The l3draw package
Core drawing support

The LaTeX3 Project*

Released 2018/02/21

1 l3draw documentation

The l3draw package provides a set of tools for creating (vector) drawings in expl3. It is heavily inspired by the pgf layer of the Ti*k*Z system, with many of the interfaces having the same form. However, the code provided here is build entirely on core expl3 ideas and uses the LaTeX3 FPU for numerical support.

Numerical expressions in l3draw are handled as floating point expressions, unless otherwise noted. This means that they may contain or omit explicit units. Where units are omitted, they will automatically be taken as given in (TeX) points.

The code here is *highly* experimental.

1.1 Drawings

`\draw_begin:`
`\draw_end:`

```
\draw_begin:
...
\draw_end:
```

Each drawing should be created within a `\draw_begin:`/`\draw_end:` function pair. The `begin` function sets up a number of key data structures for the rest of the functions here: unless otherwise specified, use of `\draw_...` functions outside of this "environment" is *not supported*.

The drawing created within the environment will be inserted into the typesetting stream by the `\draw_end:` function, which will switch out of vertical mode if required.

1.2 Graphics state

Within the drawing environment, a number of functions control how drawings will appear. Note that these all apply *globally*, though some are rest at the start of each drawing (`\draw_begin:`).

`\g_draw_linewidth_default_dim`

The default value of the linewidth for stokes, set at the start of every drawing (`\draw_begin:`).

*E-mail: latex-team@latex-project.org

`\draw_linewidth:n`
`\draw_inner_linewidth:n`

`\draw_linewidth:n {⟨width⟩}`

Sets the width to be used for stroking to the ⟨*width*⟩ (an ⟨*fp expr*⟩).

`\draw_nonzero_rule:`
`\draw_evenodd_rule:`

`\draw_nonzero_rule:`

Active either the non-zero winding number or the even-odd rule, respectively, for determining what is inside a fill or clip area. For technical reasons, these command are not influenced by scoping and apply on an ongoing basis.

`\draw_cap_butt:`
`\draw_cap_rectangle:`
`\draw_cap_round:`

`\draw_cap_butt:`

Sets the style of terminal stroke position to one of butt, rectangle or round.

`\draw_join_bevel:`
`\draw_join_miter:`
`\draw_join_round:`

`\draw_cap_butt:`

Sets the style of stroke joins to one of bevel, miter or round.

`\draw_miterlimit:n`

`\draw_miterlimit:n {⟨limit⟩}`

Sets the miter ⟨*limit*⟩ of lines joined as a miter, as described in the PDF and PostScript manuals. The ⟨*limit*⟩ is an ⟨*fp expr*⟩.

1.3 Points

Functions supporting the calculation of points (co-ordinates) are expandable and may be used outside of the drawing environment. When used in this way, they all yield a co-ordinate tuple, for example

```
\tl_set:Nx \l_tmpa_tl { \draw_point:nn { 1 } { 2 } }
\tl_show:N \l_tmpa_tl
```

gives

```
> \l_tmpa_tl=1pt,2pt.
<recently read> }
```

This output form is then suitable as *input* for subsequent point calculations, *i.e.* where a ⟨*point*⟩ is required it may be given as a tuple. This *may* include units and surrounding parentheses, for example

```
1,2
(1,2)
1cm,3pt
(1pt,2cm)
2 * sind(30), 2^4in
```

are all valid input forms. Notice that each part of the tuple may itself be a float point expression.

Point co-ordinates are relative to the canvas axes, but can be transformed by `\draw_point_transform:n`. These manipulation is applied by many higher-level functions, for example path construction, and allows parts of a drawing to be rotated, scaled or skewed. This occurs before writing any data to the driver, and so such manipulations are tracked by the drawing mechanisms. See `\driver_draw_transformcm:nnnnnn` for driver-level manipulation of the canvas axes themselves.

Notice that in contrast to `pgf` it is possible to give the positions of points *directly*.

1.3.1 Basic point functions

\draw_point:nn ⋆ `\draw_point:nn {⟨x⟩} {⟨y⟩}`

Gives the co-ordinates of the point at ⟨*x*⟩ and ⟨*y*⟩, both of which are ⟨*fp expr*⟩.

\draw_point_polar:nn ⋆ `\draw_point_polar:nn {⟨angle⟩} {⟨radius⟩}`
\draw_point_polar:nnn ⋆ `\draw_point_polar:nnn {⟨angle⟩} {⟨radius-a⟩} {⟨radius-b⟩}`

Gives the co-ordinates of the point at ⟨*angle*⟩ (an ⟨*fp expr*⟩ in *degrees*) and ⟨*radius*⟩. The three-argument version accepts two radii of different lengths.

Note the interface here is somewhat different from that in pgf: the one- and two-radii versions in l3draw use separate functions, whilst in pgf they use the same function and a keyword.

