File I
Implementation

1 **l3draw** implementation

1 ⟨*initex | package⟩

2 ⟨@@=draw⟩

3 ⟨*package⟩
4 \ProvidesExplPackage{l3draw}{2018-11-19}{}
5   {L3 Experimental core drawing support}
6 ⟨/package⟩

7 \RequirePackage { l3color }

Everything else is in the sub-files!

8 ⟨/initex | package⟩

2 **l3draw-paths** implementation

9 ⟨*initex | package⟩

10 ⟨@@=draw⟩

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

- \pgfpatharcto, \pgfpatharctoprecomputed: These are extremely specialised and are very complex in implementation. If the functionality is required, it is likely that it will be set up from scratch here.

- \pgfpathparabola: Seems to be unused other than defining a TikZ interface, which itself is then not used further.

- \pgfpathsine, \pgfpathcosine: Need to see exactly how these need to work, in particular whether a wider input range is needed and what approximation to make.

- \pgfpathcurvebetweentime, \pgfpathcurvebetweentimecontinue: These don't seem to be used at all.

\l__draw_path_tmp_tl
\l__draw_path_tmpa_fp
\l__draw_path_tmpb_fp

Scratch space.

11 \tl_new:N \l__draw_path_tmp_tl
12 \fp_new:N \l__draw_path_tmpa_fp
13 \fp_new:N \l__draw_path_tmpb_fp

(*End definition for* \l__draw_path_tmp_tl*,* \l__draw_path_tmpa_fp*, and* \l__draw_path_tmpb_fp*.*)

1

2.1 Tracking paths

`\g__draw_path_lastx_dim`
`\g__draw_path_lasty_dim`

The last point visited on a path.

```
14 \dim_new:N \g__draw_path_lastx_dim
15 \dim_new:N \g__draw_path_lasty_dim
```

(*End definition for* `\g__draw_path_lastx_dim` *and* `\g__draw_path_lasty_dim`.)

`\g__draw_path_xmax_dim`
`\g__draw_path_xmin_dim`
`\g__draw_path_ymax_dim`
`\g__draw_path_ymin_dim`

The limiting size of a path.

```
16 \dim_new:N \g__draw_path_xmax_dim
17 \dim_new:N \g__draw_path_xmin_dim
18 \dim_new:N \g__draw_path_ymax_dim
19 \dim_new:N \g__draw_path_ymin_dim
```

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

`\__draw_path_update_limits:nn`
`\__draw_path_reset_limits:`

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.)

```
20 \cs_new_protected:Npn \__draw_path_update_limits:nn #1#2
21   {
22     \dim_gset:Nn \g__draw_path_xmax_dim
23       { \dim_max:nn \g__draw_path_xmax_dim {#1} }
24     \dim_gset:Nn \g__draw_path_xmin_dim
25       { \dim_min:nn \g__draw_path_xmin_dim {#1} }
26     \dim_gset:Nn \g__draw_path_ymax_dim
27       { \dim_max:nn \g__draw_path_ymax_dim {#2} }
28     \dim_gset:Nn \g__draw_path_ymin_dim
29       { \dim_min:nn \g__draw_path_ymin_dim {#2} }
30     \bool_if:NT \l_draw_bb_update_bool
31       {
32         \dim_gset:Nn \g__draw_xmax_dim
33           { \dim_max:nn \g__draw_xmax_dim {#1} }
34         \dim_gset:Nn \g__draw_xmin_dim
35           { \dim_min:nn \g__draw_xmin_dim {#1} }
36         \dim_gset:Nn \g__draw_ymax_dim
37           { \dim_max:nn \g__draw_ymax_dim {#2} }
38         \dim_gset:Nn \g__draw_ymin_dim
39           { \dim_min:nn \g__draw_ymin_dim {#2} }
40       }
41   }
42 \cs_new_protected:Npn \__draw_path_reset_limits:
43   {
44     \dim_gset:Nn \g__draw_path_xmax_dim { -\c_max_dim }
45     \dim_gset:Nn \g__draw_path_xmin_dim {  \c_max_dim }
46     \dim_gset:Nn \g__draw_path_ymax_dim { -\c_max_dim }
47     \dim_gset:Nn \g__draw_path_ymin_dim {  \c_max_dim }
48   }
```

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

`\__draw_path_update_last:nn`

A simple auxiliary to avoid repetition.

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

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

2.2 Corner arcs

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

`\l__draw_corner_xarc_dim`
`\l__draw_corner_yarc_dim`

The two arcs in use.

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

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

`\l__draw_corner_arc_bool`

A flag to speed up the repeated checks.

```
56 \bool_new:N \l__draw_corner_arc_bool
```

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

`\draw_path_corner_arc:nn`

Calculate the arcs, check they are non-zero.

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

(*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.

```
67 \cs_new_protected:Npn \__draw_path_mark_corner:
68   {
69     \bool_if:NT \l__draw_corner_arc_bool
70       {
71         \__draw_softpath_roundpoint:VV
72           \l__draw_corner_xarc_dim
73           \l__draw_corner_yarc_dim
74       }
75   }
```

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

2.3 Basic path constructions

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

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

```
76 \cs_new_protected:Npn \draw_path_moveto:n #1
77   {
78     \__draw_point_process:nn
79       { \__draw_path_moveto:nn }
80       { \draw_point_transform:n {#1} }
81   }
82 \cs_new_protected:Npn \__draw_path_moveto:nn #1#2
83   {
84     \__draw_path_update_limits:nn {#1} {#2}
85     \__draw_softpath_moveto:nn {#1} {#2}
86     \__draw_path_update_last:nn {#1} {#2}
87   }
88 \cs_new_protected:Npn \draw_path_lineto:n #1
89   {
90     \__draw_point_process:nn
91       { \__draw_path_lineto:nn }
92       { \draw_point_transform:n {#1} }
93   }
94 \cs_new_protected:Npn \__draw_path_lineto:nn #1#2
95   {
96     \__draw_path_mark_corner:
97     \__draw_path_update_limits:nn {#1} {#2}
98     \__draw_softpath_lineto:nn {#1} {#2}
99     \__draw_path_update_last:nn {#1} {#2}
100  }
101 \cs_new_protected:Npn \draw_path_curveto:nnn #1#2#3
102  {
103    \__draw_point_process:nnn
104      {
105        \__draw_point_process:nn
106          {
107            \__draw_path_mark_corner:
108            \__draw_path_curveto:nnnnnn
109          }
110          { \draw_point_transform:n {#1} }
111      }
112      { \draw_point_transform:n {#2} }
113      { \draw_point_transform:n {#3} }
114  }
115 \cs_new_protected:Npn \__draw_path_curveto:nnnnnn #1#2#3#4#5#6
116  {
117    \__draw_path_update_limits:nn {#1} {#2}
118    \__draw_path_update_limits:nn {#3} {#4}
119    \__draw_path_update_limits:nn {#5} {#6}
120    \__draw_softpath_curveto:nnnnnn {#1} {#2} {#3} {#4} {#5} {#6}
121    \__draw_path_update_last:nn {#5} {#6}
122  }
```

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

4

`\draw_path_close:` A simple wrapper.

```
123 \cs_new_protected:Npn \draw_path_close:
124   {
125     \__draw_path_mark_corner:
126     \__draw_softpath_closepath:
127   }
```

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

2.4   Canvas path constructions

`\draw_path_canvas_moveto:n`
`\draw_path_canvas_lineto:n`
`\draw_path_canvas_curveto:nnn`

Operations with no application of the transformation matrix.

```
128 \cs_new_protected:Npn \draw_path_canvas_moveto:n #1
129   { \__draw_point_process:nn { \__draw_path_moveto:nn } {#1} }
130 \cs_new_protected:Npn \draw_path_canvas_lineto:n #1
131   { \__draw_point_process:nn { \__draw_path_lineto:nn } {#1} }
132 \cs_new_protected:Npn \draw_path_canvas_curveto:nnn #1#2#3
133   {
134     \__draw_point_process:nnn
135       {
136         \__draw_point_process:nn
137           {
138             \__draw_path_mark_corner:
139             \__draw_path_curveto:nnnnnn
140           }
141         {#1}
142       }
143     {#2} {#3}
144   }
```

(*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* **??**.)

2.5   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_s, y_s).

```
145 \cs_new_protected:Npn \draw_path_curveto:nn #1#2
146   {
147     \__draw_point_process:nnn
148       { \__draw_path_curveto:nnnn }
149       { \draw_point_transform:n {#1} }
150       { \draw_point_transform:n {#2} }
151   }
```

```
152 \cs_new_protected:Npn \__draw_path_curveto:nnnn #1#2#3#4
153   {
154     \fp_set:Nn \l__draw_path_tmpa_fp { \c__draw_path_curveto_b_fp * #1 }
155     \fp_set:Nn \l__draw_path_tmpb_fp { \c__draw_path_curveto_b_fp * #2 }
156     \use:x
157       {
158         \__draw_path_mark_corner:
159         \__draw_path_curveto:nnnnnn
160           {
161             \fp_to_dim:n
162               {
163                 \c__draw_path_curveto_a_fp * \g__draw_path_lastx_dim
164                 + \l__draw_path_tmpa_fp
165               }
166           }
167           {
168             \fp_to_dim:n
169               {
170                 \c__draw_path_curveto_a_fp * \g__draw_path_lasty_dim
171                 + \l__draw_path_tmpb_fp
172               }
173           }
174           {
175             \fp_to_dim:n
176               { \c__draw_path_curveto_a_fp * #3 + \l__draw_path_tmpa_fp }
177           }
178           {
179             \fp_to_dim:n
180               { \c__draw_path_curveto_a_fp * #4 + \l__draw_path_tmpb_fp }
181           }
182           {#3}
183           {#4}
184       }
185   }
186 \fp_const:Nn \c__draw_path_curveto_a_fp { 1 / 3 }
187 \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* **??***.*)

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°.

```
188 \cs_new_protected:Npn \draw_path_arc:nnn #1#2#3
189   { \draw_path_arc:nnnn {#1} {#2} {#3} {#3} }
190 \cs_new_protected:Npn \draw_path_arc:nnnn #1#2#3#4
191   {
192     \use:x
193       {
194         \__draw_path_arc:nnnn
195           { \fp_eval:n {#1} }
196           { \fp_eval:n {#2} }
197           { \fp_to_dim:n {#3} }
198           { \fp_to_dim:n {#4} }
199       }
```

6

```
200    }
201 \cs_new_protected:Npn \__draw_path_arc:nnnn #1#2#3#4
202    {
203      \fp_compare:nNnTF {#1} > {#2}
204        { \__draw_path_arc:nnNnn {#1} {#2} - {#3} {#4} }
205        { \__draw_path_arc:nnNnn {#1} {#2} + {#3} {#4} }
206    }
207 \cs_new_protected:Npn \__draw_path_arc:nnNnn #1#2#3#4#5
208    {
209      \fp_set:Nn \l__draw_path_arc_start_fp {#1}
210      \fp_set:Nn \l__draw_path_arc_delta_fp { abs( #1 - #2 ) }
211      \fp_while_do:nNnn { \l__draw_path_arc_delta_fp } > { 90 }
212        {
213          \fp_compare:nNnTF \l__draw_path_arc_delta_fp > { 115 }
214            {
215              \__draw_path_arc_auxi:ffnnNnn
216                { \fp_to_decimal:N \l__draw_path_arc_start_fp }
217                { \fp_eval:n { \l__draw_path_arc_start_fp #3 90 } }
218                { 90 } {#2}
219                #3 {#4} {#5}
220            }
221            {
222              \__draw_path_arc_auxi:ffnnNnn
223                { \fp_to_decimal:N \l__draw_path_arc_start_fp }
224                { \fp_eval:n { \l__draw_path_arc_start_fp #3 60 } }
225                { 60 } {#2}
226                #3 {#4} {#5}
227            }
228        }
229      \__draw_path_mark_corner:
230      \__draw_path_arc_auxi:fnfnNnn
231        { \fp_to_decimal:N \l__draw_path_arc_start_fp }
232        {#2}
233        { \fp_eval:n { abs( \l__draw_path_arc_start_fp - #2 ) } }
234        {#2}
235        #3 {#4} {#5}
236    }
```

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.

