 688 , 693 , 699 , 708 , 713
__draw_path_ellipse_arciiii:nnnnnn 489	\g__draw_path_xmax_dim 180 , 186 , 187 , 208 , 751 , 773
__draw_path_ellipse_arciv:nnnnnn 489	\l__draw_path_xmax_dim	737 , 751 , 773
\c__draw_path_ellipse_fp	489	\g__draw_path_xmin_dim 180 , 188 , 189 , 209 , 752 , 774
__draw_path_grid_auxi:nnnnnn . .	591		
__draw_path_grid_auxii:nnnnnn . .	591		

<code>\l__draw_path_xmin_dim .</code>	737 , 752 , 774	<code>__draw_point_intersect_circles_-</code>	
<code>\g__draw_path_ymax_dim</code>		<code>auxvii:nnnnnnn</code>	896
<code>.</code>	180 , 190 , 191 , 210 , 753 , 775	<code>__draw_point_intersect_lines:nnnnnn</code>	
<code>\l__draw_path_ymax_dim .</code>	737 , 753 , 775	<code>.</code>	871
<code>\g__draw_path_ymin_dim</code>		<code>__draw_point_intersect_lines:nnnnnnnn</code>	
<code>.</code>	180 , 192 , 193 , 211 , 754 , 776	<code>.</code>	871
<code>\l__draw_path_ymin_dim .</code>	737 , 754 , 776	<code>__draw_point_intersect_lines_-</code>	
<code>__draw_point_interpolate_-</code>		<code>aux:nnnnnn</code>	871
<code>arcaxes_auxi:nnnnnnnn</code>	985	<code>__draw_point_process:nn</code>	
<code>__draw_point_interpolate_-</code>		<code>.</code>	23 , 26 , 29 , 32 , 242 , 254 ,
<code>arcaxes_auxii:nnnnnnnn</code>	985	<code>.</code>	290 , 292 , 424 , 440 , 790 , 855 , 969 ,
<code>__draw_point_interpolate_-</code>		<code>.</code>	975 , 1101 , 1156 , 1188 , 1774 , 1806 , 1852
<code>arcaxes_auxiii:nnnnnn</code>	985	<code>__draw_point_process:nnn</code>	304 ,
<code>__draw_point_interpolate_-</code>		<code>.</code>	430 , 556 , 585 , 593 , 790 , 898 , 954 , 1850
<code>arcaxes_auxiv:nnnnnnnn</code>	985	<code>__draw_point_process:nnnn</code>	
<code>__draw_point_interpolate_curve_-</code>		<code>.</code>	267 , 295 , 491 , 790 , 987
<code>auxi:nnnnnnnn</code>	1020 , 1023	<code>__draw_point_process:nnnnn</code>	
<code>__draw_point_interpolate_curve_-</code>		<code>.</code>	790 , 873 , 1019
<code>auxii:nnnnnnnn</code>	1025 , 1029 , 1037	<code>__draw_point_process_auxi:nn</code>	790
<code>__draw_point_interpolate_curve_-</code>		<code>__draw_point_process_auxii:nw</code>	790
<code>auxiii:nnnnnn</code>	1032 , 1043 , 1051	<code>__draw_point_process_auxiii:nnn</code>	790
<code>__draw_point_interpolate_curve_-</code>		<code>__draw_point_process_auxiv:nw</code>	790
<code>auxiv:nnnnnn</code>	1045 , 1046 , 1047 , 1052	<code>__draw_point_process_auxv:nnnn</code>	790
<code>__draw_point_interpolate_curve_-</code>		<code>__draw_point_process_auxvi:nw</code>	790
<code>auxv:nnw</code>	1054 , 1058 , 1065	<code>__draw_point_process_auxvii:nnnnn</code>	790
<code>__draw_point_interpolate_curve_-</code>		<code>.</code>	790
<code>auxvi:n</code>	1049 , 1070	<code>__draw_point_process_auxviii:nw</code>	790
<code>__draw_point_interpolate_curve_-</code>		<code>__draw_point_to_dim:n</code>	841
<code>auxvii:nnnnnnnn</code>	1071 , 1072	<code>__draw_point_to_dim:w</code>	841
<code>__draw_point_interpolate_curve_-</code>		<code>__draw_point_transform:nn</code>	1154
<code>auxviii:nnnnnn</code>	1074 , 1081 , 1086	<code>__draw_point_transform_noshift:n</code>	427 , 443 , 494 , 495 , 1186
<code>__draw_point_interpolate_-</code>		<code>__draw_point_transform_noshift:nn</code>	1186
<code>distance:nnnn</code>	970 , 973	<code>__draw_point_unit_vector:nn</code>	854
<code>__draw_point_interpolate_-</code>		<code>__draw_point_unit_vector:nnnn</code>	854
<code>distance:nnnn</code>	967 , 977	<code>__draw_point_vec:nn</code>	1111
<code>__draw_point_interpolate_-</code>		<code>__draw_point_vec:nnn</code>	1111
<code>distance:nnnnnn</code>	967	<code>__draw_point_vec_polar:nnn</code>	1141
<code>__draw_point_interpolate_line_-</code>		<code>__draw_reset_bb:</code>	1221 , 1234 , 1349
<code>aux:nnnnn</code>	952	<code>__draw_scope_bb_begin:</code>	1342 , 1361
<code>__draw_point_interpolate_line_-</code>		<code>__draw_scope_bb_end:</code>	1342 , 1371
<code>aux:nnnnnn</code>	952	<code>__draw_softpath_add:n</code>	
<code>__draw_point_intersect_circles_-</code>		<code>.</code>	770 , 1379 , 1398 ,
<code>auxi:nnnnnnnn</code>	896	<code>.</code>	1407 , 1416 , 1421 , 1431 , 1439 , 1694
<code>__draw_point_intersect_circles_-</code>		<code>\c__draw_softpath_arc_fp</code>	
<code>auxii:nnnnnnnn</code>	896	<code>.</code>	1473 , 1636 , 1640 , 1645 , 1649
<code>__draw_point_intersect_circles_-</code>		<code>__draw_softpath_clear:</code>	
<code>auxiii:nnnnnnnn</code>	896	<code>.</code>	657 , 704 , 762 , 766 , 1239 , 1382
<code>__draw_point_intersect_circles_-</code>		<code>__draw_softpath_close_op:nn</code>	
<code>auxiv:nnnnnnnn</code>	896	<code>.</code>	1400 , 1444 , 1535 , 1571 , 1673
<code>__draw_point_intersect_circles_-</code>		<code>__draw_softpath_closepath:</code>	
<code>auxv:nnnnnnnnnn</code>	896	<code>.</code>	287 , 508 , 1396
<code>__draw_point_intersect_circles_-</code>			
<code>auxvi:nnnnnnnn</code>	896		

\l__draw_softpath_corneri_dim ...	\l__draw_softpath_part_tl
..... 1467 , 1520 , 1523 , 1609 1465 , 1480 ,
\l__draw_softpath_cornerii_dim ..	1509 , 1512 , 1514 , 1548 , 1603 , 1692
..... 1467 , 1521 , 1524 , 1600	__draw_softpath_rectangle:nnnn ..