\draw_point_add:nn ⋆ `\draw_point_add:nn {⟨point1⟩} {⟨point2⟩}`

Adds ⟨*point1*⟩ to ⟨*point2*⟩.

\draw_point_diff:nn ⋆ `\draw_point_diff:nn {⟨point1⟩} {⟨point2⟩}`

Subtracts ⟨*point1*⟩ from ⟨*point2*⟩.

\draw_point_scale:nn ⋆ `\draw_point_scale:nn {⟨scale⟩} {⟨point⟩}`

Scales the ⟨*point*⟩ by the ⟨*scale*⟩ (an ⟨*fp expr*⟩).

\draw_point_unit_vector:n ⋆ `\draw_point_unit_vector:n {⟨point⟩}`

Expands to the co-ordinates of a unit vector joining the ⟨*point*⟩ with the origin.

\draw_point_transform:n ⋆ `\draw_point_transform:n {⟨point⟩}`

Evaluates the position of the ⟨*point*⟩ subject to the current transformation matrix. This operation is applied automatically by most higher-level functions (*e.g.* path manipulations).

1.3.2 Points on a vector basis

As well as giving explicit values, it is possible to describe points in terms of underlying direction vectors. The latter are initially co-incident with the standard Cartesian axes, but may be altered by the user.

\draw_xvec_set:n `\draw_xvec_set:n {⟨point⟩}`
\draw_yvec_set:n
\draw_zvec_set:n

Defines the appropriate base vector to point toward the ⟨*point*⟩ on the canvas. The standard settings for the x- and y-vectors are 1 cm along the relevant canvas axis, whilst for the z-vector an appropriate direction is taken.

\draw_point_vec:nn ⋆ `\draw_point_vec:nn {⟨xscale⟩} {⟨yscale⟩}`
\draw_point_vec:nnn ⋆ `\draw_point_vec:nnn {⟨xscale⟩} {⟨yscale⟩} {⟨zscale⟩}`

Expands to the co-ordinate of the point at ⟨*xscale*⟩ times the x-vector and ⟨*yscale*⟩ times the y-vector. The three-argument version extends this to include the z-vector.

\draw_point_vec_polar:nn ⋆ \draw_point_vec_polar:nn {⟨angle⟩} {⟨radius⟩}
\draw_point_vec_polar:nnn ⋆ \draw_point_vec_polar:nnn {⟨angle⟩} {⟨radius-a⟩} {⟨radius-b⟩}

Gives the co-ordinates of the point at ⟨*angle*⟩ (an ⟨*fp expr*⟩ in *degrees*) and ⟨*radius*⟩, relative to the prevailing x- and y-vectors. The three-argument version accepts two radii of different lengths.

Note the interface here is somewhat different from that in pgf: the one- and two-radii versions in l3draw use separate functions, whilst in pgf they use the same function and a keyword.

1.3.3 Intersections

\draw_point_intersect_lines:nnnn ⋆ \draw_point_intersect_lines:nnnn {⟨*point1*⟩} {⟨*point2*⟩} {⟨*point3*⟩} {⟨*point4*⟩}

Evaluates the point at the intersection of one line, joining ⟨*point1*⟩ and ⟨*point2*⟩, and a second line joining ⟨*point3*⟩ and ⟨*point4*⟩. If the lines do not intersect, or are coincident, and error will occur.

\draw_point_intersect_circles:nnnn ⋆ \draw_point_intersect_circles:nnnnn {⟨*center1*⟩} {⟨*radius1*⟩} {⟨*center2*⟩} {⟨*radius2*⟩} {⟨*root*⟩}

Evaluates the point at the intersection of one circle with ⟨*center1*⟩ and ⟨*radius1*⟩, and a second circle with ⟨*center2*⟩ and ⟨*radius2*⟩. If the circles do not intersect, or are coincident, and error will occur.

Note the interface here has a different argument ordering from that in pgf, which has the two centers then the two radii.

1.3.4 Interpolations

\draw_point_interpolate_line:nnn ⋆ \draw_point_interpolate_line:nnn {⟨*part*⟩} {⟨*point1*⟩} {⟨*point2*⟩}

Expands to the point which is ⟨*part*⟩ way along the line joining ⟨*point1*⟩ and ⟨*point2*⟩. The ⟨*part*⟩ may be an interpolation or an extrapolation, and is a floating point value expressing a percentage along the line, *e.g.* a value of `0.5` would be half-way between the two points.

\draw_point_interpolate_distance:nnn ⋆ \draw_point_interpolate_distance:nnn {⟨*distance*⟩} {⟨*point expr1*⟩} {⟨*point expr2*⟩}

Expands to the point which is ⟨*distance*⟩ way along the line joining ⟨*point1*⟩ and ⟨*point2*⟩. The ⟨*distance*⟩ may be an interpolation or an extrapolation.

