File I
Implementation

1 l3draw implementation

```
⟨*initex | package⟩
⟨@@=draw⟩
⟨*package⟩
\ProvidesExplPackage{l3draw}{2019-03-05}{}
  {L3 Experimental core drawing support}
⟨/package⟩
\RequirePackage { l3color }
```

Everything else is in the sub-files!

```
⟨/initex | package⟩
```

2 l3draw-boxes implementation

```
⟨*initex | package⟩
⟨@@=draw⟩
```

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

\l__draw_tmp_box

```
\box_new:N \l__draw_tmp_box
```

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

\draw_box_use:N
__draw_box_use:Nnnnn

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

```
\cs_new_protected:Npn \draw_box_use:N #1
  {
    \__draw_box_use:Nnnnn #1
      { 0pt } { -\box_dp:N #1 } { \box_wd:N #1 } { \box_ht:N #1 }
  }
\cs_new_protected:Npn \__draw_box_use:Nnnnn #1#2#3#4#5
  {
    \bool_if:NT \l_draw_bb_update_bool
      {
        \__draw_point_process:nn
          { \__draw_path_update_limits:nn }
          { \draw_point_transform:n { #2 , #3 } }
        \__draw_point_process:nn
          { \__draw_path_update_limits:nn }
          { \draw_point_transform:n { #4 , #3 } }
        \__draw_point_process:nn
          { \__draw_path_update_limits:nn }
          { \draw_point_transform:n { #4 , #5 } }
        \__draw_point_process:nn
          { \__draw_path_update_limits:nn }
          { \draw_point_transform:n { #2 , #5 } }
      }
    \group_begin:
      \hbox_set:Nn \l__draw_tmp_box
        {
          \use:x
            {
              \driver_draw_box_use:Nnnnn #1
                { \fp_use:N \l__draw_matrix_a_fp }
                { \fp_use:N \l__draw_matrix_b_fp }
                { \fp_use:N \l__draw_matrix_c_fp }
                { \fp_use:N \l__draw_matrix_d_fp }
            }
        }
      \hbox_set:Nn \l__draw_tmp_box
        {
          \tex_kern:D \l__draw_xshift_dim
          \box_move_up:nn { \l__draw_yshift_dim }
            { \box_use_drop:N \l__draw_tmp_box }
        }
      \box_set_ht:Nn \l__draw_tmp_box { 0pt }
      \box_set_dp:Nn \l__draw_tmp_box { 0pt }
      \box_set_wd:Nn \l__draw_tmp_box { 0pt }
      \box_use_drop:N \l__draw_tmp_box
    \group_end:
  }
```

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

`\draw_coffin_use:Nnn` Slightly more than a shortcut: we have to allow for the fact that coffins have no apparent width before the reference point.

```
\cs_new_protected:Npn \draw_coffin_use:Nnn #1#2#3
  {
    \group_begin:
      \hbox_set:Nn \l__draw_tmp_box
        { \coffin_typeset:Nnnnn #1 {#2} {#3} { 0pt } { 0pt } }
      \__draw_box_use:Nnnnn \l__draw_tmp_box
        { \box_wd:N \l__draw_tmp_box - \coffin_wd:N #1 }
        { -\box_dp:N \l__draw_tmp_box }
        { \box_wd:N \l__draw_tmp_box }
        { \box_ht:N \l__draw_tmp_box }
    \group_end:
  }
```

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

```
⟨/initex | package⟩
```

3 l3draw-layers implementation

```
⟨*initex | package⟩
```

```
⟨@@=draw⟩
```

3.1 User interface

\draw_layer_new:n

```
\cs_new_protected:Npn \draw_layer_new:n #1
  {
    \str_if_eq:nnTF {#1} { main }
      { \msg_error:nnn { draw } { main-reserved } }
      {
        \box_new:c { g__draw_layer_ #1 _box }
        \box_new:c { l__draw_layer_ #1 _box }
      }
  }
```

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

\l__draw_layer_tl

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

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

(*End definition for* \l__draw_layer_tl*.*)

\l__draw_layer_close_bool

Used to track if a layer needs to be closed.

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

(*End definition for* \l__draw_layer_close_bool*.*)

\l_draw_layers_clist
\g__draw_layers_clist

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

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

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

\draw_layer_begin:n
\draw_layer_end:

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

```
\cs_new_protected:Npn \draw_layer_begin:n #1
  {
    \group_begin:
      \box_if_exist:cTF { g__draw_layer_ #1 _box }
        {
          \str_if_eq:VnTF \l__draw_layer_tl {#1}
            { \bool_set_false:N \l__draw_layer_close_bool }
            {
              \bool_set_true:N \l__draw_layer_close_bool
              \tl_set:Nn \l__draw_layer_tl {#1}
              \box_gset_wd:cn { g__draw_layer_ #1 _box } { 0pt }
              \hbox_gset:cw { g__draw_layer_ #1 _box }
                \box_use_drop:c { g__draw_layer_ #1 _box }
                \group_begin:
            }
          \draw_linewidth:n { \l_draw_default_linewidth_dim }
        }
        {
          \str_if_eq:nnTF {#1} { main }
            { \msg_error:nnn { draw } { unknown-layer } {#1} }
            { \msg_error:nnn { draw } { main-layer } }
        }
  }
\cs_new_protected:Npn \draw_layer_end:
  {
      \bool_if:NT \l__draw_layer_close_bool
        {
            \group_end:
          \hbox_gset_end:
        }
    \group_end:
  }
```

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

3.2 Internal cross-links

`\__draw_layers_insert:`

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

```
\cs_new_protected:Npn \__draw_layers_insert:
  {
    \clist_map_inline:Nn \l_draw_layers_clist
      {
        \str_if_eq:nnTF {##1} { main }
          {
            \box_set_wd:Nn \l__draw_layer_main_box { 0pt }
            \box_use_drop:N \l__draw_layer_main_box
          }
          {
            \driver_draw_scope_begin:
            \box_gset_wd:cn { g__draw_layer_ ##1 _box } { 0pt }
            \box_use_drop:c { g__draw_layer_ ##1 _box }
            \driver_draw_scope_end:
          }
      }
  }
```

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

`\__draw_layers_save:`
`\__draw_layers_restore:`

Simple save/restore functions.

```
\cs_new_protected:Npn \__draw_layers_save:
  {
    \clist_map_inline:Nn \l_draw_layers_clist
      {
        \str_if_eq:nnF {##1} { main }
          {
            \box_set_eq:cc { l__draw_layer_ ##1 _box }
              { g__draw_layer_ ##1 _box }
          }
      }
  }
\cs_new_protected:Npn \__draw_layers_restore:
  {
    \clist_map_inline:Nn \l_draw_layers_clist
      {
        \str_if_eq:nnF {##1} { main }
          {
            \box_gset_eq:cc { g__draw_layer_ ##1 _box }
              { l__draw_layer_ ##1 _box }
          }
      }
  }
```

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

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

4 l3draw-paths implementation

```
⟨*initex | package⟩
⟨@@=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 Ti*k*Z 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.

```
\tl_new:N \l__draw_path_tmp_tl
\fp_new:N \l__draw_path_tmpa_fp
\fp_new:N \l__draw_path_tmpb_fp
```

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

4.1 Tracking paths

\g__draw_path_lastx_dim
\g__draw_path_lasty_dim

The last point visited on a path.

```
\dim_new:N \g__draw_path_lastx_dim
\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.

```
\dim_new:N \g__draw_path_xmax_dim
\dim_new:N \g__draw_path_xmin_dim
\dim_new:N \g__draw_path_ymax_dim
\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.)

```
\cs_new_protected:Npn \__draw_path_update_limits:nn #1#2
  {
    \dim_gset:Nn \g__draw_path_xmax_dim
      { \dim_max:nn \g__draw_path_xmax_dim {#1} }
    \dim_gset:Nn \g__draw_path_xmin_dim
      { \dim_min:nn \g__draw_path_xmin_dim {#1} }
    \dim_gset:Nn \g__draw_path_ymax_dim
      { \dim_max:nn \g__draw_path_ymax_dim {#2} }
    \dim_gset:Nn \g__draw_path_ymin_dim
      { \dim_min:nn \g__draw_path_ymin_dim {#2} }
    \bool_if:NT \l_draw_bb_update_bool
      {
        \dim_gset:Nn \g__draw_xmax_dim
          { \dim_max:nn \g__draw_xmax_dim {#1} }
        \dim_gset:Nn \g__draw_xmin_dim
          { \dim_min:nn \g__draw_xmin_dim {#1} }
        \dim_gset:Nn \g__draw_ymax_dim
          { \dim_max:nn \g__draw_ymax_dim {#2} }
        \dim_gset:Nn \g__draw_ymin_dim
          { \dim_min:nn \g__draw_ymin_dim {#2} }
      }
  }
\cs_new_protected:Npn \__draw_path_reset_limits:
  {
    \dim_gset:Nn \g__draw_path_xmax_dim { -\c_max_dim }
    \dim_gset:Nn \g__draw_path_xmin_dim {  \c_max_dim }
    \dim_gset:Nn \g__draw_path_ymax_dim { -\c_max_dim }
    \dim_gset:Nn \g__draw_path_ymin_dim {  \c_max_dim }
  }
```

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

__draw_path_update_last:nn

A simple auxiliary to avoid repetition.

```
\cs_new_protected:Npn \__draw_path_update_last:nn #1#2
  {
    \dim_gset:Nn \g__draw_path_lastx_dim {#1}
    \dim_gset:Nn \g__draw_path_lasty_dim {#2}
  }
```

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

4.2 Corner arcs

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

`\l__draw_corner_xarc_dim`
`\l__draw_corner_yarc_dim`

The two arcs in use.

```
\dim_new:N \l__draw_corner_xarc_dim
\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.

```
\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.

```
\cs_new_protected:Npn \draw_path_corner_arc:nn #1#2
  {
    \dim_set:Nn \l__draw_corner_xarc_dim {#1}
    \dim_set:Nn \l__draw_corner_yarc_dim {#2}
    \bool_lazy_and:nnTF
      { \dim_compare_p:nNn \l__draw_corner_xarc_dim = { 0pt } }
      { \dim_compare_p:nNn \l__draw_corner_yarc_dim = { 0pt } }
      { \bool_set_false:N \l__draw_corner_arc_bool }
      { \bool_set_true:N \l__draw_corner_arc_bool }
  }
```

(*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.

```
\cs_new_protected:Npn \__draw_path_mark_corner:
  {
    \bool_if:NT \l__draw_corner_arc_bool
      {
        \__draw_softpath_roundpoint:VV
          \l__draw_corner_xarc_dim
          \l__draw_corner_yarc_dim
      }
  }
```

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

4.3 Basic path constructions

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

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

```
\cs_new_protected:Npn \draw_path_moveto:n #1
  {
    \__draw_point_process:nn
      { \__draw_path_moveto:nn }
      { \draw_point_transform:n {#1} }
  }
\cs_new_protected:Npn \__draw_path_moveto:nn #1#2
  {
     \__draw_path_update_limits:nn {#1} {#2}
     \__draw_softpath_moveto:nn {#1} {#2}
     \__draw_path_update_last:nn {#1} {#2}
  }
\cs_new_protected:Npn \draw_path_lineto:n #1
  {
    \__draw_point_process:nn
      { \__draw_path_lineto:nn }
      { \draw_point_transform:n {#1} }
  }
\cs_new_protected:Npn \__draw_path_lineto:nn #1#2
  {
     \__draw_path_mark_corner:
     \__draw_path_update_limits:nn {#1} {#2}
     \__draw_softpath_lineto:nn {#1} {#2}
     \__draw_path_update_last:nn {#1} {#2}
  }
\cs_new_protected:Npn \draw_path_curveto:nnn #1#2#3
  {
    \__draw_point_process:nnnn
      {
        \__draw_path_mark_corner:
        \__draw_path_curveto:nnnnnn
      }
      { \draw_point_transform:n {#1} }
      { \draw_point_transform:n {#2} }
      { \draw_point_transform:n {#3} }
  }
\cs_new_protected:Npn \__draw_path_curveto:nnnnnn #1#2#3#4#5#6
  {
     \__draw_path_update_limits:nn {#1} {#2}
     \__draw_path_update_limits:nn {#3} {#4}
     \__draw_path_update_limits:nn {#5} {#6}
     \__draw_softpath_curveto:nnnnnn {#1} {#2} {#3} {#4} {#5} {#6}
     \__draw_path_update_last:nn {#5} {#6}
  }
```

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

\draw_path_close:

A simple wrapper.

```
\cs_new_protected:Npn \draw_path_close:
  {
    \__draw_path_mark_corner:
    \__draw_softpath_closepath:
  }
```

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

4.4 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.

```
\cs_new_protected:Npn \draw_path_canvas_moveto:n #1
  { \__draw_point_process:nn { \__draw_path_moveto:nn } {#1} }
\cs_new_protected:Npn \draw_path_canvas_lineto:n #1
  { \__draw_point_process:nn { \__draw_path_lineto:nn } {#1} }
\cs_new_protected:Npn \draw_path_canvas_curveto:nnn #1#2#3
  {
    \__draw_point_process:nnnn
      {
        \__draw_path_mark_corner:
        \__draw_path_curveto:nnnnnn
      }
      {#1} {#2} {#3}
  }
```

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

4.5 Computed curves

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

`\draw_path_curveto:nn`
`\__draw_path_curveto:nnnn`
`\c__draw_path_curveto_a_fp`
`\c__draw_path_curveto_b_fp`

A quadratic curve with one control point $(x_{\mathrm{c}}, y_{\mathrm{c}})$. The two required control points are then

$$x_1 = \frac{1}{3}x_{\mathrm{s}} + \frac{2}{3}x_{\mathrm{c}} \quad y_1 = \frac{1}{3}y_{\mathrm{s}} + \frac{2}{3}y_{\mathrm{c}}$$

and

$$x_2 = \frac{1}{3}x_{\mathrm{e}} + \frac{2}{3}x_{\mathrm{c}} \quad x_2 = \frac{1}{3}y_{\mathrm{e}} + \frac{2}{3}y_{\mathrm{c}}$$

using the start (last) point $(x_{\mathrm{s}}, y_{\mathrm{s}})$ and the end point $(x_{\mathrm{s}}, y_{\mathrm{s}})$.

