File I
Implementation

1 l3draw implementation

(*inpro|package)
(*@@=draw)
(*package)
\ProvidesExplPackage{l3draw}{2019-03-05}{ }
{L3 Experimental core drawing support}
(/package)
\RequirePackage{ l3color }
Everything else is in the sub-files!
(*inpro|package)

2 l3draw-boxes implementation

(*inpro|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 opera-
tions, 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 , #3 } }
  }
}
\_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:Nnnn. 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:Nnnn \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
\section*{User interface}

\begin{verbatim}
\draw_layer_new:n
\end{verbatim}

\begin{verbatim}
\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{verbatim}

\begin{verbatim}(End definition for \draw_layer_new:n. This function is documented on page ??.)\end{verbatim}

\begin{verbatim}
\l__draw_layer_tl
\end{verbatim}

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

\begin{verbatim}
\tl_new:N \l__draw_layer_tl
\tl_set:Nn \l__draw_layer_tl { main }
\end{verbatim}

\begin{verbatim}(End definition for \l__draw_layer_tl.)\end{verbatim}

\begin{verbatim}
\l__draw_layer_close_bool
\end{verbatim}

Used to track if a layer needs to be closed.

\begin{verbatim}
\bool_new:N \l__draw_layer_close_bool
\end{verbatim}

\begin{verbatim}(End definition for \l__draw_layer_close_bool.)\end{verbatim}

\begin{verbatim}
\l_draw_layers_clist \g__draw_layers_clist
\end{verbatim}

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

\begin{verbatim}
\clist_new:N \l_draw_layers_clist
\clist_set:Nn \l_draw_layers_clist { main }
\clist_new:N \g__draw_layers_clist
\end{verbatim}

\begin{verbatim}(End definition for \l_draw_layers_clist and \g__draw_layers_clist. This variable is documented on page ??.)\end{verbatim}

\begin{verbatim}
\draw_layer_begin:n \draw_layer_end:
\end{verbatim}

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.

\begin{verbatim}
\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 }
        }
        \draw_linewidth:n { \l_draw_default_linewidth_dim }
        \group_begin:
        \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 }
        \end{verbatim}

\begin{verbatim}(End definition for \draw_layer_begin:n and \draw_layer_end.)\end{verbatim}
\begin{verbatim}
\cs_new_protected:Npn \draw_layer_end:
{ \bool_if:NT \l__draw_layer_close_bool
  { \group_end:
    \hbox_gset_end:
  }
  \group_end:
}
\end{verbatim}

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

3.2 Internal cross-links

\begin{verbatim}
\__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 }
  { \msg_error:nnn { draw } { unknown-layer } {##1} }
  { \msg_error:nnn { draw } { main-layer } }
}
\end{verbatim}

(End definition for \__draw_layers_insert.)

\begin{verbatim}
\__draw_layers_save: 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 } }
}
\end{verbatim}

(End definition for \__draw_layers_save.)
\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 }

\msg_new:nnn { draw } { main-reserved }

\msg_new:nnnn { draw } { unknown-layer }

\msg_new:nnn { draw } { main-layer-is-reserved. }

\msg_new:nnn { draw } { main-layer-may-only-be-accessed-at-the-top-level. }

\msg_new:nnn { draw } { material-cannot-be-added-to-`main'-layer. }

\msg_new:nnn { draw } { the-main-layer-may-only-be-accessed-at-the-top-level. }

\msg_new:nnn { draw } { the-`main’-layer-is-reserved. }

\msg_new:nnn { draw } { layer-`#1’-has-not-been-created. }

\msg_new:nnn { draw } { you-have-tried-to-use-layer-`#1’,-but-it-was-never-set-up. }

\%
% \end{macrocode}
%
% \begin{macrocode}
//initex | package

\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.)

\l3draw-paths implementation

(*initex | package)
\@@=draw

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

- \texttt{\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.

- \texttt{\pgfpathparabola}: Seems to be unused other than defining a Ti\textit{k}Z interface, which
  itself is then not used further.

- \texttt{\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.

- \texttt{\pgfpathcurvebetweentime, \pgfpathcurvebetweentimecontinue}: These don’t
  seem to be used at all.
4.1 Tracking paths

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.)

