

The `braids` package: codebase

Andrew Stacey
loopspace@mathforge.org

v2.3 from 2024/01/09

1 Introduction

This is a package for drawing braid diagrams using PGF/TikZ. Its inspiration was a question and answer on the website <http://tex.stackexchange.com>.

2 History

- v1.0 First public release.
- v1.1 Added ability to configure the gap size, the control points, and the “nudge”. Added ability to add labels to strands between crossings.
- v2 Reimplemented as TikZ library rather than a standalone package.

3 Implementation

Issue a notice that this is a deprecated version of the `braids` package.

```
1 \NeedsTeXFormat{LaTeX2e}
2 \ProvidesPackage{braids}[2011/10/18 v1.1 Tikz/PGF commands for drawing braid diagrams (depre
3 \newif\if@braids@warning@
4 \@braids@warning@true
5 \DeclareOption{nowarning}{%
6 \@braids@warning@false
7 }
8
9 \ProcessOptions
10
11 \if@braids@warning@
12 \PackageWarning{braids}{%
13   This package is frozen at v1.1 from 2011/10/18 and has been reimplemented as a TikZ libran
14 }%
15 \fi
```

`\ge@addto@macro` This is an expanded version of `\g@addto@macro`. Namely, it adds the *expansion* of the second argument to the first.

```
16 \long\def\ge@addto@macro#1#2{%
17   \begingroup
18   \toks@\expandafter\expandafter\expandafter{\expandafter#1#2}%
```

```

19 \xdef#1{\the\toks@}%
20 \endgroup}

```

(End of definition for \ge@addto@macro.)

`\braid` This is the user command. We start a group to ensure that all our assignments are local, and then call our initialisation code. The optional argument is for any keys to be set.

```

21 \newcommand{\braid}[1] [] {%
22 \begingroup
23 \braid@start{#1}}

```

(End of definition for \braid. This function is documented on page ??.)

`\braid@process` This is the token swallower. This takes the next token on the braid specification and passes it to the handler command (in the macro `\braid@token`) which decides what to do next. (Incidentally, the code here is heavily influenced by TikZ. That's probably not very surprising.)

```

24 \def\braid@process{%
25 \afterassignment\braid@handle\let\braid@token=%
26 }

```

(End of definition for \braid@process.)

`\braid@process@start` This is a variant of `\braid@process` which is used at the start where we might have a few extra bits and pieces before the braid itself starts. Specifically, we test for the `at` and `(name)` possibilities.

```

27 \def\braid@process@start{%
28 \afterassignment\braid@handle@start\let\braid@token=%
29 }

```

(End of definition for \braid@process@start.)

`\braid@handle@start` This is the handler in use at the start. It looks for the tokens `a` or `(` which (might) signal the start of an `at` (`coordinate`) or `(name)`. If we get anything else (modulo spaces) we decide that we've reached the end of the initialisation stuff and it is time to get started on the braid itself.

```

30 \def\braid@handle@start{%
31 \let\braid@next=\braid@handle
32 \ifx\braid@token a

```

We got an `a` so we might have an `at` (`coordinate`)

```

33 \let\braid@next=\braid@maybe@locate
34 \else
35 \ifx\braid@token(

```

We got an `(` so we have a `name`

```

36 \iffalse)\fi %Indentation hack!
37 \let\braid@next=\braid@assign@name
38 \else
39 \ifx\braid@token\sptoken

```

Space; boring, redo from start

```
40 \let\braid@next=\braid@process@start
41 \fi
42 \fi
43 \fi
44 \braid@next%
45 }
```

(End of definition for `\braid@handle@start`.)

`\braid@handle` This is the main handler for parsing the braid word. It decides what action to take depending on what the token is. We have to be a bit careful with catcodes, some packages set `;` and `|` to be active. We should probably also be careful with `^` and `_`.

```
46 \let\braid@semicolon=;
47 \let\braid@bar=|
48 \def\braid@handle{%
49 \let\braid@next=\braid@process
```

Start by checking our catcodes to see what we should check against

```
50 \ifnum\the\catcode'\;=\active
51 \expandafter\let\expandafter\braid@semicolon\tikz@activesemicolon
52 \fi
53 \ifnum\the\catcode'\|=\active
54 \expandafter\let\expandafter\braid@bar\tikz@activebar
55 \fi
56 \ifx\braid@token\braid@semicolon
```

Semicolon, means that we're done reading our braid. It's time to render it.

```
57 \let\braid@next=\braid@render
58 \else
59 \ifx\braid@token^
```

Superscript character, the next token tells us whether it's an over-crossing or an under-crossing.

```
60 \let\braid@next=\braid@sup
61 \else
62 \ifx\braid@token_
```

Subscript character, the next token tells us which strands cross.

```
63 \let\braid@next=\braid@sub
64 \else
65 \ifx\braid@token-
```

Hyphen, this is so that we can have more than one crossing on the same level.

```
66 \braid@increase@levelfalse
67 \else
68 \ifx\braid@token1%
```

1: this means the "identity" crossing, so no crossing here. Increase the level, unless overridden, and add to the label.

```
69 \ifbraid@increase@level
70 \stepcounter{braid@level}
71 \fi
72 \braid@increase@leveltrue
73 \ge@addto@macro\braid@label{\braid@token}%
74 \else
75 \ifx\braid@token[%
```

Open bracket, this means we have some more options to process.

```
76 \let\braid@next=\braid@process@options
77 \else
78 \ifx\braid@token\braid@bar
```

Bar, this tells us that we want a “floor” at this point.

```
79 \edef\braid@tmp{\expandafter\the\value\braid@level}%
80 \ge@addto@macro\braid@floors\braid@tmp%
81 \else
82 \ifx\braid@token\bgroup
```

Begin group, which we reinterpret as beginning a scope.

```
83 \braid@beginscope
84 \else
85 \ifx\braid@token\egroup
```

End group, which ends the scope

```
86 \braid@endscope
87 \else
88 \ifx\braid@token\braid@olabel@strand
89 \let\braid@next=\braid@olabel@strand
90 \else
91 \ifx\braid@token\braid@clabel@strand
92 \let\braid@next=\braid@clabel@strand
93 \else
```

Otherwise, we add the token to the braid label.

```
94 \ge@addto@macro\braid@label{\braid@token}%
95 \fi
96 \fi
97 \fi
98 \fi
99 \fi
100 \fi
101 \fi
102 \fi
103 \fi
104 \fi
105 \fi
106 \braid@next%
107 }
```

(End of definition for `\braid@handle`.)

`\braid@maybe@locate` If we got an a token in the `\braid@handle@start` then it *might* mean we’re looking at `at` (coordinate) or it might mean that the user has decided to use `a` as the braid parameter. So we examine the next token for a `t`.

```
108 \def\braid@maybe@locate{%
109 \afterassignment\braid@@maybe@locate\let\braid@token=%
110 }
```

(End of definition for `\braid@maybe@locate`.)

`\braid@@maybe@locate` This is where we test for `t` and act appropriately.

```
111 \def\braid@@maybe@locate{%
112   \let\braid@next=\braid@handle
113   \ifx\braid@token t
114     \let\braid@next=\braid@find@location
115   \fi
116   \braid@next%
117 }
```

(End of definition for \braid@@maybe@locate.)

`\braid@find@location` This macro starts us looking for a coordinate.

```
118 \def\braid@find@location{%
119   \afterassignment\braid@@find@location\let\braid@token=%
120 }
```

(End of definition for \braid@find@location.)

`\braid@@find@location` This is the test for the start of a coordinate. If we get a `(` that means we've reached the coordinate. A space means "carry on". Anything else is a (non-fatal) error.

```
121 \def\braid@@find@location{%
122   \let\braid@next=\braid@location@error
123   \ifx\braid@token(
124     \let\braid@next=\braid@locate
125   \else
126     \ifx\braid@token@
127       \let\braid@next=\braid@find@location
128     \fi
129   \fi
130   \braid@next%
131 }
```

(End of definition for \braid@@find@location.)

`\braid@location@error` This is our error message for not getting a location.

```
132 \def\braid@location@error{%
133   \PackageWarning{braids}{Could not figure out location for braid}%
134   \braid@process@start%
135 }
```

(End of definition for \braid@location@error.)

`\braid@locate` If we reached a `(` when looking for a coordinate, everything up to the next `)` is that coordinate. Then we parse the coordinate and call the relocation macro.

```
136 \def\braid@locate#1){%
137   \tikz@scan@one@point\braid@relocate(#1)%
138 }
```

(End of definition for \braid@locate.)

`\braid@relocate` This is the macro that actually does the relocation.

```
139 \def\braid@relocate#1){%
140   #1\relax
141   \advance\pgf@x by -\braid@width
142   \pgftransformshift{\pgfqpoint{\pgf@x}{\pgf@y}}
143   \braid@process@start%
144 }
```

(End of definition for `\braid@relocate`.)

`\braid@assign@name` This macro saves our name.

```
145 \def\braid@assign@name#1){%
146   \def\braid@name{#1}%
147   \braid@process@start%
148 }
```

(End of definition for `\braid@assign@name`.)

`\braid@process@options` The intention of this macro is to allow setting of style options mid-braid. (At present, this wouldn't make a lot of sense.)

```
149 \def\braid@process@options#1){%
150   \tikzset{#1}%
151   \braid@process%
152 }
```

(End of definition for `\braid@process@options`.)

The next macros handle the actual braid elements. Everything has to have a subscript, but the superscript is optional and can come before or after the subscript.

`\braid@sup` This handles braid elements of the form a^{-1}_2 .

```
153 \def\braid@sup#1_#2{%
154   \g@addto@macro\braid@label_{_#2}^{#1}%
155   \braid@add@crossing{#2}{#1}%
156 }
```

(End of definition for `\braid@sup`.)

`\braid@sub`

```
157 % This handles braid elements of the form \Verb+a_1+ or \Verb+a_1^{-1}+.
158 \def\braid@sub#1{%
159   \@ifnextchar^{\braid@@sub{#1}}%
160   {\g@addto@macro\braid@label_{_#1}}\braid@add@crossing{#1}{1}}%
161 }
```

(End of definition for `\braid@sub`.)

`\braid@@sub` Helper macro for `\braid@sub`.

```
162 \def\braid@@sub#1^#2{%
163   \g@addto@macro\braid@label_{_#1}^{#2}}%
164   \braid@add@crossing{#1}{#2}%
165 }
```

(End of definition for `\braid@@sub`.)

`\braid@ne` Remember what 1 looks like for testing against.

```
166 \def\braid@ne{1}
```

(End of definition for `\braid@ne`.)