```
\cs_new_protected:Npn \draw_path_curveto:nn #1#2
  {
    \__draw_point_process:nnn
      { \__draw_path_curveto:nnnn }
      { \draw_point_transform:n {#1} }
      { \draw_point_transform:n {#2} }
  }
\cs_new_protected:Npn \__draw_path_curveto:nnnn #1#2#3#4
  {
    \fp_set:Nn \l__draw_path_tmpa_fp { \c__draw_path_curveto_b_fp * #1 }
    \fp_set:Nn \l__draw_path_tmpb_fp { \c__draw_path_curveto_b_fp * #2 }
    \use:x
      {
         \__draw_path_mark_corner:
         \__draw_path_curveto:nnnnnn
```

```
          {
            \fp_to_dim:n
              {
                  \c__draw_path_curveto_a_fp * \g__draw_path_lastx_dim
                + \l__draw_path_tmpa_fp
              }
          }
          {
            \fp_to_dim:n
              {
                  \c__draw_path_curveto_a_fp * \g__draw_path_lasty_dim
                + \l__draw_path_tmpb_fp
              }
          }
          {
            \fp_to_dim:n
              { \c__draw_path_curveto_a_fp * #3 + \l__draw_path_tmpa_fp }
          }
          {
            \fp_to_dim:n
              { \c__draw_path_curveto_a_fp * #4 + \l__draw_path_tmpb_fp }
          }
          {#3}
          {#4}
      }
  }
\fp_const:Nn \c__draw_path_curveto_a_fp { 1 / 3 }
\fp_const:Nn \c__draw_path_curveto_b_fp { 2 / 3 }
```

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

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

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

```
\cs_new_protected:Npn \draw_path_arc:nnn #1#2#3
  { \draw_path_arc:nnnn {#1} {#2} {#3} {#3} }
\cs_new_protected:Npn \draw_path_arc:nnnn #1#2#3#4
  {
    \use:x
      {
        \__draw_path_arc:nnnn
          { \fp_eval:n {#1} }
          { \fp_eval:n {#2} }
          { \fp_to_dim:n {#3} }
          { \fp_to_dim:n {#4} }
       }
  }
\cs_new_protected:Npn \__draw_path_arc:nnnn #1#2#3#4
  {
    \fp_compare:nNnTF {#1} > {#2}
      { \__draw_path_arc:nnNnn {#1} {#2} - {#3} {#4} }
      { \__draw_path_arc:nnNnn {#1} {#2} + {#3} {#4} }
  }
\cs_new_protected:Npn \__draw_path_arc:nnNnn #1#2#3#4#5
```

```
  {
    \fp_set:Nn \l__draw_path_arc_start_fp {#1}
    \fp_set:Nn \l__draw_path_arc_delta_fp { abs( #1 - #2 ) }
    \fp_while_do:nNnn { \l__draw_path_arc_delta_fp } > { 90 }
      {
        \fp_compare:nNnTF \l__draw_path_arc_delta_fp > { 115 }
          {
            \__draw_path_arc_auxi:ffnnNnn
              { \fp_to_decimal:N \l__draw_path_arc_start_fp }
              { \fp_eval:n { \l__draw_path_arc_start_fp #3 90 } }
              { 90 } {#2}
              #3 {#4} {#5}
          }
          {
            \__draw_path_arc_auxi:ffnnNnn
              { \fp_to_decimal:N \l__draw_path_arc_start_fp }
              { \fp_eval:n { \l__draw_path_arc_start_fp #3 60 } }
              { 60 } {#2}
              #3 {#4} {#5}
          }
      }
    \__draw_path_mark_corner:
    \__draw_path_arc_auxi:fnfnNnn
      { \fp_to_decimal:N \l__draw_path_arc_start_fp }
      {#2}
      { \fp_eval:n { abs( \l__draw_path_arc_start_fp - #2 ) } }
      {#2}
      #3 {#4} {#5}
  }
```

The auxiliary is responsible for calculating the required points. The “magic” number required to determine the length of the control vectors is well-established for a right-angle: $\frac{4}{3}(\sqrt{2}-1) = 0.552\,284\,75$. For other cases, we follow the calculation used by pgf but with the second common case of $60°$ pre-calculated for speed.

```
\cs_new_protected:Npn \__draw_path_arc_auxi:nnnnNnn #1#2#3#4#5#6#7
  {
    \use:x
      {
        \__draw_path_arc_auxii:nnnNnnnn
          {#1} {#2} {#4} #5 {#6} {#7}
          {
            \fp_to_dim:n
              {
                \cs_if_exist_use:cF
                  { c__draw_path_arc_ #3 _fp }
                  { 4/3 * tand( 0.25 * #3 ) }
                  * #6
              }
          }
          {
            \fp_to_dim:n
              {
                \cs_if_exist_use:cF
                  { c__draw_path_arc_ #3 _fp }
                  { 4/3 * tand( 0.25 * #3 ) }
                  * #7
              }
          }
      }
  }
\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

```
\cs_new_protected:Npn \__draw_path_arc_auxii:nnnNnnnn #1#2#3#4#5#6#7#8
  {
    \tl_clear:N \l__draw_path_tmp_tl
    \__draw_point_process:nn
      { \__draw_path_arc_auxiii:nn }
      {
        \__draw_point_transform_noshift:n
          { \draw_point_polar:nnn { #1 #4 90 } {#7} {#8} }
      }
    \__draw_point_process:nnn
      { \__draw_path_arc_auxiv:nnnn }
      {
        \draw_point_transform:n
          { \draw_point_polar:nnn {#1} {#5} {#6} }
      }
      {
        \draw_point_transform:n
          { \draw_point_polar:nnn {#2} {#5} {#6} }
      }
    \__draw_point_process:nn
      { \__draw_path_arc_auxv:nn }
      {
        \__draw_point_transform_noshift:n
          { \draw_point_polar:nnn { #2 #4 -90 } {#7} {#8} }
      }
    \exp_after:wN \__draw_path_curveto:nnnnnn \l__draw_path_tmp_tl
    \fp_set:Nn \l__draw_path_arc_delta_fp { abs ( #2 - #3 ) }
    \fp_set:Nn \l__draw_path_arc_start_fp {#2}
  }
```

The first control point.

```
\cs_new_protected:Npn \__draw_path_arc_auxiii:nn #1#2
  {
    \__draw_path_arc_aux_add:nn
      { \g__draw_path_lastx_dim + #1 }
      { \g__draw_path_lasty_dim + #2 }
  }
```

The end point: simple arithmetic.

```
\cs_new_protected:Npn \__draw_path_arc_auxiv:nnnn #1#2#3#4
  {
    \__draw_path_arc_aux_add:nn
```

```
      { \g__draw_path_lastx_dim - #1 + #3 }
      { \g__draw_path_lasty_dim - #2 + #4 }
  }
```

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

```
\cs_new_protected:Npn \__draw_path_arc_auxv:nn #1#2
  {
    \exp_after:wN \__draw_path_arc_auxvi:nn
      \l__draw_path_tmp_tl {#1} {#2}
  }
\cs_new_protected:Npn \__draw_path_arc_auxvi:nn #1#2#3#4#5#6
  {
    \tl_set:Nn \l__draw_path_tmp_tl { {#1} {#2} }
    \__draw_path_arc_aux_add:nn
      { #5 + #3 }
      { #6 + #4 }
    \tl_put_right:Nn \l__draw_path_tmp_tl { {#3} {#4} }
  }
\cs_new_protected:Npn \__draw_path_arc_aux_add:nn #1#2
  {
    \tl_put_right:Nx \l__draw_path_tmp_tl
      { { \fp_to_dim:n {#1} } { \fp_to_dim:n {#2} } }
  }
\fp_new:N \l__draw_path_arc_delta_fp
\fp_new:N \l__draw_path_arc_start_fp
\fp_const:cn { c__draw_path_arc_90_fp } { 4/3 * (sqrt(2) - 1) }
\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.

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

(*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.

```
\cs_new_protected:Npn \draw_path_ellipse:nnn #1#2#3
  {
    \__draw_point_process:nnnn
      { \__draw_path_ellipse:nnnnnn }
      { \draw_point_transform:n {#1} }
      { \__draw_point_transform_noshift:n {#2} }
      { \__draw_point_transform_noshift:n {#3} }
  }
\cs_new_protected:Npn \__draw_path_ellipse:nnnnnn #1#2#3#4#5#6
  {
    \use:x
      {
        \__draw_path_moveto:nn
          { \fp_to_dim:n { #1 + #3 } } { \fp_to_dim:n { #2 + #4 } }
        \__draw_path_ellipse_arci:nnnnnn   {#1} {#2} {#3} {#4} {#5} {#6}
        \__draw_path_ellipse_arcii:nnnnnn  {#1} {#2} {#3} {#4} {#5} {#6}
        \__draw_path_ellipse_arciii:nnnnnn {#1} {#2} {#3} {#4} {#5} {#6}
        \__draw_path_ellipse_arciv:nnnnnn  {#1} {#2} {#3} {#4} {#5} {#6}
      }
    \__draw_softpath_closepath:
    \__draw_path_moveto:nn {#1} {#2}
  }
\cs_new:Npn \__draw_path_ellipse_arci:nnnnnn #1#2#3#4#5#6
  {
    \__draw_path_curveto:nnnnnn
      { \fp_to_dim:n { #1 + #3 + #5 * \c__draw_path_ellipse_fp } }
      { \fp_to_dim:n { #2 + #4 + #6 * \c__draw_path_ellipse_fp } }
      { \fp_to_dim:n { #1 + #3 * \c__draw_path_ellipse_fp + #5 } }
      { \fp_to_dim:n { #2 + #4 * \c__draw_path_ellipse_fp + #6 } }
      { \fp_to_dim:n { #1 + #5 } }
      { \fp_to_dim:n { #2 + #6 } }
  }
\cs_new:Npn \__draw_path_ellipse_arcii:nnnnnn #1#2#3#4#5#6
  {
    \__draw_path_curveto:nnnnnn
      { \fp_to_dim:n { #1 - #3 * \c__draw_path_ellipse_fp + #5 } }
      { \fp_to_dim:n { #2 - #4 * \c__draw_path_ellipse_fp + #6 } }
      { \fp_to_dim:n { #1 - #3 + #5 * \c__draw_path_ellipse_fp } }
      { \fp_to_dim:n { #2 - #4 + #6 * \c__draw_path_ellipse_fp } }
      { \fp_to_dim:n { #1 - #3 } }
      { \fp_to_dim:n { #2 - #4 } }
  }
\cs_new:Npn \__draw_path_ellipse_arciii:nnnnnn #1#2#3#4#5#6
  {
    \__draw_path_curveto:nnnnnn
      { \fp_to_dim:n { #1 - #3 - #5 * \c__draw_path_ellipse_fp } }
      { \fp_to_dim:n { #2 - #4 - #6 * \c__draw_path_ellipse_fp } }
      { \fp_to_dim:n { #1 - #3 * \c__draw_path_ellipse_fp - #5 } }
      { \fp_to_dim:n { #2 - #4 * \c__draw_path_ellipse_fp - #6 } }
      { \fp_to_dim:n { #1 - #5 } }
      { \fp_to_dim:n { #2 - #6 } }
  }
\cs_new:Npn \__draw_path_ellipse_arciv:nnnnnn #1#2#3#4#5#6
  {
    \__draw_path_curveto:nnnnnn
      { \fp_to_dim:n { #1 + #3 * \c__draw_path_ellipse_fp - #5 } }
      { \fp_to_dim:n { #2 + #4 * \c__draw_path_ellipse_fp - #6 } }
      { \fp_to_dim:n { #1 + #3 - #5 * \c__draw_path_ellipse_fp } }
      { \fp_to_dim:n { #2 + #4 - #6 * \c__draw_path_ellipse_fp } }
      { \fp_to_dim:n { #1 + #3 } }
      { \fp_to_dim:n { #2 + #4 } }
  }
\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.

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

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

4.6 Rectangles

\draw_path_rectangle:nn
__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.

```
\cs_new_protected:Npn \draw_path_rectangle:nn #1#2
  {
    \__draw_point_process:nnn
      {
        \bool_lazy_or:nnTF
          { \l__draw_corner_arc_bool }
          { \l__draw_matrix_active_bool }
          { \__draw_path_rectangle_rounded:nnnn }
          { \__draw_path_rectangle:nnnn }
      }
      { \draw_point_transform:n {#1} }
      {#2}
  }
\cs_new_protected:Npn \__draw_path_rectangle:nnnn #1#2#3#4
  {
    \__draw_path_update_limits:nn {#1} {#2}
    \__draw_path_update_limits:nn { #1 + #3 } { #2 + #4 }
    \__draw_softpath_rectangle:nnnn {#1} {#2} {#3} {#4}
    \__draw_path_update_last:nn {#1} {#2}
  }
\cs_new_protected:Npn \__draw_path_rectangle_rounded:nnnn #1#2#3#4
  {
    \draw_path_moveto:n { #1 + #3 , #2 + #4 }
    \draw_path_lineto:n { #1 , #2 + #4 }
    \draw_path_lineto:n { #1 , #2 }
    \draw_path_lineto:n { #1 + #3 , #2 }
    \draw_path_close:
    \draw_path_moveto:n { #1 , #2 }
  }
```

(*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.

```
\cs_new_protected:Npn \draw_path_rectangle_corners:nn #1#2
  {
    \__draw_point_process:nnn
      { \__draw_path_rectangle_corners:nnnnn {#1} }
      {#1} {#2}
  }
\cs_new_protected:Npn \__draw_path_rectangle_corners:nnnnn #1#2#3#4#5
  { \draw_path_rectangle:nn {#1} { #4 - #2 , #5 - #3 } }
```

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

4.7 Grids

\draw_path_grid:nnnn
__draw_path_grid_auxi:nnnnnn
__draw_path_grid_auxi:ffnnnn
__draw_path_grid_auxii:nnnnnn
__draw_path_grid_auxiii:nnnnnn
__draw_path_grid_auxiiii:ffnnnn
__draw_path_grid_auxiv:nnnnnnnn
__draw_path_grid_auxiv:ffnnnnnn

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

```
\cs_new_protected:Npn \draw_path_grid:nnnn #1#2#3#4
  {
    \__draw_point_process:nnn
      {
        \__draw_path_grid_auxi:ffnnnn
          { \dim_eval:n { \dim_abs:n {#1} } }
          { \dim_eval:n { \dim_abs:n {#2} } }
      }
      {#3} {#4}
  }
\cs_new_protected:Npn \__draw_path_grid_auxi:nnnnnn #1#2#3#4#5#6
  {
    \dim_compare:nNnTF {#3} > {#5}
      { \__draw_path_grid_auxii:nnnnnn {#1} {#2} {#5} {#4} {#3} {#6} }
      { \__draw_path_grid_auxii:nnnnnn {#1} {#2} {#3} {#4} {#5} {#6} }
  }
\cs_generate_variant:Nn \__draw_path_grid_auxi:nnnnnn { ff }
\cs_new_protected:Npn \__draw_path_grid_auxii:nnnnnn #1#2#3#4#5#6
  {
    \dim_compare:nNnTF {#4} > {#6}
      { \__draw_path_grid_auxiii:nnnnnn {#1} {#2} {#3} {#6} {#5} {#4} }
      { \__draw_path_grid_auxiii:nnnnnn {#1} {#2} {#3} {#4} {#5} {#6} }
  }
\cs_new_protected:Npn \__draw_path_grid_auxiii:nnnnnn #1#2#3#4#5#6
  {
    \__draw_path_grid_auxiv:ffnnnnnn
      { \fp_to_dim:n { #1 * trunc(#3/(#1)) } }
      { \fp_to_dim:n { #2 * trunc(#4/(#2)) } }
      {#1} {#2} {#3} {#4} {#5} {#6}
  }
\cs_new_protected:Npn \__draw_path_grid_auxiv:nnnnnnnn #1#2#3#4#5#6#7#8
  {
    \dim_step_inline:nnnn
      {#1}
      {#3}
      {#7}
      {
        \draw_path_moveto:n { ##1 , #6 }
        \draw_path_lineto:n { ##1 , #8 }
      }
    \dim_step_inline:nnnn
      {#2}
      {#4}
      {#8}
      {
        \draw_path_moveto:n { #5 , ##1 }
        \draw_path_lineto:n { #7 , ##1 }
      }
  }
\cs_generate_variant:Nn \__draw_path_grid_auxiv:nnnnnnnn { ff }
```

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

4.8 Using paths

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

Actions to pass to the driver.

```
\bool_new:N \l__draw_path_use_clip_bool
\bool_new:N \l__draw_path_use_fill_bool
\bool_new:N \l__draw_path_use_stroke_bool
```

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

`\l__draw_path_use_bb_bool`
`\l__draw_path_use_clear_bool`

Actions handled at the macro layer.

```
\bool_new:N \l__draw_path_use_bb_bool
\bool_new:N \l__draw_path_use_clear_bool
```

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

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

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

```
\cs_new_protected:Npn \draw_path_use:n #1
  {
    \tl_if_blank:nF {#1}
      { \__draw_path_use:n {#1} }
  }
\cs_new_protected:Npn \draw_path_use_clear:n #1
  {
    \bool_lazy_or:nnTF
      { \tl_if_blank_p:n {#1} }
      { \str_if_eq_p:nn {#1} { clear } }
      {
        \__draw_softpath_clear:
        \__draw_path_reset_limits:
      }
      { \__draw_path_use:n { #1 , clear } }
   }
```

