

The `braids` package: codebase

Andrew Stacey
loopspace@mathforge.org

v2.2 from 2022/10/26

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 \PackageWarning{braids}{%  
2   This package has been reimplemented as a TikZ library; if starting with a fresh document,  
3 }%
```

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

```
4 \long\def\ge@addto@macro#1#2{%  
5   \begingroup  
6   \toks@\expandafter\expandafter\expandafter{\expandafter#1#2}%  
7   \xdef#1{\the\toks@}%  
8   \endgroup}
```

(End 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.

```
9 \newcommand{\braid}[1] [] {%  
10  \begingroup  
11  \braid@start{#1}}
```

(End 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.)

```

12 \def\braid@process{%
13   \afterassignment\braid@handle\let\braid@token=%
14 }

```

(End 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.

```

15 \def\braid@process@start{%
16   \afterassignment\braid@handle@start\let\braid@token=%
17 }

```

(End 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.

```

18 \def\braid@handle@start{%
19   \let\braid@next=\braid@handle
20   \ifx\braid@token a

```

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

```

21     \let\braid@next=\braid@maybe@locate
22   \else
23     \ifx\braid@token(

```

We got an `(` so we have a name

```

24     \iffalse)\fi %Indentation hack!
25     \let\braid@next=\braid@assign@name
26   \else
27     \ifx\braid@token\@sptoken

```

Space; boring, redo from start

```

28     \let\braid@next=\braid@process@start
29   \fi
30 \fi
31 \fi
32 \braid@next%
33 }

```

(End 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 `_`.

```

34 \let\braid@semicolon=;
35 \let\braid@bar=|
36 \def\braid@handle{%
37   \let\braid@next=\braid@process

```

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

```
38 \ifnum\the\catcode'\;=\active
39 \expandafter\let\expandafter\braid@semicolon\tikz@activesemicolon
40 \fi
41 \ifnum\the\catcode'\|=\active
42 \expandafter\let\expandafter\braid@bar\tikz@activebar
43 \fi
44 \ifx\braid@token\braid@semicolon
```

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

```
45 \let\braid@next=\braid@render
46 \else
47 \ifx\braid@token^
```

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

```
48 \let\braid@next=\braid@sup
49 \else
50 \ifx\braid@token_
```

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

```
51 \let\braid@next=\braid@sub
52 \else
53 \ifx\braid@token-
```

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

```
54 \braid@increase@levelfalse
55 \else
56 \ifx\braid@token1%
```

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

```
57 \ifbraid@increase@level
58 \stepcounter{braid@level}
59 \fi
60 \braid@increase@leveltrue
61 \ge@addto@macro\braid@label{\braid@token}%
62 \else
63 \ifx\braid@token[%
```

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

```
64 \let\braid@next=\braid@process@options
65 \else
66 \ifx\braid@token\braid@bar
```

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

```
67 \edef\braid@tmp{\expandafter\the\value{braid@level}}%
68 \ge@addto@macro\braid@floors\braid@tmp%
69 \else
70 \ifx\braid@token\bgroup
```

Begin group, which we reinterpret as beginning a scope.

```
71 \braid@beginscope
72 \else
73 \ifx\braid@token\egroup
```

End group, which ends the scope

```
74 \braid@endscope
75 \else
76 \ifx\braid@token\braid@olabel@strand
77 \let\braid@next=\braid@olabel@strand
78 \else
79 \ifx\braid@token\braid@clabel@strand
80 \let\braid@next=\braid@clabel@strand
81 \else
```

Otherwise, we add the token to the braid label.

```
82 \ge@addto@macro\braid@label{\braid@token}%
83 \fi
84 \fi
85 \fi
86 \fi
87 \fi
88 \fi
89 \fi
90 \fi
91 \fi
92 \fi
93 \fi
94 \braid@next%
95 }
```

(End 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`.

```
96 \def\braid@maybe@locate{%
97 \afterassignment\braid@@maybe@locate\let\braid@token=%
98 }
```

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

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

```
99 \def\braid@@maybe@locate{%
100 \let\braid@next=\braid@handle
101 \ifx\braid@token t
102 \let\braid@next=\braid@find@location
103 \fi
104 \braid@next%
105 }
```

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

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

```
106 \def\braid@find@location{%
107 \afterassignment\braid@@find@location\let\braid@token=%
108 }
```

(End 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.

```

109 \def\braid@@find@location{%
110   \let\braid@next=\braid@location@error
111   \ifx\braid@token(%)
112     \let\braid@next=\braid@locate
113   \else
114     \ifx\braid@token\@sptoken
115       \let\braid@next=\braid@find@location
116     \fi
117   \fi
118   \braid@next%
119 }

```

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

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

```

120 \def\braid@location@error{%
121   \PackageWarning{braids}{Could not figure out location for braid}%
122   \braid@process@start%
123 }

```

(End 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.