`\braid@add@crossing` This is the macro which adds the crossing to the current list of strands. The strands are stored as *soft paths* (see the TikZ/PGF documentation). So this selects the right strands and then extends them according to the crossing type.

```
167 \def\braid@add@crossing#1#2{%
```

Our crossing type, which is #2, is one of 1 or -1. Our strands are #1 and #1+1.

```

168 \edef\braid@crossing@type{#2}%
169 \edef\braid@this@strand{#1}%
170 \pgfmathtruncatemacro{\braid@next@strand}{#1+1}

```

Increment the level counter, if requested. The controls whether the crossing is on the same level as the previous one or is one level further on.

```

171 \ifbraid@increase@level
172 \stepcounter{braid@level}
173 \fi

```

Default is to request increment so we set it for next time.

```

174 \braid@increase@leveltrue

```

Now we figure out the coordinates of the crossing. (`\braid@tx`,`\braid@ty`) is the top-left corner (assuming the braid flows down the page). (`\braid@nx`,`\braid@ny`) is the bottom-right corner (assuming the braid flows down the page). We start by setting (`\braid@tx`,`\braid@ty`) according to the level and strand number, then shift `\braid@ty` by `\braid@eh` which is the “edge height” (the little extra at the start and end of each strand). Then from these values, we set (`\braid@nx`,`\braid@ny`) by adding on the appropriate amount. The heights `\braid@cy` and `\braid@dy` are for the control points for the strands as they cross. They’re actually the same height, but using two gives us the possibility of changing them independently in a later version of this package. Lastly, we bring `\braid@ty` and `\braid@ny` towards each other just a little so that there is “clear water” between subsequent crossings (makes it look a bit better if the same strand is used in subsequent crossings).

```

175 \braid@tx=\braid@this@strand\braid@width
176 \braid@ty=\value{braid@level}\braid@height
177 \advance\braid@ty by \braid@eh
178 \braid@nx=\braid@tx
179 \braid@ny=\braid@ty
180 \advance\braid@nx by \braid@width
181 \advance\braid@ny by \braid@height
182 \advance\braid@ty by \braid@nf\braid@height
183 \advance\braid@ny by -\braid@nf\braid@height
184 \braid@cy=\braid@ty
185 \braid@dy=\braid@ny
186 \advance\braid@cy by \braid@cf\braid@height
187 \advance\braid@dy by -\braid@cf\braid@height

```

Now we try to find a starting point for the strand ending here. We might not have used this strand before, so it might not exist.

```

188 \expandafter\let\expandafter\braid@this@path@origin%
189 \csname braid@strand@\braid@this@strand @origin\endcsname

```

If we haven’t seen this strand before, that one will be `\relax`.

```

190 \ifx\braid@this@path@origin\relax

```

Haven’t seen this strand before, so initialise it. Record the initial position of the strand.

```

191 \let\braid@this@path@origin\braid@this@strand

```

Start a new soft path.

```

192 \pgfsyssoftpath@setcurrentpath{\@empty}
193 \pgfpathmoveto{\pgfpoint{\braid@tx}{0pt}}

```

Save the path as `\braid@this@path`.

```
194 \pgfsyssoftpath@getcurrentpath{\braid@this@path}
195 \else
```

We have seen this before, so we simply copy the associated path in to `\braid@this@path`.

```
196 \expandafter\let\expandafter\braid@this@path%
197 \csname braid@strand@\braid@this@path@origin\endcsname
198 \fi
```

Now we do the same again with the other strand in the crossing.

```
199 \expandafter\let\expandafter\braid@next@path@origin%
200 \csname braid@strand@\braid@next@strand @origin\endcsname
201 \ifx\braid@next@path@origin\relax
202 \let\braid@next@path@origin\braid@next@strand
203 \pgfsyssoftpath@setcurrentpath{\@empty}
204 \pgfpathmoveto{\pgfpoint{\braid@nx}{0pt}}
205 \pgfsyssoftpath@getcurrentpath{\braid@next@path}
206 \else
207 \expandafter\let\expandafter\braid@next@path%
208 \csname braid@strand@\braid@next@path@origin\endcsname
209 \fi
```

Now that we have the paths for our two strands, we extend them to the next level. We start by selecting the first path.

```
210 \pgfsyssoftpath@setcurrentpath{\braid@this@path}
```

Draw a line down to the current level, note that this line is always non-trivial since we shifted the corners of the crossing in a little.

```
211 \pgfpathlineto{\pgfpoint{\braid@tx}{\braid@ty}}
```

Curve across to the next position. Depending on the crossing type, we either have a single curve or we have to break it in two. Our gap is to interrupt at times determined by the gap key.

```
212 \pgfmathsetmacro{\braid@gst}{0.5 - \pgfkeysvalueof{/pgf/braid/gap}}%
213 \pgfmathsetmacro{\braid@gend}{0.5 + \pgfkeysvalueof{/pgf/braid/gap}}%
214 \ifx\braid@crossing@type\braid@over@cross
```

We're on the overpass, so just one curve needed.

```
215 \pgfpathcurveto{\pgfpoint{\braid@tx}{\braid@cy}}%
216 {\pgfpoint{\braid@nx}{\braid@dy}}%
217 {\pgfpoint{\braid@nx}{\braid@ny}}
218 \else
```

We're on the underpass, so we need to interrupt our path to allow the other curve to go past.

```
219 \pgfpathcurvebetweentimecontinue{0}{\braid@gst}%
220 {\pgfpoint{\braid@tx}{\braid@ty}}%
221 {\pgfpoint{\braid@tx}{\braid@cy}}%
222 {\pgfpoint{\braid@nx}{\braid@dy}}%
223 {\pgfpoint{\braid@nx}{\braid@ny}}%
224 \pgfpathcurvebetweentime{\braid@gend}{1}%
225 {\pgfpoint{\braid@tx}{\braid@ty}}%
226 {\pgfpoint{\braid@tx}{\braid@cy}}%
227 {\pgfpoint{\braid@nx}{\braid@dy}}%
228 {\pgfpoint{\braid@nx}{\braid@ny}}
229 \fi
```

We're done with this path, so now we save it.

```
230 \pgfsyssoftpath@getcurrentpath{\braid@this@path}
```

Now do the same with the second path.

```
231 \pgfsyssoftpath@setcurrentpath{\braid@next@path}
232 \pgfpathlineto{\pgfqpoint{\braid@nx}{\braid@ty}}
233 \ifx\braid@crossing@type\braid@over@cross
234 \pgfpathcurvebetweentimecontinue{0}{\braid@gst}%
235 {\pgfqpoint{\braid@nx}{\braid@ty}}%
236 {\pgfqpoint{\braid@nx}{\braid@cy}}%
237 {\pgfqpoint{\braid@tx}{\braid@dy}}%
238 {\pgfqpoint{\braid@tx}{\braid@ny}}
239 \pgfpathcurvebetweentime{\braid@gend}{1}%
240 {\pgfqpoint{\braid@nx}{\braid@ty}}%
241 {\pgfqpoint{\braid@nx}{\braid@cy}}%
242 {\pgfqpoint{\braid@tx}{\braid@dy}}%
243 {\pgfqpoint{\braid@tx}{\braid@ny}}
244 \else
245 \pgfpathcurveto{\pgfqpoint{\braid@nx}{\braid@cy}}%
246 {\pgfqpoint{\braid@tx}{\braid@dy}}%
247 {\pgfqpoint{\braid@tx}{\braid@ny}}
248 \fi
249 \pgfsyssoftpath@getcurrentpath{\braid@next@path}
```

Now save the paths to their proper macros again.

```
250 \expandafter\let%
251 \csname braid@strand@\braid@this@path@origin \endcsname%
252 \braid@this@path
253 \expandafter\let%
254 \csname braid@strand@\braid@next@path@origin \endcsname%
255 \braid@next@path
```

Now update the origins

```
256 \expandafter\let%
257 \csname braid@strand@\braid@this@strand @origin\endcsname%
258 \braid@next@path@origin
259 \expandafter\let%
260 \csname braid@strand@\braid@next@strand @origin\endcsname%
261 \braid@this@path@origin
```

increment the strand counter, if necessary

```
262 \pgfmathparse{\value{braid@strands} < \braid@next@strand ?
263   "\noexpand\setcounter{braid@strands}{\braid@next@strand}" : ""}
264 \pgfmathresult
```

And merrily go on our way with the next bit of the braid specification.

```
265 \braid@process%
266 }
```

(End of definition for `\braid@add@crossing`.)

`\braid@olabel@strand` This macro allows us to label a strand just before a crossing. The first argument is the strand number at that particular crossing and the second is the label. We also save the current height. This version takes the strand number as meaning the *original* ordering.

```
267 \newcommand{\braid@olabel@strand}[3] [] {%
268   \edef\braid@tmp{\the\value{braid@level}}%}
```

```

269 \expandafter\ifx\csname braid@strand@#2@origin\endcsname\relax
270 \g@addto@macro\braid@tmp{#2}%
271 \else
272 \edef\braid@tmpa{\csname braid@strand@#2@origin\endcsname}%
273 \g@addto@macro\braid@tmp{\braid@tmpa}%
274 \fi
275 \g@addto@macro\braid@tmp{#3-#1}%
276 \g@addto@macro{\braid@strand@labels}{\braid@tmp}%
277 \braid@process%
278 }

```

(End of definition for \braid@clabel@strand.)

`\braid@clabel@strand` This macro allows us to label a strand just before a crossing. The first argument is the strand number at that particular crossing and the second is the label. We also save the current height. This version takes the strand number as meaning the *current* ordering.

```

279 \newcommand{\braid@clabel@strand}[3][]{%
280 \edef\braid@tmp{\the\value{braid@level}}%
281 \g@addto@macro\braid@tmp{#2-#3-#1}%
282 \g@addto@macro{\braid@strand@labels}{\braid@tmp}%
283 \braid@process%
284 }

```

(End of definition for \braid@clabel@strand.)

`\braid@floors@trim` The list of floors, if given, will start with a superfluous comma. This removes it.

```

285 \def\braid@floors@trim,{}

```

(End of definition for \braid@floors@trim.)

`\braid@render@floor` This is the default rendering for floors: it draws a rectangle.

```

286 \def\braid@render@floor{%
287 \draw (\floorsx,\floorsy) rectangle (\floorex,\floorey);
288 }

```

(End of definition for \braid@render@floor.)

`\braid@render@strand@labels` This starts rendering the labels on the strands at the crossings.

```

289 \def\braid@render@strand@labels#1{%
290 \def\braid@tmp{#1}%
291 \ifx\braid@tmp\pgfutil@empty
292 \let\braid@next=\pgfutil@gobble
293 \else
294 \let\braid@next=\braid@render@strand@labels
295 \fi
296 \braid@next{#1}%
297 }

```

(End of definition for \braid@render@strand@labels.)

`\braid@render@strand@labels` This is the actual renderer.

```

298 \def\braid@render@strand@labels#1#2#3#4{%
299 \beginpgfgroup
300 \pgfscope
301 \let\tikz@options=\pgfutil@empty

```

```

302 \let\tikz@mode=\pgfutil@empty
303 \let\tik@transform=\pgfutil@empty
304 \let\tikz@fig@name=\pgfutil@empty
305 \tikzset{/pgf/braid/strand label,#4}%
306 \braid@nx=#2\braid@width
307 \braid@ny=#1\braid@height
308 \advance\braid@ny by \braid@eh
309 \advance\braid@ny by \braid@height
310 \pgftransformshift{\pgfqpoint{\braid@nx}{\braid@ny}}%
311 \tikz@options
312 \setbox\pgfnodeparttextbox=\hbox%
313 \bgroup%
314 \tikzset{every text node part/.try}%
315 \ifx\tikz@textopacity\pgfutil@empty%
316 \else%
317 \pgfsetfillopacity{\tikz@textopacity}%
318 \pgfsetstrokeopacity{\tikz@textopacity}%
319 \fi%
320 \pgfinterruptpicture%
321 \tikz@textfont%
322 \ifx\tikz@text@width\pgfutil@empty%
323 \else%
324 \begingroup%
325 \pgfmathsetlength{\pgf@x}{\tikz@text@width}%
326 \pgfutil@minipage[t]{\pgf@x}\leavevmode\hbox{}%
327 \tikz@text@action%
328 \fi%
329 \tikz@atbegin@node%
330 \bgroup%
331 \aftergroup\unskip%
332 \ifx\tikz@textcolor\pgfutil@empty%
333 \else%
334 \pgfutil@colorlet{.}{\tikz@textcolor}%
335 \fi%
336 \pgfsetcolor{.}%
337 \setbox\tikz@figbox=\box\pgfutil@voidb@x%
338 \tikz@uninstallcommands%
339 \tikz@halign@check%
340 \ignorespaces%
341 #3
342 \egroup
343 \tikz@atend@node%
344 \ifx\tikz@text@width\pgfutil@empty%
345 \else%
346 \pgfutil@endminipage%
347 \endgroup%
348 \fi%
349 \endpgfinterruptpicture%
350 \egroup%
351 \ifx\tikz@text@width\pgfutil@empty%
352 \else%
353 \pgfmathsetlength{\pgf@x}{\tikz@text@width}%
354 \wd\pgfnodeparttextbox=\pgf@x%
355 \fi%

```

```

356     \ifx\tikz@text@height\pgfutil@empty%
357     \else%
358         \pgfmathsetlength{\pgf@x}{\tikz@text@height}%
359         \ht\pgfnodeparttextbox=\pgf@x%
360     \fi%
361     \ifx\tikz@text@depth\pgfutil@empty%
362     \else%
363         \pgfmathsetlength{\pgf@x}{\tikz@text@depth}%
364         \dp\pgfnodeparttextbox=\pgf@x%
365     \fi%
366     \pgfmultipartnode{\tikz@shape}{\tikz@anchor}{\tikz@fig@name}{%
367         {\begingroup\tikz@finish}%
368     }%
369     \endpgfscope
370     \endgroup
371     \braid@render@strand@labels%
372 }

```

(End of definition for \braid@@render@strand@labels.)