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).

```
\cs_new_protected:Npn \__draw_path_use:n #1
  {
    \bool_set_false:N \l__draw_path_use_clip_bool
    \bool_set_false:N \l__draw_path_use_fill_bool
    \bool_set_false:N \l__draw_path_use_stroke_bool
    \clist_map_inline:nn {#1}
      {
        \cs_if_exist:cTF { l__draw_path_use_ ##1 _ bool }
          { \bool_set_true:c { l__draw_path_use_ ##1 _ bool } }
          {
            \cs_if_exist_use:cF { __draw_path_use_action_ ##1 : }
              { \ERROR }
          }
      }
    \__draw_softpath_round_corners:
    \bool_lazy_and:nnT
      { \l_draw_bb_update_bool }
      { \l__draw_path_use_stroke_bool }
      { \__draw_path_use_stroke_bb: }
    \bool_if:NTF \l__draw_path_use_clear_bool
      { \__draw_softpath_use_clear: }
      { \__draw_softpath_use: }
    \bool_if:NT \l__draw_path_use_clip_bool
      { \driver_draw_clip: }
    \bool_lazy_or:nnT
      { \l__draw_path_use_fill_bool }
      { \l__draw_path_use_stroke_bool }
      {
        \use:c
          {
            driver_draw_
            \bool_if:NT \l__draw_path_use_fill_bool { fill }
            \bool_if:NT \l__draw_path_use_stroke_bool { stroke }
            :
          }
      }
  }
\cs_new_protected:Npn \__draw_path_use_action_draw:
  {
    \bool_set_true:N \l__draw_path_use_stroke_bool
  }
\cs_new_protected:Npn \__draw_path_use_action_fillstroke:
  {
    \bool_set_true:N \l__draw_path_use_fill_bool
    \bool_set_true:N \l__draw_path_use_stroke_bool
  }
```

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.

```
\cs_new_protected:Npn \__draw_path_use_stroke_bb:
  {
    \__draw_path_use_stroke_bb_aux:NnN x { max } +
    \__draw_path_use_stroke_bb_aux:NnN y { max } +
    \__draw_path_use_stroke_bb_aux:NnN x { min } -
    \__draw_path_use_stroke_bb_aux:NnN y { min } -
  }
\cs_new_protected:Npn \__draw_path_use_stroke_bb_aux:NnN #1#2#3
  {
    \dim_compare:nNnF { \dim_use:c { g__draw_ #1#2 _dim } } = { #3 -\c_max_dim }
      {
        \dim_gset:cn { g__draw_ #1#2 _dim }
          {
            \use:c { dim_ #2 :nn }
              { \dim_use:c { g__draw_ #1#2 _dim } }
              {
                  \dim_use:c { g__draw_path_ #1#2 _dim }
                #3 0.5 \g__draw_linewidth_dim
                }
            }
        }
    }
```

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

4.9 Scoping paths

`\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`

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).

```
\dim_new:N \l__draw_path_lastx_dim
\dim_new:N \l__draw_path_lasty_dim
\dim_new:N \l__draw_path_xmax_dim
\dim_new:N \l__draw_path_xmin_dim
\dim_new:N \l__draw_path_ymax_dim
\dim_new:N \l__draw_path_ymin_dim
\dim_new:N \l__draw_softpath_lastx_dim
\dim_new:N \l__draw_softpath_lasty_dim
\bool_new:N \l__draw_softpath_corners_bool
```

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

`\draw_path_scope_begin:`
`\draw_path_scope_end:`

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

```
\cs_new_protected:Npn \draw_path_scope_begin:
  {
    \group_begin:
      \dim_set_eq:NN \l__draw_path_lastx_dim \g__draw_path_lastx_dim
      \dim_set_eq:NN \l__draw_path_lasty_dim \g__draw_path_lasty_dim
      \dim_set_eq:NN \l__draw_path_xmax_dim \g__draw_path_xmax_dim
      \dim_set_eq:NN \l__draw_path_xmin_dim \g__draw_path_xmin_dim
      \dim_set_eq:NN \l__draw_path_ymax_dim \g__draw_path_ymax_dim
      \dim_set_eq:NN \l__draw_path_ymin_dim \g__draw_path_ymin_dim
      \dim_set_eq:NN \l__draw_softpath_lastx_dim \g__draw_softpath_lastx_dim
      \dim_set_eq:NN \l__draw_softpath_lasty_dim \g__draw_softpath_lasty_dim
      \__draw_path_reset_limits:
      \tl_build_get:NN \g__draw_softpath_main_tl \l__draw_softpath_main_tl
      \bool_set_eq:NN
        \l__draw_softpath_corners_bool
        \g__draw_softpath_corners_bool
      \__draw_softpath_clear:
  }
\cs_new_protected:Npn \draw_path_scope_end:
  {
      \__draw_softpath_clear:
      \bool_gset_eq:NN
        \g__draw_softpath_corners_bool
        \l__draw_softpath_corners_bool
      \__draw_softpath_add:o \l__draw_softpath_main_tl
      \dim_gset_eq:NN \g__draw_softpath_lastx_dim \l__draw_softpath_lastx_dim
      \dim_gset_eq:NN \g__draw_softpath_lasty_dim \l__draw_softpath_lasty_dim
      \dim_gset_eq:NN \g__draw_path_xmax_dim \l__draw_path_xmax_dim
      \dim_gset_eq:NN \g__draw_path_xmin_dim \l__draw_path_xmin_dim
      \dim_gset_eq:NN \g__draw_path_ymax_dim \l__draw_path_ymax_dim
      \dim_gset_eq:NN \g__draw_path_ymin_dim \l__draw_path_ymin_dim
      \dim_gset_eq:NN \g__draw_path_lastx_dim \l__draw_path_lastx_dim
      \dim_gset_eq:NN \g__draw_path_lasty_dim \l__draw_path_lasty_dim
    \group_end:
  }
```

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

```
⟨/initex | package⟩
```

5 l3draw-points implementation

```
⟨*initex | package⟩
⟨@@=draw⟩
```

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

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

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

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

5.1 Support functions

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

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

```
\cs_new:Npn \__draw_point_process:nn #1#2
  {
    \exp_args:Nf \__draw_point_process_auxi:nn
```

```
      { \__draw_point_to_dim:n {#2} }
      {#1}
  }
\cs_new:Npn \__draw_point_process_auxi:nn #1#2
  { \__draw_point_process_auxii:nw {#2} #1 \q_stop }
\cs_new:Npn \__draw_point_process_auxii:nw #1 #2 , #3 \q_stop
  { #1 {#2} {#3} }
\cs_new:Npn \__draw_point_process:nnn #1#2#3
  {
    \exp_args:Nff \__draw_point_process_auxiii:nnn
      { \__draw_point_to_dim:n {#2} }
      { \__draw_point_to_dim:n {#3} }
      {#1}
  }
\cs_new:Npn \__draw_point_process_auxiii:nnn #1#2#3
  { \__draw_point_process_auxiv:nw {#3} #1 \q_mark #2 \q_stop }
\cs_new:Npn \__draw_point_process_auxiv:nw #1 #2 , #3 \q_mark #4 , #5 \q_stop
  { #1 {#2} {#3} {#4} {#5} }
\cs_new:Npn \__draw_point_process:nnnn #1#2#3#4
  {
    \exp_args:Nfff \__draw_point_process_auxv:nnnn
      { \__draw_point_to_dim:n {#2} }
      { \__draw_point_to_dim:n {#3} }
      { \__draw_point_to_dim:n {#4} }
      {#1}
  }
\cs_new:Npn \__draw_point_process_auxv:nnnn #1#2#3#4
  { \__draw_point_process_auxvi:nw {#4} #1 \q_mark #2 \q_mark #3 \q_stop }
\cs_new:Npn \__draw_point_process_auxvi:nw
  #1 #2 , #3 \q_mark #4 , #5 \q_mark #6 , #7 \q_stop
  { #1 {#2} {#3} {#4} {#5} {#6} {#7} }
\cs_new:Npn \__draw_point_process:nnnnn #1#2#3#4#5
  {
    \exp_args:Nffff \__draw_point_process_auxvii:nnnnn
      { \__draw_point_to_dim:n {#2} }
      { \__draw_point_to_dim:n {#3} }
      { \__draw_point_to_dim:n {#4} }
      { \__draw_point_to_dim:n {#5} }
      {#1}
  }
\cs_new:Npn \__draw_point_process_auxvii:nnnnn #1#2#3#4#5
  {
    \__draw_point_process_auxviii:nw
      {#5} #1 \q_mark #2 \q_mark #3 \q_mark #4 \q_stop
  }
\cs_new:Npn \__draw_point_process_auxviii:nw
  #1 #2 , #3 \q_mark #4 , #5 \q_mark #6 , #7 \q_mark #8 , #9 \q_stop
  { #1 {#2} {#3} {#4} {#5} {#6} {#7} {#8} {#9} }
```

(*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.

```
\cs_new:Npn \__draw_point_to_dim:n #1
  { \__draw_point_to_dim_aux:f { \fp_eval:n {#1} } }
```

```
\cs_new:Npn \__draw_point_to_dim_aux:n #1
  { \__draw_point_to_dim_aux:w #1 }
\cs_generate_variant:Nn \__draw_point_to_dim_aux:n { f }
\cs_new:Npn \__draw_point_to_dim_aux:w ( #1 , ~ #2 ) { #1pt , #2pt }
```

5.2 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.

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

5.3 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.*

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

```
\cs_new:Npn \draw_point_unit_vector:n #1
  { \__draw_point_process:nn { \__draw_point_unit_vector:nn } {#1} }
\cs_new:Npn \__draw_point_unit_vector:nn #1#2
  {
    \__draw_point_to_dim:n
      { ( #1 , #2 ) / (sqrt(#1 * #1 + #2 * #2)) }
  }
```

5.4 Intersection calculations

`\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`

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

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

and

$$P_y = \frac{(x_1y_2-y_1x_2)(y_3-y_5)-(x_3y_4-y_3x_4)(y_1-y_2)}{(x_1-x_2)(y_3-y_4)-(y_1-y_2)(x_3-x_4)}$$

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

```
\cs_new:Npn \draw_point_intersect_lines:nnnn #1#2#3#4
  {
    \__draw_point_process:nnnnn
      { \__draw_point_intersect_lines:nnnnnnnn }
      {#1} {#2} {#3} {#4}
  }
```

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

#1 x_1

#2 y_1

#3 x_2

#4 y_2

#5 x_3

#6 y_3

#7 x_4

#8 y_4

so now just have to do all of the calculation.

```
\cs_new:Npn \__draw_point_intersect_lines:nnnnnnnn #1#2#3#4#5#6#7#8
  {
    \__draw_point_intersect_lines_aux:ffffff
      { \fp_eval:n { #1 * #4 - #2 * #3 } }
      { \fp_eval:n { #5 * #8 - #6 * #7 } }
      { \fp_eval:n { #1 - #3 } }
      { \fp_eval:n { #5 - #7 } }
      { \fp_eval:n { #2 - #4 } }
      { \fp_eval:n { #6 - #8 } }
  }
\cs_new:Npn \__draw_point_intersect_lines_aux:nnnnnn #1#2#3#4#5#6
  {
    \__draw_point_to_dim:n
      {
        ( #2 * #3 - #1 * #4 , #2 * #5 - #1 * #6 )
          / ( #4 * #5 - #6 * #3 )
      }
  }
\cs_generate_variant:Nn \__draw_point_intersect_lines_aux:nnnnnn { ffffff }
```

\draw_point_intersect_circles:nnnnn
__draw_point_intersect_circles_auxi:nnnnnnn
__draw_point_intersect_circles_auxii:nnnnnnn
__draw_point_intersect_circles_auxii:ffnnnnn
_draw_point_intersect_circles_auxiii:nnnnnnn
_draw_point_intersect_circles_auxiii:ffnnnn
_draw_point_intersect_circles_auxiv:nnnnnnnn
_draw_point_intersect_circles_auxiv:fnnnnnnn
_draw_point_intersect_circles_auxv:nnnnnnnnn
_draw_point_intersect_circles_auxv:ffnnnnnnn
_draw_point_intersect_circles_auxvi:nnnnnnnn
_draw_point_intersect_circles_auxvi:fnnnnnnn
_draw_point_intersect_circles_auxvii:nnnnnnn
_draw_point_intersect_circles_auxvii:fffnnnn

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

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

in either

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

or

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

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

```
\cs_new:Npn \draw_point_intersect_circles:nnnnn #1#2#3#4#5
  {
    \__draw_point_process:nnn
      { \__draw_point_intersect_circles_auxi:nnnnnnn {#2} {#4} {#5} }
      {#1} {#3}
  }
\cs_new:Npn \__draw_point_intersect_circles_auxi:nnnnnnn #1#2#3#4#5#6#7
  {
    \__draw_point_intersect_circles_auxii:ffnnnnn
      { \fp_eval:n {#1} } { \fp_eval:n {#2} } {#4} {#5} {#6} {#7} {#3}
  }
```

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.

```
\cs_new:Npn \__draw_point_intersect_circles_auxii:nnnnnnn #1#2#3#4#5#6#7
  {
    \__draw_point_intersect_circles_auxiii:ffnnnnn
      { \fp_eval:n { #5 - #3 } }
      { \fp_eval:n { #6 - #4 } }
      {#1} {#2} {#3} {#4} {#7}
  }
\cs_generate_variant:Nn \__draw_point_intersect_circles_auxii:nnnnnnn { ff }
\cs_new:Npn \__draw_point_intersect_circles_auxiii:nnnnnnn #1#2#3#4#5#6#7
  {
    \__draw_point_intersect_circles_auxiv:fnnnnnnn
      { \fp_eval:n { sqrt( #1 * #1 + #2 * #2 ) } }
      {#1} {#2} {#3} {#4} {#5} {#6} {#7}
  }
\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.

```
\cs_new:Npn \__draw_point_intersect_circles_auxiv:nnnnnnnn #1#2#3#4#5#6#7#8
  {
    \__draw_point_intersect_circles_auxv:ffnnnnnnn
      { \fp_eval:n { 1 / #1 } }
      { \fp_eval:n { #4 * #4 } }
      {#1} {#2} {#3} {#5} {#6} {#7} {#8}
  }
\cs_generate_variant:Nn \__draw_point_intersect_circles_auxiv:nnnnnnnn { f }
\cs_new:Npn \__draw_point_intersect_circles_auxv:nnnnnnnnn #1#2#3#4#5#6#7#8#9
  {
    \__draw_point_intersect_circles_auxvi:fnnnnnnn
      { \fp_eval:n { 0.5 * #1 * ( #2 + #3 * #3 - #6 * #6 ) } }
      {#1} {#2} {#4} {#5} {#7} {#8} {#9}
  }
\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.

```
\cs_new:Npn \__draw_point_intersect_circles_auxvi:nnnnnnnn #1#2#3#4#5#6#7#8
  {
    \__draw_point_intersect_circles_auxvii:fffnnnn
      { \fp_eval:n { #1 * #2 } }
      { \int_if_odd:nTF {#8} { 1 } { -1 } }
      { \fp_eval:n { sqrt ( #3 - #1 * #1 ) * #2 } }
      {#4} {#5} {#6} {#7}
  }
\cs_generate_variant:Nn \__draw_point_intersect_circles_auxvi:nnnnnnnn { f }
\cs_new:Npn \__draw_point_intersect_circles_auxvii:nnnnnnn #1#2#3#4#5#6#7
  {
    \__draw_point_to_dim:n
      { #6 + #4 * #1 + #2 * #3 * #5 , #7 + #5 * #1 + -1 * #2 * #3 * #4 }
  }
\cs_generate_variant:Nn \__draw_point_intersect_circles_auxvii:nnnnnnn { fff }
```

5.5 Interpolation on a line (vector) or arc

\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

Simple maths after expansion.

```
\cs_new:Npn \draw_point_interpolate_line:nnn #1#2#3
  {
    \__draw_point_process:nnn
      { \__draw_point_interpolate_line_aux:fnnnn { \fp_eval:n {#1} } }
      {#2} {#3}
  }
\cs_new:Npn \__draw_point_interpolate_line_aux:nnnnn #1#2#3#4#5
  {
    \__draw_point_interpolate_line_aux:fnnnnn { \fp_eval:n { 1 - #1 } }
      {#1} {#2} {#3} {#4} {#5}
  }
\cs_generate_variant:Nn \__draw_point_interpolate_line_aux:nnnnn { f }
\cs_new:Npn \__draw_point_interpolate_line_aux:nnnnnn #1#2#3#4#5#6
  { \__draw_point_to_dim:n { #2 * #3 + #1 * #5 , #2 * #4 + #1 * #6 } }
\cs_generate_variant:Nn \__draw_point_interpolate_line_aux:nnnnnn { f }
```

\draw_point_interpolate_distance:nnn
__draw_point_interpolate_distance:nnnnn
__draw_point_interpolate_distance:nnnnnn
__draw_point_interpolate_distance:fnnnn

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.

```
\cs_new:Npn \draw_point_interpolate_distance:nnn #1#2#3
  {
    \__draw_point_process:nn
      { \__draw_point_interpolate_distance:nnnn {#1} {#3} }
      {#2}
  }
\cs_new:Npn \__draw_point_interpolate_distance:nnnn #1#2#3#4
  {
    \__draw_point_process:nn
      {
        \__draw_point_interpolate_distance:fnnnn
          { \fp_eval:n {#1} } {#3} {#4}
      }
      { \draw_point_unit_vector:n { ( #2 ) - ( #3 , #4 ) } }
  }
\cs_new:Npn \__draw_point_interpolate_distance:nnnnn #1#2#3#4#5
  { \__draw_point_to_dim:n { #2 + #1 * #4 , #3 + #1 * #5 } }
\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
raw_point_interpolate_arcaxes_auxi:nnnnnnnnn
aw_point_interpolate_arcaxes_auxii:nnnnnnnnn
aw_point_interpolate_arcaxes_auxii:fnnnnnnnn
raw_point_interpolate_arcaxes_auxiii:nnnnnnn
raw_point_interpolate_arcaxes_auxiii:fnnnnnn
raw_point_interpolate_arcaxes_auxiv:nnnnnnnn
raw_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.