```
237 \cs_new_protected:Npn \__draw_path_arc_auxi:nnnnNnn #1#2#3#4#5#6#7
238    {
239      \use:x
240        {
241          \__draw_path_arc_auxii:nnnNnnnn
242            {#1} {#2} {#4} #5 {#6} {#7}
243            {
244              \fp_to_dim:n
245                {
246                  \cs_if_exist_use:cF
247                    { c__draw_path_arc_ #3 _fp }
248                    { 4/3 * tand( 0.25 * #3 ) }
```

```
249              * #6
250          }
251        }
252        {
253          \fp_to_dim:n
254            {
255              \cs_if_exist_use:cF
256                { c__draw_path_arc_ #3 _fp }
257                { 4/3 * tand( 0.25 * #3 ) }
258              * #7
259            }
260        }
261      }
262    }
263 \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

```
264 \cs_new_protected:Npn \__draw_path_arc_auxii:nnnNnnnn #1#2#3#4#5#6#7#8
265   {
266     \tl_clear:N \l__draw_path_tmp_tl
267     \__draw_point_process:nn
268       { \__draw_path_arc_auxiii:nn }
269       {
270         \__draw_point_transform_noshift:n
271           { \draw_point_polar:nnn { #1 #4 90 } {#7} {#8} }
272       }
273     \__draw_point_process:nn
274       {
275         \__draw_point_process:nn
276           { \__draw_path_arc_auxiv:nnnn }
277           {
278             \draw_point_transform:n
279               { \draw_point_polar:nnn {#1} {#5} {#6} }
280           }
281       }
282       {
283         \draw_point_transform:n
284           { \draw_point_polar:nnn {#2} {#5} {#6} }
285       }
286     \__draw_point_process:nn
287       { \__draw_path_arc_auxv:nn }
288       {
289         \__draw_point_transform_noshift:n
290           { \draw_point_polar:nnn { #2 #4 -90 } {#7} {#8} }
291       }
292     \exp_after:wN \__draw_path_curveto:nnnnnn \l__draw_path_tmp_tl
293     \fp_set:Nn \l__draw_path_arc_delta_fp { abs ( #2 - #3 ) }
294     \fp_set:Nn \l__draw_path_arc_start_fp {#2}
295   }
```

The first control point.

```
296 \cs_new_protected:Npn \__draw_path_arc_auxiii:nn #1#2
297   {
298     \__draw_path_arc_aux_add:nn
299       { \g__draw_path_lastx_dim + #1 }
300       { \g__draw_path_lasty_dim + #2 }
301   }
```

The end point: simple arithmetic.

```
302 \cs_new_protected:Npn \__draw_path_arc_auxiv:nnnn #1#2#3#4
303   {
304     \__draw_path_arc_aux_add:nn
305       { \g__draw_path_lastx_dim - #1 + #3 }
306       { \g__draw_path_lasty_dim - #2 + #4 }
307   }
```

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

```
308 \cs_new_protected:Npn \__draw_path_arc_auxv:nn #1#2
309   {
310     \exp_after:wN \__draw_path_arc_auxvi:nn
311       \l__draw_path_tmp_tl {#1} {#2}
312   }
313 \cs_new_protected:Npn \__draw_path_arc_auxvi:nn #1#2#3#4#5#6
314   {
315     \tl_set:Nn \l__draw_path_tmp_tl { {#1} {#2} }
316     \__draw_path_arc_aux_add:nn
317       { #5 + #3 }
318       { #6 + #4 }
319     \tl_put_right:Nn \l__draw_path_tmp_tl { {#3} {#4} }
320   }
321 \cs_new_protected:Npn \__draw_path_arc_aux_add:nn #1#2
322   {
323     \tl_put_right:Nx \l__draw_path_tmp_tl
324       { { \fp_to_dim:n {#1} } { \fp_to_dim:n {#2} } }
325   }
326 \fp_new:N \l__draw_path_arc_delta_fp
327 \fp_new:N \l__draw_path_arc_start_fp
328 \fp_const:cn { c__draw_path_arc_90_fp } { 4/3 * (sqrt(2) - 1) }
329 \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.

```
330 \cs_new_protected:Npn \draw_path_arc_axes:nnnn #1#2#3#4
331   {
332     \draw_transform_triangle:nnn { 0cm , 0cm } {#3} {#4}
333     \draw_path_arc:nnn {#1} {#2} { 1pt }
334   }
```

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

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

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

```
335 \cs_new_protected:Npn \draw_path_ellipse:nnn #1#2#3
```

9

```
336    {
337      \__draw_point_process:nnn
338        {
339          \__draw_point_process:nn
340            { \__draw_path_ellipse:nnnnnn }
341            { \draw_point_transform:n {#1} }
342        }
343        { \__draw_point_transform_noshift:n {#2} }
344        { \__draw_point_transform_noshift:n {#3} }
345    }
346  \cs_new_protected:Npn \__draw_path_ellipse:nnnnnn #1#2#3#4#5#6
347    {
348      \use:x
349        {
350          \__draw_path_moveto:nn
351            { \fp_to_dim:n { #1 + #3 } } { \fp_to_dim:n { #2 + #4 } }
352          \__draw_path_ellipse_arci:nnnnnn    {#1} {#2} {#3} {#4} {#5} {#6}
353          \__draw_path_ellipse_arcii:nnnnnn   {#1} {#2} {#3} {#4} {#5} {#6}
354          \__draw_path_ellipse_arciii:nnnnnn  {#1} {#2} {#3} {#4} {#5} {#6}
355          \__draw_path_ellipse_arciv:nnnnnn   {#1} {#2} {#3} {#4} {#5} {#6}
356        }
357      \__draw_softpath_closepath:
358      \__draw_path_moveto:nn {#1} {#2}
359    }
360  \cs_new:Npn \__draw_path_ellipse_arci:nnnnnn #1#2#3#4#5#6
361    {
362      \__draw_path_curveto:nnnnnn
363        { \fp_to_dim:n { #1 + #3 + #5 * \c__draw_path_ellipse_fp } }
364        { \fp_to_dim:n { #2 + #4 + #6 * \c__draw_path_ellipse_fp } }
365        { \fp_to_dim:n { #1 + #3 * \c__draw_path_ellipse_fp + #5 } }
366        { \fp_to_dim:n { #2 + #4 * \c__draw_path_ellipse_fp + #6 } }
367        { \fp_to_dim:n { #1 + #5 } }
368        { \fp_to_dim:n { #2 + #6 } }
369    }
370  \cs_new:Npn \__draw_path_ellipse_arcii:nnnnnn #1#2#3#4#5#6
371    {
372      \__draw_path_curveto:nnnnnn
373        { \fp_to_dim:n { #1 - #3 * \c__draw_path_ellipse_fp + #5 } }
374        { \fp_to_dim:n { #2 - #4 * \c__draw_path_ellipse_fp + #6 } }
375        { \fp_to_dim:n { #1 - #3 + #5 * \c__draw_path_ellipse_fp } }
376        { \fp_to_dim:n { #2 - #4 + #6 * \c__draw_path_ellipse_fp } }
377        { \fp_to_dim:n { #1 - #3 } }
378        { \fp_to_dim:n { #2 - #4 } }
379    }
380  \cs_new:Npn \__draw_path_ellipse_arciii:nnnnnn #1#2#3#4#5#6
381    {
382      \__draw_path_curveto:nnnnnn
383        { \fp_to_dim:n { #1 - #3 - #5 * \c__draw_path_ellipse_fp } }
384        { \fp_to_dim:n { #2 - #4 - #6 * \c__draw_path_ellipse_fp } }
385        { \fp_to_dim:n { #1 - #3 * \c__draw_path_ellipse_fp - #5 } }
386        { \fp_to_dim:n { #2 - #4 * \c__draw_path_ellipse_fp - #6 } }
387        { \fp_to_dim:n { #1 - #5 } }
388        { \fp_to_dim:n { #2 - #6 } }
389    }
```

```
390 \cs_new:Npn \__draw_path_ellipse_arciv:nnnnnn #1#2#3#4#5#6
391   {
392     \__draw_path_curveto:nnnnnn
393       { \fp_to_dim:n { #1 + #3 * \c__draw_path_ellipse_fp - #5 } }
394       { \fp_to_dim:n { #2 + #4 * \c__draw_path_ellipse_fp - #6 } }
395       { \fp_to_dim:n { #1 + #3 - #5 * \c__draw_path_ellipse_fp } }
396       { \fp_to_dim:n { #2 + #4 - #6 * \c__draw_path_ellipse_fp } }
397       { \fp_to_dim:n { #1 + #3 } }
398       { \fp_to_dim:n { #2 + #4 } }
399   }
400 \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.

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

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

2.6 Rectangles

\draw_path_rectangle:nn
__draw_path_rectangle:nnnn
__draw_path_rectangle_rounded:nnnn

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

```
403 \cs_new_protected:Npn \draw_path_rectangle:nn #1#2
404   {
405     \__draw_point_process:nnn
406       {
407         \bool_lazy_or:nnTF
408           { \l__draw_corner_arc_bool }
409           { \l__draw_matrix_active_bool }
410           { \__draw_path_rectangle_rounded:nnnn }
411           { \__draw_path_rectangle:nnnn }
412       }
413       { \draw_point_transform:n {#1} }
414       {#2}
415   }
416 \cs_new_protected:Npn \__draw_path_rectangle:nnnn #1#2#3#4
417   {
418     \__draw_path_update_limits:nn {#1} {#2}
419     \__draw_path_update_limits:nn { #1 + #3 } { #2 + #4 }
420     \__draw_softpath_rectangle:nnnn {#1} {#2} {#3} {#4}
421     \__draw_path_update_last:nn {#1} {#2}
422   }
423 \cs_new_protected:Npn \__draw_path_rectangle_rounded:nnnn #1#2#3#4
424   {
425     \draw_path_moveto:n { #1 + #3 , #2 + #4 }
426     \draw_path_lineto:n { #1 , #2 + #4 }
427     \draw_path_lineto:n { #1 , #2 }
428     \draw_path_lineto:n { #1 + #3 , #2 }
429     \draw_path_close:
430     \draw_path_moveto:n { #1 , #2 }
431   }
```

(*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.

```
432 \cs_new_protected:Npn \draw_path_rectangle_corners:nn #1#2
433   {
434     \__draw_point_process:nnn
435       { \__draw_path_rectangle_corners:nnnnn {#1} }
436       {#1} {#2}
437   }
438 \cs_new_protected:Npn \__draw_path_rectangle_corners:nnnnn #1#2#3#4#5
439   { \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* **??**.)

2.7 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.

```
440 \cs_new_protected:Npn \draw_path_grid:nnnn #1#2#3#4
441   {
442     \__draw_point_process:nnn
443       {
444         \__draw_path_grid_auxi:ffnnnn
445           { \dim_eval:n { \dim_abs:n {#1} } }
446           { \dim_eval:n { \dim_abs:n {#2} } }
447       }
448       {#3} {#4}
449   }
450 \cs_new_protected:Npn \__draw_path_grid_auxi:nnnnnn #1#2#3#4#5#6
451   {
452     \dim_compare:nNnTF {#3} > {#5}
453       { \__draw_path_grid_auxii:nnnnnn {#1} {#2} {#5} {#4} {#3} {#6} }
454       { \__draw_path_grid_auxii:nnnnnn {#1} {#2} {#3} {#4} {#5} {#6} }
455   }
456 \cs_generate_variant:Nn \__draw_path_grid_auxi:nnnnnn { ff }
457 \cs_new_protected:Npn \__draw_path_grid_auxii:nnnnnn #1#2#3#4#5#6
458   {
459     \dim_compare:nNnTF {#4} > {#6}
460       { \__draw_path_grid_auxiii:nnnnnn {#1} {#2} {#3} {#6} {#5} {#4} }
461       { \__draw_path_grid_auxiii:nnnnnn {#1} {#2} {#3} {#4} {#5} {#6} }
462   }
463 \cs_new_protected:Npn \__draw_path_grid_auxiii:nnnnnn #1#2#3#4#5#6
464   {
465     \__draw_path_grid_auxiv:ffnnnnnn
466       { \fp_to_dim:n { #1 * trunc(#3/(#1)) } }
467       { \fp_to_dim:n { #2 * trunc(#4/(#2)) } }
468       {#1} {#2} {#3} {#4} {#5} {#6}
469   }
470 \cs_new_protected:Npn \__draw_path_grid_auxiv:nnnnnnnn #1#2#3#4#5#6#7#8
471   {
472     \dim_step_inline:nnnn
473       {#1}
```

12

```
474        {#3}
475        {#7}
476        {
477          \draw_path_moveto:n { ##1 , #6 }
478          \draw_path_lineto:n { ##1 , #8 }
479        }
480     \dim_step_inline:nnnn
481        {#2}
482        {#4}
483        {#8}
484        {
485          \draw_path_moveto:n { #5 , ##1 }
486          \draw_path_lineto:n { #7 , ##1 }
487        }
488   }
489 \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* **??**.)