\braid@render This is called at the end of the braid and it renders the braids and floors according to whatever has been built up up to now.

```

373 \def\braid@render{
Check for floors since we do them first.
374     \ifx\braid@floors\@empty
375     \else
Have some floors, start a scope and prepare to render them.
376     \pgfsys@beginscope
Clear the path (just to be sure).
377     \pgfsyssoftpath@setcurrentpath{\empty}
Trim the initial comma off the list of floors.
378     \edef\braid@floors{\expandafter\braid@floors@trim\braid@floors}
Initialise our horizontal coordinates.
379     \braid@tx=\braid@width
380     \advance\braid@tx by \braid@eh
381     \braid@nx=\value\braid@strands\braid@width
382     \advance\braid@nx by -\braid@eh
Loop over the list of floors.
383     \foreach \braid@f in \braid@floors {
384         \pgfsys@beginscope
Figure out the vertical coordinates for the current floor.
385         \braid@ty=\braid@f\braid@height
386         \advance\braid@ty by \braid@eh
387         \advance\braid@ty by \braid@height
388         \braid@ny=\braid@ty
389         \advance\braid@ny by \braid@height

```

Save the coordinates for use in the floor rendering macro.

```
390     \edef\floorsx{\the\braid@tx}
391     \edef\floorsy{\the\braid@ty}
392     \edef\floorex{\the\braid@nx}
393     \edef\floorey{\the\braid@ny}
394     \let\tikz@options=\pgfutil@empty
```

Load general floor style options.

```
395     \expandafter\tikzset\expandafter{\braid@floors@style}
```

Load any style options specific to this floor. We're actually offset by 2 from what the user thinks the floor level is.

```
396     \pgfmathtruncatemacro{\braid@ff}{\braid@f+2}
```

Load the relevant floor style, if it exists.

```
397     \expandafter\let\expandafter\braid@floor@style%
398     \csname braid@options@floor@\braid@ff\endcsname
399     \ifx\braid@floor@style\relax
400     \else
```

There is a floor style for this level, so process it.

```
401     \expandafter\tikzset\expandafter{\braid@floor@style}%
402     \fi
```

The `\tikzset` just parses the options, we need to call `\tikz@options` to actually set them.

```
403     \tikz@options
```

Now we call the rendering code.

```
404     \braid@render@floor
```

Done! End the scope for *this* floor and go again.

```
405     \pgfsys@endscope
406     }
```

Done rendering floors, end the scope.

```
407     \pgfsys@endscope
408     \fi
```

Finished with floors (if we had them), now get on with the strands.

```
409     \stepcounter{braid@level}
410     \foreach \braid@k in {1,...,\value{braid@strands}} {
```

Start a local scope to ensure we don't mess with other braids

```
411     \pgfsys@beginscope
```

Default is to draw each braid

```
412     \tikz@mode@drawtrue%
413     \let\tikz@mode=\pgfutil@empty
414     \let\tikz@options=\pgfutil@empty
```

(x,y) coordinates of bottom of strand

```
415     \braid@tx=\braid@k\braid@width
416     \braid@ty=\value{braid@level}\braid@height
417     \advance\braid@ty by 2\braid@eh
```

Try to find the starting point of this strand

```
418 \expandafter\let\expandafter\braid@path@origin%
419 \csname braid@strand@\braid@k @origin\endcsname
420 \ifx\braid@path@origin\relax
```

If that doesn't exist, we'll just draw a straight line so we move to the top of the current position

```
421 \pgfsyssoftpath@setcurrentpath{\@empty}
422 \pgfpathmoveto{\pgfpoint{\braid@tx}{0pt}}
423 \let\braid@path@origin\braid@k
424 \else
```

If the path does exist, we load it

```
425 \expandafter\let\expandafter\braid@path%
426 \csname braid@strand@\braid@path@origin\endcsname
427 \pgfsyssoftpath@setcurrentpath{\braid@path}
428 \fi
```

Extend the path to the bottom

```
429 \pgflineto{\pgfpoint{\braid@tx}{\braid@ty}}
```

Load common style options

```
430 \expandafter\tikzset\expandafter{\braid@style}
```

Load any style options specific to this strand

```
431 \expandafter\let\expandafter\braid@style%
432 \csname braid@options@strand@\braid@path@origin\endcsname
433 \ifx\braid@style\relax
434 \else
435 \expandafter\tikzset\expandafter{\braid@style}
436 \fi
437 \braid@options
438 \tikz@mode
439 \tikz@options
```

This is the command that actually draws the strand.

```
440 \edef\tikz@temp{\noexpand\pgfusepath%
441 \iftikz@mode@draw draw\fi%
442 }}%
443 \tikz@temp
```

If our braid has a name, we label the ends of the strand.

```
444 \ifx\braid@name\pgfutil@empty
445 \else
```

Label the ends of the strand.

```
446 \coordinate (\braid@name-\braid@path@origin-e) at (\braid@tx,\braid@ty);
447 \coordinate (\braid@name-rev-\braid@k-e) at (\braid@tx,\braid@ty);
448 \braid@nx=\braid@path@origin\braid@width
449 \coordinate (\braid@name-\braid@path@origin-s) at (\braid@nx,0pt);
450 \coordinate (\braid@name-rev-\braid@k-s) at (\braid@nx,0pt);
451 \fi
```

Done with this strand, close the scope and do the next one.

```
452 \pgfsys@endscope
453 }
```

If our braid has a name, we also want to label the centre.

```

454     \ifx\braid@name\pgfutil@empty
455     \else
456     \braid@tx=\value{braid@strands}\braid@width
457     \braid@ty=\value{braid@level}\braid@height
458     \advance\braid@ty by 2\braid@eh
459     \advance\braid@tx by \braid@width
460     \braid@tx=.5\braid@tx
461     \braid@ty=.5\braid@ty
462     \coordinate (\braid@name) at (\braid@tx,\braid@ty);
463     \fi

```

Now we label the strands if needed.

```

464     \ifx\braid@strand@labels\pgfutil@empty
465     \else
466     \expandafter\braid@render@strand@labels\braid@strand@labels{}%
467     \fi

```

All done now, close the scope and end the group (which was opened right at the start).

```

468     \pgfsys@endscope
469     \endgroup}

```

(End of definition for \braid@render.)

`\braid@start` This starts off the braid, initialising a load of stuff. We start a PGF scope, set the level to -1 , the label, floors, and name to empty, process any options we're given, and save certain lengths for later use..

```

470 \def\braid@start#1{%
471   \pgfsys@beginscope
472   \setcounter{braid@level}{-1}%
473   \let\braid@label\@empty
474   \let\braid@strand@labels\@empty
475   \let\braid@floors\@empty
476   \let\braid@name\empty
477   \let\clabel=\braid@clabel@strand
478   \let\olabel=\braid@olabel@strand
479   \pgfkeys{/pgf/braid/.cd,#1}%
480   \ifbraid@strand@labels@origin
481   \let\label=\braid@olabel@strand
482   \else
483   \let\label=\braid@clabel@strand
484   \fi
485   \let\braid@options\tikz@options
486   \tikz@transform
487   \setcounter{braid@strands}{%
488     \pgfkeysvalueof{/pgf/braid/number of strands}}%
489   \braid@width=\pgfkeysvalueof{/pgf/braid/width}%
490   \braid@height=\pgfkeysvalueof{/pgf/braid/height}%
491   \braid@eh=\pgfkeysvalueof{/pgf/braid/border height}%
492   \pgfkeysgetvalue{/pgf/braid/control factor}{\braid@cf}%
493   \pgfkeysgetvalue{/pgf/braid/nudge factor}{\braid@nf}%
494   \braid@height--\braid@height
495   \braid@eh--\braid@eh
496   \braid@increase@leveltrue
497   \braid@process@start

```

```

498 }
(End of definition for \braid@start.)
    These are the lengths we'll use as we construct the braid
499 \newdimen\braid@width
500 \newdimen\braid@height
501 \newdimen\braid@tx
502 \newdimen\braid@ty
503 \newdimen\braid@nx
504 \newdimen\braid@ny
505 \newdimen\braid@cy
506 \newdimen\braid@dy
507 \newdimen\braid@eh
    An if to decide whether or not to step to the next level or not
508 \newif\ifbraid@increase@level
An if to decide whether label indices should be absolute or not
509 \newif\ifbraid@strand@labels@origin
    Some initial values
510 \let\braid@style\pgfutil@empty
511 \let\braid@floors@style\pgfutil@empty
512 \def\braid@over@cross{1}
    Counters to track the strands and the levels.
513 \newcounter{braid@level}
514 \newcounter{braid@strands}
    All the keys we'll use.
515 \pgfkeys{
Handle unknown keys by passing them to pgf and tikz.
516     /tikz/braid/.search also={/pgf},
517     /pgf/braid/.search also={/pgf,/tikz},
Our "namespace" is /pgf/braid.
518     /pgf/braid/.cd,
519     number of strands/.initial=0,
520     height/.initial=1cm,
521     width/.initial=1cm,
522     gap/.initial=.1,
523     border height/.initial=.25cm,
524     control factor/.initial=.5,
525     nudge factor/.initial=.05,
526     name/.code={%
527         \def\braid@name{#1}%
528     },
529     at/.code={%
530         \braid@relocate{#1}%
531     },
532     floor command/.code={%
533         \def\braid@render@floor{#1}%
534     },
535     style strands/.code 2 args={%
536         \def\braid@temp{#2}%
537         \braidset{style each strand/.list={#1}}%

```

```

538   },
539   style each strand/.code={%
540     \expandafter\edef%
541     \csname braid@options@strand@#1\endcsname{\braid@temp}%
542   },
543   style floors/.code 2 args={%
544     \def\braid@temp{#2}%
545     \braidset{style each floor/.list={#1}}%
546   },
547   style each floor/.code={%
548     \expandafter\edef%
549     \csname braid@options@floor@#1\endcsname{\braid@temp}%
550   },
551   style all floors/.code={%
552     \def\braid@floors@style{#1}
553   },
554   strand label/.style={},
555   strand label by origin/.is if=braid@strand@labels@origin,
556 }

```

`\braidset` Shorthand for setting braid-specific keys.

```

557 \def\braidset#1{%
558   \pgfkeys{/pgf/braid/.cd,#1}}

```

(End of definition for `\braidset`. This function is documented on page ??.)

```

559 <*library>
560 <@@=braid>

```

4 Reimplementation as a TikZ Library

Life is so much easier with L^AT_EX3.

```

561 \ProvidesFile{tikzlibrarybraids.code.tex}[%
562   2024/01/09 v2.3 Tikz/PGF library for drawing braid diagrams%
563 ]
564 \RequirePackage{expl3}
565 \ExplSyntaxOn

```

Define all the variables we'll be using.

```

566 \tl_new:N \l__braid_tmpa_tl
567 \tl_new:N \l__braid_tmpb_tl
568 \tl_new:N \l__braid_tmpe_tl
569 \tl_new:N \l__braid_tmpe_tl
570 \tl_new:N \l__braid_anchor_strand_tl
571 \tl_new:N \l__braid_anchor_level_tl
572 \fp_new:N \l__braid_height_fp
573 \fp_new:N \l__braid_width_fp
574 \fp_new:N \l__braid_nudge_fp
575 \fp_new:N \l__braid_control_fp
576 \fp_new:N \l__braid_ctrlax_fp
577 \fp_new:N \l__braid_ctrlay_fp
578 \fp_new:N \l__braid_ctrlbx_fp
579 \fp_new:N \l__braid_ctrlby_fp
580 \fp_new:N \l__braid_endx_fp