```

124 \def\braid@locate#1){%
125   \tikz@scan@one@point\braid@relocate(#1)%
126 }

```

(End definition for \braid@locate.)

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

```

127 \def\braid@relocate#1){%
128   #1\relax
129   \advance\pgf@x by -\braid@width
130   \pgftransformshift{\pgfpoint{\pgf@x}{\pgf@y}}
131   \braid@process@start%
132 }

```

(End definition for \braid@relocate.)

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

```

133 \def\braid@assign@name#1){%
134   \def\braid@name{#1}%
135   \braid@process@start%
136 }

```

(End 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.)

```

137 \def\braid@process@options#1){%
138   \tikzset{#1}%
139   \braid@process%
140 }

```

(End 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 .

```
141 \def\braid@sup#1_#2{%
142   \g@addto@macro\braid@label_{#2}^{#1}%
143   \braid@add@crossing{#2}{#1}%
144 }
```

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

`\braid@sub`

```
145 % This handles braid elements of the form \Verb+a_1+ or \Verb+a_1^{-1}+.
146 \def\braid@sub#1{%
147   \@ifnextchar^{\braid@sub{#1}}%
148   {\g@addto@macro\braid@label_{#1}\braid@add@crossing{#1}{1}}%
149 }
```

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

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

```
150 \def\braid@@sub#1^#2{%
151   \g@addto@macro\braid@label_{#1}^{#2}%
152   \braid@add@crossing{#1}{#2}%
153 }
```

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

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

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

(End 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.

```
155 \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.

```
156   \edef\braid@crossing@type{#2}%
157   \edef\braid@this@strand{#1}%
158   \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.

```
159   \ifbraid@increase@level
160     \stepcounter{braid@level}
161   \fi
```

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

```
162   \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).

```

163 \braid@tx=\braid@this@strand\braid@width
164 \braid@ty=\value{braid@level}\braid@height
165 \advance\braid@ty by \braid@eh
166 \braid@nx=\braid@tx
167 \braid@ny=\braid@ty
168 \advance\braid@nx by \braid@width
169 \advance\braid@ny by \braid@height
170 \advance\braid@ty by \braid@nf\braid@height
171 \advance\braid@ny by -\braid@nf\braid@height
172 \braid@cy=\braid@ty
173 \braid@dy=\braid@ny
174 \advance\braid@cy by \braid@cf\braid@height
175 \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.

```

176 \expandafter\let\expandafter\braid@this@path@origin%
177 \csname braid@strand@\braid@this@strand @origin\endcsname

```

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

```

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

```

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

```

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

```

Start a new soft path.

```

180 \pgfsyssoftpath@setcurrentpath{\@empty}
181 \pgfpathmoveto{\pgfpoint{\braid@tx}{0pt}}

```

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

```

182 \pgfsyssoftpath@getcurrentpath{\braid@this@path}
183 \else

```

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

```

184 \expandafter\let\expandafter\braid@this@path%
185 \csname braid@strand@\braid@this@path@origin\endcsname
186 \fi

```

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

```

187 \expandafter\let\expandafter\braid@next@path@origin%
188 \csname braid@strand@\braid@next@strand @origin\endcsname
189 \ifx\braid@next@path@origin\relax
190 \let\braid@next@path@origin\braid@next@strand

```

```

191 \pgfsyssoftpath@setcurrentpath{\@empty}
192 \pgfpathmoveto{\pgfpoint{\braid@nx}{0pt}}
193 \pgfsyssoftpath@getcurrentpath{\braid@next@path}
194 \else
195 \expandafter\let\expandafter\braid@next@path%
196 \csname braid@strand@\braid@next@path@origin\endcsname
197 \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.

```

198 \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.

```

199 \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.

```

200 \pgfmathsetmacro{\braid@gst}{0.5 - \pgfkeysvalueof{/pgf/braid/gap}}%
201 \pgfmathsetmacro{\braid@gend}{0.5 + \pgfkeysvalueof{/pgf/braid/gap}}%
202 \ifx\braid@crossing@type\braid@over@cross

```

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

```

203 \pgfpathcurveto{\pgfpoint{\braid@tx}{\braid@cy}}%
204 {\pgfpoint{\braid@nx}{\braid@dy}}%
205 {\pgfpoint{\braid@nx}{\braid@ny}}
206 \else

```

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

```

207 \pgfpathcurvebetweentimecontinue{0}{\braid@gst}%
208 {\pgfpoint{\braid@tx}{\braid@ty}}%
209 {\pgfpoint{\braid@tx}{\braid@cy}}%
210 {\pgfpoint{\braid@nx}{\braid@dy}}%
211 {\pgfpoint{\braid@nx}{\braid@ny}}%
212 \pgfpathcurvebetweentime{\braid@gend}{1}%
213 {\pgfpoint{\braid@tx}{\braid@ty}}%
214 {\pgfpoint{\braid@tx}{\braid@cy}}%
215 {\pgfpoint{\braid@nx}{\braid@dy}}%
216 {\pgfpoint{\braid@nx}{\braid@ny}}
217 \fi