2.8   Using paths

\l__draw_path_use_clip_bool   Actions to pass to the driver.
\l__draw_path_use_fill_bool
\l__draw_path_use_stroke_bool
```
490 \bool_new:N \l__draw_path_use_clip_bool
491 \bool_new:N \l__draw_path_use_fill_bool
492 \bool_new:N \l__draw_path_use_stroke_bool
```

(*End definition for* \l__draw_path_use_clip_bool, \l__draw_path_use_fill_bool, *and* \l__draw_-
path_use_stroke_bool.)

\l__draw_path_use_bb_bool   Actions handled at the macro layer.
\l__draw_path_use_clear_bool
```
493 \bool_new:N \l__draw_path_use_bb_bool
494 \bool_new:N \l__draw_path_use_clear_bool
```

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

\draw_path_use:n   There are a range of actions which can apply to a path: they are handled in a single
\draw_path_use_clear:n   function which can carry out several of them. The first step is to deal with the special
__draw_path_use:n   case of clearing the path.
__draw_path_use_action_draw:
__draw_path_use_action_fillstroke:
__draw_path_use_stroke_bb:
__draw_path_use_stroke_bb_aux:NnN
```
495 \cs_new_protected:Npn \draw_path_use:n #1
496   {
497     \tl_if_blank:nF {#1}
498       { \__draw_path_use:n {#1} }
499   }
500 \cs_new_protected:Npn \draw_path_use_clear:n #1
501   {
502     \bool_lazy_or:nnTF
503       { \tl_if_blank_p:n {#1} }
504       { \str_if_eq_p:nn {#1} { clear } }
505       {
506         \__draw_softpath_clear:
507         \__draw_path_reset_limits:
508       }
509       { \__draw_path_use:n { #1 , clear } }
510   }
```

13

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).

```
511 \cs_new_protected:Npn \__draw_path_use:n #1
512   {
513     \bool_set_false:N \l__draw_path_use_clip_bool
514     \bool_set_false:N \l__draw_path_use_fill_bool
515     \bool_set_false:N \l__draw_path_use_stroke_bool
516     \clist_map_inline:nn {#1}
517       {
518         \cs_if_exist:cTF { l__draw_path_use_ ##1 _ bool }
519           { \bool_set_true:c { l__draw_path_use_ ##1 _ bool } }
520           {
521             \cs_if_exist_use:cF { __draw_path_use_action_ ##1 : }
522               { \ERROR }
523           }
524       }
525     \__draw_softpath_round_corners:
526     \bool_lazy_and:nnT
527       { \l_draw_bb_update_bool }
528       { \l_draw_path_use_stroke_bool }
529       { \__draw_path_use_stroke_bb: }
530     \bool_if:NTF \l__draw_path_use_clear_bool
531       { \__draw_softpath_use_clear: }
532       { \__draw_softpath_use: }
533     \bool_if:NT \l__draw_path_use_clip_bool
534       { \driver_draw_clip: }
535     \bool_lazy_or:nnT
536       { \l_draw_path_use_fill_bool }
537       { \l_draw_path_use_stroke_bool }
538       {
539         \use:c
540           {
541             driver_draw_
542             \bool_if:NT \l__draw_path_use_fill_bool { fill }
543             \bool_if:NT \l__draw_path_use_stroke_bool { stroke }
544             :
545           }
546       }
547   }
548 \cs_new_protected:Npn \__draw_path_use_action_draw:
549   {
550     \bool_set_true:N \l_draw_path_use_stroke_bool
551   }
552 \cs_new_protected:Npn \__draw_path_use_action_fillstroke:
553   {
554     \bool_set_true:N \l_draw_path_use_fill_bool
555     \bool_set_true:N \l_draw_path_use_stroke_bool
556   }
```

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.

```
557 \cs_new_protected:Npn \__draw_path_use_stroke_bb:
558   {
```

```
559     \__draw_path_use_stroke_bb_aux:NnN x { max } +
560     \__draw_path_use_stroke_bb_aux:NnN y { max } +
561     \__draw_path_use_stroke_bb_aux:NnN x { min } -
562     \__draw_path_use_stroke_bb_aux:NnN y { min } -
563   }
564 \cs_new_protected:Npn \__draw_path_use_stroke_bb_aux:NnN #1#2#3
565   {
566     \dim_compare:nNnF { \dim_use:c { g__draw_ #1#2 _dim } } = { #3 -\c_max_dim }
567       {
568         \dim_gset:cn { g__draw_ #1#2 _dim }
569           {
570             \use:c { dim_ #2 :nn }
571               { \dim_use:c { g__draw_ #1#2 _dim } }
572               {
573                 \dim_use:c { g__draw_path_ #1#2 _dim }
574               #3 0.5 \g__draw_linewidth_dim
575               }
576           }
577       }
578   }
```

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

2.9 Scoping paths

<div style="float:left">

`\l__draw_path_lastx_dim`
`\l__draw_path_lasty_dim`
`\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`

</div>

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).

```
579 \dim_new:N \l__draw_path_lastx_dim
580 \dim_new:N \l__draw_path_lasty_dim
581 \dim_new:N \l__draw_path_xmax_dim
582 \dim_new:N \l__draw_path_xmin_dim
583 \dim_new:N \l__draw_path_ymax_dim
584 \dim_new:N \l__draw_path_ymin_dim
585 \dim_new:N \l__draw_softpath_lastx_dim
586 \dim_new:N \l__draw_softpath_lasty_dim
587 \bool_new:N \l__draw_softpath_corners_bool
```

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

<div style="float:left">

`\draw_path_scope_begin:`
`\draw_path_scope_end:`

</div>

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

```
588 \cs_new_protected:Npn \draw_path_scope_begin:
589   {
590     \group_begin:
591       \dim_set_eq:NN \l__draw_path_lastx_dim \g__draw_path_lastx_dim
592       \dim_set_eq:NN \l__draw_path_lasty_dim \g__draw_path_lasty_dim
593       \dim_set_eq:NN \l__draw_path_xmax_dim \g__draw_path_xmax_dim
594       \dim_set_eq:NN \l__draw_path_xmin_dim \g__draw_path_xmin_dim
595       \dim_set_eq:NN \l__draw_path_ymax_dim \g__draw_path_ymax_dim
596       \dim_set_eq:NN \l__draw_path_ymin_dim \g__draw_path_ymin_dim
597       \dim_set_eq:NN \l__draw_softpath_lastx_dim \g__draw_softpath_lastx_dim
598       \dim_set_eq:NN \l__draw_softpath_lasty_dim \g__draw_softpath_lasty_dim
599       \__draw_path_reset_limits:
```

```
600        \tl_build_get:NN \g__draw_softpath_main_tl \l__draw_softpath_main_tl
601        \bool_set_eq:NN
602          \l__draw_softpath_corners_bool
603          \g__draw_softpath_corners_bool
604        \__draw_softpath_clear:
605      }
606  \cs_new_protected:Npn \draw_path_scope_end:
607      {
608        \__draw_softpath_clear:
609        \bool_gset_eq:NN
610          \g__draw_softpath_corners_bool
611          \l__draw_softpath_corners_bool
612        \__draw_softpath_add:o \l__draw_softpath_main_tl
613        \dim_gset_eq:NN \g__draw_softpath_lastx_dim \l__draw_softpath_lastx_dim
614        \dim_gset_eq:NN \g__draw_softpath_lasty_dim \l__draw_softpath_lasty_dim
615        \dim_gset_eq:NN \g__draw_path_xmax_dim \l__draw_path_xmax_dim
616        \dim_gset_eq:NN \g__draw_path_xmin_dim \l__draw_path_xmin_dim
617        \dim_gset_eq:NN \g__draw_path_ymax_dim \l__draw_path_ymax_dim
618        \dim_gset_eq:NN \g__draw_path_ymin_dim \l__draw_path_ymin_dim
619        \dim_gset_eq:NN \g__draw_path_lastx_dim \l__draw_path_lastx_dim
620        \dim_gset_eq:NN \g__draw_path_lasty_dim \l__draw_path_lasty_dim
621      \group_end:
622      }
```

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

623 ⟨/initex | package⟩

3   **l3draw-points** implementation

624 ⟨*initex | package⟩

625 ⟨@@=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`.

- `\pgfpoint`, `\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.

16

- \pgfpointborderrectangle, \pgfpointborderellipse: To be revisited once the semantics and use cases are clear.

- \pgfqpoint, \pgfqpointscale, \pgfqpointpolar, \pgfqpointxy, \pgfqpointxyz: The expandable approach taken in the code here, along with the absolute requirement for ε-TeX, means it is likely many use cases for these commands may be covered in other ways. This may be revisited as higher-level structures are constructed.

3.1 Support functions

__draw_point_process:nn
__draw_point_process_auxi:nn
__draw_point_process_auxi:fn
__draw_point_process_auxii:nw
__draw_point_process:nnn
__draw_point_process_auxiii:nnn
__draw_point_process_auxiii:ffn
__draw_point_process_auxiv:nw

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.

```
626 \cs_new:Npn \__draw_point_process:nn #1#2
627   {
628     \__draw_point_process_auxi:fn
629       { \__draw_point_to_dim:n {#2} }
630       {#1}
631   }
632 \cs_new:Npn \__draw_point_process_auxi:nn #1#2
633   { \__draw_point_process_auxii:nw {#2} #1 \q_stop }
634 \cs_generate_variant:Nn \__draw_point_process_auxi:nn { f }
635 \cs_new:Npn \__draw_point_process_auxii:nw #1 #2 , #3 \q_stop
636   { #1 {#2} {#3} }
637 \cs_new:Npn \__draw_point_process:nnn #1#2#3
638   {
639     \__draw_point_process_auxiii:ffn
640       { \__draw_point_to_dim:n {#2} }
641       { \__draw_point_to_dim:n {#3} }
642       {#1}
643   }
644 \cs_new:Npn \__draw_point_process_auxiii:nnn #1#2#3
645   { \__draw_point_process_auxiv:nw {#3} #1 \q_mark #2 \q_stop }
646 \cs_generate_variant:Nn \__draw_point_process_auxiii:nnn { ff }
647 \cs_new:Npn \__draw_point_process_auxiv:nw #1 #2 , #3 \q_mark #4 , #5 \q_stop
648   { #1 {#2} {#3} {#4} {#5} }
```

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

__draw_point_to_dim:n
__draw_point_to_dim_aux:n
__draw_point_to_dim_aux:f
__draw_point_to_dim_aux:w

Co-ordinates are always returned as two dimensions.

```
649 \cs_new:Npn \__draw_point_to_dim:n #1
650   { \__draw_point_to_dim_aux:f { \fp_eval:n {#1} } }
651 \cs_new:Npn \__draw_point_to_dim_aux:n #1
652   { \__draw_point_to_dim_aux:w #1 }
653 \cs_generate_variant:Nn \__draw_point_to_dim_aux:n { f }
654 \cs_new:Npn \__draw_point_to_dim_aux:w ( #1 , ~ #2 ) { #1pt , #2pt }
```

3.2 Polar co-ordinates

\draw_point_polar:nn
\draw_point_polar:nnn
__draw_draw_polar:nnn
__draw_draw_polar:fnn

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.

```
655  \cs_new:Npn \draw_point_polar:nn #1#2
656    { \draw_point_polar:nnn {#1} {#2} {#2} }
657  \cs_new:Npn \draw_point_polar:nnn #1#2#3
658    { \__draw_draw_polar:fnn { \fp_eval:n {#1} } {#2} {#3} }
659  \cs_new:Npn \__draw_draw_polar:nnn #1#2#3
660    { \__draw_point_to_dim:n { cosd(#1) * (#2) , sind(#1) * (#3) } }
661  \cs_generate_variant:Nn \__draw_draw_polar:nnn { f }
```

3.3   Point expression arithmetic

These functions all take point expressions as arguments.