\g__draw_softpath_corners_bool 571 , 1396
761 , 768 , 1378 , 1390 , 1441 , 1476 , 1492	__draw_softpath_rectangle_-
\l__draw_softpath_corners_bool ..	opi:nn 1433 , 1444
..... 737 , 760 , 769	__draw_softpath_rectangle_-
\l__draw_softpath_curve_end_tl ..	opi:nnNnn 1444
..... 1466 , 1597 , 1616 , 1665 , 1676	__draw_softpath_rectangle_-
__draw_softpath_curveto:nnnnnn ..	opii:nn 1434 , 1444
..... 281 , 1396	__draw_softpath_round_action:nn
__draw_softpath_curveto_opi:nn 1474
..... 1409 , 1444 , 1532 , 1570 , 1633	__draw_softpath_round_action:Nnn
__draw_softpath_curveto_- 1474
opi:nnNnnNnn 1444	__draw_softpath_round_action_-
__draw_softpath_curveto_opii:nn	close: 1474
..... 1410 , 1444 , 1642	__draw_softpath_round_action_-
__draw_softpath_curveto_-	curveto:NnnNnn 1474
opiii:nn 1411 , 1444 , 1651	__draw_softpath_round_calc:NnnNnn
\l__draw_softpath_first_tl 1474
..... 1467 , 1483 , 1500 ,	__draw_softpath_round_calc:nnnnnn
1501 , 1511 , 1530 , 1531 , 1557 , 1561 1474
\l__draw_softpath_internal_tl ...	__draw_softpath_round_calc:nnnnw
..... 1377 , 1384 , 1385 , 1485 , 1487 1474
\g__draw_softpath_lastx_dim	__draw_softpath_round_close:nn 1474
..... 755 , 771 , 1392 , 1401 , 1425	__draw_softpath_round_close:w 1474
\l__draw_softpath_lastx_dim	__draw_softpath_round_corners: ..
..... 743 , 755 , 771 676 , 1474
\l__draw_softpath_lastx_fp	__draw_softpath_round_end: .. 1474
.. 1467 , 1481 , 1502 , 1550 , 1612 , 1619	__draw_softpath_round_lookahead:NnnNnn
\g__draw_softpath_lasty_dim 1474
..... 756 , 772 , 1392 , 1402 , 1426	__draw_softpath_round_loop:Nnn 1474
\l__draw_softpath_lasty_dim	__draw_softpath_round_roundpoint:NnnNnnNnn
..... 744 , 756 , 772 1474
\l__draw_softpath_lasty_fp	__draw_softpath_roundpoint:nn ..
.. 1467 , 1482 , 1503 , 1551 , 1613 , 1620 235 , 1396
__draw_softpath_lineto:nn 262 , 1396	__draw_softpath_roundpoint_-
__draw_softpath_lineto_op:nn ...	op:nn 1440 , 1444 , 1497 , 1579
..... 1417 , 1444 , 1538 , 1569 , 1682	__draw_softpath_use: 681 , 1382
\g__draw_softpath_main_tl	\l__draw_stroke_color_tl 1321
758 , 1376 , 1380 , 1384 , 1389 , 1485 , 1693	\l__draw_tmp_box 13 , 37 , 48 ,
\l__draw_softpath_main_tl	52 , 54 , 55 , 56 , 57 , 63 , 65 , 66 , 67 , 68 , 69
..... 19 , 758 , 770 , 1464 ,	\l__draw_tmp_seq 1707
1479 , 1506 , 1508 , 1689 , 1691 , 1694	__draw_tranform_triangle:nnnnnn
\g__draw_softpath_move_bool 1853 , 1858
..... 1394 , 1423	__draw_transform:nnnn 1782
\l__draw_softpath_move_tl	__draw_transform_invert:n ... 1814
..... 1467 , 1484 ,	__draw_transform_rotate:n ... 1884
1507 , 1510 , 1558 , 1660 , 1683 , 1690	__draw_transform_rotate:nn .. 1884
__draw_softpath_moveto:nn 249 , 1396	__draw_transform_shift:nn ... 1782
__draw_softpath_moveto_op:nn ...	__draw_transform_shift_absolute:nn
..... 1422 , 1444 , 1504 , 1662 1756
	__draw_vec:nn 1093