```

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

```

218 \pgfsyssoftpath@getcurrentpath{\braid@this@path}

```

Now do the same with the second path.

```

219 \pgfsyssoftpath@setcurrentpath{\braid@next@path}
220 \pgfpathlineto{\pgfpoint{\braid@nx}{\braid@ty}}
221 \ifx\braid@crossing@type\braid@over@cross
222 \pgfpathcurvebetweentimecontinue{0}{\braid@gst}%
223 {\pgfpoint{\braid@nx}{\braid@ty}}%
224 {\pgfpoint{\braid@nx}{\braid@cy}}%
225 {\pgfpoint{\braid@tx}{\braid@dy}}%
226 {\pgfpoint{\braid@tx}{\braid@ny}}
227 \pgfpathcurvebetweentime{\braid@gend}{1}%

```



```

228   {\pgfqpoint{\braid@nx}{\braid@ty}}%
229   {\pgfqpoint{\braid@nx}{\braid@cy}}%
230   {\pgfqpoint{\braid@tx}{\braid@dy}}%
231   {\pgfqpoint{\braid@tx}{\braid@ny}}
232 \else
233   \pgfpathcurveto{\pgfqpoint{\braid@nx}{\braid@cy}}%
234   {\pgfqpoint{\braid@tx}{\braid@dy}}%
235   {\pgfqpoint{\braid@tx}{\braid@ny}}
236 \fi
237 \pgfsyssoftpath@getcurrentpath{\braid@next@path}

```

Now save the paths to their proper macros again.

```

238 \expandafter\let%
239 \csname braid@strand@\braid@this@path@origin \endcsname%
240 \braid@this@path
241 \expandafter\let%
242 \csname braid@strand@\braid@next@path@origin \endcsname%
243 \braid@next@path

```

Now update the origins

```

244 \expandafter\let%
245 \csname braid@strand@\braid@this@strand @origin\endcsname%
246 \braid@next@path@origin
247 \expandafter\let%
248 \csname braid@strand@\braid@next@strand @origin\endcsname%
249 \braid@this@path@origin

```

increment the strand counter, if necessary

```

250 \pgfmathparse{\value{braid@strands} < \braid@next@strand ?
251   "\noexpand\setcounter{braid@strands}{\braid@next@strand}" : ""}
252 \pgfmathresult

```

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

```

253 \braid@process%
254 }

```

(End 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.

```

255 \newcommand{\braid@olabel@strand}[3] [] {%
256   \edef\braid@tmp{\the\value{braid@level}}%
257   \expandafter\ifx\csname braid@strand@#2@origin\endcsname\relax
258   \g@addto@macro\braid@tmp{#2}%
259   \else
260   \edef\braid@tmpa{\csname braid@strand@#2@origin\endcsname}%
261   \g@addto@macro\braid@tmp{\braid@tmpa}%
262   \fi
263   \g@addto@macro\braid@tmp{#3-#1}%
264   \g@addto@macro{\braid@strand@labels}{\braid@tmp}%
265   \braid@process%
266 }

```

(End definition for `\braid@olabel@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.

```

267 \newcommand{\braid@clabel@strand}[3] [] {%
268   \edef\braid@tmp{\the\value\braid@level}}%
269   \g@addto@macro\braid@tmp{{#2}{#3}{#1}}%
270   \g@addto@macro{\braid@strand@labels}{\braid@tmp}%
271   \braid@process%
272 }

```

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

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

```

273 \def\braid@floors@trim,{

```

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

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

```

274 \def\braid@render@floor{%
275   \draw (\floorsx,\floorsy) rectangle (\floorex,\floorey);
276 }

```

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

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

```

277 \def\braid@render@strand@labels#1{%
278   \def\braid@tmp{#1}%
279   \ifx\braid@tmp\pgfutil@empty
280     \let\braid@next=\pgfutil@gobble
281   \else
282     \let\braid@next=\braid@@render@strand@labels
283   \fi
284   \braid@next{#1}%
285 }

```

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

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

```

286 \def\braid@@render@strand@labels#1#2#3#4{%
287   \begingroup
288   \pgfscope
289   \let\tikz@options=\pgfutil@empty
290   \let\tikz@mode=\pgfutil@empty
291   \let\tik@transform=\pgfutil@empty
292   \let\tikz@fig@name=\pgfutil@empty
293   \tikzset{/pgf/braid/strand label,#4}%
294   \braid@nx=#2\braid@width
295   \braid@ny=#1\braid@height
296   \advance\braid@ny by \braid@eh
297   \advance\braid@ny by \braid@height
298   \pgftransformshift{\pgfpoint{\braid@nx}{\braid@ny}}%
299   \tikz@options
300   \setbox\pgfnodeparttextbox=\hbox%
301   \bgroup%