\draw_point_unit_vector:n
__draw_point_unit_vector:nn

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

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

```
662  \cs_new:Npn \draw_point_unit_vector:n #1
663    { \__draw_point_process:nn { \__draw_point_unit_vector:nn } {#1} }
664  \cs_new:Npn \__draw_point_unit_vector:nn #1#2
665    {
666      \__draw_point_to_dim:n
667        { ( #1 , #2 ) / (sqrt(#1 * #1 + #2 * #2)) }
668    }
```

3.4   Intersection calculations

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

\draw_point_intersect_lines:nnnn
__draw_point_intersect_lines:nnnnnn
__draw_point_intersect_lines:nnnnnnnn
__draw_point_intersect_lines_aux:nnnnnn
__draw_point_intersect_lines_aux:ffffff

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

and

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

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

```
669  \cs_new:Npn \draw_point_intersect_lines:nnnn #1#2#3#4
670    {
671      \__draw_point_process:nnn
672        {
673          \__draw_point_process:nnn
674            { \__draw_point_intersect_lines:nnnnnnnn } {#3} {#4}
675        }
676      {#1} {#2}
677    }
```

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

#1 x_3

18

#2 y_3

#3 x_4

#4 y_4

#5 x_1

#6 y_1

#7 x_2

#8 y_2

so now just have to do all of the calculation.

```
678 \cs_new:Npn \__draw_point_intersect_lines:nnnnnnnn #1#2#3#4#5#6#7#8
679   {
680     \__draw_point_intersect_lines_aux:ffffff
681       { \fp_eval:n { #1 * #4 - #2 * #3 } }
682       { \fp_eval:n { #5 * #8 - #6 * #7 } }
683       { \fp_eval:n { #1 - #3 } }
684       { \fp_eval:n { #5 - #7 } }
685       { \fp_eval:n { #2 - #4 } }
686       { \fp_eval:n { #6 - #8 } }
687   }
688 \cs_new:Npn \__draw_point_intersect_lines_aux:nnnnnn #1#2#3#4#5#6
689   {
690     \__draw_point_to_dim:n
691       {
692         ( #2 * #3 - #1 * #4 , #2 * #5 - #1 * #6 )
693           / ( #4 * #5 - #6 * #3 )
694       }
695   }
696 \cs_generate_variant:Nn \__draw_point_intersect_lines_aux:nnnnnn { ffffff }
```

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

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

in either

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

19

or

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

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

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

```
697 \cs_new:Npn \draw_point_intersect_circles:nnnnn #1#2#3#4#5
698   {
699     \__draw_point_process:nnn
700       { \__draw_point_intersect_circles_auxi:nnnnnnn {#2} {#4} {#5} }
701       {#1} {#3}
702   }
703 \cs_new:Npn \__draw_point_intersect_circles_auxi:nnnnnnn #1#2#3#4#5#6#7
704   {
705     \__draw_point_intersect_circles_auxii:ffnnnnn
706       { \fp_eval:n {#1} } { \fp_eval:n {#2} } {#4} {#5} {#6} {#7} {#3}
707   }
```

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.

```
708 \cs_new:Npn \__draw_point_intersect_circles_auxii:nnnnnnn #1#2#3#4#5#6#7
709   {
710     \__draw_point_intersect_circles_auxiii:ffnnnnn
711       { \fp_eval:n { #5 - #3 } }
712       { \fp_eval:n { #6 - #4 } }
713       {#1} {#2} {#3} {#4} {#7}
714   }
715 \cs_generate_variant:Nn \__draw_point_intersect_circles_auxii:nnnnnnn { ff }
716 \cs_new:Npn \__draw_point_intersect_circles_auxiii:nnnnnnn #1#2#3#4#5#6#7
717   {
718     \__draw_point_intersect_circles_auxiv:fnnnnnnn
719       { \fp_eval:n { sqrt( #1 * #1 + #2 * #2 ) } }
720       {#1} {#2} {#3} {#4} {#5} {#6} {#7}
721   }
722 \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.

```
723 \cs_new:Npn \__draw_point_intersect_circles_auxiv:nnnnnnnn #1#2#3#4#5#6#7#8
724   {
725     \__draw_point_intersect_circles_auxv:ffnnnnnnn
726       { \fp_eval:n { 1 / #1 } }
727       { \fp_eval:n { #4 * #4 } }
728       {#1} {#2} {#3} {#5} {#6} {#7} {#8}
729   }
730 \cs_generate_variant:Nn \__draw_point_intersect_circles_auxiv:nnnnnnnn { f }
731 \cs_new:Npn \__draw_point_intersect_circles_auxv:nnnnnnnnn #1#2#3#4#5#6#7#8#9
732   {
733     \__draw_point_intersect_circles_auxvi:fnnnnnnn
734       { \fp_eval:n { 0.5 * #1 * ( #2 + #3 * #3 - #6 * #6 ) } }
735       {#1} {#2} {#4} {#5} {#7} {#8} {#9}
736   }
737 \cs_generate_variant:Nn \__draw_point_intersect_circles_auxv:nnnnnnnnn { ff }
```

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

#1 k

#2 $1/p$

#3 r^2

#4 e

#5 f

#6 a

#7 b

#8 n

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

```
738 \cs_new:Npn \__draw_point_intersect_circles_auxvi:nnnnnnnn #1#2#3#4#5#6#7#8
739   {
740     \__draw_point_intersect_circles_auxvii:fffnnnn
741       { \fp_eval:n { #1 * #2 } }
742       { \int_if_odd:nTF {#8} { 1 } { -1 } }
743       { \fp_eval:n { sqrt ( #3 - #1 * #1 ) * #2 } }
744       {#4} {#5} {#6} {#7}
745   }
746 \cs_generate_variant:Nn \__draw_point_intersect_circles_auxvi:nnnnnnnn { f }
747 \cs_new:Npn \__draw_point_intersect_circles_auxvii:nnnnnnn #1#2#3#4#5#6#7
748   {
749     \__draw_point_to_dim:n
750       { #6 + #4 * #1 + #2 * #3 * #5 , #7 + #5 * #1 + -1 * #2 * #3 * #4 }
751   }
752 \cs_generate_variant:Nn \__draw_point_intersect_circles_auxvii:nnnnnnn { fff }
```

3.5 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`

```
753 \cs_new:Npn \draw_point_interpolate_line:nnn #1#2#3
754   {
755     \__draw_point_process:nnn
756       { \__draw_point_interpolate_line_aux:fnnnn { \fp_eval:n {#1} } }
757       {#2} {#3}
758   }
759 \cs_new:Npn \__draw_point_interpolate_line_aux:nnnnn #1#2#3#4#5
760   {
761     \__draw_point_interpolate_line_aux:fnnnnn { \fp_eval:n { 1 - #1 } }
762       {#1} {#2} {#3} {#4} {#5}
763   }
764 \cs_generate_variant:Nn \__draw_point_interpolate_line_aux:nnnnn { f }
765 \cs_new:Npn \__draw_point_interpolate_line_aux:nnnnnn #1#2#3#4#5#6
766   { \__draw_point_to_dim:n { #2 * #3 + #1 * #5 , #2 * #4 + #1 * #6 } }
767 \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`

```
768 \cs_new:Npn \draw_point_interpolate_distance:nnn #1#2#3
769   {
770     \__draw_point_process:nn
771       { \__draw_point_interpolate_distance:nnnn {#1} {#3} }
772       {#2}
773   }
774 \cs_new:Npn \__draw_point_interpolate_distance:nnnn #1#2#3#4
775   {
776     \__draw_point_process:nn
777       {
778         \__draw_point_interpolate_distance:fnnnn
779           { \fp_eval:n {#1} } {#3} {#4}
780       }
781       { \draw_point_unit_vector:n { ( #2 ) - ( #3 , #4 ) } }
782   }
783 \cs_new:Npn \__draw_point_interpolate_distance:nnnnn #1#2#3#4#5
784   { \__draw_point_to_dim:n { #2 + #1 * #4 , #3 + #1 * #5 } }
785 \cs_generate_variant:Nn \__draw_point_interpolate_distance:nnnnn { f }
```

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

`\draw_point_interpolate_arcaxes:nnnnnn`
`\draw_point_interpolate_arcaxes_auxi:nnnnnnnnn`
`\draw_point_interpolate_arcaxes_auxii:nnnnnnnnn`
`\draw_point_interpolate_arcaxes_auxii:fnnnnnnnn`
`\draw_point_interpolate_arcaxes_auxiii:nnnnnnnnn`
`\draw_point_interpolate_arcaxes_auxiii:fnnnnnnn`
`\draw_point_interpolate_arcaxes_auxiv:nnnnnnnnn`
`\draw_point_interpolate_arcaxes_auxiv:ffnnnnnn`

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.

```
786 \cs_new:Npn \draw_point_interpolate_arcaxes:nnnnnn #1#2#3#4#5#6
787   {
788     \__draw_point_process:nnn
789       {
790         \__draw_point_process:nn
791           { \__draw_point_interpolate_arcaxes_auxi:nnnnnnnnn {#1} {#5} {#6} }
792           {#4}
793       }
794       {#2} {#3}
```

```
795     }
796  \cs_new:Npn \__draw_point_interpolate_arcaxes_auxi:nnnnnnnnn #1#2#3#4#5#6#7#8#9
797    {
798      \__draw_point_interpolate_arcaxes_auxii:fnnnnnnnn
799        { \fp_eval:n {#1} } {#2} {#3} {#6} {#7} {#8} {#9} {#4} {#5}
800    }
```

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

#1 p

#2 θ_1

#3 θ_2

#4 x_c

#5 y_c

#6 x_{a1}

#7 y_{a1}

#8 x_{a2}

#9 y_{a2}

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

```
801  \cs_new:Npn \__draw_point_interpolate_arcaxes_auxii:nnnnnnnnn #1#2#3#4#5#6#7#8#9
802    {
803      \__draw_point_interpolate_arcaxes_auxiii:fnnnnnn
804        { \fp_eval:n { #1 * (#3) + ( 1 - #1 ) * (#2) } }
805        {#4} {#5} {#6} {#7} {#8} {#9}
806    }
807  \cs_generate_variant:Nn \__draw_point_interpolate_arcaxes_auxii:nnnnnnnnn { f }
808  \cs_new:Npn \__draw_point_interpolate_arcaxes_auxiii:nnnnnnn #1#2#3#4#5#6#7
809    {
810      \__draw_point_interpolate_arcaxes_auxiv:ffnnnnnn
811        { \fp_eval:n { cosd (#1) } }
812        { \fp_eval:n { sind (#1) } }
813        {#2} {#3} {#4} {#5} {#6} {#7}
814    }
815  \cs_generate_variant:Nn \__draw_point_interpolate_arcaxes_auxiii:nnnnnnn { f }
816  \cs_new:Npn \__draw_point_interpolate_arcaxes_auxiv:nnnnnnnn #1#2#3#4#5#6#7#8
817    {
818      \__draw_point_to_dim:n
819        { #3 + #1 * #5 + #2 * #7 , #4 + #1 * #6 + #2 * #8 }
820    }
821  \cs_generate_variant:Nn \__draw_point_interpolate_arcaxes_auxiv:nnnnnnnn { ff }
```

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

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

1. The initial point (x_1, y_1)

23

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.

```
822 \cs_new:Npn \draw_point_interpolate_curve:nnnnnn #1#2#3#4#5
823   {
824     \__draw_point_process:nnn
825       {
826         \__draw_point_process:nnn
827           { \__draw_point_interpolate_curve_auxi:nnnnnnnnn {#1} }
828           {#4} {#5}
829       }
830       {#2} {#3}
831   }
832 \cs_new:Npn \__draw_point_interpolate_curve_auxi:nnnnnnnnn #1#2#3#4#5#6#7#8#9
833   {
834     \__draw_point_interpolate_curve_auxii:fnnnnnnnn
835       { \fp_eval:n {#1} }
836       {#6} {#7} {#8} {#9} {#2} {#3} {#4} {#5}
837   }
```