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}
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 \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

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

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

(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

Operations with no application of the transformation matrix.
\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:nnn
\c__draw_path_curveto_a_fp
\c__draw_path_curveto_b_fp

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

\[
\begin{align*}
x_1 &= \frac{1}{3} x_s + \frac{2}{3} x_c \\
y_1 &= \frac{1}{3} y_s + \frac{2}{3} y_c
\end{align*}
\]

and

\[
\begin{align*}
x_2 &= \frac{1}{3} x_e + \frac{2}{3} x_c \\
y_2 &= \frac{1}{3} y_e + \frac{2}{3} y_c
\end{align*}
\]

using the start (last) point \((x_s, y_s)\) and the end point \((x_e, y_e)\).
\cs_new_protected:Npn \__draw_path_curveto:nn #1#2
{ \__draw_point_process:nnnn
  \__draw_path_mark_corner:
  \__draw_path_curveto:nnnnnnn
}
\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:nnnnnnn
}
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
\cs_new_protected:Npn \draw_path_arc:nnnn #1#2#3#4
\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:nnNnn #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_add:nnnn {\__draw_path_arc:nnn {\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_auxii:nnnn {\__draw_path_arc:nnn {\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_auxvi:nnnn {\__draw_path_arc:nnn {\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_add:nnnn {\fp_compare:nNnTF {\#1} > {\#2} {\__draw_path_arc:nnNnn {\#1} {\#2} - {\#3} {\#4}} {\__draw_path_arc:nnNnn {\#1} {\#2} + {\#3} {\#4}}}

(End definition for \draw_path_curveto:nn and others. This function is documented on page ??.)
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: \(\frac{1}{2}(\sqrt{2} - 1) = 0.55228475\). For other cases, we follow the calculation used by \textit{pgf} but with the second common case of 60° pre-calculated for speed.
\begin{verbatim}
{ 4/3 * tand( 0.25 * #3 ) }
* #7
}
}
}
}
}
\cs_generate_variant:Nn \__draw_path_arcAuxi: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

\begin{verbatim}
\cs_new_protected:Npn \__draw_path_arcAuxii:nnn #1#2#3#4#5#6#7#8
{
\tl_clear:N \l__draw_path_tmp_tl
\__draw_point_process:nn
{ \__draw_path_auxiii:nn }
{ }
\__draw_point_transform_noshift:n
{ \draw_point_polar:nnn { #1 #4 90 } \{#7\} \{#8\} }
\__draw_point_process:nnn
{ \__draw_path_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_auxv:nn }
{ }
\__draw_point_transform_noshift:n
{ \draw_point_polar:nnn { #2 #4 -90 } \{#7\} \{#8\} }
\exp_after:wN \__draw_path_curveto:nnnnn \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}
}
\end{verbatim}

The first control point.

\begin{verbatim}
\cs_new_protected:Npn \__draw_path_auxiii:nn #1#2
{
 \__draw_path_aux_add:nn
{ \g__draw_path_lastx_dim + #1 }
{ \g__draw_path_lasty_dim + #2 }
}
\end{verbatim}

The end point: simple arithmetic.

\begin{verbatim}
\cs_new_protected:Npn \__draw_path_auxiv:nnnn #1#2#3#4
{
 \__draw_path_aux_add:nn
}
\end{verbatim}

12
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 } }
\cs_new_protected:Npn \__draw_path_ellipse:nnnnnn #1#2#3#4#5#6#7
{ \use:x { \__draw_point_process:nnnn { \__draw_path_ellipse:nnnnnnnn } { \draw_point_transform:n {#1} } { \draw_point_transform:noshift:n {#2} } { \draw_point_transform:noshift:n {#3} } } }
\draw_path_ellipse:nnn Drawing an ellipse is an optimised version of drawing an arc, in particular reusing the same constant. We need to deal with the ellipse in four parts and also deal with moving to the right place, closing it and ending up back at the center. That is handled on a per-arc basis, each in a separate auxiliary for readability.
\__draw_path_moveto:nn

\__draw_path_ellipse_arci:nnnnnn {#1} {#2} {#3} {#4} {#5} {#6}
\__draw_path_ellipse_arclii: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_arclii: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_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 \c__draw_path_ellipse_fp - #5 } }
{ \fp_to_dim:n { #2 + #4 \c__draw_path_ellipse_fp - #6 } }
{ \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\quad A shortcut.
\cs_new_protected:Npn \draw_path_circle:nn #1#2
\{ \draw_path_ellipse:nnn {#1} { #2 , Opt } { 0pt , #2 } \}