```

```

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

```

```

356 }%
357 \endpgfscope
358 \endgroup
359 \braid@render@strand@labels%
360 }

```

(End 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 to now.

```

361 \def\braid@render{
Check for floors since we do them first.
362   \ifx\braid@floors\@empty
363   \else
Have some floors, start a scope and prepare to render them.
364   \pgfsys@beginscope
Clear the path (just to be sure).
365   \pgfsyssoftpath@setcurrentpath{\empty}
Trim the initial comma off the list of floors.
366   \edef\braid@floors{\expandafter\braid@floors@trim\braid@floors}
Initialise our horizontal coordinates.
367   \braid@tx=\braid@width
368   \advance\braid@tx by \braid@eh
369   \braid@nx=\value\braid@strands\braid@width
370   \advance\braid@nx by -\braid@eh
Loop over the list of floors.
371   \foreach \braid@f in \braid@floors {
372     \pgfsys@beginscope
Figure out the vertical coordinates for the current floor.
373     \braid@ty=\braid@f\braid@height
374     \advance\braid@ty by \braid@eh
375     \advance\braid@ty by \braid@height
376     \braid@ny=\braid@ty
377     \advance\braid@ny by \braid@height
Save the coordinates for use in the floor rendering macro.
378     \edef\floorsx{\the\braid@tx}
379     \edef\floorsy{\the\braid@ty}
380     \edef\floorex{\the\braid@nx}
381     \edef\floorey{\the\braid@ny}
382     \let\tikz@options=\pgfutil@empty
Load general floor style options.
383     \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.
384     \pgfmathtruncatemacro{\braid@ff}{\braid@f+2}

```

Load the relevant floor style, if it exists.

```
385 \expandafter\let\expandafter\braid@floor@style%
386 \csname braid@options@floor@\braid@ff\endcsname
387 \ifx\braid@floor@style\relax
388 \else
```

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

```
389 \expandafter\tikzset\expandafter{\braid@floor@style}%
390 \fi
```

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

```
391 \tikz@options
```

Now we call the rendering code.

```
392 \braid@render@floor
```

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

```
393 \pgfsys@endscope
394 }
```

Done rendering floors, end the scope.

```
395 \pgfsys@endscope
396 \fi
```

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

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

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

```
399 \pgfsys@beginscope
```

Default is to draw each braid

```
400 \tikz@mode@drawtrue%
401 \let\tikz@mode=\pgfutil@empty
402 \let\tikz@options=\pgfutil@empty
```

(x,y) coordinates of bottom of strand

```
403 \braid@tx=\braid@k\braid@width
404 \braid@ty=\value{braid@level}\braid@height
405 \advance\braid@ty by 2\braid@eh
```

Try to find the starting point of this strand

```
406 \expandafter\let\expandafter\braid@path@origin%
407 \csname braid@strand@\braid@k @origin\endcsname
408 \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

```
409 \pgfsyssoftpath@setcurrentpath{\@empty}
410 \pgfpathmoveto{\pgfqpoint{\braid@tx}{0pt}}
411 \let\braid@path@origin\braid@k
412 \else
```

If the path does exist, we load it

```
413 \expandafter\let\expandafter\braid@path%
414 \csname braid@strand@\braid@path@origin\endcsname
415 \pgfsyssoftpath@setcurrentpath{\braid@path}
416 \fi
```

Extend the path to the bottom

```
417 \pgflineto{\pgfqpoint{\braid@tx}{\braid@ty}}
```

Load common style options

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

Load any style options specific to this strand

```
419 \expandafter\let\expandafter\braid@style%
420 \csname braid@options@strand@\braid@path@origin\endcsname
421 \ifx\braid@style\relax
422 \else
423 \expandafter\tikzset\expandafter{\braid@style}
424 \fi
425 \braid@options
426 \tikz@mode
427 \tikz@options
```

This is the command that actually draws the strand.

```
428 \edef\tikz@temp{\noexpand\pgfusepath%
429 \iftikz@mode@draw draw\fi%
430 }}%
431 \tikz@temp
```

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

```
432 \ifx\braid@name\pgfutil@empty
433 \else
```

Label the ends of the strand.