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!

```
838 \cs_new:Npn \__draw_point_interpolate_curve_auxii:nnnnnnnnn
839   #1#2#3#4#5#6#7#8#9
840   {
841     \__draw_point_interpolate_curve_auxiii:fnnnnn
```

```
842        { \fp_eval:n { 1 - #1 } }
843        {#1}
844        { {#2} {#3} } { {#4} {#5} } { {#6} {#7} } { {#8} {#9} }
845      }
846  \cs_generate_variant:Nn \__draw_point_interpolate_curve_auxii:nnnnnnnnn { f }
847  %    \begin{macrocode}
848  %   We need to do the first cycle, but haven't got enough arguments to keep
849  %   everything in play at once. So her ewe use a but of argument re-ordering
850  %   and a single auxiliary to get the job done.
851  %    \begin{macrocode}
852  \cs_new:Npn \__draw_point_interpolate_curve_auxiii:nnnnnn #1#2#3#4#5#6
853    {
854      \__draw_point_interpolate_curve_auxiv:nnnnnn {#1} {#2} #3 #4
855      \__draw_point_interpolate_curve_auxiv:nnnnnn {#1} {#2} #4 #5
856      \__draw_point_interpolate_curve_auxiv:nnnnnn {#1} {#2} #5 #6
857      \prg_do_nothing:
858      \__draw_point_interpolate_curve_auxvi:n { {#1} {#2} }
859    }
860  \cs_generate_variant:Nn \__draw_point_interpolate_curve_auxiii:nnnnnn { f }
861  \cs_new:Npn \__draw_point_interpolate_curve_auxiv:nnnnnn #1#2#3#4#5#6
862    {
863      \__draw_point_interpolate_curve_auxv:ffw
864        { \fp_eval:n { #1 * #3 + #2 * #5 } }
865        { \fp_eval:n { #1 * #4 + #2 * #6 } }
866    }
867  \cs_new:Npn \__draw_point_interpolate_curve_auxv:nnw
868    #1#2#3 \prg_do_nothing: #4#5
869    {
870      #3
871      \prg_do_nothing:
872      #4 { #5 {#1} {#2} }
873    }
874  \cs_generate_variant:Nn \__draw_point_interpolate_curve_auxv:nnw { ff }
875  %    \begin{macrocode}
876  %   Get the arguments back into the right places and to the second and
877  %   third cycles directly.
878  %    \begin{macrocode}
879  \cs_new:Npn \__draw_point_interpolate_curve_auxvi:n #1
880    { \__draw_point_interpolate_curve_auxvii:nnnnnnnn #1 }
881  \cs_new:Npn \__draw_point_interpolate_curve_auxvii:nnnnnnnn #1#2#3#4#5#6#7#8
882    {
883      \__draw_point_interpolate_curve_auxviii:ffffnn
884        { \fp_eval:n { #1 * #5 + #2 * #3 } }
885        { \fp_eval:n { #1 * #6 + #2 * #4 } }
886        { \fp_eval:n { #1 * #7 + #2 * #5 } }
887        { \fp_eval:n { #1 * #8 + #2 * #6 } }
888        {#1} {#2}
889    }
890  \cs_new:Npn \__draw_point_interpolate_curve_auxviii:nnnnnn #1#2#3#4#5#6
891    {
892      \__draw_point_to_dim:n
893        { #5 * #3 + #6 * #1 , #5 * #4 + #6 * #2 }
894    }
895  \cs_generate_variant:Nn \__draw_point_interpolate_curve_auxviii:nnnnnn { ffff }
```

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

3.6 Vector support

As well as co-ordinates relative to the drawing

`\l__draw_xvec_x_dim`
`\l__draw_xvec_y_dim`
`\l__draw_yvec_x_dim`
`\l__draw_yvec_y_dim`
`\l__draw_zvec_x_dim`
`\l__draw_zvec_y_dim`

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

```
896 \dim_new:N \l__draw_xvec_x_dim
897 \dim_new:N \l__draw_xvec_y_dim
898 \dim_new:N \l__draw_yvec_x_dim
899 \dim_new:N \l__draw_yvec_y_dim
900 \dim_new:N \l__draw_zvec_x_dim
901 \dim_new:N \l__draw_zvec_y_dim
```

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

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

Calculate the underlying position and store it.

```
902 \cs_new_protected:Npn \draw_xvec:n #1
903   { \__draw_vec:nn { x } {#1} }
904 \cs_new_protected:Npn \draw_yvec:n #1
905   { \__draw_vec:nn { y } {#1} }
906 \cs_new_protected:Npn \draw_zvec:n #1
907   { \__draw_vec:nn { z } {#1} }
908 \cs_new_protected:Npn \__draw_vec:nn #1#2
909   {
910     \__draw_point_process:nn { \__draw_vec:nnn {#1} } {#2}
911   }
912 \cs_new_protected:Npn \__draw_vec:nnn #1#2#3
913   {
914     \dim_set:cn { l__draw_ #1 vec_x_dim } {#2}
915     \dim_set:cn { l__draw_ #1 vec_y_dim } {#3}
916   }
```

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

Initialise the vectors.

```
917 \draw_xvec:n { 1cm , 0cm }
918 \draw_yvec:n { 0cm , 1cm }
919 \draw_zvec:n { -0.385cm , -0.385cm }
```

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

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

```
920 \cs_new:Npn \draw_point_vec:nn #1#2
921   { \__draw_point_vec:ff { \fp_eval:n {#1} } { \fp_eval:n {#2} } }
922 \cs_new:Npn \__draw_point_vec:nn #1#2
923   {
924     \__draw_point_to_dim:n
925       {
926         #1 * \l__draw_xvec_x_dim + #2 * \l__draw_yvec_x_dim ,
927         #1 * \l__draw_xvec_y_dim + #2 * \l__draw_yvec_y_dim
928       }
929   }
930 \cs_generate_variant:Nn \__draw_point_vec:nn { ff }
931 \cs_new:Npn \draw_point_vec:nnn #1#2#3
932   {
```

```
933        \__draw_point_vec:fff
934          { \fp_eval:n {#1} } { \fp_eval:n {#2} } { \fp_eval:n {#3} }
935      }
936    \cs_new:Npn \__draw_point_vec:nnn #1#2#3
937      {
938        \__draw_point_to_dim:n
939          {
940                  #1 * \l__draw_xvec_x_dim
941              + #2 * \l__draw_yvec_x_dim
942              + #3 * \l__draw_zvec_x_dim
943            ,
944                  #1 * \l__draw_xvec_y_dim
945              + #2 * \l__draw_yvec_y_dim
946              + #3 * \l__draw_zvec_y_dim
947          }
948      }
949    \cs_generate_variant:Nn \__draw_point_vec:nnn { fff }
```

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

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

Much the same as the core polar approach.

```
950    \cs_new:Npn \draw_point_vec_polar:nn #1#2
951      { \draw_point_vec_polar:nnn {#1} {#2} {#2} }
952    \cs_new:Npn \draw_point_vec_polar:nnn #1#2#3
953      { \__draw_draw_vec_polar:fnn { \fp_eval:n {#1} } {#2} {#3} }
954    \cs_new:Npn \__draw_draw_vec_polar:nnn #1#2#3
955      {
956        \__draw_point_to_dim:n
957          {
958            cosd(#1) * (#2) * \l__draw_xvec_x_dim ,
959            sind(#1) * (#3) * \l__draw_yvec_y_dim
960          }
961      }
962    \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* **??**.)

3.7 Transformations

\draw_point_transform:n
__draw_point_transform:nn

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

```
963    \cs_new:Npn \draw_point_transform:n #1
964      {
965        \__draw_point_process:nn
966          { \__draw_point_transform:nn } {#1}
967      }
968    \cs_new:Npn \__draw_point_transform:nn #1#2
969      {
970        \bool_if:NTF \l__draw_matrix_active_bool
971          {
972            \__draw_point_to_dim:n
973              {
974                (
```

27

```
975            \l__draw_matrix_a_fp * #1
976          + \l__draw_matrix_c_fp * #2
977          + \l__draw_xshift_dim
978        )
979        ,
980        (
981            \l__draw_matrix_b_fp * #1
982          + \l__draw_matrix_d_fp * #2
983          + \l__draw_yshift_dim
984        )
985      }
986    }
987    {
988      \__draw_point_to_dim:n
989        {
990          (#1, #2)
991          + ( \l__draw_xshift_dim , \l__draw_yshift_dim )
992        }
993    }
994  }
```

(*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.

```
995 \cs_new:Npn \__draw_point_transform_noshift:n #1
996  {
997    \__draw_point_process:nn
998      { \__draw_point_transform_noshift:nn } {#1}
999  }
1000 \cs_new:Npn \__draw_point_transform_noshift:nn #1#2
1001  {
1002    \bool_if:NTF \l__draw_matrix_active_bool
1003      {
1004        \__draw_point_to_dim:n
1005          {
1006            (
1007                \l__draw_matrix_a_fp * #1
1008              + \l__draw_matrix_c_fp * #2
1009            )
1010            ,
1011            (
1012                \l__draw_matrix_b_fp * #1
1013              + \l__draw_matrix_d_fp * #2
1014            )
1015          }
1016      }
1017      { \__draw_point_to_dim:n { (#1, #2) } }
1018  }
```

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

```
1019 ⟨/initex | package⟩
```

4 l3draw-scopes implementation

```
1020 ⟨*initex | package⟩
```

```
1021 ⟨@@=draw⟩
```

4.1 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.

```
1022 \dim_new:N \g__draw_xmax_dim
1023 \dim_new:N \g__draw_xmin_dim
1024 \dim_new:N \g__draw_ymax_dim
1025 \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.

```
1026 \bool_new:N \l_draw_bb_update_bool
```

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

\l__draw_main_box   Box for setting the drawing.

```
1027 \box_new:N \l__draw_main_box
```

(*End definition for* \l__draw_main_box.)

\g__draw_id_int   The drawing number.

```
1028 \int_new:N \g__draw_id_int
```

(*End definition for* \g__draw_id_int.)

__draw_reset_bb: A simple auxiliary.

```
1029 \cs_new_protected:Npn \__draw_reset_bb:
1030   {
1031     \dim_gset:Nn \g__draw_xmax_dim { -\c_max_dim }
1032     \dim_gset:Nn \g__draw_xmin_dim {  \c_max_dim }
1033     \dim_gset:Nn \g__draw_ymax_dim { -\c_max_dim }
1034     \dim_gset:Nn \g__draw_ymin_dim {  \c_max_dim }
1035   }
```

(*End definition for* __draw_reset_bb:.)

\draw_begin:
\draw_end:

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.

```
1036 \cs_new_protected:Npn \draw_begin:
1037   {
1038     \group_begin:
1039       \int_gincr:N \g__draw_id_int
1040       \hbox_set:Nw \l__draw_main_box
1041         \driver_draw_begin:
1042         \__draw_reset_bb:
1043         \__draw_path_reset_limits:
```

```
1044          \bool_set_true:N \l_draw_bb_update_bool
1045          \draw_transform_matrix_reset:
1046          \draw_transform_shift_reset:
1047          \__draw_softpath_clear:
1048          \draw_linewidth:n { \l_draw_default_linewidth_dim }
1049          \draw_color:n { . }
1050          \draw_nonzero_rule:
1051          \draw_cap_butt:
1052          \draw_join_miter:
1053          \draw_miterlimit:n { 10 }
1054          \draw_dash_pattern:nn { } { 0cm }
1055      }
1056  \cs_new_protected:Npn \draw_end:
1057      {
1058          \driver_draw_end:
1059          \hbox_set_end:
1060          \dim_compare:nNnT \g__draw_xmin_dim = \c_max_dim
1061              {
1062                  \dim_gzero:N \g__draw_xmax_dim
1063                  \dim_gzero:N \g__draw_xmin_dim
1064                  \dim_gzero:N \g__draw_ymax_dim
1065                  \dim_gzero:N \g__draw_ymin_dim
1066              }
1067          \hbox_set:Nn \l__draw_main_box
1068              {
1069                  \skip_horizontal:n { -\g__draw_xmin_dim }
1070                  \box_move_down:nn { \g__draw_ymin_dim }
1071                      { \box_use_drop:N \l__draw_main_box }
1072              }
1073          \box_set_ht:Nn \l__draw_main_box
1074              { \g__draw_ymax_dim - \g__draw_ymin_dim }
1075          \box_set_dp:Nn \l__draw_main_box { 0pt }
1076          \box_set_wd:Nn \l__draw_main_box
1077              { \g__draw_xmax_dim - \g__draw_xmin_dim }
1078          \mode_leave_vertical:
1079          \box_use_drop:N \l__draw_main_box
1080      \group_end:
1081      }
```

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

4.2 Scopes

`\l__draw_linewidth_dim`
`\l__draw_fill_color_tl`
`\l__draw_stroke_color_tl`

Storage for local variables.

```
1082  \dim_new:N \l__draw_linewidth_dim
1083  \tl_new:N \l__draw_fill_color_tl
1084  \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:`
`\draw_scope_begin:`

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

```
1085  \cs_new_protected:Npn \draw_scope_begin:
1086      {
```

```
1087        \driver_draw_scope_begin:
1088        \group_begin:
1089          \dim_set_eq:NN \l__draw_linewidth_dim \g__draw_linewidth_dim
1090          \draw_path_scope_begin:
1091      }
1092  \cs_new_protected:Npn \draw_scope_end:
1093      {
1094        \draw_path_scope_end:
1095        \dim_gset_eq:NN \g__draw_linewidth_dim \l__draw_linewidth_dim
1096      \group_end:
1097      \driver_draw_scope_end:
1098      }
```

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

`\l__draw_xmax_dim`
`\l__draw_xmin_dim`
`\l__draw_ymax_dim`
`\l__draw_ymin_dim`

Storage for the bounding box.