```
\cs_new:Npn \draw_point_interpolate_arcaxes:nnnnnn #1#2#3#4#5#6
  {
    \__draw_point_process:nnnn
```

```
      { \__draw_point_interpolate_arcaxes_auxi:nnnnnnnnn {#1} {#5} {#6} }
      {#2} {#3} {#4}
  }
\cs_new:Npn \__draw_point_interpolate_arcaxes_auxi:nnnnnnnnn #1#2#3#4#5#6#7#8#9
  {
    \__draw_point_interpolate_arcaxes_auxii:fnnnnnnnn
      { \fp_eval:n {#1} } {#2} {#3} {#4} {#5} {#6} {#7} {#8} {#9}
  }
```

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.

```
\cs_new:Npn \__draw_point_interpolate_arcaxes_auxii:nnnnnnnnn #1#2#3#4#5#6#7#8#9
  {
    \__draw_point_interpolate_arcaxes_auxiii:fnnnnnn
      { \fp_eval:n { #1 * (#3) + ( 1 - #1 ) * (#2) } }
      {#4} {#5} {#6} {#7} {#8} {#9}
  }
\cs_generate_variant:Nn \__draw_point_interpolate_arcaxes_auxii:nnnnnnnnn { f }
\cs_new:Npn \__draw_point_interpolate_arcaxes_auxiii:nnnnnnn #1#2#3#4#5#6#7
  {
    \__draw_point_interpolate_arcaxes_auxiv:ffnnnnnn
      { \fp_eval:n { cosd (#1) } }
      { \fp_eval:n { sind (#1) } }
      {#2} {#3} {#4} {#5} {#6} {#7}
  }
\cs_generate_variant:Nn \__draw_point_interpolate_arcaxes_auxiii:nnnnnnn { f }
\cs_new:Npn \__draw_point_interpolate_arcaxes_auxiv:nnnnnnnn #1#2#3#4#5#6#7#8
  {
    \__draw_point_to_dim:n
      { #3 + #1 * #5 + #2 * #7 , #4 + #1 * #6 + #2 * #8 }
  }
\cs_generate_variant:Nn \__draw_point_interpolate_arcaxes_auxiv:nnnnnnnn { ff }
```

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

\draw_point_interpolate_curve:nnnnn
\draw_point_interpolate_curve_auxi:nnnnnnnnn
draw_point_interpolate_curve_auxii:nnnnnnnnn
draw_point_interpolate_curve_auxii:fnnnnnnnn
\draw_point_interpolate_curve_auxiii:nnnnnn
\draw_point_interpolate_curve_auxiii:fnnnnn
\draw_point_interpolate_curve_auxiv:nnnnnn
\draw_point_interpolate_curve_auxv:nnw
\draw_point_interpolate_curve_auxv:ffw
\draw_point_interpolate_curve_auxvi:n
draw_point_interpolate_curve_auxvii:nnnnnnnn

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

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

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

```
\cs_new:Npn \draw_point_interpolate_curve:nnnnnn #1#2#3#4#5
  {
    \__draw_point_process:nnnnn
      { \__draw_point_interpolate_curve_auxi:nnnnnnnnn {#1} }
      {#2} {#3} {#4} {#5}
  }
\cs_new:Npn \__draw_point_interpolate_curve_auxi:nnnnnnnnn #1#2#3#4#5#6#7#8#9
  {
    \__draw_point_interpolate_curve_auxii:fnnnnnnnn
      { \fp_eval:n {#1} }
      {#2} {#3} {#4} {#5} {#6} {#7} {#8} {#9}
  }
```

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!

```
\cs_new:Npn \__draw_point_interpolate_curve_auxii:nnnnnnnnn
  #1#2#3#4#5#6#7#8#9
  {
    \__draw_point_interpolate_curve_auxiii:fnnnnn
      { \fp_eval:n { 1 - #1 } }
      {#1}
      { {#2} {#3} } { {#4} {#5} } { {#6} {#7} } { {#8} {#9} }
  }
\cs_generate_variant:Nn \__draw_point_interpolate_curve_auxii:nnnnnnnnn { f }
%    \begin{macrocode}
%   We need to do the first cycle, but haven't got enough arguments to keep
%   everything in play at once. So her ewe use a but of argument re-ordering
%   and a single auxiliary to get the job done.
%    \begin{macrocode}
\cs_new:Npn \__draw_point_interpolate_curve_auxiii:nnnnnn #1#2#3#4#5#6
  {
    \__draw_point_interpolate_curve_auxiv:nnnnnn {#1} {#2} #3 #4
    \__draw_point_interpolate_curve_auxiv:nnnnnn {#1} {#2} #4 #5
    \__draw_point_interpolate_curve_auxiv:nnnnnn {#1} {#2} #5 #6
    \prg_do_nothing:
    \__draw_point_interpolate_curve_auxvi:n { {#1} {#2} }
  }
\cs_generate_variant:Nn \__draw_point_interpolate_curve_auxiii:nnnnnn { f }
\cs_new:Npn \__draw_point_interpolate_curve_auxiv:nnnnnn #1#2#3#4#5#6
  {
    \__draw_point_interpolate_curve_auxv:ffw
      { \fp_eval:n { #1 * #3 + #2 * #5 } }
      { \fp_eval:n { #1 * #4 + #2 * #6 } }
  }
\cs_new:Npn \__draw_point_interpolate_curve_auxv:nnw
  #1#2#3 \prg_do_nothing: #4#5
  {
    #3
    \prg_do_nothing:
    #4 { #5 {#1} {#2} }
  }
\cs_generate_variant:Nn \__draw_point_interpolate_curve_auxv:nnw { ff }
%    \begin{macrocode}
%   Get the arguments back into the right places and to the second and
%   third cycles directly.
%    \begin{macrocode}
\cs_new:Npn \__draw_point_interpolate_curve_auxvi:n #1
  { \__draw_point_interpolate_curve_auxvii:nnnnnnnn #1 }
\cs_new:Npn \__draw_point_interpolate_curve_auxvii:nnnnnnnn #1#2#3#4#5#6#7#8
  {
    \__draw_point_interpolate_curve_auxviii:ffffnn
      { \fp_eval:n { #1 * #5 + #2 * #3 } }
      { \fp_eval:n { #1 * #6 + #2 * #4 } }
      { \fp_eval:n { #1 * #7 + #2 * #5 } }
      { \fp_eval:n { #1 * #8 + #2 * #6 } }
      {#1} {#2}
  }
\cs_new:Npn \__draw_point_interpolate_curve_auxviii:nnnnnn #1#2#3#4#5#6
  {
    \__draw_point_to_dim:n
      { #5 * #3 + #6 * #1 , #5 * #4 + #6 * #2 }
  }
\cs_generate_variant:Nn \__draw_point_interpolate_curve_auxviii:nnnnnn { ffff }
```

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

5.6 Vector support

As well as co-ordinates relative to the drawing

\l__draw_xvec_x_dim
\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.

```
\dim_new:N \l__draw_xvec_x_dim
\dim_new:N \l__draw_xvec_y_dim
\dim_new:N \l__draw_yvec_x_dim
\dim_new:N \l__draw_yvec_y_dim
\dim_new:N \l__draw_zvec_x_dim
\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.

```
\cs_new_protected:Npn \draw_xvec:n #1
  { \__draw_vec:nn { x } {#1} }
\cs_new_protected:Npn \draw_yvec:n #1
  { \__draw_vec:nn { y } {#1} }
\cs_new_protected:Npn \draw_zvec:n #1
  { \__draw_vec:nn { z } {#1} }
\cs_new_protected:Npn \__draw_vec:nn #1#2
  {
    \__draw_point_process:nn { \__draw_vec:nnn {#1} } {#2}
  }
\cs_new_protected:Npn \__draw_vec:nnn #1#2#3
  {
    \dim_set:cn { l__draw_ #1 vec_x_dim } {#2}
    \dim_set:cn { l__draw_ #1 vec_y_dim } {#3}
  }
```

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

Initialise the vectors.

```
\draw_xvec:n { 1cm , 0cm }
\draw_yvec:n { 0cm , 1cm }
\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.

```
\cs_new:Npn \draw_point_vec:nn #1#2
  { \__draw_point_vec:ff { \fp_eval:n {#1} } { \fp_eval:n {#2} } }
\cs_new:Npn \__draw_point_vec:nn #1#2
  {
    \__draw_point_to_dim:n
      {
        #1 * \l__draw_xvec_x_dim + #2 * \l__draw_yvec_x_dim ,
        #1 * \l__draw_xvec_y_dim + #2 * \l__draw_yvec_y_dim
      }
  }
\cs_generate_variant:Nn \__draw_point_vec:nn { ff }
\cs_new:Npn \draw_point_vec:nnn #1#2#3
  {
    \__draw_point_vec:fff
      { \fp_eval:n {#1} } { \fp_eval:n {#2} } { \fp_eval:n {#3} }
  }
```

```
\cs_new:Npn \__draw_point_vec:nnn #1#2#3
  {
    \__draw_point_to_dim:n
      {
              #1 * \l__draw_xvec_x_dim
            + #2 * \l__draw_yvec_x_dim
            + #3 * \l__draw_zvec_x_dim
        ,
             #1 * \l__draw_xvec_y_dim
           + #2 * \l__draw_yvec_y_dim
           + #3 * \l__draw_zvec_y_dim
    }
  }
\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.

```
\cs_new:Npn \draw_point_vec_polar:nn #1#2
  { \draw_point_vec_polar:nnn {#1} {#2} {#2} }
\cs_new:Npn \draw_point_vec_polar:nnn #1#2#3
  { \__draw_draw_vec_polar:fnn { \fp_eval:n {#1} } {#2} {#3} }
\cs_new:Npn \__draw_draw_vec_polar:nnn #1#2#3
  {
    \__draw_point_to_dim:n
      {
        cosd(#1) * (#2) * \l__draw_xvec_x_dim ,
        sind(#1) * (#3) * \l__draw_yvec_y_dim
     }
  }
\cs_generate_variant:Nn \__draw_draw_vec_polar:nnn { f }
```

(*End definition for* `\draw_point_vec_polar:nn`, `\draw_point_vec_polar:nnn`, *and* `\__draw_point_-vec_polar:nnn`. *These functions are documented on page* **??**.)

5.7 Transformations

`\draw_point_transform:n`
`\__draw_point_transform:nn`

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

```
\cs_new:Npn \draw_point_transform:n #1
  {
    \__draw_point_process:nn
      { \__draw_point_transform:nn } {#1}
  }
\cs_new:Npn \__draw_point_transform:nn #1#2
  {
    \bool_if:NTF \l__draw_matrix_active_bool
      {
        \__draw_point_to_dim:n
          {
            (
                \l__draw_matrix_a_fp * #1
              + \l__draw_matrix_c_fp * #2
              + \l__draw_xshift_dim
            )
            ,
            (
                \l__draw_matrix_b_fp * #1
              + \l__draw_matrix_d_fp * #2
              + \l__draw_yshift_dim
            )
        }
      }
      {
        \__draw_point_to_dim:n
          {
              (#1, #2)
            + ( \l__draw_xshift_dim , \l__draw_yshift_dim )
          }
      }
  }
```

(*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.

```
\cs_new:Npn \__draw_point_transform_noshift:n #1
  {
    \__draw_point_process:nn
      { \__draw_point_transform_noshift:nn } {#1}
  }
\cs_new:Npn \__draw_point_transform_noshift:nn #1#2
  {
    \bool_if:NTF \l__draw_matrix_active_bool
      {
        \__draw_point_to_dim:n
          {
            (
                \l__draw_matrix_a_fp * #1
              + \l__draw_matrix_c_fp * #2
            )
            ,
            (
                \l__draw_matrix_b_fp * #1
              + \l__draw_matrix_d_fp * #2
            )
        }
      }
      { \__draw_point_to_dim:n { (#1, #2) } }
  }
```

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

```
⟨/initex | package⟩
```

6 l3draw-scopes implementation

```
⟨*initex | package⟩
```

```
⟨@@=draw⟩
```

6.1 Drawing environment

\g__draw_xmax_dim
\g__draw_xmin_dim
\g__draw_ymax_dim
\g__draw_ymin_dim

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

```
\dim_new:N \g__draw_xmax_dim
\dim_new:N \g__draw_xmin_dim
\dim_new:N \g__draw_ymax_dim
\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.

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

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

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

```
\box_new:N \l__draw_main_box
\box_new:N \l__draw_layer_main_box
```

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

\g__draw_id_int The drawing number.

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

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

__draw_reset_bb: A simple auxiliary.

```
\cs_new_protected:Npn \__draw_reset_bb:
  {
    \dim_gset:Nn \g__draw_xmax_dim { -\c_max_dim }
    \dim_gset:Nn \g__draw_xmin_dim {  \c_max_dim }
    \dim_gset:Nn \g__draw_ymax_dim { -\c_max_dim }
    \dim_gset:Nn \g__draw_ymin_dim {  \c_max_dim }
  }
```

(*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`. To allow for layers, there is some box nesting: notice that we

```
\cs_new_protected:Npn \draw_begin:
  {
    \group_begin:
      \int_gincr:N \g__draw_id_int
      \hbox_set:Nw \l__draw_main_box
        \driver_draw_begin:
        \__draw_reset_bb:
        \__draw_path_reset_limits:
        \bool_set_true:N \l_draw_bb_update_bool
        \draw_transform_matrix_reset:
        \draw_transform_shift_reset:
        \__draw_softpath_clear:
        \draw_linewidth:n { \l_draw_default_linewidth_dim }
        \draw_color:n { . }
        \draw_nonzero_rule:
        \draw_cap_butt:
        \draw_join_miter:
        \draw_miterlimit:n { 10 }
        \draw_dash_pattern:nn { } { 0cm }
        \hbox_set:Nw \l__draw_layer_main_box
  }
\cs_new_protected:Npn \draw_end:
  {
          \exp_args:NNNV \hbox_set_end:
          \clist_set:Nn \l_draw_layers_clist \l_draw_layers_clist
          \__draw_layers_insert:
        \driver_draw_end:
      \hbox_set_end:
      \dim_compare:nNnT \g__draw_xmin_dim = \c_max_dim
        {
          \dim_gzero:N \g__draw_xmax_dim
          \dim_gzero:N \g__draw_xmin_dim
          \dim_gzero:N \g__draw_ymax_dim
          \dim_gzero:N \g__draw_ymin_dim
        }
      \hbox_set:Nn \l__draw_main_box
        {
          \skip_horizontal:n { -\g__draw_xmin_dim }
          \box_move_down:nn { \g__draw_ymin_dim }
            { \box_use_drop:N \l__draw_main_box }
        }
      \box_set_ht:Nn \l__draw_main_box
        { \g__draw_ymax_dim - \g__draw_ymin_dim }
      \box_set_dp:Nn \l__draw_main_box { 0pt }
      \box_set_wd:Nn \l__draw_main_box
        { \g__draw_xmax_dim - \g__draw_xmin_dim }
      \mode_leave_vertical:
      \box_use_drop:N \l__draw_main_box
    \group_end:
  }
```

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

6.2 Scopes

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

Storage for local variables.