```
434 \coordinate (\braid@name-\braid@path@origin-e) at (\braid@tx,\braid@ty);
435 \coordinate (\braid@name-rev-\braid@k-e) at (\braid@tx,\braid@ty);
436 \braid@nx=\braid@path@origin\braid@width
437 \coordinate (\braid@name-\braid@path@origin-s) at (\braid@nx,0pt);
438 \coordinate (\braid@name-rev-\braid@k-s) at (\braid@nx,0pt);
439 \fi
```

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

```
440 \pgfsys@endscope
441 }
```

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

```
442 \ifx\braid@name\pgfutil@empty
443 \else
444 \braid@tx=\value{braid@strands}\braid@width
445 \braid@ty=\value{braid@level}\braid@height
446 \advance\braid@ty by 2\braid@eh
447 \advance\braid@tx by \braid@width
448 \braid@tx=.5\braid@tx
449 \braid@ty=.5\braid@ty
450 \coordinate (\braid@name) at (\braid@tx,\braid@ty);
451 \fi
```

Now we label the strands if needed.

```
452 \ifx\braid@strand@labels\pgfutil@empty
453 \else
454 \expandafter\braid@render@strand@labels\braid@strand@labels{}%
455 \fi
```

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

```
456     \pgfsys@endscope
457     \endgroup}
```

(End 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..

```
458 \def\braid@start#1{%
459   \pgfsys@beginscope
460   \setcounter{braid@level}{-1}%
461   \let\braid@label\@empty
462   \let\braid@strand@labels\@empty
463   \let\braid@floors\@empty
464   \let\braid@name\@empty
465   \let\clabel=\braid@clabel@strand
466   \let\olabel=\braid@olabel@strand
467   \pgfkeys{/pgf/braid/.cd,#1}%
468   \ifbraid@strand@labels@origin
469   \let\label=\braid@olabel@strand
470   \else
471   \let\label=\braid@clabel@strand
472   \fi
473   \let\braid@options\tikz@options
474   \tikz@transform
475   \setcounter{braid@strands}{%
476     \pgfkeysvalueof{/pgf/braid/number of strands}}%
477   \braid@width=\pgfkeysvalueof{/pgf/braid/width}%
478   \braid@height=\pgfkeysvalueof{/pgf/braid/height}%
479   \braid@eh=\pgfkeysvalueof{/pgf/braid/border height}%
480   \pgfkeysgetvalue{/pgf/braid/control factor}{\braid@cf}%
481   \pgfkeysgetvalue{/pgf/braid/nudge factor}{\braid@nf}%
482   \braid@height=-\braid@height
483   \braid@eh=-\braid@eh
484   \braid@increase@leveltrue
485   \braid@process@start
486 }
```

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

These are the lengths we'll use as we construct the braid

```
487 \newdimen\braid@width
488 \newdimen\braid@height
489 \newdimen\braid@tx
490 \newdimen\braid@ty
491 \newdimen\braid@nx
492 \newdimen\braid@ny
493 \newdimen\braid@cy
494 \newdimen\braid@dy
495 \newdimen\braid@eh
```

An if to decide whether or not to step to the next level or not

```
496 \newif\ifbraid@increase@level
```

An if to decide whether label indices should be absolute or not

```
497 \newif\ifbraid@strand@labels@origin
    Some initial values
498 \let\braid@style\pgfutil@empty
499 \let\braid@floors@style\pgfutil@empty
500 \def\braid@over@cross{1}
    Counters to track the strands and the levels.
501 \newcounter{braid@level}
502 \newcounter{braid@strands}
    All the keys we'll use.
503 \pgfkeys{
```

Handle unknown keys by passing them to pgf and tikz.

```
504     /tikz/braid/.search also={/pgf},
505     /pgf/braid/.search also={/pgf,/tikz},
```

Our “namespace” is /pgf/braid.

```
506     /pgf/braid/.cd,
507     number of strands/.initial=0,
508     height/.initial=1cm,
509     width/.initial=1cm,
510     gap/.initial=.1,
511     border height/.initial=.25cm,
512     control factor/.initial=.5,
513     nudge factor/.initial=.05,
514     name/.code={%
515         \def\braid@name{#1}%
516     },
517     at/.code={%
518         \braid@relocate{#1}%
519     },
520     floor command/.code={%
521         \def\braid@render@floor{#1}%
522     },
523     style strands/.code 2 args={%
524         \def\braid@temp{#2}%
525         \braidset{style each strand/.list={#1}}%
526     },
527     style each strand/.code={%
528         \expandafter\edef%
529         \csname braid@options@strand@#1\endcsname{\braid@temp}%
530     },
531     style floors/.code 2 args={%
532         \def\braid@temp{#2}%
533         \braidset{style each floor/.list={#1}}%
534     },
535     style each floor/.code={%
536         \expandafter\edef%
537         \csname braid@options@floor@#1\endcsname{\braid@temp}%
538     },
539     style all floors/.code={%
540         \def\braid@floors@style{#1}
541     },
```



```

542     strand label/.style={},
543     strand label by origin/.is if=braid@strand@labels@origin,
544 }

```

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

```

545 \def\braidset#1{%
546   \pgfkeys{/pgf/braid/.cd,#1}}

```

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

```

547 <*library>
548 <@=braid>

```

4 Reimplementation as a TikZ Library

Life is so much easier with L^AT_EX₃.