```
1099  \dim_new:N \l__draw_xmax_dim
1100  \dim_new:N \l__draw_xmin_dim
1101  \dim_new:N \l__draw_ymax_dim
1102  \dim_new:N \l__draw_ymin_dim
```

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

`\__draw_scope_bb_begin:`
`\__draw_scope_bb_end:`

The bounding box is simple: a straight group-based save and restore approach.

```
1103  \cs_new_protected:Npn \__draw_scope_bb_begin:
1104      {
1105        \group_begin:
1106          \dim_set_eq:NN \l__draw_xmax_dim \g__draw_xmax_dim
1107          \dim_set_eq:NN \l__draw_xmin_dim \g__draw_xmin_dim
1108          \dim_set_eq:NN \l__draw_ymax_dim \g__draw_ymax_dim
1109          \dim_set_eq:NN \l__draw_ymin_dim \g__draw_ymin_dim
1110          \__draw_reset_bb:
1111      }
1112  \cs_new_protected:Npn \__draw_scope_bb_end:
1113      {
1114        \dim_gset_eq:NN \g__draw_xmax_dim \l__draw_xmax_dim
1115        \dim_gset_eq:NN \g__draw_xmin_dim \l__draw_xmin_dim
1116        \dim_gset_eq:NN \g__draw_ymax_dim \l__draw_ymax_dim
1117        \dim_gset_eq:NN \g__draw_ymin_dim \l__draw_ymin_dim
1118      \group_end:
1119      }
```

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

`\draw_suspend_begin:`
`\draw_suspend_end:`

Suspend all parts of a drawing.

```
1120  \cs_new_protected:Npn \draw_suspend_begin:
1121      {
1122        \__draw_scope_bb_begin:
1123        \draw_path_scope_begin:
1124        \draw_transform_matrix_reset:
1125        \draw_transform_shift_reset:
1126      }
1127  \cs_new_protected:Npn \draw_suspend_end:
1128      {
1129        \draw_path_scope_end:
```

```
1130        \__draw_scope_bb_end:
1131    }
```

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

1132 ⟨/initex | package⟩

5 l3draw-softpath implementation

1133 ⟨*initex | package⟩

1134 ⟨@@=draw⟩

5.1 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.

```
1135 \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.

```
1136 \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.

```
1137 \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

```
1138 \cs_new_protected:Npn \__draw_softpath_add:n
1139    { \tl_build_gput_right:Nn \g__draw_softpath_main_tl }
1140 \cs_generate_variant:Nn \__draw_softpath_add:n { o, x }
```

(*End definition for* __draw_softpath_add:n.)

__draw_softpath_use:
__draw_softpath_clear:
__draw_softpath_use_clear:

Using and clearing is trivial.

```
1141 \cs_new_protected:Npn \__draw_softpath_use:
1142   {
1143     \tl_build_get:NN \g__draw_softpath_main_tl \l__draw_softpath_internal_tl
1144     \l__draw_softpath_internal_tl
1145   }
1146 \cs_new_protected:Npn \__draw_softpath_clear:
1147   {
1148     \tl_build_gclear:N \g__draw_softpath_main_tl
1149     \bool_gset_false:N \g__draw_softpath_corners_bool
1150   }
1151 \cs_new_protected:Npn \__draw_softpath_use_clear:
1152   {
1153     \__draw_softpath_use:
1154     \__draw_softpath_clear:
1155   }
```

(*End definition for* __draw_softpath_use:, __draw_softpath_clear:, *and* __draw_softpath_use_-
clear:.)

\g__draw_softpath_lastx_dim
\g__draw_softpath_lasty_dim

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

```
1156 \dim_new:N \g__draw_softpath_lastx_dim
1157 \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.

```
1158 \bool_new:N \g__draw_softpath_move_bool
1159 \bool_gset_true:N \g__draw_softpath_move_bool
```

(*End definition for* \g__draw_softpath_move_bool.)

__draw_softpath_curveto:nnnnnn
__draw_softpath_lineto:nn
__draw_softpath_moveto:nn
__draw_softpath_rectangle:nnnn
__draw_softpath_roundpoint:nn
__draw_softpath_roundpoint:VV

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

```
1160 \cs_new_protected:Npn \__draw_softpath_closepath:
1161   {
1162     \__draw_softpath_add:x
1163       {
1164         \__draw_softpath_close_op:nn
1165           { \dim_use:N \g__draw_softpath_lastx_dim }
1166           { \dim_use:N \g__draw_softpath_lasty_dim }
1167       }
1168   }
1169 \cs_new_protected:Npn \__draw_softpath_curveto:nnnnnn #1#2#3#4#5#6
1170   {
1171     \__draw_softpath_add:n
1172       {
1173         \__draw_softpath_curveto_opi:nn {#1} {#2}
1174         \__draw_softpath_curveto_opii:nn {#3} {#4}
1175         \__draw_softpath_curveto_opiii:nn {#5} {#6}
1176       }
1177   }
1178 \cs_new_protected:Npn \__draw_softpath_lineto:nn #1#2
1179   {
1180     \__draw_softpath_add:n
1181       { \__draw_softpath_lineto_op:nn {#1} {#2} }
```

```
1182     }
1183 \cs_new_protected:Npn \__draw_softpath_moveto:nn #1#2
1184   {
1185     \__draw_softpath_add:n
1186       { \__draw_softpath_moveto_op:nn {#1} {#2} }
1187     \bool_if:NT \g__draw_softpath_move_bool
1188       {
1189         \dim_gset:Nn \g__draw_softpath_lastx_dim {#1}
1190         \dim_gset:Nn \g__draw_softpath_lasty_dim {#2}
1191       }
1192   }
1193 \cs_new_protected:Npn \__draw_softpath_rectangle:nnnn #1#2#3#4
1194   {
1195     \__draw_softpath_add:n
1196       {
1197         \__draw_softpath_rectangle_opi:nn {#1} {#2}
1198         \__draw_softpath_rectangle_opii:nn {#3} {#4}
1199       }
1200   }
1201 \cs_new_protected:Npn \__draw_softpath_roundpoint:nn #1#2
1202   {
1203     \__draw_softpath_add:n
1204       { \__draw_softpath_roundpoint_op:nn {#1} {#2} }
1205     \bool_gset_true:N \g__draw_softpath_corners_bool
1206   }
1207 \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.

```
1208 \cs_new_protected:Npn \__draw_softpath_close_op:nn #1#2
1209   { \driver_draw_closepath: }
1210 \cs_new_protected:Npn \__draw_softpath_curveto_opi:nn #1#2
1211   { \__draw_softpath_curveto_opi:nnNnnNnn {#1} {#2} }
1212 \cs_new_protected:Npn \__draw_softpath_curveto_opi:nnNnnNnn #1#2#3#4#5#6#7#8
1213   { \driver_draw_curveto:nnnnnn {#1} {#2} {#4} {#5} {#7} {#8} }
1214 \cs_new_protected:Npn \__draw_softpath_curveto_opii:nn #1#2
1215   { \__draw_softpath_curveto_opii:nn }
1216 \cs_new_protected:Npn \__draw_softpath_curveto_opiii:nn #1#2
1217   { \__draw_softpath_curveto_opiii:nn }
1218 \cs_new_protected:Npn \__draw_softpath_lineto_op:nn #1#2
1219   { \driver_draw_lineto:nn {#1} {#2} }
1220 \cs_new_protected:Npn \__draw_softpath_moveto_op:nn #1#2
1221   { \driver_draw_moveto:nn {#1} {#2} }
1222 \cs_new_protected:Npn \__draw_softpath_roundpoint_op:nn #1#2 { }
1223 \cs_new_protected:Npn \__draw_softpath_rectangle_opi:nn #1#2
1224   { \__draw_softpath_rectangle_opi:nnNnn {#1} {#2} }
1225 \cs_new_protected:Npn \__draw_softpath_rectangle_opi:nnNnn #1#2#3#4#5
1226   { \driver_draw_rectangle:nnnn {#1} {#2} {#4} {#5} }
1227   \cs_new_protected:Npn \__draw_softpath_rectangle_opii:nn #1#2 { }
```

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

34

5.2 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.

```
1228 \tl_new:N \l__draw_softpath_main_tl
```

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

`\l__draw_softpath_part_tl`   Data structures.

```
1229 \tl_new:N \l__draw_softpath_part_tl
1230 \tl_new:N \l__draw_softpath_curve_end_tl
```

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

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

```
1231 \fp_new:N \l__draw_softpath_lastx_fp
1232 \fp_new:N \l__draw_softpath_lasty_fp
1233 \dim_new:N \l__draw_softpath_corneri_dim
1234 \dim_new:N \l__draw_softpath_cornerii_dim
1235 \tl_new:N \l__draw_softpath_first_tl
1236 \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.

```
1237 \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:`   Rounding corners on a path means going through the entire path and adjusting it. As
`\__draw_softpath_round_loop:Nnn`   such, we avoid this entirely if we know there are no corners to deal with. Assuming there
`\__draw_softpath_round_action:nn`   is work to do, we recover the existing path and start a loop.
`\__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:`

```
1238 \cs_new_protected:Npn \__draw_softpath_round_corners:
1239   {
1240     \bool_if:NT \g__draw_softpath_corners_bool
1241       {
1242         \group_begin:
1243           \tl_clear:N \l__draw_softpath_main_tl
1244           \tl_clear:N \l__draw_softpath_part_tl
1245           \fp_zero:N \l__draw_softpath_lastx_fp
1246           \fp_zero:N \l__draw_softpath_lasty_fp
1247           \tl_clear:N \l__draw_softpath_first_tl
1248           \tl_clear:N \l__draw_softpath_move_tl
1249           \tl_build_get:NN \g__draw_softpath_main_tl \l__draw_softpath_internal_tl
1250           \exp_after:wN \__draw_softpath_round_loop:Nnn
1251             \l__draw_softpath_internal_tl
1252             \q_recursion_tail ? ?
1253             \q_recursion_stop
1254         \group_end:
1255       }
1256     \bool_gset_false:N \g__draw_softpath_corners_bool
1257   }
```