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

4.6 Rectangles

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 ??.)
4.7 Grids

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

\begin{verbatim}
\draw_path_grid:nnnn
  \_draw_path_grid_auxi:nnnn
  \_draw_path_grid_auxi:ffnnnn
  \_draw_path_grid_auxii:nnnnnn
  \_draw_path_grid_auxii:ffnnnn
  \_draw_path_grid_auxiiii:nnnnnn
  \_draw_path_grid_auxiiii:ffnnnn
  \_draw_path_grid_auxiv:nnnnnnnn
  \_draw_path_grid_auxiv:ffnnnnnn
\end{verbatim}
4.8 Using paths

Actions to pass to the driver.

\begin{verbatim}
\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{verbatim}

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

Actions handled at the macro layer.

\begin{verbatim}
\bool_new:N \l__draw_path_use_bb_bool
\bool_new:N \l__draw_path_use_clear_bool
\end{verbatim}

(End definition for \l__draw_path_use_bb_bool and \l__draw_path_use_clear_bool.)

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

\begin{verbatim}
\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 } } }
\end{verbatim}

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).

\begin{verbatim}
\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 } }
}
\end{verbatim}

(End definition for \_\_draw_path_use_clip_bool, \_\_draw_path_use_fill_bool, and \_\_draw_path_use_stroke_bool.)

\begin{verbatim}
\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 } } }
\end{verbatim}

(End definition for \_\_draw_path_use_bb_bool and \_\_draw_path_use_clear_bool.)

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

\begin{verbatim}
\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 } }
}
\end{verbatim}

(End definition for \_\_draw_path_use_clip_bool, \_\_draw_path_use_fill_bool, and \_\_draw_path_use_stroke_bool.)

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).

\begin{verbatim}
\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 } }
}
\end{verbatim}

(End definition for \_\_\_draw_path_use_clip_bool, \_\_\_draw_path_use_fill_bool, and \_\_\_draw_path_use_stroke_bool.)

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).
\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 } = { #3 -\c_max_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
4.9 Scoping paths

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

\cs_new_protected:Npn \draw_path_scope_begin:
  {\group_begin:}
\cs_new_protected:Npn \draw_path_scope_end:
  {\__draw_softpath_clear:}

(End definition for \_\_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:}
\cs_new_protected:Npn \draw_path_scope_end:
  {\__draw_softpath_clear:}

(End definition for \_\_draw_path_lastx_dim and others.)
\dim_gset_eq:NN \g__draw_path_xmin_dim \l__draw_path_xmin_dim
\dim_gset_eq:NN \g__draw_path_ymin_dim \l__draw_path_ymin_dim
\dim_gset_eq:NN \g__draw_path_ymax_dim \l__draw_path_ymax_dim
\dim_gset_eq:NN \g__draw_path_maxx_dim \l__draw_path_maxx_dim
\dim_gset_eq:NN \g__draw_path_maxy_dim \l__draw_path_maxy_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 ??.)

\l3draw-points implementation

5 (*tex | package)

\@@=draw

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

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

- \texttt{\pgfpointcylindrical}, \texttt{\pgfpointspherical}: The usefulness of these commands is not currently clear.
- \texttt{\pgfpointborderrectangle}, \texttt{\pgfpointborderellipse}: To be revisited once the semantics and use cases are clear.
- \texttt{\pgfpoint}, \texttt{\pgfpointscale}, \texttt{\pgfpointpolar}, \texttt{\pgfpointxy}, \texttt{\pgfpointxyz}: The expandable approach taken in the code here, along with the absolute requirement for \texttt{\epsilon-\LaTeX}, means it is likely many use cases for these commands may be covered in other ways. This may be revisited as higher-level structures are constructed.