```
\dim_new:N \l__draw_linewidth_dim
\tl_new:N \l__draw_fill_color_tl
\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.

```
\cs_new_protected:Npn \draw_scope_begin:
  {
    \driver_draw_scope_begin:
    \group_begin:
      \dim_set_eq:NN \l__draw_linewidth_dim \g__draw_linewidth_dim
      \draw_path_scope_begin:
  }
\cs_new_protected:Npn \draw_scope_end:
  {
      \draw_path_scope_end:
      \dim_gset_eq:NN \g__draw_linewidth_dim \l__draw_linewidth_dim
    \group_end:
    \driver_draw_scope_end:
  }
```

(*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.

```
\dim_new:N \l__draw_xmax_dim
\dim_new:N \l__draw_xmin_dim
\dim_new:N \l__draw_ymax_dim
\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.

```
\cs_new_protected:Npn \__draw_scope_bb_begin:
  {
    \group_begin:
      \dim_set_eq:NN \l__draw_xmax_dim \g__draw_xmax_dim
      \dim_set_eq:NN \l__draw_xmin_dim \g__draw_xmin_dim
      \dim_set_eq:NN \l__draw_ymax_dim \g__draw_ymax_dim
      \dim_set_eq:NN \l__draw_ymin_dim \g__draw_ymin_dim
      \__draw_reset_bb:
  }
\cs_new_protected:Npn \__draw_scope_bb_end:
  {
      \dim_gset_eq:NN \g__draw_xmax_dim \l__draw_xmax_dim
      \dim_gset_eq:NN \g__draw_xmin_dim \l__draw_xmin_dim
      \dim_gset_eq:NN \g__draw_ymax_dim \l__draw_ymax_dim
      \dim_gset_eq:NN \g__draw_ymin_dim \l__draw_ymin_dim
    \group_end:
  }
```

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

\draw_suspend_begin:
\draw_suspend_end:

Suspend all parts of a drawing.

```
\cs_new_protected:Npn \draw_suspend_begin:
  {
    \__draw_scope_bb_begin:
    \draw_path_scope_begin:
    \draw_transform_matrix_reset:
    \draw_transform_shift_reset:
```

```
    \__draw_layers_save:
  }
\cs_new_protected:Npn \draw_suspend_end:
  {
    \__draw_layers_restore:
    \draw_path_scope_end:
    \__draw_scope_bb_end:
  }
```

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

```
⟨/initex | package⟩
```

7 l3draw-softpath implementation

```
⟨*initex | package⟩
⟨@@=draw⟩
```

7.1 Managing soft paths

There are two linked aims in the code here. The most significant is to provide a way to modify paths, for example to shorten the ends or round the corners. This means that the path cannot be written piecemeal as specials, but rather needs to be held in macros. The second aspect that follows from this is performance: simply adding to a single macro a piece at a time will have poor performance as the list gets long so we use `\tl_build_...` functions.

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

`\g__draw_softpath_main_tl`

The soft path itself.

```
\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.

```
\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.

```
\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`

```
\cs_new_protected:Npn \__draw_softpath_add:n
  { \tl_build_gput_right:Nn \g__draw_softpath_main_tl }
\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.

```
\cs_new_protected:Npn \__draw_softpath_use:
  {
    \tl_build_get:NN \g__draw_softpath_main_tl \l__draw_softpath_internal_tl
    \l__draw_softpath_internal_tl
  }
\cs_new_protected:Npn \__draw_softpath_clear:
  {
    \tl_build_gclear:N \g__draw_softpath_main_tl
    \bool_gset_false:N \g__draw_softpath_corners_bool
  }
\cs_new_protected:Npn \__draw_softpath_use_clear:
  {
    \__draw_softpath_use:
    \__draw_softpath_clear:
  }
```

(*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).

```
\dim_new:N \g__draw_softpath_lastx_dim
\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.

```
\bool_new:N \g__draw_softpath_move_bool
\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.

```
\cs_new_protected:Npn \__draw_softpath_closepath:
  {
    \__draw_softpath_add:x
      {
        \__draw_softpath_close_op:nn
          { \dim_use:N \g__draw_softpath_lastx_dim }
          { \dim_use:N \g__draw_softpath_lasty_dim }
      }
  }
\cs_new_protected:Npn \__draw_softpath_curveto:nnnnnn #1#2#3#4#5#6
  {
    \__draw_softpath_add:n
      {
        \__draw_softpath_curveto_opi:nn {#1} {#2}
        \__draw_softpath_curveto_opii:nn {#3} {#4}
        \__draw_softpath_curveto_opiii:nn {#5} {#6}
      }
  }
\cs_new_protected:Npn \__draw_softpath_lineto:nn #1#2
  {
    \__draw_softpath_add:n
      { \__draw_softpath_lineto_op:nn {#1} {#2} }
  }
\cs_new_protected:Npn \__draw_softpath_moveto:nn #1#2
  {
    \__draw_softpath_add:n
      { \__draw_softpath_moveto_op:nn {#1} {#2} }
    \bool_if:NT \g__draw_softpath_move_bool
      {
        \dim_gset:Nn \g__draw_softpath_lastx_dim {#1}
        \dim_gset:Nn \g__draw_softpath_lasty_dim {#2}
      }
  }
\cs_new_protected:Npn \__draw_softpath_rectangle:nnnn #1#2#3#4
  {
    \__draw_softpath_add:n
      {
        \__draw_softpath_rectangle_opi:nn {#1} {#2}
        \__draw_softpath_rectangle_opii:nn {#3} {#4}
      }
  }
\cs_new_protected:Npn \__draw_softpath_roundpoint:nn #1#2
  {
    \__draw_softpath_add:n
      { \__draw_softpath_roundpoint_op:nn {#1} {#2} }
    \bool_gset_true:N \g__draw_softpath_corners_bool
  }
\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.

```
\cs_new_protected:Npn \__draw_softpath_close_op:nn #1#2
  { \driver_draw_closepath: }
\cs_new_protected:Npn \__draw_softpath_curveto_opi:nn #1#2
  { \__draw_softpath_curveto_opi:nnNnnNnn {#1} {#2} }
\cs_new_protected:Npn \__draw_softpath_curveto_opi:nnNnnNnn #1#2#3#4#5#6#7#8
  { \driver_draw_curveto:nnnnnn {#1} {#2} {#4} {#5} {#7} {#8} }
\cs_new_protected:Npn \__draw_softpath_curveto_opii:nn #1#2
  { \__draw_softpath_curveto_opii:nn }
\cs_new_protected:Npn \__draw_softpath_curveto_opiii:nn #1#2
  { \__draw_softpath_curveto_opiii:nn }
\cs_new_protected:Npn \__draw_softpath_lineto_op:nn #1#2
  { \driver_draw_lineto:nn {#1} {#2} }
\cs_new_protected:Npn \__draw_softpath_moveto_op:nn #1#2
  { \driver_draw_moveto:nn {#1} {#2} }
\cs_new_protected:Npn \__draw_softpath_roundpoint_op:nn #1#2 { }
\cs_new_protected:Npn \__draw_softpath_rectangle_opi:nn #1#2
  { \__draw_softpath_rectangle_opi:nnNnn {#1} {#2} }
\cs_new_protected:Npn \__draw_softpath_rectangle_opi:nnNnn #1#2#3#4#5
  { \driver_draw_rectangle:nnnn {#1} {#2} {#4} {#5} }
  \cs_new_protected:Npn \__draw_softpath_rectangle_opii:nn #1#2 { }
```

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

7.2 Rounding soft path corners

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

\l__draw_softpath_main_tl

For constructing the updated path.

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

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

\l__draw_softpath_part_tl

Data structures.

```
\tl_new:N \l__draw_softpath_part_tl
\tl_new:N \l__draw_softpath_curve_end_tl
```

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

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

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

```
\fp_new:N \l__draw_softpath_lastx_fp
\fp_new:N \l__draw_softpath_lasty_fp
\dim_new:N \l__draw_softpath_corneri_dim
\dim_new:N \l__draw_softpath_cornerii_dim
\tl_new:N \l__draw_softpath_first_tl
\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.

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

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

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

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

```
\cs_new_protected:Npn \__draw_softpath_round_corners:
  {
    \bool_if:NT \g__draw_softpath_corners_bool
      {
        \group_begin:
          \tl_clear:N \l__draw_softpath_main_tl
          \tl_clear:N \l__draw_softpath_part_tl
          \fp_zero:N \l__draw_softpath_lastx_fp
          \fp_zero:N \l__draw_softpath_lasty_fp
          \tl_clear:N \l__draw_softpath_first_tl
          \tl_clear:N \l__draw_softpath_move_tl
          \tl_build_get:NN \g__draw_softpath_main_tl \l__draw_softpath_internal_tl
          \exp_after:wN \__draw_softpath_round_loop:Nnn
            \l__draw_softpath_internal_tl
            \q_recursion_tail ? ?
            \q_recursion_stop
        \group_end:
      }
    \bool_gset_false:N \g__draw_softpath_corners_bool
  }
```

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`.

```
\cs_new_protected:Npn \__draw_softpath_round_loop:Nnn #1#2#3
  {
    \quark_if_recursion_tail_stop_do:Nn #1 { \__draw_softpath_round_end: }
    \token_if_eq_meaning:NNTF #1 \__draw_softpath_roundpoint_op:nn
      { \__draw_softpath_round_action:nn {#2} {#3} }
      {
        \tl_if_empty:NT \l__draw_softpath_first_tl
          { \tl_set:Nn \l__draw_softpath_first_tl { {#2} {#3} } }
        \fp_set:Nn \l__draw_softpath_lastx_fp {#2}
        \fp_set:Nn \l__draw_softpath_lasty_fp {#3}
        \token_if_eq_meaning:NNTF #1 \__draw_softpath_moveto_op:nn
          {
            \tl_put_right:No \l__draw_softpath_main_tl
              \l__draw_softpath_move_tl
            \tl_put_right:No \l__draw_softpath_main_tl
              \l__draw_softpath_part_tl
            \tl_set:Nn \l__draw_softpath_move_tl { #1 {#2} {#3} }
            \tl_clear:N \l__draw_softpath_first_tl
            \tl_clear:N \l__draw_softpath_part_tl
          }
          { \tl_put_right:Nn \l__draw_softpath_part_tl { #1 {#2} {#3} } }
        \__draw_softpath_round_loop:Nnn
      }
  }
\cs_new_protected:Npn \__draw_softpath_round_action:nn #1#2
  {
    \dim_set:Nn \l__draw_softpath_corneri_dim {#1}
    \dim_set:Nn \l__draw_softpath_cornerii_dim {#2}
    \bool_lazy_and:nnTF
      { \dim_compare_p:nNn \l__draw_softpath_corneri_dim = { 0pt } }
      { \dim_compare_p:nNn \l__draw_softpath_cornerii_dim = { 0pt } }
      { \__draw_softpath_round_loop:Nnn }
      { \__draw_softpath_round_action:Nnn }
  }
```

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.

```
\cs_new_protected:Npn \__draw_softpath_round_action:Nnn #1#2#3
  {
    \tl_if_empty:NT \l__draw_softpath_first_tl
      { \tl_set:Nn \l__draw_softpath_first_tl { {#2} {#3} } }
    \token_if_eq_meaning:NNTF #1 \__draw_softpath_curveto_opi:nn
      { \__draw_softpath_round_action_curveto:NnnNnn }
      {
        \token_if_eq_meaning:NNTF #1 \__draw_softpath_close_op:nn
          { \__draw_softpath_round_action_close: }
          {
            \token_if_eq_meaning:NNTF #1 \__draw_softpath_lineto_op:nn
            { \__draw_softpath_round_lookahead:NnnNnn }
            { \__draw_softpath_round_loop:Nnn }
        }
    }
    #1 {#2} {#3}
}
```

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.

```
\cs_new_protected:Npn \__draw_softpath_round_action_curveto:NnnNnn
  #1#2#3#4#5#6
  {
    \tl_put_right:Nn \l__draw_softpath_part_tl
      { #1 {#2} {#3} #4 {#5} {#6} }
    \fp_set:Nn \l__draw_softpath_lastx_fp {#5}
    \fp_set:Nn \l__draw_softpath_lasty_fp {#6}
    \__draw_softpath_round_lookahead:NnnNnn
  }
\cs_new_protected:Npn \__draw_softpath_round_action_close:
  {
    \bool_lazy_and:nnTF
      { ! \tl_if_empty_p:N \l__draw_softpath_first_tl }
      { ! \tl_if_empty_p:N \l__draw_softpath_move_tl }
      {
        \exp_after:wN \__draw_softpath_round_close:nn
          \l__draw_softpath_first_tl
      }
      { \__draw_softpath_round_loop:Nnn }
  }
```

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.

```
\cs_new_protected:Npn \__draw_softpath_round_lookahead:NnnNnn #1#2#3#4#5#6
  {
    \bool_lazy_any:nTF
      {
        { \token_if_eq_meaning_p:NN #4 \__draw_softpath_lineto_op:nn }
        { \token_if_eq_meaning_p:NN #4 \__draw_softpath_curveto_opi:nn }
        { \token_if_eq_meaning_p:NN #4 \__draw_softpath_close_op:nn }
      }
      {
        \__draw_softpath_round_calc:nnnNnn
          \__draw_softpath_round_loop:Nnn {#5} {#6}
      }
      {
        \token_if_eq_meaning:NNTF #4 \__draw_softpath_roundpoint_op:nn
          { \__draw_softpath_round_roundpoint:NnnNnnNnn }
          { \__draw_softpath_round_loop:Nnn }
      }
    #1 {#2} {#3}
    #4 {#5} {#6}
  }
\cs_new_protected:Npn \__draw_softpath_round_roundpoint:NnnNnnNnn
  #1#2#3#4#5#6#7#8#9
  {
    \__draw_softpath_round_calc:nnnNnn
      \__draw_softpath_round_loop:Nnn
      {#8} {#9} #1 {#2} {#3}
    #4 {#5} {#6} #7 {#8} {#9}
  }
```

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.

```
\cs_new_protected:Npn \__draw_softpath_round_calc:nnnNnn #1#2#3#4#5#6
  {
    \tl_set:Nx \l__draw_softpath_curve_end_tl
      {
        \draw_point_interpolate_distance:nnn
          \l__draw_softpath_cornerii_dim
          { #5 , #6 } { #2 , #3 }
      }
    \tl_put_right:Nx \l__draw_softpath_part_tl
      {
        \exp_not:N #4
        \__draw_softpath_round_calc:fVnnnn
          {
            \draw_point_interpolate_distance:nnn
              \l__draw_softpath_corneri_dim
              { #5 , #6 }
              {
                \l__draw_softpath_lastx_fp ,
                \l__draw_softpath_lasty_fp
              }
          }
          \l__draw_softpath_curve_end_tl
          {#5} {#6} {#2} {#3}
      }
    \fp_set:Nn \l__draw_softpath_lastx_fp {#5}
    \fp_set:Nn \l__draw_softpath_lasty_fp {#6}
    #1
  }
```

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).