\draw_point_interpolate_curve:nnnnnn ⋆ \draw_point_interpolate_curve:nnnnnn {⟨*part*⟩} {⟨*start*⟩} {⟨*control1*⟩} {⟨*control2*⟩} {⟨*end*⟩}

Expands to the point which is ⟨*part*⟩ way along the curve between ⟨*start*⟩ and ⟨*end*⟩ and defined by ⟨*control1*⟩ and ⟨*control2*⟩. The ⟨*part*⟩ may be an interpolation or an extrapolation, and is a floating point value expressing a percentage along the curve, *e.g.* a value of `0.5` would be half-way along the curve.

1.4 Paths

Paths are constructed by combining one or more operations before applying one or more actions. Thus until a path is "used", it may be manipulated or indeed discarded entirely. Only one path is active at any one time, and the path is *not* affected by TEX grouping.

`\draw_path_corner_arc:n` `\draw_path_corner_arc:n {⟨length⟩}`

Sets the degree of rounding applied to corners in a path: if the ⟨*length*⟩ is `0pt` then no rounding applies. The value of the ⟨*length*⟩ is local to the current TEX group. *At present, corner arcs are not activated in the code.*

`\draw_path_moveto:n` `\draw_path_moveto:n {⟨point⟩}`

Moves the reference point of the path to the ⟨*point*⟩, but will not join this to any previous point.

`\draw_path_lineto:n` `\draw_path_lineto:n {⟨point⟩}`

Joins the current path to the ⟨*point*⟩ with a straight line.

`\draw_path_curveto:nnn` `\draw_path_curveto:nnn {⟨control1⟩} {⟨control2⟩} {⟨end⟩}`

Joins the current path to the ⟨*end*⟩ with a curved line defined by cubic BÃľzier points ⟨*control1*⟩ and ⟨*control2*⟩.

`\draw_path_curveto:nn` `\draw_path_curveto:nn {⟨control⟩} {⟨end⟩}`

Joins the current path to the ⟨*end*⟩ with a curved line defined by quadratic BÃľzier point ⟨*control*⟩.

`\draw_path_arc:nnn` `\draw_path_arc:nnn {⟨angle1⟩} {⟨angle2⟩} {⟨radius⟩}`
`\draw_path_arc:nnnn` `\draw_path_arc:nnnn {⟨angle1⟩} {⟨angle2⟩} {⟨radius-a⟩} {⟨radius-b⟩}`

Joins the current path with an arc between ⟨*angle1*⟩ and ⟨*angle2*⟩ and of ⟨*radius*⟩. The four-argument version accepts two radii of different lengths.

Note the interface here has a different argument ordering from that in pgf, which has the two centers then the two radii.

`\draw_path_arc_axes:nnnn` `\draw_path_arc_axes:nnn {⟨angle1⟩} {⟨angle2⟩} {⟨vector1⟩} {⟨vector2⟩}`

Appends the portion of an ellipse from ⟨*angle1*⟩ to ⟨*angle2*⟩ of an ellipse with axes along ⟨*vector1*⟩ and ⟨*vector2*⟩ to the current path.

`\draw_path_ellipse:nnnn` `\draw_path_ellipse:nnn {⟨center⟩} {⟨vector1⟩} {⟨vector2⟩}`

Appends an ellipse at ⟨*center*⟩ with axes along ⟨*vector1*⟩ and ⟨*vector2*⟩ to the current path.

`\draw_path_circle:nn` `\draw_path_circle:nn {⟨center⟩} {⟨radius⟩}`

Appends a circle of ⟨*radius*⟩ at ⟨*center*⟩ to the current path.

\draw_path_rectangle:nn
\draw_path_rectangle_corners:nn

\draw_path_rectangle:nn {⟨*lower-left*⟩} {⟨*displacement*⟩}
\draw_path_rectangle_corners:nn {⟨*lower-left*⟩} {⟨*top-right*⟩}

Appends a rectangle starting at ⟨*lower-left*⟩ to the current path, with the size of the rectangle determined either by a ⟨*displacement*⟩ or the position of the ⟨*top-right*⟩.

\draw_path_grid:nnnn

\draw_path_grid:nnnn {⟨*xspace*⟩} {⟨*yspace*⟩} {⟨*lower-left*⟩} {⟨*upper-right*⟩}

Constructs a grid of ⟨*xspace*⟩ and ⟨*yspace*⟩ from the ⟨*lower-left*⟩ to the ⟨*upper-right*⟩, and appends this to the current path.

\draw_path_close:

\draw_path_close:

Closes the current part of the path by appending a straight line from the current point to the starting point of the path.