5.1 Support functions

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

\cs_new:Npn \__draw_point_process:nn #1#2
\exp_args:Nf \__draw_point_process_auxi:nn
\__draw_point_process:nnn
\exp_args:Nf \__draw_point_process_auxii:nnn
\__draw_point_process:nnnn
\exp_args:Nf \__draw_point_process_auxiii:nnnn
\__draw_point_process:nnnnn
\exp_args:Nf \__draw_point_process_auxiv:nnnnn
\__draw_point_process:nnnnnn
\exp_args:Nf \__draw_point_process_auxv:nnnnnn
\__draw_point_process:nnnnnnn
\exp_args:Nf \__draw_point_process_auxvi:nnnnnnn
\__draw_point_process:nnnnnnnn
\exp_args:Nf \__draw_point_process_auxvii:nnnnnnnn
\__draw_point_process:nnnnnnnnn
\exp_args:Nf \__draw_point_process_auxviii:nnnnnnnnn
\__draw_point_process:nnnnnnnnnn
\exp_args:Nf \__draw_point_process_auxix:nnnnnnnnnn
\__draw_point_process:nnnnnnnnnnn
\exp_args:Nf \__draw_point_process_auxx:nnnnnnnnnnn
\__draw_point_process:nnnnnnnnnnnn
\exp_args:Nf \__draw_point_process_auxxi:nnnnnnnnnnnn
\__draw_point_process:nnnnnnnnnnnnn
\exp_args:Nf \__draw_point_process_auxxii:nnnnnnnnnnnnn
\__draw_point_process:nnnnnnnnnnnnnn
\exp_args:Nf \__draw_point_process_auxxiii:nnnnnnnnnnnnnn
\__draw_point_process:nnnnnnnnnnnnnnn
\exp_args:Nf \__draw_point_process_auxxiv:nnnnnnnnnnnnnnn
\__draw_point_process:nnnnnnnnnnnnnnnn
\exp_args:Nf \__draw_point_process_auxxv:nnnnnnnnnnnnnnnn
\__draw_point_process:nnnnnnnnnnnnnnnnn
\exp_args:Nf \__draw_point_process_auxxvi:nnnnnnnnnnnnnnnnn
\__draw_point_process:nnnnnnnnnnnnnnnnnn
\exp_args:Nf \__draw_point_process_auxxvii:nnnnnnnnnnnnnnnnnnn
\__draw_point_process:nnnnnnnnnnnnnnnnnnn
\exp_args:Nf \__draw_point_process_auxxviii:nnnnnnnnnnnnnnnnnnn
\__draw_point_process:nnnnnnnnnnnnnnnnnnn
Co-ordinates are always returned as two dimensions.
5.2 Polar co-ordinates

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

5.3 Point expression arithmetic

These functions all take point expressions as arguments. 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.

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

5.4 Intersection calculations

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

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

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

\begin{align*}
#1 & \quad x_1 \\
#2 & \quad y_1 \\
#3 & \quad x_2 \\
#4 & \quad y_2 \\
#5 & \quad x_3 \\
#6 & \quad y_3 \\
#7 & \quad x_4 \\
#8 & \quad y_4
\end{align*}

so now just have to do all of the calculation.

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{align*}
e &= c - a \\
f &= d - b \\
p &= \sqrt{e^2 + f^2} \\
k &= \frac{p^2 + r^2 - s^2}{2p}
\end{align*}
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.

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.
\_\_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 ) \} \}
\fp_eval:n { sqrt ( \#3 - \#1 * \#1 ) * \#2 }, {\#4} {\#5} {\#7} {\#8} {\#9}
\cs_generate_variant:Nn \_\_draw_point_intersect_circles_auxvi:nnnnnnnn \{ ff \}

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

\begin{align*}
&\#1 \quad k \\
&\#2 \quad 1/p \\
&\#3 \quad r^2 \\
&\#4 \quad e \\
&\#5 \quad f \\
&\#6 \quad a \\
&\#7 \quad b \\
&\#8 \quad n
\end{align*}

There are some final pre-calculations, \( k/p \), \( \sqrt{r^2 - k^2} \), 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:fffnnnnn
\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:nnnnnnnn \#1#2#3#4#5#6#7
{

25
923 \_\_draw_point_to_dim:n
924 { \#6 + \#4 * \#1 + \#2 * \#3 * \#5 , \#7 + \#5 * \#1 + -1 * \#2 * \#3 * \#4 }
925 }
926 \cs_generate_variant:Nn \_\_draw_point_intersect_circles_auxvii:nnnnnn { fff }

5.5 Interpolation on a line (vector) or arc

Simple maths after expansion.