```
\cs_new:Npn \__draw_softpath_round_calc:nnnnnn #1#2#3#4#5#6
  {
    \__draw_softpath_round_calc:nnnnw {#3} {#4} {#5} {#6}
      #1 \q_mark #2 \q_stop
  }
\cs_generate_variant:Nn \__draw_softpath_round_calc:nnnnnn { fV }
```

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

```
\cs_new:Npn \__draw_softpath_round_calc:nnnnw
  #1#2#3#4 #5 , #6 \q_mark #7 , #8 \q_stop
  {
    {#5} {#6}
    \exp_not:N \__draw_softpath_curveto_opi:nn
      {
        \fp_to_dim:n
          { #5 + \c__draw_softpath_arc_fp * ( #1 - #5 ) }
      }
      {
        \fp_to_dim:n
          { #6 + \c__draw_softpath_arc_fp * ( #2 - #6 ) }
      }
    \exp_not:N \__draw_softpath_curveto_opii:nn
      {
        \fp_to_dim:n
          { #7 + \c__draw_softpath_arc_fp * ( #1 - #7 ) }
      }
      {
        \fp_to_dim:n
          { #8 + \c__draw_softpath_arc_fp* ( #2 - #8 ) }
      }
    \exp_not:N \__draw_softpath_curveto_opiii:nn
      {#7} {#8}
  }
```

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.

```
\cs_new_protected:Npn \__draw_softpath_round_close:nn #1#2
  {
    \use:x
      {
        \__draw_softpath_round_calc:nnnNnn
          {
            \tl_set:Nx \exp_not:N \l__draw_softpath_move_tl
              {
                \__draw_softpath_moveto_op:nn
                \exp_not:N \exp_after:wN
                  \exp_not:N \__draw_softpath_round_close:w
                  \exp_not:N \l__draw_softpath_curve_end_tl
                    \exp_not:N \q_stop
              }
            \use:x
              {
                \exp_not:N \exp_not:N \exp_not:N \use_i:nnnn
                  {
                    \__draw_softpath_round_loop:Nnn
                      \__draw_softpath_close_op:nn
                      \exp_not:N \exp_after:wN
                        \exp_not:N \__draw_softpath_round_close:w
                        \exp_not:N \l__draw_softpath_curve_end_tl
                          \exp_not:N \q_stop
                  }
              }
          }
          {#1} {#2}
          \__draw_softpath_lineto_op:nn
          \exp_after:wN \use_none:n \l__draw_softpath_move_tl
      }
  }
\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.

```
\cs_new_protected:Npn \__draw_softpath_round_end:
  {
    \tl_put_right:No \l__draw_softpath_main_tl
      \l__draw_softpath_move_tl
    \tl_put_right:No \l__draw_softpath_main_tl
      \l__draw_softpath_part_tl
    \tl_build_gclear:N \g__draw_softpath_main_tl
    \__draw_softpath_add:o \l__draw_softpath_main_tl
  }
```

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

```
⟨/initex | package⟩
```

8 l3draw-state implementation

```
⟨*initex | package⟩
⟨@@=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`, `\pgfsetinnerstrok`
  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.

```
\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.

```
\dim_new:N \l_draw_default_linewidth_dim
\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.

```
\cs_new_protected:Npn \draw_linewidth:n #1
  {
    \dim_gset:Nn \g__draw_linewidth_dim { \fp_to_dim:n {#1} }
    \driver_draw_linewidth:n \g__draw_linewidth_dim
  }
```

(*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.

```
\cs_new_protected:Npn \draw_dash_pattern:nn #1#2
  {
    \group_begin:
      \seq_set_from_clist:Nn \l__draw_tmp_seq {#1}
      \seq_set_map:NNn \l__draw_tmp_seq \l__draw_tmp_seq
        { \fp_to_dim:n {##1} }
      \use:x
        {
          \driver_draw_dash_pattern:nn
            { \seq_use:Nn \l__draw_tmp_seq { , } }
            { \fp_to_dim:n {#2} }
        }
    \group_end:
  }
\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.

```
\cs_new_protected:Npn \draw_miterlimit:n #1
  { \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.

```
\cs_new_protected:Npn \draw_cap_butt: { \driver_draw_cap_butt: }
\cs_new_protected:Npn \draw_cap_rectangle: { \driver_draw_cap_rectangle: }
\cs_new_protected:Npn \draw_cap_round: { \driver_draw_cap_round: }
\cs_new_protected:Npn \draw_evenodd_rule: { \driver_draw_evenodd_rule: }
\cs_new_protected:Npn \draw_nonzero_rule: { \driver_draw_nonzero_rule: }
\cs_new_protected:Npn \draw_join_bevel: { \driver_draw_join_bevel: }
\cs_new_protected:Npn \draw_join_miter: { \driver_draw_join_miter: }
\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.

```
\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.

```
\cs_new_eq:NN \draw_color:n \color_select:n
\cs_new_protected:Npn \draw_color_fill:n #1
  { \__draw_color:nn { fill } {#1} }
\cs_new_protected:Npn \draw_color_stroke:n #1
  { \__draw_color:nn { stroke } {#1} }
\cs_new_protected:Npn \__draw_color:nn #1#2
  {
    \color_parse:nN {#2} \l__draw_color_tmp_tl
    \__draw_color_aux:Vn \l__draw_color_tmp_tl {#1}
  }
\cs_new_protected:Npn \__draw_color_aux:nn #1#2
```

```
  { \__draw_color:nw {#2} #1 \q_stop }
\cs_generate_variant:Nn \__draw_color_aux:nn { V }
\cs_new_protected:Npn \__draw_color:nw #1#2 ~ #3 \q_stop
  { \use:c { __draw_color_ #2 :nw } {#1} #3 \q_stop }
\cs_new_protected:Npn \__draw_color_cmyk:nw #1#2 ~ #3 ~ #4 ~ #5 \q_stop
  { \use:c { driver_draw_color_ #1 _cmyk:nnnn } {#2} {#3} {#4} {#5} }
\cs_new_protected:Npn \__draw_color_gray:nw #1#2 \q_stop
  { \use:c { driver_draw_color_ #1 _gray:n } {#2} }
\cs_new_protected:Npn \__draw_color_rgb:nw #1#2 ~ #3 ~ #4 \q_stop
  { \use:c { driver_draw_color_ #1 _rgb:nnn } {#2} {#3} {#4} }
\cs_new_protected:Npn \__draw_color_spot:nw #1#2 ~ #3 \q_stop
  { \use:c { driver_draw_color_ #1 _spot:nn } {#2} {#3} }
```

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

```
⟨/initex | package⟩
```

9 l3draw-transforms implementation

```
⟨*initex | package⟩
⟨@@=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.

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

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

`\l__draw_matrix_a_fp`
`\l__draw_matrix_b_fp`
`\l__draw_matrix_c_fp`
`\l__draw_xshift_dim`
`\l__draw_yshift_dim`

The active matrix and shifts.

```
\fp_new:N \l__draw_matrix_a_fp
\fp_new:N \l__draw_matrix_b_fp
\fp_new:N \l__draw_matrix_c_fp
\fp_new:N \l__draw_matrix_d_fp
\dim_new:N \l__draw_xshift_dim
\dim_new:N \l__draw_yshift_dim
```

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

`\draw_transform_matrix_reset:`
`\draw_transform_shift_reset:`

Fast resetting.

```
\cs_new_protected:Npn \draw_transform_matrix_reset:
  {
    \fp_set:Nn \l__draw_matrix_a_fp { 1 }
    \fp_zero:N \l__draw_matrix_b_fp
    \fp_zero:N \l__draw_matrix_c_fp
    \fp_set:Nn \l__draw_matrix_d_fp { 1 }
  }
\cs_new_protected:Npn \draw_transform_shift_reset:
  {
    \dim_zero:N \l__draw_xshift_dim
    \dim_zero:N \l__draw_yshift_dim
  }
\draw_transform_matrix_reset:
\draw_transform_shift_reset:
```

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

`\draw_transform_matrix_absolute:nnnn`
`\draw_transform_shift_absolute:n`
`\__draw_transform_shift_absolute:nn`

Setting the transform matrix is straight-forward, with just a bit of expansion to sort out. With the mechanism active, the identity matrix is set.

```
\cs_new_protected:Npn \draw_transform_matrix_absolute:nnnn #1#2#3#4
  {
    \fp_set:Nn \l__draw_matrix_a_fp {#1}
    \fp_set:Nn \l__draw_matrix_b_fp {#2}
    \fp_set:Nn \l__draw_matrix_c_fp {#3}
    \fp_set:Nn \l__draw_matrix_d_fp {#4}
    \bool_lazy_all:nTF
      {
        { \fp_compare_p:nNn \l__draw_matrix_a_fp = \c_one_fp }
        { \fp_compare_p:nNn \l__draw_matrix_b_fp = \c_zero_fp }
        { \fp_compare_p:nNn \l__draw_matrix_c_fp = \c_zero_fp }
        { \fp_compare_p:nNn \l__draw_matrix_d_fp = \c_one_fp }
      }
      { \bool_set_false:N \l__draw_matrix_active_bool }
      { \bool_set_true:N \l__draw_matrix_active_bool }
  }
\cs_new_protected:Npn \draw_transform_shift_absolute:n #1
  {
    \__draw_point_process:nn
      { \__draw_transform_shift_absolute:nn } {#1}
  }
\cs_new_protected:Npn \__draw_transform_shift_absolute:nn #1#2
  {
    \dim_set:Nn \l__draw_xshift_dim {#1}
    \dim_set:Nn \l__draw_yshift_dim {#2}
  }
```

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

`\draw_transform_matrix:nnnn`
`\__draw_transform:nnnn`
`\draw_transform_shift:n`
`\__draw_transform_shift:nn`

Much the same story for adding to an existing matrix, with a bit of pre-expansion so that the calculation uses "frozen" values.

```
\cs_new_protected:Npn \draw_transform_matrix:nnnn #1#2#3#4
  {
    \use:x
      {
        \__draw_transform:nnnn
          { \fp_eval:n {#1} }
          { \fp_eval:n {#2} }
          { \fp_eval:n {#3} }
          { \fp_eval:n {#4} }
      }
  }
\cs_new_protected:Npn \__draw_transform:nnnn #1#2#3#4
  {
    \use:x
      {
        \draw_transform_matrix_absolute:nnnn
          { #1 * \l__draw_matrix_a_fp + #2 * \l__draw_matrix_c_fp }
          { #1 * \l__draw_matrix_b_fp + #2 * \l__draw_matrix_d_fp }
          { #3 * \l__draw_matrix_a_fp + #4 * \l__draw_matrix_c_fp }
          { #3 * \l__draw_matrix_b_fp + #4 * \l__draw_matrix_d_fp }
        }
  }
\cs_new_protected:Npn \draw_transform_shift:n #1
  {
    \__draw_point_process:nn
      { \__draw_transform_shift:nn } {#1}
  }
\cs_new_protected:Npn \__draw_transform_shift:nn #1#2
  {
    \dim_set:Nn \l__draw_xshift_dim { \l__draw_xshift_dim + #1 }
    \dim_set:Nn \l__draw_yshift_dim { \l__draw_yshift_dim + #2 }
  }
```

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

\draw_transform_matrix_invert:
__draw_transform_invert:n
__draw_transform_invert:f
\draw_transform_shift_invert:

Standard mathematics: calculate the inverse matrix and use that, then undo the shifts.

```
\cs_new_protected:Npn \draw_transform_matrix_invert:
  {
    \bool_if:NT \l__draw_matrix_active_bool
      {
        \__draw_transform_invert:f
          {
            \fp_eval:n
              {
                1 /
                  (
                      \l__draw_matrix_a_fp * \l__draw_matrix_d_fp
                    - \l__draw_matrix_b_fp * \l__draw_matrix_c_fp
                  )
              }
          }
      }
  }
\cs_new_protected:Npn \__draw_transform_invert:n #1
  {
    \fp_set:Nn \l__draw_matrix_a_fp
      { \l__draw_matrix_d_fp * #1 }
    \fp_set:Nn \l__draw_matrix_b_fp
      { -\l__draw_matrix_b_fp * #1 }
    \fp_set:Nn \l__draw_matrix_c_fp
      { -\l__draw_matrix_c_fp * #1 }
    \fp_set:Nn \l__draw_matrix_d_fp
      { \l__draw_matrix_a_fp * #1 }
  }
\cs_generate_variant:Nn \__draw_transform_invert:n { f }
\cs_new_protected:Npn \draw_transform_shift_invert:
  {
    \dim_set:Nn \l__draw_xshift_dim { -\l__draw_xshift_dim }
    \dim_set:Nn \l__draw_yshift_dim { -\l__draw_yshift_dim }
  }
```

(*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.

```
\cs_new_protected:Npn \draw_transform_triangle:nnn #1#2#3
  {
    \__draw_point_process:nnn
      {
        \__draw_point_process:nn
          { \__draw_tranform_triangle:nnnnnn }
          {#1}
      }
      {#2} {#3}
  }
\cs_new_protected:Npn \__draw_tranform_triangle:nnnnnn #1#2#3#4#5#6
  {
    \use:x
      {
        \draw_transform_matrix_absolute:nnnn
          { #3 - #1 }
          { #4 - #2 }
          { #5 - #1 }
          { #6 - #2 }
        \draw_transform_shift_absolute:n { #1 , #2 }
      }
  }
```

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

`\draw_transform_scale:n`
`\draw_transform_xscale:n`
`\draw_transform_yscale:n`
`\draw_transform_xshift:n`
`\draw_transform_yshift:n`
`\draw_transform_xslant:n`
`\draw_transform_yslant:n`

Lots of shortcuts.

```
\cs_new_protected:Npn \draw_transform_scale:n #1
  { \draw_transform_matrix:nnnn { #1 } { 0 } { 0 } { #1 } }
\cs_new_protected:Npn \draw_transform_xscale:n #1
  { \draw_transform_matrix:nnnn { #1 } { 0 } { 0 } { 1 } }
\cs_new_protected:Npn \draw_transform_yscale:n #1
  { \draw_transform_matrix:nnnn { 1 } { 0 } { 0 } { #1 } }
\cs_new_protected:Npn \draw_transform_xshift:n #1
  { \draw_transform_shift:n { #1 , 0pt } }
\cs_new_protected:Npn \draw_transform_yshift:n #1
  { \draw_transform_shift:n { 0pt , #1 } }
\cs_new_protected:Npn \draw_transform_xslant:n #1
  { \draw_transform_matrix:nnnn { 1 } { 0 } { #1 } { 1 } }
\cs_new_protected:Npn \draw_transform_yslant:n #1
  { \draw_transform_matrix:nnnn { 1 } { #1 } { 0 } { 1 } }
```

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

`\draw_transform_rotate:n`
`\__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.

```
\cs_new_protected:Npn \draw_transform_rotate:n #1
  { \__draw_transform_rotate:f { \fp_eval:n {#1} } }
\cs_new_protected:Npn \__draw_transform_rotate:n #1
  {
    \__draw_transform_rotate:ff
      { \fp_eval:n { cosd(#1) } }
      { \fp_eval:n { sind(#1) } }
  }
\cs_generate_variant:Nn \__draw_transform_rotate:n { f }
\cs_new_protected:Npn \__draw_transform_rotate:nn #1#2
  { \draw_transform_matrix:nnnn {#1} {#2} { -#2 } { #1 } }
\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* **??**.)