```

```

581 \fp_new:N \l__braid_endy_fp
582 \fp_new:N \l__braid_anchor_x_fp
583 \fp_new:N \l__braid_anchor_y_fp
584 \int_new:N \l__braid_tmpa_int
585 \int_new:N \l__braid_tmpb_int
586 \int_new:N \l__braid_length_int
587 \int_new:N \l__braid_strands_int
588 \int_new:N \l__braid_crossing_int
589 \int_new:N \l__braid_crossing_start_int
590 \int_new:N \l__braid_crossing_end_int
591 \int_new:N \l__braid_crossing_width_int
592 \int_new:N \l__braid_crossing_long_int
593 \int_new:N \l__braid_crossing_start_factor_int
594 \int_new:N \l__braid_crossing_end_factor_int
595 \int_new:N \l__braid_anchor_level_int
596 \int_new:N \l__braid_floor_int
597 \seq_new:N \l__braid_tmpa_seq
598 \seq_new:N \l__braid_word_seq
599 \seq_new:N \l__braid_crossing_seq
600 \seq_new:N \l__braid_anchor_seq
601 \seq_new:N \l__braid_floors_seq
602 \str_new:N \l__braid_tmpa_str
603 \bool_new:N \l__braid_step_level_bool
604 \bool_new:N \l__braid_swap_crossing_bool
605 \bool_new:N \l__braid_default_crossing_bool
606 \bool_new:N \l__braid_default_symbol_bool
607 \bool_set_true:N \l__braid_default_crossing_bool
608 \bool_set_true:N \l__braid_default_symbol_bool
609 \bool_new:N \l__braid_floor_bool
610 \bool_new:N \l__braid_height_bool
611 \bool_new:N \l__braid_crossing_height_bool
612 \prop_new:N \l__braid_strands_prop
613 \prop_new:N \l__braid_permutation_prop
614 \prop_new:N \l__braid_crossing_permutation_prop
615 \prop_new:N \l__braid_inverse_prop
616 \prop_new:N \l__braid_anchor_prop
617 \msg_new:nnn {braids} {height} {The~ keys~ "height"~ and~ "crossing~ height"~ shouldn't~ be~}
618 \cs_generate_variant:Nn \seq_set_split:Nnn {NVn}
619 \cs_generate_variant:Nn \str_compare:nNnTF {VnNnTF}

```

Our interface is through a TikZ pic.

```

620 \tikzset{
621   braid/.pic={
622     \begin{scope}[every~ braid/.try]
623       \__braid_parse_word:n {#1}
624       \__braid_count:
625       \__braid_render:
626     \end{scope}
627   },
628   floor/.pic={
629     \path[pic~ actions, draw=none] (0,0) rectangle (1,1);
630     \path[pic~ actions, fill=none] (0,0) -- (1,0) (0,1) -- (1,1);
631   },
632   /tikz/braid/.search~ also={/tikz},
633   braid/.cd,

```

The various TikZ parameters for the braid.

The anchor determines which part of the braid is located at the position specified by the pic. It can be of the form `n-m` where `n` is a strand number and `+m+` is a crossing level. The strand number can be either a number or `rev-n` to use the ending numbering of the strands. The crossing level can also be `s` or `e` which means the actual start or end of the strand (including the border).

```
634 anchor/.initial=1-s,
```

`number of strands` sets a minimum for the number of strands in the braid (otherwise, it is set by the strands used in the specified crossings).

```
635 number~ of~ strands/.initial=0,
```

These keys determine whether crossings are over or under by default.

```
636 crossing~ convention/.is~choice,
637 crossing~ convention/over/.code={
638   \bool_set_true:N \l__braid_default_crossing_bool
639 },
640 crossing~ convention/down/.code={
641   \bool_set_true:N \l__braid_default_crossing_bool
642 },
643 crossing~ convention/under/.code={
644   \bool_set_false:N \l__braid_default_crossing_bool
645 },
646 crossing~ convention/up/.code={
647   \bool_set_false:N \l__braid_default_crossing_bool
648 },
649 crossing~ convention/.default=over,
650 flip~ crossing~ convention/.code={
651   \bool_set_inverse:N \l__braid_default_crossing_bool
652 },
```

These keys determine whether elements should be inverted by default.

```
653 set~ symbols/.is~choice,
654 set~ symbols/over/.code={
655   \bool_set_true:N \l__braid_default_symbol_bool
656 },
657 set~ symbols/down/.code={
658   \bool_set_true:N \l__braid_default_symbol_bool
659 },
660 set~ symbols/under/.code={
661   \bool_set_false:N \l__braid_default_symbol_bool
662 },
663 set~ symbols/up/.code={
664   \bool_set_false:N \l__braid_default_symbol_bool
665 },
666 set~ symbols/.default=over,
667 flip~ symbols/.code={
668   \bool_set_inverse:N \l__braid_default_symbol_bool
669 },
```

The next two keys are used to control the separation between the crossings. The original braid package used the `height` as part of how it determined the direction of the braid on the page. In particular, the `height` could be negative, and indeed the

default is for it to be so since a vertical braid usually flows from top to bottom. Now that the drawing is reimplemented as a `pic` then the direction is better controlled using transformations. So then `height` should be simply to set the gap between crossings in whatever orientation. In particular, `height` should be a positive length.

To avoid backwards incompatibility, the original (counter-intuitive) version of `height` is retained but a new key, `crossing height`, is introduced which can only be positive (rather, the code will take its absolute value so you can *declare* it to be negative but the code will laugh at you and ignore the sign).

To ensure that `crossing height` wins, we use a boolean to see if it has been invoked by the user.

```

670   height/.initial=-1cm,
671   height/.code={
672     \bool_if:NTF \l__braid_crossing_height_bool
673     {
674       \msg_term:nn {braids} {height}
675     }
676     {
677       \pgfkeyssetvalue{/tikz/braid/height}{#1}
678       \bool_set_true:N \l__braid_height_bool
679     }
680   },
681   crossing~ height/.code={
682     \bool_if:NT \l__braid_height_bool
683     {
684       \msg_term:nn {braids} {height}
685     }
686     \exp_args:Nnx \pgfkeyssetvalue {/tikz/braid/height}
687     {\dim_eval:n {-\dim_abs:n{#1}}}
688     \bool_set_true:N \l__braid_crossing_height_bool
689   },

```

`width` is the distance between strands (can be negative).

```

690   width/.initial=1cm,

```

`gap` is for determining the gap in the under-strand of a crossing.

```

691   gap/.initial=.05,

```

`border height` is a length added at the start and end of each strand.

```

692   border~ height/.initial=.25cm,

```

`floor border` is added to the width of any floors

```

693   floor~ border/.initial=.25cm,

```

`floors` is a list of floors to draw, specified as a `csl` of coordinates as (x,y,w,h,a) in which the units are numbers of strands and crossing levels. The parameters are: coordinates of lower left corner, width, height, (optional) name for styling.

```

694   add~ floor/.code={
695     \seq_push:Nn \l__braid_floors_seq {#1}
696   },

```

`control factor` determines the proportion of the `height` used for the control points.

```

697   control~ factor/.initial=.5,

```

`nudge factor` is used to compress each crossing slightly within its rectangle.

```

698   nudge~ factor/.initial=.05

```

```

699 }

```

`__braid_parse_word:Nn` Parse the braid word as a token list and convert it into a sequence.

```

700 \cs_new_nopar:Npn \__braid_parse_word:n #1
701 {
702   \seq_clear:N \l__braid_word_seq
703   \tl_clear:N \l__braid_tmpa_tl
704   \tl_set:Nn \l__braid_tmpb_tl {#1}
705
706   \bool_until_do:nn { \tl_if_empty_p:N \l__braid_tmpb_tl }
707   {

```

We step through the braid specification, looking for special characters. To avoid catcode issues, the comparison is as strings. Some actions may involve consuming more tokens from the list so we can't do a simple `map_inline` but have to keep stripping off the head token.

The idea is to store information about the current crossing in a token list (noting that it may be specified in a variety of orders) and then when we're sure we have all the information we add it to our sequence of crossings.

```

708   \str_set:Nx \l__braid_tmpa_str {\tl_head:N \l__braid_tmpb_tl}
709   \tl_set:Nx \l__braid_tmpb_tl {\tl_tail:N \l__braid_tmpb_tl}
710   \str_case_e:nnTF {\l__braid_tmpa_str}
711   {

```

Underscore introduces the crossing numbers

```

712   {_}
713   {
714     \tl_put_right:Nx \l__braid_tmpa_tl
715     {
716       \exp_not:N \__braid_parse_index:n {\tl_head:N \l__braid_tmpb_tl}
717     }
718     \tl_set:Nx \l__braid_tmpb_tl {\tl_tail:N \l__braid_tmpb_tl}
719   }

```

Power is used to indicate inverse.

```

720   {^}
721   {
722     \tl_put_left:Nx \l__braid_tmpa_tl
723     {
724       \exp_not:N \__braid_parse_exponent:n {\tl_head:N \l__braid_tmpb_tl}
725     }
726     \tl_set:Nx \l__braid_tmpb_tl {\tl_tail:N \l__braid_tmpb_tl}
727   }

```

Bar is for floors.

```

728   {|}
729   {
730     \tl_if_empty:NF \l__braid_tmpa_tl
731     {
732       \seq_put_right:NV \l__braid_word_seq \l__braid_tmpa_tl
733       \tl_clear:N \l__braid_tmpa_tl
734     }
735
736     \tl_set:Nn \l__braid_tmpa_tl {
737       \bool_set_false:N \l__braid_step_level_bool
738       \bool_set_true:N \l__braid_floor_bool

```

```

739     }
740     \seq_put_right:NV \l__braid_word_seq \l__braid_tmpa_tl
741     \tl_clear:N \l__braid_tmpa_tl
742   }

```

Hyphen says the next crossing is on the same level as the current one.

```

743   {-}
744   {
745     \tl_put_right:Nn \l__braid_tmpa_tl
746     {
747       \bool_set_false:N \l__braid_step_level_bool
748     }
749   }

```

1 is for the identity (i.e., no crossing but still have a level). We put a nop token on the list so that it is no longer empty.

```

750     {1}
751     {
752       \tl_if_empty:NF \l__braid_tmpa_tl
753       {
754         \seq_put_right:NV \l__braid_word_seq \l__braid_tmpa_tl
755         \tl_clear:N \l__braid_tmpa_tl
756       }
757       \tl_put_right:Nn \l__braid_tmpa_tl {\__braid_do_identity:}
758     }

```

Ignore spaces.

```

759     {~}
760     {
761     }
762   }
763   {
764   }
765   {

```

If we get an unrecognised token, it's our trigger to start accumulating information for the next crossing.

```

766     \tl_if_empty:NF \l__braid_tmpa_tl
767     {
768       \seq_put_right:NV \l__braid_word_seq \l__braid_tmpa_tl
769       \tl_clear:N \l__braid_tmpa_tl
770     }
771   }
772 }

```

At the end, we also put our current token list on the word sequence.

```

773 \tl_if_empty:NF \l__braid_tmpa_tl
774 {
775   \seq_put_right:NV \l__braid_word_seq \l__braid_tmpa_tl
776   \tl_clear:N \l__braid_tmpa_tl
777 }
778 }

```

(End of definition for __braid_parse_word:Nn.)

`__braid_parse_index:n` Parse an index, saving it in a sequence with the two indices such that the first goes over the second.

```

779 \cs_new_nopar:Npn \__braid_parse_index:n #1
780 {
781   \seq_clear:N \l__braid_crossing_seq
782   \clist_map_inline:nn {#1}
783   {
784     \tl_if_in:nnTF {##1} {-}
785     {
786       \seq_set_split:Nnn \l__braid_tmpa_seq {-} {##1}
787       \int_compare:nTF {\seq_item:Nn \l__braid_tmpa_seq {1} < \seq_item:Nn \l__braid_tmpa_seq {2}}
788       {
789         \int_set:Nn \l__braid_tmpa_int {1}
790       }
791       {
792         \int_set:Nn \l__braid_tmpa_int {-1}
793       }
794       \int_step_inline:nnnn {\seq_item:Nn \l__braid_tmpa_seq {1}} {\l__braid_tmpa_int} {\seq_item:Nn \l__braid_tmpa_seq {2}}
795       {
796         \seq_put_right:Nn \l__braid_crossing_seq {####1}
797       }
798     }
799     {
800       \seq_put_right:Nn \l__braid_crossing_seq {##1}
801     }
802   }
803   \int_compare:nT {\seq_count:N \l__braid_crossing_seq == 1}
804   {
805     \seq_put_right:Nx \l__braid_crossing_seq {\int_eval:n {#1 + 1}}
806   }
807   \bool_xor:nnT {\l__braid_swap_crossing_bool} {\l__braid_default_symbol_bool}
808   {
809     \seq_reverse:N \l__braid_crossing_seq
810   }
811 }

```

(End of definition for `__braid_parse_index:n`.)