\cs_new:Npn \draw_point_interpolate_line:nnn #1#2#3
937 { \__draw_point_process:nnn { \__draw_point_interpolate_line_aux:fnnnn { \fp_eval:n {#1} } } #2 #3 }
938 \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
958 \__draw_point_to_dim:n { #2 + #1 * #4 , #3 + #1 * #5 }
959 \cs_generate_variant:Nn \__draw_point_interpolate_line_aux:nnnnnn { f }

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

\begin{itemize}
\item \#1 \( p \)
\item \#2 \( \theta_1 \)
\item \#3 \( \theta_2 \)
\item \#4 \( x_c \)
\item \#5 \( y_c \)
\item \#6 \( x_{a1} \)
\item \#7 \( y_{a1} \)
\item \#8 \( x_{a2} \)
\item \#9 \( y_{a2} \)
\end{itemize}

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

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

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

\[
\text{\texttt{\textbackslash cs\_new:Npn \textbackslash draw\_point\_interpolate\_curve:nnnnnn \#1\#2\#3\#4\#5}}
\]

\[
\text{\texttt{\textbackslash \_\_draw\_point\_process:nnnn}}
\]

\[
\text{\texttt{\{ \textbackslash \_\_draw\_point\_interpolate\_curve\_auxi:nnnnnnn \{#1\} \}}}
\]

\[
\text{\texttt{\{#2\} \{#3\} \{#4\} \{#5\} \}}
\]

\[
\text{\texttt{\textbackslash cs\_new:Npn \textbackslash \_\_draw\_point\_interpolate\_curve\_auxi:nnnnnnnn \#1\#2\#3\#4\#5\#6\#7\#8\#9}}
\]

\[
\text{\texttt{\{ \texttt{fp\_eval:n} \ (#1) \}}}
\]

\[
\text{\texttt{\{#2\} \{#3\} \{#4\} \{#5\} \{#6\} \{#7\} \{#8\} \{#9\} \}}
\]

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

\[
x'_1 = (1 - p)x_1 + px_2
\]

\[
y'_1 = (1 - p)y_1 + py_2
\]

\[
x'_2 = (1 - p)x_2 + px_3
\]

\[
y'_2 = (1 - p)y_2 + py_3
\]

\[
x'_3 = (1 - p)x_3 + px_4
\]

\[
y'_3 = (1 - p)y_3 + py_4
\]

In the second stage, we can drop the final point

\[
x''_1 = (1 - p)x'_1 + px'_2
\]

\[
y''_1 = (1 - p)y'_1 + py'_2
\]

\[
x''_2 = (1 - p)x'_2 + px'_3
\]

\[
y''_2 = (1 - p)y'_2 + py'_3
\]

and for the final stage only need one set of calculations

\[
P_x = (1 - p)x''_1 + px''_2
\]

\[
P_y = (1 - p)y''_1 + py''_2
\]

Of course, this does mean a lot of calculations and expansion!
\begin{macrocode}
% We need to do the first cycle, but haven’t got enough arguments to keep
% everything in play at once. So here we use a but of argument re-ordering
% and a single auxiliary to get the job done.
% \begin{macrocode}
\cs_generate_variant:Nn \__draw_point_interpolate_curve_auxii:nnnnnn { f }
% \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} }
}
% \begin{macrocode}
\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 } }
}
% \begin{macrocode}
\cs_new:Npn \__draw_point_interpolate_curve_auxv:nnw
#1#2#3 \prg_do_nothing: #4#5
{ #3
  #4 { #5 {#1} {#2} }
}
% \begin{macrocode}
\cs_generate_variant:Nn \__draw_point_interpolate_curve_auxv:nnw { ff }
% \begin{macrocode}
\cs_new:Npn \__draw_point_interpolate_curve_auxvi:n #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}
}
% \begin{macrocode}
\cs_new:Npn \__draw_point_interpolate_curve_auxviii:ffffnn
{ \__draw_point_to_dim:n
  { \fp_eval:n { #5 * #3 + #6 * #1 , #5 * #4 + #6 * #2 } }
}
% \begin{macrocode}
\cs_generate_variant:Nn \__draw_point_interpolate_curve_auxviii:ffffnn { ffff }
\end{macrocode}
(End definition for \texttt{\textbackslash 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

As base vectors to map to the underlying two-dimensional drawing space.

\begin{verbatim}
\draw_xvec:n { 1cm, 0cm }
\draw_yvec:n { 0cm, 1cm }
\draw_zvec:n { -0.385cm, -0.385cm }
\end{verbatim}