```

549 \ProvidesFile{tikzlibrarybraids.code.tex}[%
550   2022/10/26 v2.2 Tikz/PGF library for drawing braid diagrams%
551 ]
552 \RequirePackage{expl3}
553 \ExplSyntaxOn

```

Define all the variables we'll be using.

```

554 \tl_new:N \l__braid_tmpa_tl
555 \tl_new:N \l__braid_tmpb_tl
556 \tl_new:N \l__braid_tmpc_tl
557 \tl_new:N \l__braid_tmpd_tl
558 \tl_new:N \l__braid_anchor_strand_tl
559 \tl_new:N \l__braid_anchor_level_tl
560 \fp_new:N \l__braid_height_fp
561 \fp_new:N \l__braid_width_fp
562 \fp_new:N \l__braid_nudge_fp
563 \fp_new:N \l__braid_control_fp
564 \fp_new:N \l__braid_ctrlax_fp
565 \fp_new:N \l__braid_ctrlay_fp
566 \fp_new:N \l__braid_ctrlbx_fp
567 \fp_new:N \l__braid_ctrlby_fp
568 \fp_new:N \l__braid_endx_fp
569 \fp_new:N \l__braid_endy_fp
570 \fp_new:N \l__braid_anchor_x_fp
571 \fp_new:N \l__braid_anchor_y_fp
572 \int_new:N \l__braid_tmpa_int
573 \int_new:N \l__braid_tmpb_int
574 \int_new:N \l__braid_length_int
575 \int_new:N \l__braid_strands_int
576 \int_new:N \l__braid_crossing_int
577 \int_new:N \l__braid_crossing_start_int
578 \int_new:N \l__braid_crossing_end_int
579 \int_new:N \l__braid_crossing_width_int
580 \int_new:N \l__braid_crossing_long_int
581 \int_new:N \l__braid_crossing_start_factor_int
582 \int_new:N \l__braid_crossing_end_factor_int
583 \int_new:N \l__braid_anchor_level_int
584 \int_new:N \l__braid_floor_int

```

```

585 \seq_new:N \l__braid_tmpa_seq
586 \seq_new:N \l__braid_word_seq
587 \seq_new:N \l__braid_crossing_seq
588 \seq_new:N \l__braid_anchor_seq
589 \seq_new:N \l__braid_floors_seq
590 \str_new:N \l__braid_tmpa_str
591 \str_new:N \l__braid_sup_str
592 \str_set:Nn \l__braid_sup_str {^}
593 \str_new:N \l__braid_sub_str
594 \str_set:Nn \l__braid_sub_str {_}
595 \str_new:N \l__braid_hyphen_str
596 \str_set:Nn \l__braid_hyphen_str {-}
597 \str_new:N \l__braid_bar_str
598 \str_set:Nn \l__braid_bar_str {|}
599 \str_new:N \l__braid_one_str
600 \str_set:Nn \l__braid_one_str {1}
601 \bool_new:N \l__braid_step_level_bool
602 \bool_new:N \l__braid_swap_crossing_bool
603 \bool_new:N \l__braid_floor_bool
604 \prop_new:N \l__braid_strands_prop
605 \prop_new:N \l__braid_permutation_prop
606 \prop_new:N \l__braid_crossing_permutation_prop
607 \prop_new:N \l__braid_inverse_prop
608 \prop_new:N \l__braid_anchor_prop
609 \cs_generate_variant:Nn \seq_set_split:Nnn {NVn}

```

Our interface is through a TikZ pic.

```

610 \tikzset{
611   braid/.pic={
612     \__braid_parse_word:n {#1}
613     \__braid_count:
614     \__braid_render:
615   },
616   floor/.pic={
617     \path[pic~ actions, draw=none] (0,0) rectangle (1,1);
618     \path[pic~ actions, fill=none] (0,0) -- (1,0) (0,1) -- (1,1);
619   },
620   /tikz/braid/.search~ also={/tikz},
621   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).

```

622   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).

```

623   number~ of~ strands/.initial=0,

```

`height` is the distance between crossings (can be negative).

```

624   height/.initial=-1cm,

```

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

```
625 width/.initial=1cm,
```

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

```
626 gap/.initial=.05,
```

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

```
627 border~ height/.initial=.25cm,
```

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

```
628 floor~ border/.initial=.25cm,
```

`floors` is a list of floors to draw, specified as a cslist 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.