`__braid_parse_exponent:n` Parse an exponent, basically testing to see if it is -1 in which case our crossing numbers should be reversed.

```

812 \cs_new_nopar:Npn \__braid_parse_exponent:n #1
813 {
814   \int_compare:nTF {#1 == -1}
815   {
816     \bool_set_true:N \l__braid_swap_crossing_bool
817   }
818   {
819     \bool_set_false:N \l__braid_swap_crossing_bool
820   }
821 }

```

(End of definition for `__braid_parse_exponent:n`.)

`__braid_do_identity:`

```

822 \cs_new_nopar:Npn \__braid_do_identity:
823 {
824 }

```

(End of definition for __braid_do_identity:.)

__braid_count:NNN Work out how big the braid is by counting strands and levels. We also figure out the permutation from the start to end of the strands. This is useful for labelling various parts of the braid.

```

825 \cs_new_nopar:Npn \__braid_count:
826 {
827   \int_zero:N \l__braid_length_int
828   \int_set:Nn \l__braid_strands_int {\__braid_value:n {number-of~strands}}
829   \prop_clear:N \l__braid_permutation_prop
830   \prop_clear:N \l__braid_crossing_permutation_prop
831   \prop_clear:N \l__braid_anchor_prop
832   \prop_clear:N \l__braid_inverse_prop
833
834   \seq_map_inline:Nn \l__braid_word_seq
835   {

```

Clear the crossing sequence and assume we're going to step the level.

```

836     \seq_clear:N \l__braid_crossing_seq
837     \bool_set_true:N \l__braid_step_level_bool
838     \bool_set_false:N \l__braid_swap_crossing_bool

```

Run the details of this crossing.

```

839     ##1

```

If we're increasing the level (no hyphen), do so.

```

840     \bool_if:NT \l__braid_step_level_bool
841     {
842       \int_incr:N \l__braid_length_int
843     }

```

If we have a crossing, check we have enough strands to cover it.

```

844     \seq_if_empty:NF \l__braid_crossing_seq
845     {
846       \seq_map_inline:Nn \l__braid_crossing_seq
847       {
848         \int_set:Nn \l__braid_strands_int
849         {
850           \int_max:nn {\l__braid_strands_int} {####1}
851         }
852       }
853     }
854 }

```

Now that we know how many strands we have, we can initialise our permutation props. One will hold the overall permutation, the other will keep track of our current permutation.

```

855   \int_step_inline:nnnn {1} {1} {\l__braid_strands_int}
856   {
857     \prop_put:Nnn \l__braid_permutation_prop {##1} {##1}
858     \prop_put:Nnn \l__braid_anchor_prop {##1} {##1}
859     \prop_put:Nnn \l__braid_crossing_permutation_prop {##1} {##1}
860   }

```

Now we step through the braid word again and record the permutations so that we can calculate the overall permutation defined by the braid.

We will also figure out our shift from the anchor, so first we need to get some information about the anchor.

If the anchor specification has a hyphen then it is of the form strand-level, otherwise it is an anchor as if the whole braid were contained in a rectangular node.

```

861 \tl_set:Nx \l__braid_tmpa_tl {\__braid_value:n {anchor}}
862 \tl_if_in:NnTF \l__braid_tmpa_tl {-}
863 {
864   \seq_set_split:NnV \l__braid_anchor_seq {-} \l__braid_tmpa_tl
865
866   \tl_set:Nx \l__braid_tmpa_tl {\seq_item:Nn \l__braid_anchor_seq {1}}
867   \tl_if_eq:VnTF \l__braid_tmpa_tl {rev}
868   {
869     \tl_set:Nx \l__braid_anchor_strand_tl {\seq_item:Nn \l__braid_anchor_seq {2}}
870     \tl_set:Nx \l__braid_anchor_level_tl {\seq_item:Nn \l__braid_anchor_seq {3}}
871   }
872   {
873     \tl_set:Nx \l__braid_anchor_strand_tl {\seq_item:Nn \l__braid_anchor_seq {1}}
874     \tl_set:Nx \l__braid_anchor_level_tl {\seq_item:Nn \l__braid_anchor_seq {2}}
875   }

```

The important information is as to the level at which the requested anchor resides. If it is at the end or start of a strand, we set the level to -1 so that it never matches a level number.

```

876 \tl_if_eq:VnTF \l__braid_anchor_level_tl {s}
877 {
878   \int_set:Nn \l__braid_anchor_level_int {-1}
879 }
880 {
881   \tl_if_eq:VnTF \l__braid_anchor_level_tl {e}
882   {
883     \int_set:Nn \l__braid_anchor_level_int {-1}
884   }
885   {
886     \int_set:Nn \l__braid_anchor_level_int
887     {\tl_use:N \l__braid_anchor_level_tl}
888   }
889 }
890 }
891 {

```

There wasn't a hyphen in the anchor specification, so assume it's an anchor on a node surrounding the entire braid. For now, set the anchor strand and level to -1 .

```

892 \int_set:Nn \l__braid_anchor_level_int {-1}
893 \tl_set:Nn \l__braid_anchor_strand_tl {-1}
894 }
895
896 \int_zero:N \l__braid_crossing_int
897 \int_incr:N \l__braid_crossing_int
898 \seq_map_inline:Nn \l__braid_word_seq
899 {
900   \bool_set_true:N \l__braid_step_level_bool

```

```

901 \seq_clear:N \l__braid_crossing_seq
902 \bool_set_false:N \l__braid_swap_crossing_bool
903 ##1
904 \seq_if_empty:NF \l__braid_crossing_seq
905 {
906   \int_step_inline:nnn {2} {\seq_count:N \l__braid_crossing_seq}
907   {
908     \int_set:Nn \l__braid_tmpa_int {####1}
909     \int_set:Nn \l__braid_tmpb_int {####1 - 1}
910
911     \prop_get:NxN \l__braid_permutation_prop
912     {
913       \seq_item:Nn \l__braid_crossing_seq {\l__braid_tmpa_int}
914     } \l__braid_tmpa_tl
915     \prop_get:NxN \l__braid_permutation_prop
916     {
917       \seq_item:Nn \l__braid_crossing_seq {\l__braid_tmpb_int}
918     } \l__braid_tmpb_tl
919
920     \prop_put:NxV \l__braid_permutation_prop
921     {
922       \seq_item:Nn \l__braid_crossing_seq {\l__braid_tmpb_int}
923     } \l__braid_tmpa_tl
924     \prop_put:NxV \l__braid_permutation_prop
925     {
926       \seq_item:Nn \l__braid_crossing_seq {\l__braid_tmpa_int}
927     } \l__braid_tmpb_tl
928   }
929 }

```

See if the current level is what was requested by the anchor.

```

930 \int_compare:nT {\l__braid_crossing_int = \l__braid_anchor_level_int}
931 {
932   \prop_set_eq:NN \l__braid_anchor_prop \l__braid_permutation_prop
933 }
934 \bool_if:NT \l__braid_step_level_bool
935 {
936   \int_incr:N \l__braid_crossing_int
937 }
938 }

```

This inverts the anchor permutation.

```

939 \int_step_inline:nnnn {1} {1} {\l__braid_strands_int}
940 {
941   \prop_get:NnN \l__braid_anchor_prop {##1} \l__braid_tmpa_tl
942   \prop_put:NVn \l__braid_inverse_prop \l__braid_tmpa_tl {##1}
943 }
944 \prop_set_eq:NN \l__braid_anchor_prop \l__braid_inverse_prop

```

This inverts the full permutation.

```

945 \int_step_inline:nnnn {1} {1} {\l__braid_strands_int}
946 {
947   \prop_get:NnN \l__braid_permutation_prop {##1} \l__braid_tmpa_tl
948   \prop_put:NVn \l__braid_inverse_prop \l__braid_tmpa_tl {##1}
949 }

```

Now that we have the inverse, we can figure out our anchor. If the strand was recorded as -1 , then we want to figure out the position from the braid as a whole so we don't bother with processing.

```
950 \tl_if_eq:VnF \l__braid_anchor_strand_tl {-1}
951 {
```

Now, see if we requested a strand by its position at the end of the braid.

```
952 \tl_set:Nx \l__braid_tmpa_tl {\seq_item:Nn \l__braid_anchor_seq {1}}
953 \tl_if_eq:VnT \l__braid_tmpa_tl {rev}
954 {
955   \prop_get:NVN \l__braid_permutation_prop
956   \l__braid_anchor_strand_tl \l__braid_anchor_strand_tl
957 }
958 \tl_if_eq:VnF \l__braid_anchor_level_tl {s}
959 {
960   \tl_if_eq:VnTF \l__braid_anchor_level_tl {e}
961   {
962     \prop_get:NVN \l__braid_inverse_prop
963     \l__braid_anchor_strand_tl \l__braid_anchor_strand_tl
964   }
965   {
966     \prop_get:NVN \l__braid_anchor_prop
967     \l__braid_anchor_strand_tl \l__braid_anchor_strand_tl
968   }
969 }
970 }
971 }
```

(End of definition for __braid_count:NNN.)

`__braid_dim_value:n` Extract a length or a value from a PGF key.

```
\__braid_value:n
972 \cs_new_nopar:Npn \__braid_dim_value:n #1
973 {
974   \dim_to_fp:n {\pgfkeysvalueof{/tikz/braid/#1}}
975 }
976 \cs_new_nopar:Npn \__braid_value:n #1
977 {
978   \pgfkeysvalueof{/tikz/braid/#1}
979 }
```

(End of definition for __braid_dim_value:n and __braid_value:n.)

`__braid_render:` This is the macro that converts the braid word into TikZ paths.

```
980 \cs_generate_variant:Nn \prop_get:NnN {NxN}
981 \cs_generate_variant:Nn \prop_put:Nnn {NxV}
982 \cs_generate_variant:Nn \tl_if_eq:nnTF {VnTF}
983 \cs_generate_variant:Nn \tl_if_eq:nnF {VnF}
984 \cs_generate_variant:Nn \tl_if_eq:nnT {VnT}
985
986 \cs_new_nopar:Npn \__braid_render:
987 {
```

Start by figuring out our anchor.

```
988 \tl_if_eq:VnTF \l__braid_anchor_strand_tl {-1}
989 {
```

The strand is -1 then we're working with the braid as if a node. We'll redefine this node later anyway.

```

990 \tl_set:cn {pgf@sh@ns@temporary braid node}{rectangle}
991 \tl_set:cx {pgf@sh@np@temporary braid node}{%
992 \exp_not:N\def
993 \exp_not:N\southwest
994 {
995 \exp_not:N\pgfqpoint
996 {0pt}
997 {0pt}
998 }
999 \exp_not:N\def
1000 \exp_not:N\northeast
1001 {
1002 \exp_not:N\pgfqpoint
1003 {
1004 \fp_to_dim:n
1005 {
1006 (\l__braid_strands_int - 1)
1007 *
1008 abs(\__braid_dim_value:n {width})
1009 }
1010 }
1011 {
1012 \fp_to_dim:n
1013 {
1014 \l__braid_length_int * abs(\__braid_dim_value:n {height})
1015 + 2 * \__braid_dim_value:n {border~ height}
1016 }
1017 }
1018 }
1019 }%
1020 \pgfgettransform\l__braid_tmpa_tl
1021 \tl_set:cV {pgf@sh@nt@temporary braid node} \l__braid_tmpa_tl
1022 \tl_set:cV {pgf@sh@pi@temporary braid node} \pgfpictureid
1023 \pgfpictureanchor{temporary braid node} {\__braid_value:n {anchor}}

```

Adjustments due to the possibility of negative widths/heights

```

1024 \fp_set:Nn \l__braid_anchor_x_fp {
1025 - \dim_use:c {pgf@x}
1026 - (1 - sign(\__braid_dim_value:n {width})) / 2
1027 * (\l__braid_strands_int - 1)
1028 * \__braid_dim_value:n {width}
1029 }
1030 \fp_set:Nn \l__braid_anchor_y_fp {
1031 - \dim_use:c {pgf@y}
1032 - (1 - sign(\__braid_dim_value:n {height})) / 2
1033 * (
1034 \l__braid_length_int * abs(\__braid_dim_value:n {height})
1035 + 2 * \__braid_dim_value:n {border~ height}
1036 ) * sign(\__braid_dim_value:n {height})
1037 }
1038 }
1039 {