```
⟨/initex | package⟩
```

Index

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

B

\begin 169, 1013, 1017, 1041, 1044
bool commands:
- \bool_gset_eq:NN 758
- \bool_gset_false:N 1322, 1429
- \bool_gset_true:N 1332, 1378
- \bool_if:NTF 19, 113, 192, 231, 679, 682, 691, 692, 1136, 1168, 1360, 1413, 1775
- \bool_lazy_all:nTF 1721
- \bool_lazy_and:nnTF 223, 675, 1459, 1493
- \bool_lazy_any:nTF 1504
- \bool_lazy_or:nnTF 556, 651, 684
- \bool_new:N . 84, 218, 639, 640, 641, 642, 643, 736, 1192, 1310, 1331, 1694
- \bool_set_eq:NN 750
- \bool_set_false:N 94, 226, 662, 663, 664, 1728
- \bool_set_true:N 96, 227, 668, 699, 703, 704, 1211, 1729

box commands:
- \box_dp:N 15, 65
- \box_gset_eq:NN 154
- \box_gset_wd:Nn 98, 131
- \box_ht:N 15, 67
- \box_if_exist:NTF 91
- \box_move_down:nn 1241
- \box_move_up:nn 49
- \box_new:N 11, 78, 79, 1193, 1194
- \box_set_dp:Nn 53, 1246
- \box_set_eq:NN 143
- \box_set_ht:Nn 52, 1244
- \box_set_wd:Nn 54, 126, 1247
- \box_use_drop:N 50, 55, 100, 127, 132, 1242, 1250
- \box_wd:N 15, 64, 66

C

clist commands:
- \clist_map_inline:Nn . . 122, 139, 150
- \clist_map_inline:nn 665
- \clist_new:N 85, 87
- \clist_set:Nn 86, 1227

coffin commands:
- \coffin_typeset:Nnnnn 62
- \coffin_wd:N 64

color commands:
- \color_parse:nN 1675
- \color_select:n 1668

cs commands:
- \cs_generate_variant:Nn 418, 605, 638, 830, 838, 870, 889, 896, 904, 911, 920, 926, 938, 941, 959, 977, 985, 991, 1012, 1026, 1040, 1061, 1096, 1115, 1128, 1313, 1380, 1563, 1680, 1801, 1851, 1854
- \cs_if_exist:NTF 667
- \cs_if_exist_use:NTF . . 401, 410, 670
- \cs_new:Npn 509, 519, 529, 539, 775, 781, 783, 785, 792, 794, 796, 804, 806, 809, 818, 823, 826, 828, 831, 832, 834, 836, 839, 841, 846, 852, 862, 871, 877, 882, 890, 897, 905, 912, 921, 927, 933, 939, 942, 948, 957, 960, 966, 971, 978, 986, 992, 998, 1004, 1018, 1027, 1033, 1045, 1047, 1056, 1086, 1088, 1097, 1102, 1116, 1118, 1120, 1129, 1134, 1161, 1166, 1558, 1564, 1621
- \cs_new_eq:NN 1668
- \cs_new_protected:Npn . 12, 17, 58, 73, 88, 111, 120, 137, 148, 182, 204, 211, 219, 229, 238, 244, 250, 256, 263, 274, 282, 287, 289, 291, 300, 307, 343, 345, 356, 362, 392, 419, 448, 454, 460, 465, 473, 482, 487, 495, 550, 552, 565, 572, 581, 587, 589, 599, 606, 612, 619, 644, 649, 660, 697, 701, 706, 713, 737, 755, 1068, 1070, 1072, 1074, 1078, 1196, 1203, 1224, 1256, 1263, 1274, 1283, 1291, 1299, 1311, 1314, 1319, 1324, 1333, 1342, 1351, 1356, 1366, 1374, 1381, 1383, 1385, 1387, 1389, 1391, 1393, 1395, 1396, 1398, 1400, 1411, 1431, 1455, 1465, 1482, 1491, 1502, 1522, 1530, 1589, 1622, 1637, 1642, 1657, 1659, 1660, 1661, 1662, 1663, 1664, 1665, 1666, 1669, 1671, 1673, 1678, 1681, 1683, 1685, 1687, 1689, 1701, 1708, 1715, 1731, 1736, 1741, 1752, 1763, 1768, 1773, 1790, 1802, 1807, 1817, 1829, 1831, 1833, 1835, 1837, 1839, 1841, 1843, 1845, 1852

D

dim commands:
- \dim_abs:n 594, 595
- \dim_compare:nNnTF 601, 608, 715, 1231
- \dim_compare_p:nNn 224, 225, 1460, 1461
- \dim_eval:n 594, 595
- \dim_gset:Nn 184, 186, 188, 190, 194, 196, 198, 200, 206, 207, 208, 209, 213, 214, 717, 1198, 1199, 1200, 1201, 1362, 1363, 1639
- \dim_gset_eq:NN 762, 763, 764, 765, 766, 767, 768, 769, 1266, 1285, 1286, 1287, 1288
- \dim_gzero:N . . 1233, 1234, 1235, 1236
- \dim_max:nn 185, 189, 195, 199
- \dim_min:nn 187, 191, 197, 201
- \dim_new:N 176, 177, 178, 179, 180, 181, 216, 217, 728, 729, 730, 731, 732, 733, 734, 735, 1062, 1063, 1064, 1065, 1066, 1067, 1188, 1189, 1190, 1191, 1253, 1270, 1271, 1272, 1273, 1329, 1330, 1406, 1407, 1634, 1635, 1699, 1700
- \dim_set:Nn 221, 222, 1080, 1081, 1457, 1458, 1636, 1738, 1739, 1770, 1771, 1804, 1805
- \dim_set_eq:NN 740, 741, 742, 743, 744, 745, 746, 747, 1260, 1277, 1278, 1279, 1280
- \dim_step_inline:nnnn 621, 629
- \dim_use:N . . 715, 720, 722, 1338, 1339
- \dim_zero:N 1710, 1711
- \c_max_dim 206, 207, 208, 209, 715, 1198, 1199, 1200, 1201, 1231

draw commands:
- \l_draw_bb_update_bool 19, 192, 676, 1192, 1211
- \draw_begin: 1203
- \draw_box_use:N 12
- \draw_cap_butt: 1218, 1659
- \draw_cap_rectangle: 1659
- \draw_cap_round: 1659
- \draw_coffin_use:Nnn 58
- \draw_color:n 1216, 1668
- \draw_color_fill:n 1668
- \draw_color_stroke:n 1668
- \draw_dash_pattern:nn . . . 1221, 1642
- \l_draw_default_linewidth_dim 103, 1215, 1635
- \draw_end: 1203
- \draw_evenodd_rule: 1659
- \draw_join_bevel: 1659
- \draw_join_miter: 1219, 1659
- \draw_join_round: 1659
- \draw_layer_begin:n 88
- \draw_layer_end: 88
- \draw_layer_new:n 73
- \l_draw_layers_clist 85, 122, 139, 150, 1227
- \draw_linewidth:n 103, 1215, 1637
- \draw_miterlimit:n 1220, 1657
- \draw_nonzero_rule: 1217, 1659
- \draw_path_arc:nnn 343, 485
- \draw_path_arc:nnnn 343
- \draw_path_arc_axes:nnnn 482
- \draw_path_canvas_curveto:nnn . . 287
- \draw_path_canvas_lineto:n 287
- \draw_path_canvas_moveto:n 287
- \draw_path_circle:nn 550
- \draw_path_close: 282, 578
- \draw_path_corner_arc:nn 219
- \draw_path_curveto:nn 300
- \draw_path_curveto:nnn 238
- \draw_path_ellipse:nnn 487, 551
- \draw_path_grid:nnnn 589
- \draw_path_lineto:n 238, 575, 576, 577, 627, 635
- \draw_path_moveto:n 238, 574, 579, 626, 634
- \draw_path_rectangle:nn 552, 588
- \draw_path_rectangle_corners:nn 581
- \draw_path_scope_begin: 737, 1261, 1294
- \draw_path_scope_end: 737, 1265, 1302
- \draw_path_use:n 644
- \draw_path_use_clear:n 644
- \draw_point_interpolate_arcaxes:nnnnnn 960
- \draw_point_interpolate_curve:nnnnn 992
- \draw_point_interpolate_curve:nnnnnn 992
- \draw_point_interpolate_curve_-auxi:nnnnnnnnn 992
- \draw_point_interpolate_curve_-auxii:nnnnnnnnn 992
- \draw_point_interpolate_curve_-auxiii:nnnnnn 992
- \draw_point_interpolate_curve_-auxiv:nnnnnn 992
- \draw_point_interpolate_curve_-auxv:nnw 992
- \draw_point_interpolate_curve_-auxvi:n 992
- \draw_point_interpolate_curve_-auxvii:nnnnnnnn 992
- \draw_point_interpolate_curve_-auxviii:nnnnnn 992

\draw_point_interpolate_distance:nnn 942, 1534, 1543
\draw_point_interpolate_line:nnn 927
\draw_point_intersect_circles:nnnnn 871
\draw_point_intersect_lines:nnnn 846
\draw_point_polar:nn 832
\draw_point_polar:nnn 426, 432, 436, 442, 832
\draw_point_transform:n 23, 26, 29, 32, 242, 254, 270, 271, 272, 304, 305, 431, 435, 491, 562, 1129
\draw_point_unit_vector:n . . 839, 955
\draw_point_vec:nn 1086
\draw_point_vec:nnn 1086
\draw_point_vec_polar:nn 1116
\draw_point_vec_polar:nnn 1116
\draw_scope_begin: 1256
\draw_scope_end: 1263
\draw_suspend_begin: 1291
\draw_suspend_end: 1291
\draw_transform_matrix:nnnn 1741, 1830, 1832, 1834, 1840, 1842, 1853
\draw_transform_matrix_absolute:nnnn 1715, 1756, 1821
\draw_transform_matrix_invert: 1773
\draw_transform_matrix_reset: 1212, 1295, 1701
\draw_transform_rotate:n 1843
\draw_transform_scale:n 1829
\draw_transform_shift:n 1741, 1836, 1838
\draw_transform_shift_absolute:n 1715, 1826
\draw_transform_shift_invert: . 1773
\draw_transform_shift_reset: 1213, 1296, 1701
\draw_transform_triangle:nnn 484, 1807
\draw_transform_xscale:n 1829
\draw_transform_xshift:n 1829
\draw_transform_xslant:n 1829
\draw_transform_yscale:n 1829
\draw_transform_yshift:n 1829
\draw_transform_yslant:n 1829
\draw_xvec:n 1068, 1083
\draw_yvec:n 1068, 1084
\draw_zvec:n 1068, 1085
draw internal commands:
__draw_box_use:Nnnnn 12, 63
__draw_color:nn 1668
__draw_color:nw 1668
__draw_color_aux:nn 1668
__draw_color_cmyk:nw 1683
__draw_color_gray:nw 1685
__draw_color_rgb:nw 1687
__draw_color_spot:nw 1689
\l__draw_color_tmp_tl 1667, 1675, 1676
\l__draw_corner_arc_bool 218, 226, 227, 231, 557
\l__draw_corner_xarc_dim 216, 221, 224, 234
\l__draw_corner_yarc_dim 216, 222, 225, 235
__draw_draw_polar:nnn 832
__draw_draw_vec_polar:nnn 1119, 1120, 1128
\l__draw_fill_color_tl 1253
\g__draw_id_int 1195, 1206
\l__draw_layer_close_bool 84, 94, 96, 113
\l__draw_layer_main_box 126, 127, 1193, 1222
\l__draw_layer_tl 82, 93, 97
\g__draw_layers_clist 85
__draw_layers_insert: 120, 1228
__draw_layers_restore: . . . 137, 1301
__draw_layers_save: 137, 1297
\g__draw_linewidth_dim 723, 1260, 1266, 1634, 1639, 1640
\l__draw_linewidth_dim 1253, 1260, 1266
\l__draw_main_box 1193, 1207, 1238, 1242, 1244, 1246, 1247, 1250
\l__draw_matrix_a_fp 40, 1141, 1173, 1695, 1703, 1717, 1723, 1757, 1759, 1783, 1792, 1799
\l__draw_matrix_active_bool 558, 1136, 1168, 1694, 1728, 1729, 1775
\l__draw_matrix_b_fp 41, 1147, 1178, 1695, 1704, 1718, 1724, 1758, 1760, 1784, 1794, 1795
\l__draw_matrix_c_fp 42, 1142, 1174, 1695, 1705, 1719, 1725, 1757, 1759, 1784, 1796, 1797
\l__draw_matrix_d_fp 43, 1148, 1179, 1698, 1706, 1720, 1726, 1758, 1760, 1783, 1793, 1798
__draw_path_arc:nnnn 343
__draw_path_arc:nnNnn 343
\c__draw_path_arc_60_fp 343
\c__draw_path_arc_90_fp 343
__draw_path_arc_add:nnnn 343
__draw_path_arc_aux_add:nn 450, 456, 468, 473
__draw_path_arc_auxi:nnnnNnn 343, 370, 377
__draw_path_arc_auxii:nnnNnnnn 343

__draw_path_arc_auxiii:nn 343
__draw_path_arc_auxiv:nnnn . . . 343
__draw_path_arc_auxv:nn 343
__draw_path_arc_auxvi:nn 343
\l__draw_path_arc_delta_fp 343
\l__draw_path_arc_start_fp 343
__draw_path_curveto:nnnn 300
__draw_path_curveto:nnnnnn 238, 296, 314, 444, 511, 521, 531, 541
\c__draw_path_curveto_a_fp 300
\c__draw_path_curveto_b_fp 300
__draw_path_ellipse:nnnnnn . . . 487
__draw_path_ellipse_arci:nnnnnn 487
__draw_path_ellipse_arcii:nnnnnn 487
__draw_path_ellipse_arciii:nnnnnn 487
__draw_path_ellipse_arciv:nnnnnn 487
\c__draw_path_ellipse_fp 487
__draw_path_grid_auxi:nnnnnn . . 589
__draw_path_grid_auxii:nnnnnn . 589
__draw_path_grid_auxiii:nnnnnn 589
__draw_path_grid_auxiiii:nnnnnn 589
__draw_path_grid_auxiv:nnnnnnnn 589
\g__draw_path_lastx_dim 176, 213, 318, 451, 457, 740, 768
\l__draw_path_lastx_dim 728, 740, 768
\g__draw_path_lasty_dim 176, 214, 325, 452, 458, 741, 769
\l__draw_path_lasty_dim 728, 741, 769
__draw_path_lineto:nn 238, 290
__draw_path_mark_corner: 229, 258, 267, 284, 295, 313, 384
__draw_path_moveto:nn 238, 288, 499, 507
__draw_path_rectangle:nnnn . . . 552
__draw_path_rectangle_corners:nnnn 581
__draw_path_rectangle_corners:nnnnn 584, 587
__draw_path_rectangle_rounded:nnnn 552
__draw_path_reset_limits: 182, 656, 748, 1210
\l__draw_path_tmp_tl 173, 421, 444, 463, 467, 471, 475
\l__draw_path_tmpa_fp 173, 309, 319, 331
\l__draw_path_tmpb_fp 173, 310, 326, 335
__draw_path_update_last:nn 211, 248, 261, 280, 570
__draw_path_update_limits:nn 22, 25, 28, 31, 182, 246, 259, 276, 277, 278, 567, 568
__draw_path_use:n 644
__draw_path_use_action_draw: . . 644
__draw_path_use_action_fillstroke: 644
\l__draw_path_use_bb_bool 642
\l__draw_path_use_clear_bool 642, 679
\l__draw_path_use_clip_bool 639, 662, 682
\l__draw_path_use_fill_bool 639, 663, 685, 691, 703
__draw_path_use_stroke_bb: . . . 644
__draw_path_use_stroke_bb_-aux:NnN 644
\l__draw_path_use_stroke_bool 639, 664, 677, 686, 692, 699, 704
\g__draw_path_xmax_dim 178, 184, 185, 206, 742, 764
\l__draw_path_xmax_dim . 728, 742, 764
\g__draw_path_xmin_dim 178, 186, 187, 207, 743, 765
\l__draw_path_xmin_dim . 728, 743, 765
\g__draw_path_ymax_dim 178, 188, 189, 208, 744, 766
\l__draw_path_ymax_dim . 728, 744, 766
\g__draw_path_ymin_dim 178, 190, 191, 209, 745, 767
\l__draw_path_ymin_dim . 728, 745, 767