```
629 add~ floor/.code={
630   \seq_push:Nn \l__braid_floors_seq {#1}
631 },
```

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

```
632 control~ factor/.initial=.5,
```

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

```
633 nudge~ factor/.initial=.05
634 }
```

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

```
635 \cs_new_nopar:Npn \__braid_parse_word:n #1
636 {
637   \seq_clear:N \l__braid_word_seq
638   \tl_clear:N \l__braid_tmpa_tl
639   \tl_set:Nn \l__braid_tmpb_tl {#1}
640
641   \bool_until_do:nm { \tl_if_empty_p:N \l__braid_tmpb_tl }
642   {
```

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.

```
643   \str_set:Nx \l__braid_tmpa_str {\tl_head:N \l__braid_tmpb_tl}
644   \tl_set:Nx \l__braid_tmpb_tl {\tl_tail:N \l__braid_tmpb_tl}
645   \str_case_e:nnTF {\l__braid_tmpa_str}
646   {
```

Underscore introduces the crossing numbers

```
647   {_}
648   {
649     \tl_put_right:Nx \l__braid_tmpa_tl
650     {
651       \exp_not:N \__braid_parse_index:n {\tl_head:N \l__braid_tmpb_tl}
652     }
653     \tl_set:Nx \l__braid_tmpb_tl {\tl_tail:N \l__braid_tmpb_tl}
654   }
```

Power is used to indicate inverse.

```

655     {~}
656     {
657       \tl_put_left:Nx \l__braid_tmpa_tl
658       {
659         \exp_not:N \__braid_parse_exponent:n {\tl_head:N \l__braid_tmpb_tl}
660       }
661       \tl_set:Nx \l__braid_tmpb_tl {\tl_tail:N \l__braid_tmpb_tl}
662     }

```

Bar is for floors.

```

663     {}
664     {
665       \tl_if_empty:NF \l__braid_tmpa_tl
666       {
667         \seq_put_right:NV \l__braid_word_seq \l__braid_tmpa_tl
668         \tl_clear:N \l__braid_tmpa_tl
669       }
670
671       \tl_set:Nn \l__braid_tmpa_tl {
672         \bool_set_false:N \l__braid_step_level_bool
673         \bool_set_true:N \l__braid_floor_bool
674       }
675       \seq_put_right:NV \l__braid_word_seq \l__braid_tmpa_tl
676       \tl_clear:N \l__braid_tmpa_tl
677     }

```

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

```

678     {-}
679     {
680       \tl_put_right:Nn \l__braid_tmpa_tl
681       {
682         \bool_set_false:N \l__braid_step_level_bool
683       }
684     }

```

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.

```

685     {1}
686     {
687       \tl_if_empty:NF \l__braid_tmpa_tl
688       {
689         \seq_put_right:NV \l__braid_word_seq \l__braid_tmpa_tl
690         \tl_clear:N \l__braid_tmpa_tl
691       }
692       \tl_put_right:Nn \l__braid_tmpa_tl {\__braid_do_identity:}
693     }

```

Ignore spaces.

```

694     {~}
695     {
696     }
697   }
698   {
699   }
700   {

```

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

```

701     \tl_if_empty:NF \l__braid_tmpa_tl
702     {
703       \seq_put_right:NV \l__braid_word_seq \l__braid_tmpa_tl
704       \tl_clear:N \l__braid_tmpa_tl
705     }
706   }
707 }

```

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

```

708 \tl_if_empty:NF \l__braid_tmpa_tl
709 {
710   \seq_put_right:NV \l__braid_word_seq \l__braid_tmpa_tl
711   \tl_clear:N \l__braid_tmpa_tl
712 }
713 }

```

(End 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.

```

714 \cs_new_nopar:Npn \__braid_parse_index:n #1
715 {
716   \seq_clear:N \l__braid_crossing_seq
717   \clist_map_inline:nn {#1}
718   {
719     \tl_if_in:nnTF {##1} {-}
720     {
721       \seq_set_split:Nnn \l__braid_tmpa_seq {-} {##1}
722       \int_compare:nTF {\seq_item:Nn \l__braid_tmpa_seq {1} < \seq_item:Nn \l__braid_tmpa_seq {2}}
723       {
724         \int_set:Nn \l__braid_tmpa_int {1}
725       }
726       {
727         \int_set:Nn \l__braid_tmpa_int {-1}
728       }
729       \int_step_inline:nnnn {\seq_item:Nn \l__braid_tmpa_seq {1}} {\l__braid_tmpa_int} {\seq_item:Nn \l__braid_tmpa_seq {2}}
730       {
731         \seq_put_right:Nn \l__braid_crossing_seq {###1}
732       }
733     }
734     {
735       \seq_put_right:Nn \l__braid_crossing_seq {##1}
736     }
737   }
738   \int_compare:nT {\seq_count:N \l__braid_crossing_seq == 1}
739   {
740     \seq_put_right:Nx \l__braid_crossing_seq {\int_eval:n {#1 + 1}}
741   }
742   \bool_if:NF \l__braid_swap_crossing_bool
743   {
744     \seq_reverse:N \l__braid_crossing_seq
745   }
746 }

```

(End 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..