```

The strand is not -1 so we're setting the anchor via strand and level numbers.

```

1040     \fp_set:Nn \l__braid_anchor_x_fp { - 1 * (\tl_use:N \l__braid_anchor_strand_tl - 1) * \_
1041
1042     \tl_if_eq:VnTF \l__braid_anchor_level_tl {s}
1043     {
1044       \fp_set:Nn \l__braid_anchor_y_fp {0}
1045     }
1046     {
1047       \tl_if_eq:VnTF \l__braid_anchor_level_tl {e}
1048       {
1049         \fp_set:Nn \l__braid_anchor_y_fp {
1050           -1 * \l__braid_length_int * \__braid_dim_value:n {height}
1051           - sign(\__braid_dim_value:n {height})
1052           * 2 * \__braid_dim_value:n {border~ height}
1053         }
1054       }
1055       {
1056         \fp_set:Nn \l__braid_anchor_y_fp {
1057           -1 * \l__braid_anchor_level_tl * \__braid_dim_value:n {height}
1058           - sign(\__braid_dim_value:n {height})
1059           * \__braid_dim_value:n {border~ height}
1060         }
1061       }
1062     }
1063   }
1064
1065   \begin{scope}[
1066     shift={
1067       (\fp_to_decimal:N \l__braid_anchor_x_fp pt,
1068        \fp_to_decimal:N \l__braid_anchor_y_fp pt
1069       )
1070     }
1071   ]

```

Initialise a prop for the individual strands.

```

1072   \prop_clear:N \l__braid_strands_prop

```

Initialise some lengths.

```

1073   \fp_zero:N \l__braid_height_fp
1074   \fp_zero:N \l__braid_nudge_fp
1075   \fp_zero:N \l__braid_control_fp

```

This holds our current height of our strands.

```

1076   \fp_set:Nn \l__braid_height_fp
1077   {
1078     sign(\__braid_dim_value:n {height})
1079     * \__braid_dim_value:n {border~ height}
1080   }

```

This holds the total width of our strands.

```

1081   \fp_set:Nn \l__braid_width_fp
1082   {
1083     (\l__braid_strands_int - 1) * \__braid_dim_value:n {width}
1084     + 2 * sign(\__braid_dim_value:n {width})
1085     * \__braid_dim_value:n {floor~ border}
1086   }

```

Each crossing actually starts a little bit into the crossing space, as defined by the `nudge factor`.

```

1087 \fp_set:Nn \l__braid_nudge_fp
1088 {
1089   \__braid_value:n {nudge~ factor} * \__braid_dim_value:n {height}
1090 }

```

This sets where the control points for the crossing curves will be.

```

1091 \fp_set:Nn \l__braid_control_fp
1092 {
1093   \__braid_value:n {control~ factor} * \__braid_dim_value:n {height}
1094 }
1095 \fp_sub:Nn \l__braid_control_fp {\l__braid_nudge_fp}

```

Initialise our strand paths with a `\draw`.

```

1096 \int_step_inline:nnnn {1} {1} {\l__braid_strands_int}
1097 {
1098   \prop_get:NnN \l__braid_inverse_prop {##1} \l__braid_tmpa_tl
1099   \prop_put:Nnx \l__braid_strands_prop {##1}
1100   {
1101     \exp_not:N \draw[
1102       braid/every~ strand/.try,
1103       braid/strand~ ##1/.try
1104     ]
1105     \exp_not:N \__braid_moveto:nn {
1106       \fp_eval:n {(##1 - 1) * \__braid_dim_value:n {width} }
1107     } {0}
1108     \exp_not:N \__braid_lineto:nn {
1109       \fp_eval:n {(##1 - 1) * \__braid_dim_value:n {width} }
1110     } { \fp_to_decimal:N \l__braid_height_fp}
1111   }

```

Add a load of coordinates at the start of each strand, indexed by both forward and backward strand numbers.

```

1112   \__braid_coordinate:xxxx {-##1-s} {-rev-\l__braid_tmpa_tl-s}
1113   {\fp_eval:n {(##1 - 1) * \__braid_dim_value:n {width} }} {0}
1114
1115   \__braid_coordinate:xxxx {-##1-0} {-rev-\l__braid_tmpa_tl-0}
1116   {\fp_eval:n {(##1 - 1) * \__braid_dim_value:n {width} }}
1117   { \fp_to_decimal:N \l__braid_height_fp}
1118 }

```

Run through any extra floors requested.

```

1119 \seq_map_inline:Nn \l__braid_floors_seq
1120 {
1121   \tl_set:Nx \l__braid_tmpa_tl {\clist_item:nn {##1} {5}}
1122   \__braid_do_floor:Vxxxx \l__braid_tmpa_tl
1123   {\fp_eval:n
1124     {
1125       -1*sign(\__braid_dim_value:n{width})
1126       * \__braid_dim_value:n {floor~ border}
1127       + (\__braid_dim_value:n {width}) * (\clist_item:nn {##1} {1} - 1)
1128     }
1129     pt
1130   }

```

```

1131   {\fp_eval:n
1132     {
1133       \l__braid_height_fp + ( \__braid_dim_value:n {height} ) * (\clist_item:nn {##1} {2})
1134     }
1135     pt
1136   }
1137   {\fp_eval:n {
1138     ( (\clist_item:nn {##1} {3}) * \__braid_dim_value:n {width}
1139     + 2 * sign(\__braid_dim_value:n{width})
1140     * \__braid_dim_value:n {floor~ border} ) / \dim_to_fp:n {1cm}
1141   }
1142   }
1143   {\fp_eval:n {
1144     (\clist_item:nn {##1} {4}) * ( \__braid_dim_value:n {height} ) / \dim_to_fp:n {1cm}
1145   }
1146   }
1147 }

```

Keep track of the crossing level for the floor.

```

1148 \int_zero:N \l__braid_crossing_int
1149 \int_incr:N \l__braid_crossing_int
1150
1151 \seq_map_inline:Nn \l__braid_word_seq
1152 {

```

Clear the flags for this segment of the braid word

```

1153   \seq_clear:N \l__braid_crossing_seq
1154   \bool_set_true:N \l__braid_step_level_bool
1155   \bool_set_false:N \l__braid_floor_bool
1156   \bool_set_false:N \l__braid_swap_crossing_bool
1157   ##1

```

If we're drawing a floor, do so straightaway.

```

1158   \bool_if:NT \l__braid_floor_bool
1159   {
1160     \__braid_do_floor:Vxxxx \l__braid_crossing_int
1161     {\fp_eval:n
1162       {
1163         -1*sign(\__braid_dim_value:n{width})
1164         * \__braid_dim_value:n {floor~ border}
1165       }
1166       pt
1167     }
1168     {\fp_to_decimal:N \l__braid_height_fp pt}
1169     {\fp_eval:n { \l__braid_width_fp / \dim_to_fp:n {1cm} }}
1170     {\fp_eval:n { ( \__braid_dim_value:n {height} ) / \dim_to_fp:n {1cm}}}
1171   }

```

If we have a crossing, process it.

```

1172   \seq_if_empty:NF \l__braid_crossing_seq
1173   {
1174     \int_set:Nn \l__braid_crossing_long_int
1175     {
1176       % \seq_item:Nn \l__braid_crossing_seq {\seq_count:N \l__braid_crossing_seq}
1177       \seq_item:Nn \l__braid_crossing_seq {1}
1178     }

```

```

1179 \int_set:Nn \l__braid_crossing_start_int
1180 {
1181   \int_min:nn
1182   {
1183     \seq_item:Nn \l__braid_crossing_seq {1}
1184   }
1185   {
1186     \seq_item:Nn \l__braid_crossing_seq {\seq_count:N \l__braid_crossing_seq}
1187   }
1188 }
1189 \int_set:Nn \l__braid_crossing_end_int
1190 {
1191   \int_max:nn
1192   {
1193     \seq_item:Nn \l__braid_crossing_seq {1}
1194   }
1195   {
1196     \seq_item:Nn \l__braid_crossing_seq {\seq_count:N \l__braid_crossing_seq}
1197   }
1198 }
1199 }
1200 \int_set:Nn \l__braid_crossing_width_int
1201 {
1202   \l__braid_crossing_end_int
1203   -
1204   \l__braid_crossing_start_int
1205 }

```

Step through the crossing

```

1206 \int_step_inline:nnn {2} {\seq_count:N \l__braid_crossing_seq}
1207 {
1208   \bool_if:NTF \l__braid_default_crossing_bool
1209   {
1210     \int_set:Nn \l__braid_tmpa_int {####1}
1211     \int_set:Nn \l__braid_tmpb_int {####1 - 1}
1212   }
1213   {
1214     \int_set:Nn \l__braid_tmpa_int {####1 - 1}
1215     \int_set:Nn \l__braid_tmpb_int {####1}
1216   }

```

Keep track of the current permutation.

```

1217 \prop_get:NxN \l__braid_crossing_permutation_prop
1218 {\seq_item:Nn \l__braid_crossing_seq {\l__braid_tmpa_int}} \l__braid_tmpa_tl
1219 \prop_get:NxN \l__braid_crossing_permutation_prop
1220 {\seq_item:Nn \l__braid_crossing_seq {\l__braid_tmpb_int}} \l__braid_tmpb_tl
1221
1222 \prop_put:NxV \l__braid_crossing_permutation_prop
1223 {\seq_item:Nn \l__braid_crossing_seq {\l__braid_tmpb_int}} \l__braid_tmpa_tl
1224 \prop_put:NxV \l__braid_crossing_permutation_prop
1225 {\seq_item:Nn \l__braid_crossing_seq {\l__braid_tmpa_int}} \l__braid_tmpb_tl

```

Now get the strands corresponding to the ones involved in the crossing.

```

1226 \prop_get:NxN \l__braid_strands_prop
1227 {\seq_item:Nn \l__braid_crossing_seq {\l__braid_tmpa_int}} \l__braid_tmpa_tl

```

```

1228     \prop_get:NxN \l__braid_strands_prop
1229     {\seq_item:Nn \l__braid_crossing_seq {\l__braid_tmpb_int}} \l__braid_tmpb_tl
The over-strand is easy as that's a single curve.
1230 %     \int_set:Nn \l__braid_crossing_start_factor_int {1}
1231 %     \int_set:Nn \l__braid_crossing_end_factor_int {1}
1232 %     \int_compare:nT {
1233 %         \seq_item:Nn \l__braid_crossing_seq {\l__braid_tmpa_int}
1234 %         =
1235 %         \l__braid_crossing_long_int
1236 %     }
1237 %     {
1238 %         \int_set:Nn \l__braid_crossing_start_factor_int {0}
1239 %         \int_set:Nn \l__braid_crossing_end_factor_int {0}
1240
1241 %         \int_compare:nT {
1242 %             ###1 = \seq_count:N \l__braid_crossing_seq
1243 %         }
1244 %         {
1245 %             \int_set:Nn \l__braid_crossing_end_factor_int {1}
1246 %         }
1247 %         \int_compare:nT {
1248 %             ###1 = 2
1249 %         }
1250 %         {
1251 %             \int_set:Nn \l__braid_crossing_start_factor_int {1}
1252 %         }
1253 %     }
1254
1255 \tl_put_right:Nx \l__braid_tmpa_tl
1256 {
1257     \exp_not:N \__braid_lineto:nn
1258
1259     {\fp_eval:n
1260     {
1261         (\seq_item:Nn \l__braid_crossing_seq {\l__braid_tmpa_int} - 1)
1262         * \__braid_dim_value:n {width}
1263     }
1264 }
1265 {\fp_eval:n { \l__braid_height_fp + \l__braid_nudge_fp * \l__braid_crossing_start_factor_int
1266     + \__braid_dim_value:n {height} * (###1 - 2)/(\seq_count:N \l__braid_crossing_seq - 1)
1267 } }
1268
1269 \exp_not:N \__braid_curveto:nnnnnn
1270
1271 {0}
1272 {\fp_eval:n { \l__braid_control_fp
1273 %     * \l__braid_crossing_start_factor_int
1274 %     * 1/(\seq_count:N \l__braid_crossing_seq - 1)}}
1275
1276 {0}
1277 {\fp_eval:n {- \l__braid_control_fp
1278 %     * \l__braid_crossing_end_factor_int
1279 %     * 1/(\seq_count:N \l__braid_crossing_seq - 1)}}
1280