\draw_path_use:n
\draw_path_use_clear:n

\draw_path_use:n {⟨*action(s)*⟩}

Inserts the current path, carrying out one ore more possible ⟨*actions*⟩ (a comma list):

- `clear` Resets the path to empty
- `clip` Clips any content outside of the path
- `draw`
- `fill` Fills the interior of the path with the current file color
- `stroke` Draws a line along the current path

1.5 Color

\draw_color:n
\draw_fill:n
\draw_stroke:n

\draw_color:n {⟨*color expression*⟩}

Evaluates the ⟨*color expression*⟩ as described for l3color.

1.6 Transformations

Points are normally used unchanged relative to the canvas axes. This can be modified by applying a transformation matrix. The canvas axes themselves may be adjusted using `\driver_draw_transformcm:nnnnnn`: note that this is transparent to the drawing code so is not tracked.

\draw_transform_reset:

\draw_transform_reset:

Resets the matrix to the identity.

\draw_transform_concat:nnnnn

\draw_transform_concat:nnnnn
{⟨a⟩} {⟨*b*⟩} {⟨*c*⟩} {⟨*d*⟩} {⟨*vector*⟩}

Appends the given transformation to the currently-active one. The transformation is made up of a matrix ⟨*a*⟩, ⟨*b*⟩, ⟨*c*⟩ and ⟨*d*⟩, and a shift by the ⟨*vector*⟩.

`\draw_transform:nnnnn`

`\draw_transform:nnnnn`
`{⟨a⟩} {⟨b⟩} {⟨c⟩} {⟨d⟩} {⟨vector⟩}`

Applies the transformation matrix specified, over-writing any existing matrix. The transformation is made up of a matrix $\langle a \rangle$, $\langle b \rangle$, $\langle c \rangle$ and $\langle d \rangle$, and a shift by the $\langle vector \rangle$.

`\draw_transform_triangle:nnn`

`\draw_transform_triangle:nnn`
`{⟨origin⟩} {⟨point1⟩} {⟨point2⟩}`

Applies a transformation such that the co-ordinates $(0,0)$, $(1,0)$ and $(0,1)$ are given by the $\langle origin \rangle$, $\langle point1 \rangle$ and $\langle point2 \rangle$, respectively.

`\draw_transform_invert:`

`\draw_transform_invert:`

Inverts the current transformation matrix and reverses the current shift vector.

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.

D

draw commands:
\draw_begin: 1, 1
\draw_cap_butt: 2, 2
\draw_cap_rectangle: 2
\draw_cap_round: 2
\draw_color:n 6
\draw_end: 1
\draw_evenodd_rule: 2
\draw_fill:n 6
\draw_inner_linewidth:n 2
\draw_join_bevel: 2
\draw_join_miter: 2
\draw_join_round: 2
\draw_linewidth:n 2
\g_draw_linewidth_default_dim 1
\draw_miterlimit:n 2
\draw_nonzero_rule: 2
\draw_path_arc:nnn 5
\draw_path_arc:nnnn 5
\draw_path_arc_axes:nnn 5
\draw_path_arc_axes:nnnn 5
\draw_path_circle:nn 5
\draw_path_close: 6
\draw_path_corner_arc:n 5
\draw_path_curveto:nn 5
\draw_path_curveto:nnn 5
\draw_path_ellipse:nnn 5
\draw_path_ellipse:nnnn 5
\draw_path_grid:nnnn 6
\draw_path_lineto:n 5
\draw_path_moveto:n 5
\draw_path_rectangle:nn 6
\draw_path_rectangle_corners:nn . . 6
\draw_path_use:n 6
\draw_path_use_clear:n 6
\draw_point:nn 3
\draw_point_add:nn 3
\draw_point_diff:nn 3
\draw_point_interpolate_curve:nnnnnn 4
\draw_point_interpolate_distance:nnn 4
\draw_point_interpolate_line:nnn . 4
\draw_point_intersect_circles:nnnn 4
\draw_point_intersect_circles:nnnnn 4
\draw_point_intersect_lines:nnnn . 4
\draw_point_polar:nn 3
\draw_point_polar:nnn 3
\draw_point_scale:nn 3
\draw_point_transform:n 2, 3
\draw_point_unit_vector:n 3
\draw_point_vec:nn 3
\draw_point_vec:nnn 3
\draw_point_vec_polar:nn 4
\draw_point_vec_polar:nnn 4
\draw_stroke:n 6
\draw_transform:nnnnn 7
\draw_transform_concat:nnnnn 6
\draw_transform_invert: 7
\draw_transform_reset: 6
\draw_transform_triangle:nnn 7

\draw_xvec_set:n 3
\draw_yvec_set:n 3
\draw_zvec_set:n 3
driver commands:
\driver_draw_transformcm:nnnnnn 2, 6