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`.

```
1258 \cs_new_protected:Npn \__draw_softpath_round_loop:Nnn #1#2#3
1259   {
1260     \quark_if_recursion_tail_stop_do:Nn #1 { \__draw_softpath_round_end: }
1261     \token_if_eq_meaning:NNTF #1 \__draw_softpath_roundpoint_op:nn
1262       { \__draw_softpath_round_action:nn {#2} {#3} }
1263       {
1264         \tl_if_empty:NT \l__draw_softpath_first_tl
1265           { \tl_set:Nn \l__draw_softpath_first_tl { {#2} {#3} } }
1266         \fp_set:Nn \l__draw_softpath_lastx_fp {#2}
1267         \fp_set:Nn \l__draw_softpath_lasty_fp {#3}
1268         \token_if_eq_meaning:NNTF #1 \__draw_softpath_moveto_op:nn
1269           {
1270             \tl_put_right:No \l__draw_softpath_main_tl
1271               \l__draw_softpath_move_tl
1272             \tl_put_right:No \l__draw_softpath_main_tl
1273               \l__draw_softpath_part_tl
1274             \tl_set:Nn \l__draw_softpath_move_tl { #1 {#2} {#3} }
1275             \tl_clear:N \l__draw_softpath_first_tl
1276             \tl_clear:N \l__draw_softpath_part_tl
1277           }
1278           { \tl_put_right:Nn \l__draw_softpath_part_tl { #1 {#2} {#3} } }
1279         \__draw_softpath_round_loop:Nnn
1280       }
1281   }
1282 \cs_new_protected:Npn \__draw_softpath_round_action:nn #1#2
1283   {
1284     \dim_set:Nn \l__draw_softpath_corneri_dim {#1}
1285     \dim_set:Nn \l__draw_softpath_cornerii_dim {#2}
1286     \bool_lazy_and:nnTF
1287       { \dim_compare_p:nNn \l__draw_softpath_corneri_dim = { 0pt } }
1288       { \dim_compare_p:nNn \l__draw_softpath_cornerii_dim = { 0pt } }
1289       { \__draw_softpath_round_loop:Nnn }
1290       { \__draw_softpath_round_action:Nnn }
1291   }
```

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.

```
1292 \cs_new_protected:Npn \__draw_softpath_round_action:Nnn #1#2#3
1293   {
1294     \tl_if_empty:NT \l__draw_softpath_first_tl
1295       { \tl_set:Nn \l__draw_softpath_first_tl { {#2} {#3} } }
1296     \token_if_eq_meaning:NNTF #1 \__draw_softpath_curveto_opi:nn
1297       { \__draw_softpath_round_action_curveto:NnnNnn }
1298       {
1299         \token_if_eq_meaning:NNTF #1 \__draw_softpath_close_op:nn
1300           { \__draw_softpath_round_action_close: }
1301           {
1302             \token_if_eq_meaning:NNTF #1 \__draw_softpath_lineto_op:nn
```

```
1303              { \__draw_softpath_round_lookahead:NnnNnn }
1304              { \__draw_softpath_round_loop:Nnn }
1305          }
1306        }
1307      #1 {#2} {#3}
1308    }
```

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.

```
1309 \cs_new_protected:Npn \__draw_softpath_round_action_curveto:NnnNnn
1310    #1#2#3#4#5#6
1311    {
1312      \tl_put_right:Nn \l__draw_softpath_part_tl
1313        { #1 {#2} {#3} #4 {#5} {#6} }
1314      \fp_set:Nn \l__draw_softpath_lastx_fp {#5}
1315      \fp_set:Nn \l__draw_softpath_lasty_fp {#6}
1316      \__draw_softpath_round_lookahead:NnnNnn
1317    }
1318 \cs_new_protected:Npn \__draw_softpath_round_action_close:
1319    {
1320      \bool_lazy_and:nnTF
1321        { ! \tl_if_empty_p:N \l__draw_softpath_first_tl }
1322        { ! \tl_if_empty_p:N \l__draw_softpath_move_tl }
1323        {
1324          \exp_after:wN \__draw_softpath_round_close:nn
1325            \l__draw_softpath_first_tl
1326        }
1327        { \__draw_softpath_round_loop:Nnn }
1328    }
```

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.

```
1329 \cs_new_protected:Npn \__draw_softpath_round_lookahead:NnnNnn #1#2#3#4#5#6
1330    {
1331      \bool_lazy_any:nTF
1332        {
1333          { \token_if_eq_meaning_p:NN #4 \__draw_softpath_lineto_op:nn }
1334          { \token_if_eq_meaning_p:NN #4 \__draw_softpath_curveto_opi:nn }
1335          { \token_if_eq_meaning_p:NN #4 \__draw_softpath_close_op:nn }
1336        }
1337        {
1338          \__draw_softpath_round_calc:nnnNnn
1339            \__draw_softpath_round_loop:Nnn {#5} {#6}
1340        }
1341        {
1342          \token_if_eq_meaning:NNTF #4 \__draw_softpath_roundpoint_op:nn
1343            { \__draw_softpath_round_roundpoint:NnnNnnNnn }
1344            { \__draw_softpath_round_loop:Nnn }
1345        }
1346      #1 {#2} {#3}
1347      #4 {#5} {#6}
1348    }
1349 \cs_new_protected:Npn \__draw_softpath_round_roundpoint:NnnNnnNnn
1350    #1#2#3#4#5#6#7#8#9
```

```
1351    {
1352      \__draw_softpath_round_calc:nnnNnn
1353        \__draw_softpath_round_loop:Nnn
1354        {#8} {#9} #1 {#2} {#3}
1355      #4 {#5} {#6} #7 {#8} {#9}
1356    }
```

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 (`#4`, `#5`), and where the next point is (`#1`, `#2`). 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.

```
1357  \cs_new_protected:Npn \__draw_softpath_round_calc:nnnNnn #1#2#3#4#5#6
1358    {
1359      \tl_set:Nx \l__draw_softpath_curve_end_tl
1360        {
1361          \draw_point_interpolate_distance:nnn
1362            \l__draw_softpath_cornerii_dim
1363            { #5 , #6 } { #2 , #3 }
1364        }
1365      \tl_put_right:Nx \l__draw_softpath_part_tl
1366        {
1367          \exp_not:N #4
1368          \__draw_softpath_round_calc:fVnnnn
1369            {
1370              \draw_point_interpolate_distance:nnn
1371                \l__draw_softpath_corneri_dim
1372                { #5 , #6 }
1373                {
1374                  \l__draw_softpath_lastx_fp ,
1375                  \l__draw_softpath_lasty_fp
1376                }
1377            }
1378            \l__draw_softpath_curve_end_tl
1379            {#5} {#6} {#2} {#3}
1380        }
1381      \fp_set:Nn \l__draw_softpath_lastx_fp {#5}
1382      \fp_set:Nn \l__draw_softpath_lasty_fp {#6}
1383      #1
1384    }
```

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).

```
1385  \cs_new:Npn \__draw_softpath_round_calc:nnnnnn #1#2#3#4#5#6
1386    {
1387      \__draw_softpath_round_calc:nnnnw {#3} {#4} {#5} {#6}
1388        #1 \q_mark #2 \q_stop
1389    }
1390  \cs_generate_variant:Nn \__draw_softpath_round_calc:nnnnnn { fV }
```

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

```
1391  \cs_new:Npn \__draw_softpath_round_calc:nnnnw
1392    #1#2#3#4 #5 , #6 \q_mark #7 , #8 \q_stop
1393    {
1394      {#5} {#6}
1395      \exp_not:N \__draw_softpath_curveto_opi:nn
1396        {
1397          \fp_to_dim:n
1398            { #5 + \c__draw_softpath_arc_fp * ( #1 - #5 ) }
1399        }
1400        {
1401          \fp_to_dim:n
1402            { #6 + \c__draw_softpath_arc_fp * ( #2 - #6 ) }
1403        }
1404      \exp_not:N \__draw_softpath_curveto_opii:nn
1405        {
1406          \fp_to_dim:n
1407            { #7 + \c__draw_softpath_arc_fp * ( #1 - #7 ) }
1408        }
1409        {
1410          \fp_to_dim:n
1411            { #8 + \c__draw_softpath_arc_fp* ( #2 - #8 ) }
1412        }
1413      \exp_not:N \__draw_softpath_curveto_opiii:nn
1414        {#7} {#8}
1415    }
```

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.

```
1416  \cs_new_protected:Npn \__draw_softpath_round_close:nn #1#2
1417    {
1418      \use:x
1419        {
1420          \__draw_softpath_round_calc:nnnNnn
1421            {
1422              \tl_set:Nx \exp_not:N \l__draw_softpath_move_tl
1423                {
1424                  \__draw_softpath_moveto_op:nn
1425                  \exp_not:N \exp_after:wN
1426                    \exp_not:N \__draw_softpath_round_close:w
1427                    \exp_not:N \l__draw_softpath_curve_end_tl
1428                      \exp_not:N \q_stop
1429                }
1430              \use:x
1431                {
1432                  \exp_not:N \exp_not:N \exp_not:N \use_i:nnnn
1433                    {
1434                      \__draw_softpath_round_loop:Nnn
1435                      \__draw_softpath_close_op:nn
1436                      \exp_not:N \exp_after:wN
1437                        \exp_not:N \__draw_softpath_round_close:w
1438                        \exp_not:N \l__draw_softpath_curve_end_tl
1439                          \exp_not:N \q_stop
```

```
1440                              }
1441                          }
1442                      }
1443                  {#1} {#2}
1444                  \__draw_softpath_lineto_op:nn
1445                  \exp_after:wN \use_none:n \l__draw_softpath_move_tl
1446          }
1447      }
1448 \cs_new:Npn \__draw_softpath_round_close:w #1 , #2 \q_stop { {#1} {#2} }
```

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

```
1449 \cs_new_protected:Npn \__draw_softpath_round_end:
1450      {
1451          \tl_put_right:No \l__draw_softpath_main_tl
1452              \l__draw_softpath_move_tl
1453          \tl_put_right:No \l__draw_softpath_main_tl
1454              \l__draw_softpath_part_tl
1455          \tl_build_gclear:N \g__draw_softpath_main_tl
1456          \__draw_softpath_add:o \l__draw_softpath_main_tl
1457      }
```

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

```
1458 ⟨/initex | package⟩
```

6   l3draw-state implementation

```
1459 ⟨*initex | package⟩
```

```
1460 ⟨@@=draw⟩
```

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

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

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

```
1461 \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.

```
1462 \dim_new:N \l_draw_default_linewidth_dim
1463 \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.

```
1464 \cs_new_protected:Npn \draw_linewidth:n #1
1465      {
1466          \dim_gset:Nn \g__draw_linewidth_dim { \fp_to_dim:n {#1} }
1467          \driver_draw_linewidth:n \g__draw_linewidth_dim
1468      }
```

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

`\draw_dash_pattern:nn`
`\l__draw_tmp_seq`

Evaluated all of the list and pass it to the driver layer.

```
1469 \cs_new_protected:Npn \draw_dash_pattern:nn #1#2
1470   {
1471     \group_begin:
1472       \seq_set_from_clist:Nn \l__draw_tmp_seq {#1}
1473       \seq_set_map:NNn \l__draw_tmp_seq \l__draw_tmp_seq
1474         { \fp_to_dim:n {##1} }
1475       \use:x
1476         {
1477           \driver_draw_dash_pattern:nn
1478             { \seq_use:Nn \l__draw_tmp_seq { , } }
1479             { \fp_to_dim:n {#2} }
1480         }
1481     \group_end:
1482   }
1483 \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.

```
1484 \cs_new_protected:Npn \draw_miterlimit:n #1
1485   { \driver_draw_miterlimit:n { \fp_eval:n {#1} } }
```

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

`\draw_cap_butt:`
`\draw_cap_rectangle:`
`\draw_cap_round:`
`\draw_evenodd_rule:`
`\draw_nonzero_rule:`
`\draw_join_bevel:`
`\draw_join_miter:`
`\draw_join_round:`

All straight wrappers.

```
1486 \cs_new_protected:Npn \draw_cap_butt: { \driver_draw_cap_butt: }
1487 \cs_new_protected:Npn \draw_cap_rectangle: { \driver_draw_cap_rectangle: }
1488 \cs_new_protected:Npn \draw_cap_round: { \driver_draw_cap_round: }
1489 \cs_new_protected:Npn \draw_evenodd_rule: { \driver_draw_evenodd_rule: }
1490 \cs_new_protected:Npn \draw_nonzero_rule: { \driver_draw_nonzero_rule: }
1491 \cs_new_protected:Npn \draw_join_bevel: { \driver_draw_join_bevel: }
1492 \cs_new_protected:Npn \draw_join_miter: { \driver_draw_join_miter: }
1493 \cs_new_protected:Npn \draw_join_round: { \driver_draw_join_round: }
```

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

`\l__draw_color_tmp_tl`

Scratch space.