```

```

1281     {\fp_eval:n
1282     {
1283       (\seq_item:Nn \l__braid_crossing_seq {\l__braid_tmpb_int} - 1)
1284       * \__braid_dim_value:n {width}
1285     }
1286   }
1287   {\fp_eval:n
1288   {
1289     \l__braid_height_fp
1290     + \__braid_dim_value:n {height} * (####1 - 1)/(\seq_count:N \l__braid_crossing_s
1291     - \l__braid_nudge_fp * \l__braid_crossing_end_factor_int
1292   }
1293   }
1294 }

```

The under-strand is a bit more complicated as we need to break it in the middle.

```

1295 %   \int_set:Nn \l__braid_crossing_start_factor_int {1}
1296 %   \int_set:Nn \l__braid_crossing_end_factor_int {1}
1297 %   \int_compare:nT {
1298 %     \seq_item:Nn \l__braid_crossing_seq {\l__braid_tmpb_int}
1299 %     =
1300 %     \l__braid_crossing_long_int
1301 %   }
1302 %   {
1303 %     \int_set:Nn \l__braid_crossing_start_factor_int {0}
1304 %     \int_set:Nn \l__braid_crossing_end_factor_int {0}
1305 %
1306 %     \int_compare:nT {
1307 %       ####1 = \seq_count:N \l__braid_crossing_seq
1308 %     }
1309 %     {
1310 %       \int_set:Nn \l__braid_crossing_end_factor_int {1}
1311 %     }
1312 %     \int_compare:nT {
1313 %       ####1 = 2
1314 %     }
1315 %     {
1316 %       \int_set:Nn \l__braid_crossing_start_factor_int {1}
1317 %     }
1318 %   }
1319
1320 \tl_put_right:Nx \l__braid_tmpb_tl
1321 {
1322   \exp_not:N \__braid_lineto:nn
1323
1324   {\fp_eval:n
1325   {
1326     (\seq_item:Nn \l__braid_crossing_seq {\l__braid_tmpb_int} - 1)
1327     * \__braid_dim_value:n {width}
1328   }
1329   }
1330   {\fp_eval:n { \l__braid_height_fp + \l__braid_nudge_fp * \l__braid_crossing_start_fa
1331     + \__braid_dim_value:n {height} * (####1 - 2)/(\seq_count:N \l__braid_crossing_s
1332   } }
1333 }

```

```

1334
1335 \exp_not:N \__braid_curveto:nnnnnn
1336
1337 {0}
1338 {
1339   \fp_eval:n {
1340     \l__braid_control_fp * (.5 - \__braid_value:n {gap} * (\seq_count:N \l__braid_c
1341 %     * \l__braid_crossing_start_factor_int
1342   }
1343 }
1344
1345 {
1346   \fp_eval:n {
1347     - (.5 - \__braid_value:n {gap} * (\seq_count:N \l__braid_crossing_seq - 1) ) / 3
1348     \__braid_bezier_tangent:nnnnn
1349     {.5 - \__braid_value:n {gap} * (\seq_count:N \l__braid_crossing_seq - 1) }
1350     {0}
1351     {0}
1352     {
1353       (\seq_item:Nn \l__braid_crossing_seq {\l__braid_tmpa_int}
1354       - \seq_item:Nn \l__braid_crossing_seq {\l__braid_tmpb_int})
1355       * \__braid_dim_value:n {width}
1356     }
1357     {
1358       (\seq_item:Nn \l__braid_crossing_seq {\l__braid_tmpa_int}
1359       - \seq_item:Nn \l__braid_crossing_seq {\l__braid_tmpb_int})
1360       * \__braid_dim_value:n {width}
1361     }
1362   }
1363 }
1364 {
1365   \fp_eval:n {
1366     -(.5 - \__braid_value:n {gap} * (\seq_count:N \l__braid_crossing_seq - 1) ) / 3
1367     \__braid_bezier_tangent:nnnnn
1368     {.5 - \__braid_value:n {gap} * (\seq_count:N \l__braid_crossing_seq - 1) }
1369     {0}
1370     {
1371       \l__braid_control_fp * 1/(\seq_count:N \l__braid_crossing_seq - 1)
1372 %     * \l__braid_crossing_start_factor_int
1373   }
1374   {
1375     \__braid_dim_value:n {height} * 1/(\seq_count:N \l__braid_crossing_seq - 1)
1376     - \l__braid_nudge_fp * \l__braid_crossing_start_factor_int
1377     - \l__braid_nudge_fp * \l__braid_crossing_end_factor_int
1378     - \l__braid_control_fp * 1/(\seq_count:N \l__braid_crossing_seq - 1)
1379 %     * \l__braid_crossing_end_factor_int
1380   }
1381   {
1382     \__braid_dim_value:n {height} * 1/(\seq_count:N \l__braid_crossing_seq - 1)
1383     - \l__braid_nudge_fp * \l__braid_crossing_start_factor_int
1384     - \l__braid_nudge_fp * \l__braid_crossing_end_factor_int
1385   }
1386 }
1387 }

```

```

1388
1389
1390 {
1391   \fp_eval:n {
1392     (\seq_item:Nn \l__braid_crossing_seq {\l__braid_tmpb_int} - 1)
1393     * \__braid_dim_value:n {width} +
1394     \__braid_bezier_point:nnnnn
1395     {0.5 - \__braid_value:n {gap} * (\seq_count:N \l__braid_crossing_seq - 1) }
1396     {0}
1397     {
1398       (\seq_item:Nn \l__braid_crossing_seq {\l__braid_tmpa_int}
1399       - \seq_item:Nn \l__braid_crossing_seq {\l__braid_tmpb_int})
1400       * \__braid_dim_value:n {width}
1401     }
1402     {
1403       (\seq_item:Nn \l__braid_crossing_seq {\l__braid_tmpa_int}
1404       - \seq_item:Nn \l__braid_crossing_seq {\l__braid_tmpb_int})
1405       * \__braid_dim_value:n {width}
1406     }
1407   }
1408 }
1409 {
1410   \fp_eval:n {
1411     \l__braid_height_fp
1412     + \l__braid_nudge_fp * \l__braid_crossing_start_factor_int
1413     + \__braid_dim_value:n {height} * (###1 - 2)/(\seq_count:N \l__braid_crossing_s
1414     +
1415     \__braid_bezier_point:nnnnn
1416     {0.5 - \__braid_value:n {gap} * (\seq_count:N \l__braid_crossing_seq - 1) }
1417     {0}
1418     {
1419       \l__braid_control_fp * 1/(\seq_count:N \l__braid_crossing_seq - 1)
1420 %     * \l__braid_crossing_start_factor_int
1421     }
1422     {
1423       \__braid_dim_value:n {height} * 1/(\seq_count:N \l__braid_crossing_seq - 1)
1424       - \l__braid_nudge_fp * \l__braid_crossing_start_factor_int
1425       - \l__braid_nudge_fp * \l__braid_crossing_end_factor_int
1426       - \l__braid_control_fp * 1/(\seq_count:N \l__braid_crossing_seq - 1)
1427 %     * \l__braid_crossing_end_factor_int
1428     }
1429     {\__braid_dim_value:n {height} * 1/(\seq_count:N \l__braid_crossing_seq - 1)
1430     - \l__braid_nudge_fp * \l__braid_crossing_start_factor_int
1431     - \l__braid_nudge_fp * \l__braid_crossing_end_factor_int
1432     }
1433   }
1434 }
1435
1436 \exp_not:N \__braid_moveto:nn
1437 {
1438   \fp_eval:n {
1439     (\seq_item:Nn \l__braid_crossing_seq {\l__braid_tmpb_int} - 1)
1440     * \__braid_dim_value:n {width} +
1441     \__braid_bezier_point:nnnnn

```

```

1442     {.5 + \__braid_value:n {gap} * (\seq_count:N \l__braid_crossing_seq - 1) }
1443     {0}
1444     {0}
1445     {
1446         (\seq_item:Nn \l__braid_crossing_seq {\l__braid_tmpa_int}
1447         - \seq_item:Nn \l__braid_crossing_seq {\l__braid_tmpb_int})
1448         * \__braid_dim_value:n {width}
1449     }
1450     {
1451         (\seq_item:Nn \l__braid_crossing_seq {\l__braid_tmpa_int}
1452         - \seq_item:Nn \l__braid_crossing_seq {\l__braid_tmpb_int})
1453         * \__braid_dim_value:n {width}
1454     }
1455 }
1456 }
1457 {
1458     \fp_eval:n {
1459         \l__braid_height_fp
1460         + \l__braid_nudge_fp * \l__braid_crossing_start_factor_int
1461         + \__braid_dim_value:n {height} * (###1 - 2)/(\seq_count:N \l__braid_crossing_s
1462         +
1463         \__braid_bezier_point:nnnnn
1464         {.5 + \__braid_value:n {gap} * (\seq_count:N \l__braid_crossing_seq - 1) }
1465         {0}
1466         {
1467             \l__braid_control_fp * 1/(\seq_count:N \l__braid_crossing_seq - 1)
1468             * \l__braid_crossing_start_factor_int
1469         }
1470         {
1471             \__braid_dim_value:n {height} * 1/(\seq_count:N \l__braid_crossing_seq - 1)
1472             - \l__braid_nudge_fp * \l__braid_crossing_start_factor_int
1473             - \l__braid_nudge_fp * \l__braid_crossing_end_factor_int
1474             - \l__braid_control_fp * 1/(\seq_count:N \l__braid_crossing_seq - 1)
1475             * \l__braid_crossing_end_factor_int
1476         }
1477         {\__braid_dim_value:n {height} * 1/(\seq_count:N \l__braid_crossing_seq - 1)
1478         - \l__braid_nudge_fp * \l__braid_crossing_start_factor_int
1479         - \l__braid_nudge_fp * \l__braid_crossing_end_factor_int
1480         }
1481     }
1482 }
1483
1484 \exp_not:N \__braid_curveto:nnnnnn
1485
1486 {
1487     \fp_eval:n {
1488         (.5 - \__braid_value:n {gap} * (\seq_count:N \l__braid_crossing_seq - 1) ) / 3 *
1489         \__braid_bezier_tangent:nnnnn
1490         {.5 + \__braid_value:n {gap} * (\seq_count:N \l__braid_crossing_seq - 1) }
1491         {0}
1492         {0}
1493         {
1494             (\seq_item:Nn \l__braid_crossing_seq {\l__braid_tmpa_int}
1495             - \seq_item:Nn \l__braid_crossing_seq {\l__braid_tmpb_int})

```

```

1496         * \l__braid_dim_value:n {width}
1497     }
1498     {
1499         (\seq_item:Nn \l__braid_crossing_seq {\l__braid_tmpa_int}
1500         - \seq_item:Nn \l__braid_crossing_seq {\l__braid_tmpb_int})
1501         * \l__braid_dim_value:n {width}
1502     }
1503 }
1504 }
1505 {
1506     \fp_eval:n {
1507         (.5 - \l__braid_value:n {gap} * (\seq_count:N \l__braid_crossing_seq - 1) ) / 3 *
1508         \l__braid_bezier_tangent:nmnnn
1509         {0.5 + \l__braid_value:n {gap} * (\seq_count:N \l__braid_crossing_seq - 1) }
1510         {0}
1511         {
1512             \l__braid_control_fp * 1/(\seq_count:N \l__braid_crossing_seq - 1)
1513 %         * \l__braid_crossing_start_factor_int
1514         }
1515         {
1516             \l__braid_dim_value:n {height} * 1/(\seq_count:N \l__braid_crossing_seq - 1)
1517             - \l__braid_nudge_fp * \l__braid_crossing_start_factor_int
1518             - \l__braid_nudge_fp * \l__braid_crossing_end_factor_int
1519             - \l__braid_control_fp * 1/(\seq_count:N \l__braid_crossing_seq - 1)
1520 %         * \l__braid_crossing_end_factor_int
1521         }
1522         {\l__braid_dim_value:n {height} * 1/(\seq_count:N \l__braid_crossing_seq - 1)
1523         - \l__braid_nudge_fp * \l__braid_crossing_start_factor_int
1524         - \l__braid_nudge_fp * \l__braid_crossing_end_factor_int
1525         }
1526     }
1527 }
1528
1529 {0}
1530 {\fp_eval:n {
1531     - \l__braid_control_fp * (.5 - \l__braid_value:n {gap} * (\seq_count:N \l__braid
1532 %     * \l__braid_crossing_end_factor_int
1533     * 1/(\seq_count:N \l__braid_crossing_seq - 1))}
1534 }
1535
1536 {\fp_eval:n
1537     {
1538         (\seq_item:Nn \l__braid_crossing_seq {\l__braid_tmpa_int} - 1)
1539         * \l__braid_dim_value:n {width}
1540     }
1541 }
1542 {\fp_eval:n
1543     {
1544         \l__braid_height_fp
1545         + \l__braid_dim_value:n {height} * (###1 - 1)/(\seq_count:N \l__braid_crossing_s
1546         - \l__braid_nudge_fp * \l__braid_crossing_end_factor_int
1547     }
1548 }
1549