Calculate the underlying position and store it.

\begin{verbatim}
\draw_xvec:n \draw_yvec:n \draw_zvec:n
\__draw_vec:nn \__draw_vec:nnn
\end{verbatim}

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

\begin{verbatim}
\draw_point_vec:nn \__draw_point_vec:nn \__draw_point_vec:ff
\draw_point_vec:nnn \__draw_point_vec:fff
\end{verbatim}

\end{document}
\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 { f ff f }

(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:nn #1 #2
\{ \_draw_point_process:nn { \_draw_point_transform:nn } {#1} {#2} \}
\cs_new:Npn \_draw_point_transform:nn #1 #2
\{ \_draw_point_to_dim:n
{ \l__draw_matrix_a_fp \* #1 
+ \l__draw_matrix_c_fp \* #2 
+ \l__draw_xshift_dim
\}
\}
\cs_generate_variant:Nn \_draw_point_transform:nn { f ff f }
\draw_point_transform:n \draw_point_transform_noshift:n

A version with no shift: used for internal purposes.

\cs_new:Npn \__draw_point_transform_noshift:n #1
\cs_new:Npn \__draw_point_transform_noshift:nn #1#2

\bool_if:NTF \l__draw_matrix_active_bool
{ \__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 ??.)

6 \l3draw-scopes implementation

(*initex | package)
6.1 Drawing environment

\g__draw_xmax_dim\ Used to track the overall (official) size of the image created: may not actually be the
\g__draw_xmin_dim\ natural size of the content.
\g__draw_ymax_dim\ \dim_new:N \g__draw_xmax_dim
\g__draw_ymin_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: Drawings are created by setting them into a box, then adjusting the box before inserting
\draw_end: into the surroundings. Color is set here using the drawing mechanism largely as it then
sets up the internal data structures. It may be that a coffin construct is better here in
the longer term: that may become clearer as the code is completed. As we need to avoid
any insertion of baseline skips, the outer box here has to be an \texttt{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
\par}
\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.)
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:
\}

\l__draw_xmax_dim \l__draw_xmin_dim \l__draw_ymax_dim \l__draw_ymin_dim

\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

\__draw_scope_bb_begin:
\__draw_scope_bb_end:

\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:
\}

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

\__draw_scope_bb_end:

\\draw_suspend_begin:
\\draw_suspend_end:

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

\\draw_suspend_end: Suspend all parts of a drawing.
\__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 ??.)

7 l3draw-softwarepath implementation

(*in tex | 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 straightforward. 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 softenpath_corners_bool
Allow for optimised path use.
\bool_new:N \g__draw softenpath_corners_bool
(End definition for \g__draw softenpath_corners_bool.)

\__draw softenpath_add:n
\__draw softenpath_add:o
\__draw softenpath_add:x
\cs_new_protected:Npn \__draw softenpath_add:n
\{ \tl_build_gput_right:Nn \g__draw softenpath_main_tl \}
\cs_generate_variant:Nn \__draw softenpath_add:n \{ o, x \}
(End definition for \__draw softenpath_add:n.)
Using and clearing is trivial.

\use{\_\_draw_softpath_use}
\clear{\_\_draw_softpath_clear}
\use\_\_draw_softpath_use\_clear

\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.)

\curveto:nnnnnn
\lineto:nn
\moveto:nn
\rectangle:nnnn
\roundpoint:nn
\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}\}
\}

37
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.
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 \texttt{pgf}: step through, find the corners, find the supporting data, do the rounding.

```latex
\l__draw_softpath_main_tl
```

For constructing the updated path.

```latex
\l__draw_softpath_part_tl
```

Data structures.

```latex
\l__draw_softpath_lastx_fp \l__draw_softpath_lasty_fp
```

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

```latex\c__draw_softpath_arc_fp
```

The magic constant.

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.

```latex \cs_new_protected:Npn \__draw_softpath_round_corners: 
```

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.
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.