__draw_point_interpolate_-arcaxes_auxi:nnnnnnnnn 960
__draw_point_interpolate_-arcaxes_auxii:nnnnnnnnn 960
__draw_point_interpolate_-arcaxes_auxiii:nnnnnnn 960
__draw_point_interpolate_-arcaxes_auxiv:nnnnnnnn 960
__draw_point_interpolate_curve_-auxi:nnnnnnnnn 995, 998
__draw_point_interpolate_curve_-auxii:nnnnnnnnn . 1000, 1004, 1012
__draw_point_interpolate_curve_-auxiii:nnnnnn . . . 1007, 1018, 1026
__draw_point_interpolate_curve_-auxiv:nnnnnn 1020, 1021, 1022, 1027
__draw_point_interpolate_curve_-auxv:nnw 1029, 1033, 1040
__draw_point_interpolate_curve_-auxvi:n 1024, 1045
__draw_point_interpolate_curve_-auxvii:nnnnnnnn 1046, 1047
__draw_point_interpolate_curve_-auxviii:nnnnnn . . 1049, 1056, 1061

__draw_point_interpolate_-distance:nnnn 945, 948
__draw_point_interpolate_-distance:nnnnn 942, 952
__draw_point_interpolate_-distance:nnnnnn 942
__draw_point_interpolate_line_-aux:nnnnn 927
__draw_point_interpolate_line_-aux:nnnnnn 927
__draw_point_intersect_circles_-auxi:nnnnnnn 871
__draw_point_intersect_circles_-auxii:nnnnnnn 871
__draw_point_intersect_circles_-auxiii:nnnnnnn 871
__draw_point_intersect_circles_-auxiv:nnnnnnnn 871
__draw_point_intersect_circles_-auxv:nnnnnnnnn 871
__draw_point_intersect_circles_-auxvi:nnnnnnnn 871
__draw_point_intersect_circles_-auxvii:nnnnnnn 871
__draw_point_intersect_lines:nnnnnn 846
__draw_point_intersect_lines:nnnnnnnn 846
__draw_point_intersect_lines_-aux:nnnnnn 846
__draw_point_process:nn 21, 24, 27, 30, 240, 252, 288, 290, 422, 438, 775, 840, 944, 950, 1076, 1131, 1163, 1733, 1765, 1811
__draw_point_process:nnn . . 302, 428, 554, 583, 591, 775, 873, 929, 1809
__draw_point_process:nnnn 265, 293, 489, 775, 962
__draw_point_process:nnnnn 775, 848, 994
__draw_point_process_auxi:nn . . 775
__draw_point_process_auxii:nw . 775
__draw_point_process_auxiii:nnn 775
__draw_point_process_auxiv:nw . 775
__draw_point_process_auxv:nnnn 775
__draw_point_process_auxvi:nw . 775
__draw_point_process_auxvii:nnnnn 775
__draw_point_process_auxviii:nw 775
__draw_point_to_dim:n 778, 788, 789, 799, 800, 801, 812, 813, 814, 815, 826, 988, 1058, 1090, 1104, 1122, 1138, 1154, 1170, 1183
__draw_point_to_dim_aux:n 826
__draw_point_to_dim_aux:w 826
__draw_point_transform:nn . . . 1129
__draw_point_transform_noshift:n 425, 441, 492, 493, 1161
__draw_point_transform_noshift:nn 1161
__draw_point_unit_vector:nn . . 839
__draw_point_vec:nn 1086
__draw_point_vec:nnn 1086
__draw_point_vec_polar:nnn . . 1116
__draw_reset_bb: . . . 1196, 1209, 1281
__draw_scope_bb_begin: . . 1274, 1293
__draw_scope_bb_end: . . . 1274, 1303
__draw_select_cmyk:nw 1668
__draw_select_gray:nw 1668
__draw_select_rgb:nw 1668
__draw_softpath_add:n 761, 1311, 1335, 1344, 1353, 1358, 1368, 1376, 1629
\c__draw_softpath_arc_fp 1410, 1571, 1575, 1580, 1584
__draw_softpath_clear: 655, 753, 757, 1214, 1314
__draw_softpath_close_op:nn 1337, 1381, 1472, 1508, 1608
__draw_softpath_closepath: 285, 506, 1333
\l__draw_softpath_corneri_dim 1404, 1457, 1460, 1544
\l__draw_softpath_cornerii_dim 1404, 1458, 1461, 1535
\g__draw_softpath_corners_bool . . 752, 759, 1310, 1322, 1378, 1413, 1429
\l__draw_softpath_corners_bool 728, 751, 760
\l__draw_softpath_curve_end_tl 1403, 1532, 1551, 1600, 1611
__draw_softpath_curveto:nnnnnn 279, 1333
__draw_softpath_curveto_opi:nn 1346, 1381, 1469, 1507, 1568
__draw_softpath_curveto_-opi:nnNnnNnn 1381
__draw_softpath_curveto_opii:nn 1347, 1381, 1577
__draw_softpath_curveto_-opiii:nn 1348, 1381, 1586
\l__draw_softpath_first_tl 1404, 1420, 1437, 1438, 1448, 1467, 1468, 1494, 1498
\l__draw_softpath_internal_tl 1309, 1316, 1317, 1422, 1424
\g__draw_softpath_lastx_dim 746, 762, 1329, 1338, 1362

\l__draw_softpath_lastx_dim 734, 746, 762
\l__draw_softpath_lastx_fp 1404, 1418, 1439, 1487, 1547, 1554
\g__draw_softpath_lasty_dim 747, 763, 1329, 1339, 1363
\l__draw_softpath_lasty_dim 735, 747, 763
\l__draw_softpath_lasty_fp 1404, 1419, 1440, 1488, 1548, 1555
__draw_softpath_lineto:nn 260, 1333
__draw_softpath_lineto_op:nn 1354, 1381, 1475, 1506, 1617
\g__draw_softpath_main_tl 749, 1308, 1312, 1316, 1321, 1422, 1628
\l__draw_softpath_main_tl *19*, 749, 761, 1401, 1416, 1443, 1445, 1624, 1626, 1629
\g__draw_softpath_move_bool 1331, 1360
\l__draw_softpath_move_tl 1404, 1421, 1444, 1447, 1495, 1595, 1618, 1625
__draw_softpath_moveto:nn 247, 1333
__draw_softpath_moveto_op:nn 1359, 1381, 1441, 1597
\l__draw_softpath_part_tl 1402, 1417, 1446, 1449, 1451, 1485, 1538, 1627
__draw_softpath_rectangle:nnnn 569, 1333
__draw_softpath_rectangle_opi:nn 1370, 1381
__draw_softpath_rectangle_opi:nnNnn 1381
__draw_softpath_rectangle_opii:nn 1371, 1381
__draw_softpath_round_action:nn 1411
__draw_softpath_round_action:Nnn 1411
__draw_softpath_round_action_close: 1411
__draw_softpath_round_action_curveto:NnnNnn 1411
__draw_softpath_round_calc:nnnNnn 1411
__draw_softpath_round_calc:nnnnnn 1411
__draw_softpath_round_calc:nnnnw 1411
__draw_softpath_round_close:nn 1411
__draw_softpath_round_close:w 1411
__draw_softpath_round_corners: 674, 1411
__draw_softpath_round_end: 1411
__draw_softpath_round_lookahead:NnnNnn 1411
__draw_softpath_round_loop:Nnn 1411
__draw_softpath_round_roundpoint:NnnNnnNnn 1411
__draw_softpath_roundpoint:nn 233, 1333
__draw_softpath_roundpoint_op:nn 1377, 1381, 1434, 1515
__draw_softpath_use: 681, 1314
__draw_softpath_use_clear: 680, 1314
__draw_split_select:nw 1668
\l__draw_stroke_color_tl 1253
\l__draw_tmp_box 11, 35, 46, 50, 52, 53, 54, 55, 61, 63, 64, 65, 66, 67
\l__draw_tmp_seq 1642
__draw_tranform_triangle:nnnnnn 1812, 1817
__draw_transform:nnnn 1741
__draw_transform_invert:n 1773
__draw_transform_rotate:n 1843
__draw_transform_rotate:nn 1843
__draw_transform_shift:nn 1741
__draw_transform_shift_absolute:nn 1715
__draw_vec:nn 1068
__draw_vec:nnn 1068
\g__draw_xmax_dim 194, 195, 1188, 1198, 1233, 1248, 1277, 1285
\l__draw_xmax_dim 1270, 1277, 1285
\g__draw_xmin_dim 196, 197, 1188, 1199, 1231, 1234, 1240, 1248, 1278, 1286
\l__draw_xmin_dim 1270, 1278, 1286
\l__draw_xshift_dim 48, 1143, 1157, 1695, 1710, 1738, 1770, 1804
\l__draw_xvec_x_dim 1062, 1092, 1106, 1124
\l__draw_xvec_y_dim 1062, 1093, 1110
\g__draw_ymax_dim 198, 199, 1188, 1200, 1235, 1245, 1279, 1287
\l__draw_ymax_dim 1270, 1279, 1287
\g__draw_ymin_dim 200, 201, 1188, 1201, 1236, 1241, 1245, 1280, 1288
\l__draw_ymin_dim 1270, 1280, 1288
\l__draw_yshift_dim 49, 1149, 1157, 1695, 1711, 1739, 1771, 1805
\l__draw_yvec_x_dim 1062, 1092, 1107
\l__draw_yvec_y_dim 1062, 1093, 1111, 1125
\l__draw_zvec_x_dim 1062, 1108

\l__draw_zvec_y_dim 1062, 1112
driver commands:
\driver_draw_begin: 1208
\driver_draw_box_use:Nnnnn 39
\driver_draw_cap_butt: 1659
\driver_draw_cap_rectangle: . . 1660
\driver_draw_cap_round: 1661
\driver_draw_clip: 683
\driver_draw_closepath: 1382
\driver_draw_curveto:nnnnnn . . 1386
\driver_draw_dash_pattern:nn . 1650
\driver_draw_end: 1229
\driver_draw_evenodd_rule: . . . 1662
\driver_draw_join_bevel: 1664
\driver_draw_join_miter: 1665
\driver_draw_join_round: 1666
\driver_draw_lineto:nn 1392
\driver_draw_linewidth:n 1640
\driver_draw_miterlimit:n 1658
\driver_draw_moveto:nn 1394
\driver_draw_nonzero_rule: . . . 1663
\driver_draw_rectangle:nnnn . . 1399
\driver_draw_scope_begin: . 130, 1258
\driver_draw_scope_end: . . . 133, 1268

E

\end 167
\ERROR 671
exp commands:
\exp_after:wN 444, 462, 1423, 1497, 1598, 1609, 1618
\exp_args:Nf 777
\exp_args:Nff 787
\exp_args:Nfff 798
\exp_args:Nffff 811
\exp_args:NNNV 1226
\exp_not:N 1540, 1568, 1577, 1586, 1595, 1598, 1599, 1600, 1601, 1605, 1609, 1610, 1611, 1612

F

fp commands:
\fp_compare:nNnTF 358, 368
\fp_compare_p:nNn 1723, 1724, 1725, 1726
\fp_const:Nn 341, 342, 480, 481, 549, 1410
\fp_eval:n 350, 351, 372, 379, 388, 827, 835, 855, 856, 857, 858, 859, 860, 880, 885, 886, 893, 900, 901, 908, 915, 917, 930, 935, 953, 969, 974, 981, 982, 1001, 1008, 1030, 1031, 1050, 1051, 1052, 1053, 1087, 1100, 1119, 1658, 1746, 1747, 1748, 1749, 1779, 1844, 1848, 1849
\fp_new:N 174, 175, 478, 479, 1404, 1405, 1695, 1696, 1697, 1698
\fp_set:Nn 309, 310, 364, 365, 445, 446, 1439, 1440, 1487, 1488, 1554, 1555, 1703, 1706, 1717, 1718, 1719, 1720, 1792, 1794, 1796, 1798
\fp_to_decimal:N 371, 378, 386
\fp_to_dim:n 316, 323, 330, 334, 352, 353, 399, 408, 476, 500, 512, 513, 514, 515, 516, 517, 522, 523, 524, 525, 526, 527, 532, 533, 534, 535, 536, 537, 542, 543, 544, 545, 546, 547, 615, 616, 1570, 1574, 1579, 1583, 1639, 1647, 1652
\fp_use:N 40, 41, 42, 43, 549
\fp_while_do:nNnn 366
\fp_zero:N 1418, 1419, 1704, 1705
\c_one_fp 1723, 1726
\c_zero_fp 1724, 1725

G

group commands:
\group_begin: 34, 60, 90, 101, 739, 1205, 1259, 1276, 1415, 1644
\group_end: 56, 68, 115, 118, 770, 1251, 1267, 1289, 1427, 1654

H

hbox commands:
\hbox_gset:Nw 99
\hbox_gset_end: 116
\hbox_set:Nn 35, 46, 61, 1238
\hbox_set:Nw 1207, 1222
\hbox_set_end: 1226, 1230

I

int commands:
\int_gincr:N 1206
\int_if_odd:nTF 916
\int_new:N 1195

M

mode commands:
\mode_leave_vertical: 1249
msg commands:
\msg_error:nnn 76, 107, 108
\msg_new:nnn 162
\msg_new:nnnn 159, 164

P

\pgfextractx *20*
\pgfextracty *20*
\pgfgetlastxy *20*

\pgfgettransform *46*
\pgfgettransformentries *46*
\pgfinnerlinewidth *44*
\pgflowlevel *46*
\pgflowlevelsynccm *46*
\pgfpatharcto *5*
\pgfpatharctoprecomputed *5*
\pgfpathcosine *5*
\pgfpathcurvebetweentime *5*
\pgfpathcurvebetweentimecontinue *5*
\pgfpathparabola *5*
\pgfpathsine *5*
\pgfpointadd *20*
\pgfpointborderellipse *20*
\pgfpointborderrectangle *20*
\pgfpointcylindrical *20*
\pgfpointdiff *20*
\pgfpointorigin *20*
\pgfpointscale *20*
\pgfpointspherical *20*
\pgfqpoint *20*
\pgfqpointpolar *20*
\pgfqpointscale *20*
\pgfqpointxy *20*
\pgfqpointxyz *20*
\pgfsetinnerlinewidth *44*
\pgfsetinnerstrokecolor *44*
\pgftransformarcaxesattime *46*
\pgftransformarrow *46*
\pgftransformcurveattime *46*
\pgftransformlineattime *46*
prg commands:
  \prg_do_nothing: 1023, 1034, 1037
\ProvidesExplPackage *4*

Q

quark commands:
  \q_mark 793, 794, 805, 807, 821, 824, 1561, 1565
  \quark_if_recursion_tail_stop_-do:Nn 1433
  \q_recursion_stop 1426
  \q_recursion_tail 1425
  \q_stop 782, 783, 793, 794, 805, 807, 821, 824, 1561, 1565, 1601, 1612, 1621, 1679, 1681, 1682, 1683, 1685, 1687, 1689

R

\RequirePackage *7*

S

seq commands:
  \seq_new:N 1656
  \seq_set_from_clist:Nn 1645
  \seq_set_map:NNn 1646
  \seq_use:Nn 1651
skip commands:
  \skip_horizontal:n 1240
str commands:
  \str_if_eq:nnTF 75, 93, 106, 124, 141, 152
  \str_if_eq_p:nn 653

T

tex commands:
  \tex_kern:D 48
tl commands:
  \tl_build_gclear:N 1321, 1628
  \tl_build_get:NN 749, 1316, 1422
  \tl_build_gput_right:Nn 1312
  \tl_clear:N 421, 1416, 1417, 1420, 1421, 1448, 1449
  \tl_if_blank:nTF 646
  \tl_if_blank_p:n 652
  \tl_if_empty:NTF 1437, 1467
  \tl_if_empty_p:N 1494, 1495
  \tl_new:N 82, 173, 1254, 1255, 1308, 1309, 1401, 1402, 1403, 1408, 1409, 1667
  \tl_put_right:Nn . . 471, 475, 1443, 1445, 1451, 1485, 1538, 1624, 1626
  \tl_set:Nn 83, 97, 467, 1438, 1447, 1468, 1532, 1595
token commands:
  \token_if_eq_meaning:NNTF 1434, 1441, 1469, 1472, 1475, 1515
  \token_if_eq_meaning_p:NN 1506, 1507, 1508

U

use commands:
  \use:N 688, 719, 1682, 1684, 1686, 1688, 1690
  \use:n 37, 311, 347, 394, 497, 1591, 1603, 1648, 1743, 1754, 1819
  \use_i:nn *20*
  \use_i:nnnn 1605
  \use_ii:nn *20*
  \use_none:n 1618