```

```
1550     }
```

Now put those new strands back in the prop.

```
1551     \prop_put:NxV \l__braid_strands_prop
1552     {\seq_item:Nn \l__braid_crossing_seq {\l__braid_tmpb_tl}} \l__braid_tmpa_tl
1553     \prop_put:NxV \l__braid_strands_prop
1554     {\seq_item:Nn \l__braid_crossing_seq {\l__braid_tmpa_int}} \l__braid_tmpb_tl
```

If the strands are more than one apart, the intermediate strands need to be broken as well.

```
1555     \int_compare:nT
1556     {
1557       \int_max:nn
1558       {
1559         \seq_item:Nn \l__braid_crossing_seq {####1 - 1}
1560       }
1561       {
1562         \seq_item:Nn \l__braid_crossing_seq {####1}
1563       }
1564       -
1565       \int_min:nn
1566       {
1567         \seq_item:Nn \l__braid_crossing_seq {####1 - 1}
1568       }
1569       {
1570         \seq_item:Nn \l__braid_crossing_seq {####1}
1571       }
1572       > 1
1573     }
1574     {
1575       \int_step_inline:nnnn
1576       {
1577         \int_min:nn
1578         {
1579           \seq_item:Nn \l__braid_crossing_seq {####1 - 1}
1580         }
1581         {
1582           \seq_item:Nn \l__braid_crossing_seq {####1}
1583         }
1584         + 1}
1585       {1}
1586     {
1587       \int_max:nn
1588       {
1589         \seq_item:Nn \l__braid_crossing_seq {####1 - 1}
1590       }
1591       {
1592         \seq_item:Nn \l__braid_crossing_seq {####1}
1593       }
1594       - 1
1595     }
1596     {
1597
1598     \prop_get:NnN \l__braid_strands_prop {#####1} \l__braid_tmpa_tl
1599     \tl_put_right:Nx \l__braid_tmpa_tl
```

```

1600     {
1601     \exp_not:N \__braid_lineto:nn
1602     {\fp_eval:n {(#####1 - 1) * \__braid_dim_value:n {width} }}
1603     {\fp_eval:n
1604     {
1605     \l__braid_height_fp + \l__braid_nudge_fp
1606     + .5 * \l__braid_control_fp / (\seq_count:N \l__braid_crossing_seq - 1)
1607     + \__braid_dim_value:n {height} * (#####1 - 2)/(\seq_count:N \l__braid_crossing_seq - 1)
1608     }
1609     }
1610     \exp_not:N \__braid_moveto:nn
1611     {\fp_eval:n {(#####1 - 1) * \__braid_dim_value:n {width} }}
1612     {\fp_eval:n
1613     {
1614     \l__braid_height_fp
1615     - \l__braid_nudge_fp - .5 * \l__braid_control_fp / (\seq_count:N \l__braid_crossing_seq - 1)
1616     + \__braid_dim_value:n {height} * (#####1 - 1)/(\seq_count:N \l__braid_crossing_seq - 1)
1617     }
1618     }
1619     }
1620
1621     \prop_put:NnV \l__braid_strands_prop {#####1} \l__braid_tmpa_tl
1622   }
1623 }

```

Reset the current long

```

1624 \int_compare:nTF
1625 {
1626   \seq_item:Nn \l__braid_crossing_seq {\l__braid_tmpa_int}
1627   =
1628   \l__braid_crossing_long_int
1629 }
1630 {
1631   \int_set:Nn \l__braid_crossing_long_int {\seq_item:Nn \l__braid_crossing_seq {\l__braid_tmpa_int}}
1632 }
1633 {
1634   \int_compare:nT
1635   {
1636     \seq_item:Nn \l__braid_crossing_seq {\l__braid_tmpb_int}
1637     =
1638     \l__braid_crossing_long_int
1639   }
1640   {
1641     \int_set:Nn \l__braid_crossing_long_int {\seq_item:Nn \l__braid_crossing_seq {\l__braid_tmpb_int}}
1642   }
1643 }
1644 }
1645 }
1646 }

```

If we're to step the level, increase the height and add a load of coordinates.

```

1647 \bool_if:NT \l__braid_step_level_bool
1648 {
1649   \fp_add:Nn \l__braid_height_fp { \__braid_dim_value:n {height} }

```

```

1650
1651 \int_step_inline:nnnn {1} {1} {\l__braid_strands_int}
1652 {
1653   \prop_get:NnN \l__braid_crossing_permutation_prop
1654   {###1} \l__braid_tmpb_tl
1655   \prop_get:NVN \l__braid_inverse_prop
1656   \l__braid_tmpb_tl \l__braid_tmpa_tl
1657
1658   \__braid_coordinate:xxxx
1659   {-\l__braid_tmpb_tl-\int_use:N \l__braid_crossing_int}
1660   {-rev-\l__braid_tmpa_tl-\int_use:N \l__braid_crossing_int }
1661   {\fp_eval:n { (###1 - 1) * \__braid_dim_value:n {width} }}
1662   {\fp_to_decimal:N \l__braid_height_fp}
1663 }
1664
1665 \int_incr:N \l__braid_crossing_int
1666 }
1667 }
1668
1669 \fp_add:Nn \l__braid_height_fp
1670 {
1671   sign(\__braid_dim_value:n {height})
1672   * \__braid_dim_value:n {border~ height}
1673 }

```

Add a little bit to the end of each strand, together with some coordinates.

```

1674 \int_step_inline:nnnn {1} {1} {\l__braid_strands_int}
1675 {
1676   \prop_get:NxN \l__braid_strands_prop {##1} \l__braid_tmpa_tl
1677   \prop_get:NxN \l__braid_permutation_prop {##1} \l__braid_tmpb_tl
1678
1679   \tl_put_right:Nx \l__braid_tmpa_tl {
1680     \exp_not:N \__braid_lineto:nn
1681     {\fp_eval:n { (##1 - 1) * \__braid_dim_value:n {width} }}
1682     {\fp_to_decimal:N \l__braid_height_fp}
1683     coordinate (-rev-##1-e)
1684     coordinate (-\l__braid_tmpb_tl-e)
1685   }
1686 }
1687
1688 \prop_put:NnV \l__braid_strands_prop {##1} \l__braid_tmpa_tl
1689 }

```

This is where we actually carry out the drawing commands.

```

1690 \int_step_inline:nnnn {1} {1} {\l__braid_strands_int}
1691 {
1692   \prop_get:NnN \l__braid_strands_prop {##1} \l__braid_tmpa_tl
1693   \tl_use:N \l__braid_tmpa_tl
1694 }

```

Finally, put a node around the whole braid if it's been named

```

1695 \tl_if_empty:cF {tikz@fig@name}
1696 {
1697   \tl_gset:cn {pgf@sh@ns@ \tl_use:c{tikz@fig@name} }{rectangle}
1698   \tl_gset:cx {pgf@sh@np@ \tl_use:c{tikz@fig@name} }{%

```

```

1699 \exp_not:N\def
1700 \exp_not:N\southwest
1701 {
1702   \exp_not:N\pgfqpoint
1703   {
1704     \fp_to_dim:n
1705     {
1706       min(0,
1707         (\l__braid_strands_int - 1)
1708         *
1709         (\__braid_dim_value:n {width})
1710         )
1711     }
1712   }
1713   {
1714     \fp_to_dim:n
1715     {
1716       min(0,
1717         \l__braid_length_int * (\__braid_dim_value:n {height})
1718         + 2 * sign(\__braid_dim_value:n {height}) *
1719         \__braid_dim_value:n {border~ height}
1720         )
1721     }
1722   }
1723 }
1724 \exp_not:N\def
1725 \exp_not:N\northeast
1726 {
1727   \exp_not:N\pgfqpoint
1728   {
1729     \fp_to_dim:n
1730     {
1731       max(0,
1732         (\l__braid_strands_int - 1)
1733         *
1734         (\__braid_dim_value:n {width})
1735         )
1736     }
1737   }
1738   {
1739     \fp_to_dim:n
1740     {
1741       max(0,
1742         \l__braid_length_int * (\__braid_dim_value:n {height})
1743         + 2 * sign(\__braid_dim_value:n {height}) *
1744         \__braid_dim_value:n {border~ height}
1745         )
1746     }
1747   }
1748 }
1749 }%
1750 \pgfgettransform\l__braid_tmpa_tl
1751 \tl_gset:cV {pgf@sh@nt@ \tl_use:c{tikz@fig@name} } \l__braid_tmpa_tl
1752 \tl_gset:cV {pgf@sh@pi@ \tl_use:c{tikz@fig@name} } \pgfpictureid

```

```

1753 }
1754 \end{scope}
1755 }

```

(End of definition for `__braid_render:`.)

```

\__braid_moveto:nn These are our interfaces to the TikZ code.
\__braid_lineto:nn 1756 \cs_new_nopar:Npn \__braid_moveto:nn #1#2
\__braid_curveto:nnnnnn 1757 {
\__braid_coordinate:nnnn 1758 (#1 pt, #2 pt)
1759 }
1760 \cs_new_nopar:Npn \__braid_lineto:nn #1#2
1761 {
1762 -- (#1 pt, #2 pt)
1763 }
1764 \cs_new_nopar:Npn \__braid_curveto:nnnnnn #1#2#3#4#5#6
1765 {
1766 % -- +(5 pt, 0) -- +(0 pt, 0pt)
1767 % -- +( #1 pt, #2 pt) -- (#5 pt + #3 pt, #6 pt + #4 pt) -- (#5 pt, #6 pt)
1768 .. controls +( #1 pt, #2 pt) and +( #3 pt, #4 pt)
1769 .. (#5 pt, #6 pt)
1770 }
1771 \cs_new_nopar:Npn \__braid_coordinate:nnnn #1#2#3#4
1772 {
1773 \coordinate[alias=#2] (#1) at (#3 pt,#4 pt);
1774 }
1775 \cs_generate_variant:Nn \__braid_coordinate:nnnn {xxxx}

```

(End of definition for `__braid_moveto:nn` and others.)

```

\__braid_bezier_point:nnnnnn Used to calculate intermediate points and tangents on a bezier curve.
\__braid_bezier_tangent:nnnnnn 1776 \cs_new_nopar:Npn \__braid_bezier_point:nnnnnn #1#2#3#4#5
1777 {
1778 \fp_eval:n
1779 {
1780 (1 - (#1)) * (1 - (#1)) * (1 - (#1)) * (#2)
1781 +
1782 3 * (1 - (#1)) * (1 - (#1)) * (#1) * (#3)
1783 +
1784 3 * (1 - (#1)) * (#1) * (#1) * (#4)
1785 +
1786 (#1) * (#1) * (#1) * (#5)
1787 }
1788 }
1789 \cs_new_nopar:Npn \__braid_bezier_tangent:nnnnnn #1#2#3#4#5
1790 {
1791 \fp_eval:n
1792 {
1793 3 * (1 - (#1)) * (1 - (#1)) * (#3 - (#2))
1794 +
1795 6 * (1 - (#1)) * (#1) * (#4 - (#3))
1796 +
1797 3 * (#1) * (#1) * (#5 - (#4))
1798 }
1799 }

```

```

1800 \cs_new_nopar:Npn \__braid_do_floor:nnnnn #1#2#3#4#5
1801 {
1802   \pic[pic~ type=floor,
1803     xscale=#4,
1804     yscale=#5,
1805     at={(#2,#3)},
1806     braid/every~ floor/.try,
1807     braid/floor~#1/.try,
1808   ];
1809 }
1810 \cs_generate_variant:Nn \__braid_do_floor:nnnnn {Vxxxx}

```

(End of definition for __braid_bezier_point:nnnnn and __braid_bezier_tangent:nnnnn.)

```

1811 \ExplSyntaxOff

```