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.
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 }

  #1#2#3#4#5#6#7#8#9 
  { \token_if_eq_meaning:p:NN \__draw_softpath_lineto_op:nn 
    \__draw_softpath_round_start:nn 
    \__draw_softpath_round_loop:Nnn } 

\cs_new_protected:Npn \__draw_softpath_round_start:nn 
  \__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
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
\begin{Verbatim}
{#8} {#9} #1 (#2) (#3)
#4 (#5) (#6) #7 (#8) (#9)
\end{Verbatim}
\}

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软path用圆角计算：nnnNnn #1#2#3#4#5#6
\begin{Verbatim}
{#8} {#9} #1 (#2) (#3)
#4 (#5) (#6) #7 (#8) (#9)
\end{Verbatim}
\}

The calculations themselves are relatively straight-forward, as we use a quadratic Bézier curve.
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.
{#1} {#2}
\__draw_softpath_lineto_op:nn
\exp_after:wN \use_none:n \l__draw_softpath_move_tl
}
\exp_after:wN \use_none:n \l__draw_softpath_lineto_op:nn
\use_none:n \l__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: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 \l__draw_softpath_round_close:w 1 \l__draw_softpath_main_tl
}

(End definition for \__draw_softpath_round_corners: and others.)

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

\draw_dash_pattern:nn \l__draw_tmp_seq

\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
\{ \draw_driver_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

\cs_new_protected:Npn \draw_miterlimit:n #1
\{ \draw_driver_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:

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

\l__draw_color_tmp_tl

\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:nn
\__draw_select_cmyk:nn
\__draw_select_gray:nn
\__draw_select_rgb:nn
\__draw_split_select:nn

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
\\l__draw_color_tmp_tl \l__draw_color_tmp_tl {#1} \}
\cs_new_protected:Npn \__draw_split_select:nn #1#2
\{ \color_parse:nN {#2} \l__draw_color_tmp_tl
\l__draw_color_aux:Vn \l__draw_color_tmp_tl {#1} \}
\cs_new_protected:Npn \__draw_color_aux:nn #1#2

45
This sub-module covers more-or-less the same ideas as \texttt{pgfcoretransformations.code.tex}.

At present, equivalents of the following are currently absent:

- \texttt{\pgfgettransform, \pgfgettransformentries}: Awaiting use cases.
- \texttt{\pgftransformlineatime, \pgftransformarcaxesatime, \pgftransformcurveatime}: Need to look at the use cases for these to fully understand them.
- \texttt{\pgftransformarrow}: Likely to be done when other arrow functions are added.
- \texttt{\pgflowlevelsynccm, \pgflowlevel}: Likely to be added when use cases are encountered in other parts of the code.

\begin{Verbatim}
\l__draw_matrix_active_bool
\end{Verbatim}

An internal flag to avoid redundant calculations.

\begin{Verbatim}
\bool_new:N \l__draw_matrix_active_bool
\end{Verbatim}

\begin{Verbatim}
\l__draw_matrix_a_fp \l__draw_matrix_b_fp \l__draw_matrix_c_fp \l__draw_xshift_dim \l__draw_yshift_dim
\end{Verbatim}

The active matrix and shifts.

\begin{Verbatim}
\draw_transform_matrix_reset: \draw_transform_shift_reset:
\end{Verbatim}

Fast resetting.
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
\use:x { \__draw_transform:nnnn { \fp_eval:n {#1} } { \fp_eval:n {#2} } { \fp_eval:n {#3} } { \fp_eval:n {#4} } }
```

(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 77.)

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} } }
```

\makeatletter
\begin{align*}
\CSnewprotected:Npn \__draw_transform:nnnn #1#2#3#4
\{ #4 \}
\}
\\use:x
\{ \draw_transform_matrix_absolute:nnnn
\{ #1 \times \l__draw_matrix_a_fp + #2 \times \l__draw_matrix_c_fp \}
\{ #1 \times \l__draw_matrix_b_fp + #2 \times \l__draw_matrix_d_fp \}
\{ #3 \times \l__draw_matrix_a_fp + #4 \times \l__draw_matrix_c_fp \}
\{ #3 \times \l__draw_matrix_b_fp + #4 \times \l__draw_matrix_d_fp \}
\}
\CSnewprotected:Npn \draw_transform_shift:n #1
\{ \__draw_point_process:nn \{ \__draw_transform_shift:nn \} {#1} \}
\CSnewprotected: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 \}
\}
\makeatother