```
1494 \tl_new:N \l__draw_color_tmp_tl
```

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

`\draw_color:n`
`\draw_color_fill:n`
`\draw_color_stroke:n`
`\__draw_color:nn`
`\__draw_color_aux:nn`
`\__draw_color_aux:Vn`
`\__draw_color:nw`
`\__draw_select_cmyk:nw`
`\__draw_select_gray:nw`
`\__draw_select_rgb:nw`
`\__draw_split_select:nw`

Much the same as for core color support but calling the relevant driver-level function.

```
1495 \cs_new_eq:NN \draw_color:n \color_select:n
1496 \cs_new_protected:Npn \draw_color_fill:n #1
1497   { \__draw_color:nn { fill } {#1} }
1498 \cs_new_protected:Npn \draw_color_stroke:n #1
1499   { \__draw_color:nn { stroke } {#1} }
1500 \cs_new_protected:Npn \__draw_color:nn #1#2
1501   {
1502     \color_parse:nN {#2} \l__draw_color_tmp_tl
1503     \__draw_color_aux:Vn \l__draw_color_tmp_tl {#1}
1504   }
1505 \cs_new_protected:Npn \__draw_color_aux:nn #1#2
```

41

```
1506    { \__draw_color:nw {#2} #1 \q_stop }
1507  \cs_generate_variant:Nn \__draw_color_aux:nn { V }
1508  \cs_new_protected:Npn \__draw_color:nw #1#2 ~ #3 \q_stop
1509    { \use:c { __draw_color_ #2 :nw } {#1} #3 \q_stop }
1510  \cs_new_protected:Npn \__draw_color_cmyk:nw #1#2 ~ #3 ~ #4 ~ #5 \q_stop
1511    { \use:c { driver_draw_color_ #1 _cmyk:nnnn } {#2} {#3} {#4} {#5} }
1512  \cs_new_protected:Npn \__draw_color_gray:nw #1#2 \q_stop
1513    { \use:c { driver_draw_color_ #1 _gray:n } {#2} }
1514  \cs_new_protected:Npn \__draw_color_rgb:nw #1#2 ~ #3 ~ #4 \q_stop
1515    { \use:c { driver_draw_color_ #1 _rgb:nnn } {#2} {#3} {#4} }
1516  \cs_new_protected:Npn \__draw_color_spot:nw #1#2 ~ #3 \q_stop
1517    { \use:c { driver_draw_color_ #1 _spot:nn } {#2} {#3} }
```

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

```
1518  ⟨/initex | package⟩
```

7 **l3draw-transforms** implementation

```
1519  ⟨*initex | package⟩
```

```
1520  ⟨@@=draw⟩
```

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

- \pgfgettransform, \pgfgettransformentries: Awaiting use cases.

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

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

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

\l__draw_matrix_active_bool An internal flag to avoid redundant calculations.

```
1521  \bool_new:N \l__draw_matrix_active_bool
```

(*End definition for* \l__draw_matrix_active_bool.)

\l__draw_matrix_a_fp The active matrix and shifts.
\l__draw_matrix_b_fp
\l__draw_matrix_c_fp
\l__draw_xshift_dim
\l__draw_yshift_dim

```
1522  \fp_new:N \l__draw_matrix_a_fp
1523  \fp_new:N \l__draw_matrix_b_fp
1524  \fp_new:N \l__draw_matrix_c_fp
1525  \fp_new:N \l__draw_matrix_d_fp
1526  \dim_new:N \l__draw_xshift_dim
1527  \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:

```
1528  \cs_new_protected:Npn \draw_transform_matrix_reset:
1529    {
1530      \fp_set:Nn \l__draw_matrix_a_fp { 1 }
1531      \fp_zero:N \l__draw_matrix_b_fp
1532      \fp_zero:N \l__draw_matrix_c_fp
```

```
1533        \fp_set:Nn \l__draw_matrix_d_fp { 1 }
1534      }
1535    \cs_new_protected:Npn \draw_transform_shift_reset:
1536      {
1537        \dim_zero:N \l__draw_xshift_dim
1538        \dim_zero:N \l__draw_yshift_dim
1539      }
1540    \draw_transform_matrix_reset:
1541    \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:nnnn   Setting the transform matrix is straight-forward, with just a bit of expansion to sort out.
\draw_transform_shift:n   With the mechanism active, the identity matrix is set.
__draw_transform_shift:nn

```
1542    \cs_new_protected:Npn \draw_transform_matrix:nnnn #1#2#3#4
1543      {
1544        \fp_set:Nn \l__draw_matrix_a_fp {#1}
1545        \fp_set:Nn \l__draw_matrix_b_fp {#2}
1546        \fp_set:Nn \l__draw_matrix_c_fp {#3}
1547        \fp_set:Nn \l__draw_matrix_d_fp {#4}
1548        \bool_lazy_all:nTF
1549          {
1550            { \fp_compare_p:nNn \l__draw_matrix_a_fp = \c_one_fp }
1551            { \fp_compare_p:nNn \l__draw_matrix_b_fp = \c_zero_fp }
1552            { \fp_compare_p:nNn \l__draw_matrix_c_fp = \c_zero_fp }
1553            { \fp_compare_p:nNn \l__draw_matrix_d_fp = \c_one_fp }
1554          }
1555          { \bool_set_false:N \l__draw_matrix_active_bool }
1556          { \bool_set_true:N \l__draw_matrix_active_bool }
1557      }
1558    \cs_new_protected:Npn \draw_transform_shift:n #1
1559      {
1560        \__draw_point_process:nn
1561          { \__draw_transform_shift:nn } {#1}
1562      }
1563    \cs_new_protected:Npn \__draw_transform_shift:nn #1#2
1564      {
1565        \dim_set:Nn \l__draw_xshift_dim {#1}
1566        \dim_set:Nn \l__draw_yshift_dim {#2}
1567      }
```

(*End definition for* \draw_transform_matrix:nnnn, \draw_transform_shift:n, *and* __draw_transform_-
shift:nn. *These functions are documented on page* **??**.)

\draw_transform_matrix_concat:nnnn   Much the same story for adding to an existing matrix, with a bit of pre-expansion so
__draw_transform_concat:nnnn   that the calculation uses "frozen" values.
\draw_transform_shift_concat:n
__draw_transform_shift_concat:nn

```
1568    \cs_new_protected:Npn \draw_transform_matrix_concat:nnnn #1#2#3#4
1569      {
1570        \use:x
1571          {
1572            \__draw_transform_concat:nnnn
1573              { \fp_eval:n {#1} }
1574              { \fp_eval:n {#2} }
1575              { \fp_eval:n {#3} }
```

43

```
1576                    { \fp_eval:n {#4} }
1577                }
1578        }
1579 \cs_new_protected:Npn \__draw_transform_concat:nnnn #1#2#3#4
1580    {
1581        \use:x
1582            {
1583                \draw_transform_matrix:nnnn
1584                    { #1 * \l__draw_matrix_a_fp + #2 * \l__draw_matrix_c_fp }
1585                    { #1 * \l__draw_matrix_b_fp + #2 * \l__draw_matrix_d_fp }
1586                    { #3 * \l__draw_matrix_a_fp + #4 * \l__draw_matrix_c_fp }
1587                    { #3 * \l__draw_matrix_b_fp + #4 * \l__draw_matrix_d_fp }
1588            }
1589        }
1590 \cs_new_protected:Npn \draw_transform_shift_concat:n #1
1591    {
1592        \__draw_point_process:nn
1593            { \__draw_transform_shift_concat:nn } {#1}
1594    }
1595 \cs_new_protected:Npn \__draw_transform_shift_concat:nn #1#2
1596    {
1597        \dim_set:Nn \l__draw_xshift_dim { \l__draw_xshift_dim + #1 }
1598        \dim_set:Nn \l__draw_yshift_dim { \l__draw_yshift_dim + #2 }
1599    }
```

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

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

```
1600 \cs_new_protected:Npn \draw_transform_matrix_invert:
1601    {
1602        \bool_if:NT \l__draw_matrix_active_bool
1603            {
1604                \__draw_transform_invert:f
1605                    {
1606                        \fp_eval:n
1607                            {
1608                                1 /
1609                                (
1610                                    \l__draw_matrix_a_fp * \l__draw_matrix_d_fp
1611                                    - \l__draw_matrix_b_fp * \l__draw_matrix_c_fp
1612                                )
1613                            }
1614                    }
1615            }
1616    }
1617 \cs_new_protected:Npn \__draw_transform_invert:n #1
1618    {
1619        \fp_set:Nn \l__draw_matrix_a_fp
1620            { \l__draw_matrix_d_fp * #1 }
1621        \fp_set:Nn \l__draw_matrix_b_fp
1622            { -\l__draw_matrix_b_fp * #1 }
1623        \fp_set:Nn \l__draw_matrix_c_fp
1624            { -\l__draw_matrix_c_fp * #1 }
```

44

```
1625        \fp_set:Nn \l__draw_matrix_d_fp
1626          { \l__draw_matrix_a_fp * #1 }
1627      }
1628   \cs_generate_variant:Nn \__draw_transform_invert:n { f }
1629   \cs_new_protected:Npn \draw_transform_shift_invert:
1630      {
1631        \dim_set:Nn \l__draw_xshift_dim { -\l__draw_xshift_dim }
1632        \dim_set:Nn \l__draw_yshift_dim { -\l__draw_yshift_dim }
1633      }
```

(*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.

```
1634   \cs_new_protected:Npn \draw_transform_triangle:nnn #1#2#3
1635      {
1636        \__draw_point_process:nnn
1637          {
1638            \__draw_point_process:nn
1639              { \__draw_tranform_triangle:nnnnnn }
1640              {#1}
1641          }
1642          {#2} {#3}
1643      }
1644   \cs_new_protected:Npn \__draw_tranform_triangle:nnnnnn #1#2#3#4#5#6
1645      {
1646        \use:x
1647          {
1648            \draw_transform_matrix:nnnn
1649              { #3 - #1 }
1650              { #4 - #2 }
1651              { #5 - #1 }
1652              { #6 - #2 }
1653            \draw_transform_shift:n { #1 , #2 }
1654          }
1655      }
```

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

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

```
1656   \cs_new_protected:Npn \draw_transform_scale:n #1
1657      { \draw_transform_matrix_concat:nnnn { #1 } { 0 } { 0 } { #1 } }
1658   \cs_new_protected:Npn \draw_transform_xscale:n #1
1659      { \draw_transform_matrix_concat:nnnn { #1 } { 0 } { 0 } { 1 } }
1660   \cs_new_protected:Npn \draw_transform_yscale:n #1
1661      { \draw_transform_matrix_concat:nnnn { 1 } { 0 } { 0 } { #1 } }
1662   \cs_new_protected:Npn \draw_transform_xshift:n #1
1663      { \draw_transform_shift_concat:n { #1 , 0 } }
1664   \cs_new_protected:Npn \draw_transform_yshift:n #1
1665      { \draw_transform_shift_concat:n { 0 , #1 } }
1666   \cs_new_protected:Npn \draw_transform_xslant:n #1
1667      { \draw_transform_matrix_concat:nnnn { 1 } { 0 } { #1 } { 1 } }
1668   \cs_new_protected:Npn \draw_transform_yslant:n #1
1669      { \draw_transform_matrix_concat:nnnn { 1 } { #1 } { 0 } { 1 } }
```

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

`\draw_transform_rotate:n`
`\__draw_transform_rotate:n`
`\__draw_transform_rotate:f`
`\__draw_transform_rotate:nn`
`\__draw_transform_rotate:ff`

Slightly more involved: evaluate the angle only once, and the sine and cosine only once.

```
1670 \cs_new_protected:Npn \draw_transform_rotate:n #1
1671   { \__draw_transform_rotate:f { \fp_eval:n {#1} } }
1672 \cs_new_protected:Npn \__draw_transform_rotate:n #1
1673   {
1674     \__draw_transform_rotate:ff
1675       { \fp_eval:n { cosd(#1) } }
1676       { \fp_eval:n { sind(#1) } }
1677   }
1678 \cs_generate_variant:Nn \__draw_transform_rotate:n { f }
1679 \cs_new_protected:Npn \__draw_transform_rotate:nn #1#2
1680   { \draw_transform_matrix_concat:nnnn {#1} {#2} { -#2 } { #1 } }
1681 \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* **??***.)*

```
1682 ⟨/initex | package⟩
```

46

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.

48

50

51

53