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

\begin{align*}
\CSnewprotected:Npn \__draw_transform_invert:n
\{ \__draw_transform_invert:f
\}
\CSnewprotected:Npn \__draw_transform_invert:f
\{ \fp_eval:n \{ \}
\}
\CSnewprotected:Npn \__draw_transform_invert:n #1
\{ \fp_set:Nn \l__draw_matrix_a_fp \{ \l__draw_matrix_a_fp \}
\fp_set:Nn \l__draw_matrix_b_fp \{ \l__draw_matrix_b_fp \}
\fp_set:Nn \l__draw_matrix_c_fp \{ \l__draw_matrix_c_fp \}
\fp_set:Nn \l__draw_matrix_d_fp \{ \l__draw_matrix_d_fp \}
\}
\makeatother

\textbf{Standard mathematics: calculate the inverse matrix and use that, then undo the shifts.}

\begin{align*}
\CSnewprotected:Npn \draw_transform_matrix_invert:
\{ \CSnewprotected:Npn \__draw_transform_invert:n
\{ \\bool_if:NT \l__draw_matrix_active_bool
\{ \__draw_transform_invert:f
\{ \fp_eval:n
\{ 1 / \}
\{ \l__draw_matrix_a_fp \times \l__draw_matrix_d_fp
\} - \l__draw_matrix_b_fp \times \l__draw_matrix_c_fp
\}
\}
\}
\CSnewprotected:Npn \__draw_transform_invert:n #1
\{ \fp_set:Nn \l__draw_matrix_a_fp \{ \l__draw_matrix_a_fp \}
\fp_set:Nn \l__draw_matrix_b_fp \{ \l__draw_matrix_b_fp \}
\fp_set:Nn \l__draw_matrix_c_fp \{ \l__draw_matrix_c_fp \}
\fp_set:Nn \l__draw_matrix_d_fp \{ \l__draw_matrix_d_fp \}
\}
\end{align*}
\__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_transform_triangle:nnnnn
  \{ #1 \}
\{ #2 \} \{ #3 \}
\cs_new_protected:Npn \__draw_transform_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 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 \} }

49
\draw_transform_rotate:n
__draw_transform_rotate:n
__draw_transform_rotate:f
__draw_transform_rotate:mm
__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 }
\__draw_transform_matrix:nnnn {#1} {#2} { -#2 } { #1 } 
\cs_generate_variant:Nn \__draw_transform_rotate:n { ff }

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

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

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, 950, 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, 1118, 1120, 1129, 1134, 1161, 1166, 1558, 1564, 1621  
\cs_new_eq:NN  1668  
\cs_set_eq:Nn  62  
\cs_set_wd:N  64  

51
\l_draw_zvec_y_dim  

\draw commands:
\draw_begin:  
\draw_box_use:Nn  
\draw_cap:  
\draw_cap_round:  
\draw_closepath:  
\draw_curveto:nnnn  
\draw_dash_pattern:nn  
\draw_end:  
\draw_evenodd_rule:  
\draw_join_bevel:  
\draw_join_miter:  
\draw_join_round:  
\draw_lineto:nn  
\draw_linewidth:n  
\draw_miterlimit:n  
\draw_moveto:nn  
\draw_nonzero_rule:  
\draw_rectangle:nnnn  
\draw_scope_begin:  
\draw_scope_end:  

\error commands:
\exp_after:wN  
\exp_args:Nf  
\exp_args:Nfff  
\exp_args:Nffff  
\exp_args:NNNV  
\exp_not:N  
\c_one_fp  
\c_zero_fp  

\group commands:
\group_begin:  
\group_end:  

\hbox commands:
\hbox_gset:Nw  
\hbox_gset_end:  
\hbox_set:Nn  
\hbox_set:Nw  

\int commands:
\int_gincr:N  
\int_if_odd:nTF  
\int_new:N  

\mode commands:
\mode_leave_vertical:  

\msg commands:
\msg_error:nnn  
\msg_new:nnn  
\msg_new:nnnn  

\pgf commands:
\pgfextractx  
\pgfextracty  
\pgfgetlastxy  

---

\end  
\ERROR  

\exp commands:
\exp_after:wN  
\exp_args:Nf  
\exp_args:Nfff  
\exp_args:Nffff  
\exp_args:NNNV  
\exp_not:N  

\fp commands:
\fp_compare:nNTF  
\fp_compare_p:nN  
\fp_const:Nn  
\fp_eval:n  

\msg commands:
\msg_error:nnn  
\msg_new:nnn  

\pgf commands:
\pgfextractx  
\pgfextracty  
\pgfgetlastxy  

---

\end