```
747 \cs_new_nopar:Npn \__braid_parse_exponent:n #1
748 {
749   \int_compare:nTF {#1 == -1}
750   {
751     \bool_set_true:N \l__braid_swap_crossing_bool
752   }
753   {
754     \bool_set_false:N \l__braid_swap_crossing_bool
755   }
756 }
```

(End definition for `__braid_parse_exponent:n`.)

`__braid_do_identity:`

```
757 \cs_new_nopar:Npn \__braid_do_identity:
758 {
759 }
```

(End 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.

```
760 \cs_new_nopar:Npn \__braid_count:
761 {
762   \int_zero:N \l__braid_length_int
763   \int_set:Nn \l__braid_strands_int {\__braid_value:n {number-of~strands}}
764   \prop_clear:N \l__braid_permutation_prop
765   \prop_clear:N \l__braid_crossing_permutation_prop
766   \prop_clear:N \l__braid_anchor_prop
767   \prop_clear:N \l__braid_inverse_prop
768
769   \seq_map_inline:Nn \l__braid_word_seq
770   {
```

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

```
771   \seq_clear:N \l__braid_crossing_seq
772   \bool_set_true:N \l__braid_step_level_bool
773   \bool_set_false:N \l__braid_swap_crossing_bool
```

Run the details of this crossing.

```
774   ##1
```

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

```
775   \bool_if:NT \l__braid_step_level_bool
776   {
777     \int_incr:N \l__braid_length_int
778   }
```

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

```

779   \seq_if_empty:NF \l__braid_crossing_seq
780   {
781     \seq_map_inline:Nn \l__braid_crossing_seq
782     {
783       \int_set:Nn \l__braid_strands_int
784       {
785         \int_max:nn {\l__braid_strands_int} {####1}
786       }
787     }
788   }
789 }

```

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.

```

790   \int_step_inline:nmmn {1} {1} {\l__braid_strands_int}
791   {
792     \prop_put:Nnn \l__braid_permutation_prop {##1} {##1}
793     \prop_put:Nnn \l__braid_anchor_prop {##1} {##1}
794     \prop_put:Nnn \l__braid_crossing_permutation_prop {##1} {##1}
795   }

```

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.

```

796   \tl_set:Nx \l__braid_tmpa_tl {\__braid_value:n {anchor}}
797   \tl_if_in:NnTF \l__braid_tmpa_tl {-}
798   {
799     \seq_set_split:NnV \l__braid_anchor_seq {-} \l__braid_tmpa_tl
800
801     \tl_set:Nx \l__braid_tmpa_tl {\seq_item:Nn \l__braid_anchor_seq {1}}
802     \tl_if_eq:VnTF \l__braid_tmpa_tl {rev}
803     {
804       \tl_set:Nx \l__braid_anchor_strand_tl {\seq_item:Nn \l__braid_anchor_seq {2}}
805       \tl_set:Nx \l__braid_anchor_level_tl {\seq_item:Nn \l__braid_anchor_seq {3}}
806     }
807     {
808       \tl_set:Nx \l__braid_anchor_strand_tl {\seq_item:Nn \l__braid_anchor_seq {1}}
809       \tl_set:Nx \l__braid_anchor_level_tl {\seq_item:Nn \l__braid_anchor_seq {2}}
810     }

```

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.

```

811   \tl_if_eq:VnTF \l__braid_anchor_level_tl {s}
812   {
813     \int_set:Nn \l__braid_anchor_level_int {-1}
814   }
815   {
816     \tl_if_eq:VnTF \l__braid_anchor_level_tl {e}

```

```

817     {
818       \int_set:Nn \l__braid_anchor_level_int {-1}
819     }
820     {
821       \int_set:Nn \l__braid_anchor_level_int
822       {\tl_use:N \l__braid_anchor_level_tl}
823     }
824   }
825 }
826 {

```

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 .

```

827   \int_set:Nn \l__braid_anchor_level_int {-1}
828   \tl_set:Nn \l__braid_anchor_strand_tl {-1}
829 }
830
831 \int_zero:N \l__braid_crossing_int
832 \int_incr:N \l__braid_crossing_int
833 \seq_map_inline:Nn \l__braid_word_seq
834 {
835   \bool_set_true:N \l__braid_step_level_bool
836   \seq_clear:N \l__braid_crossing_seq
837   \bool_set_false:N \l__braid_swap_crossing_bool
838   ##1
839   \seq_if_empty:NF \l__braid_crossing_seq
840   {
841     \int_step_inline:nnn {2} {\seq_count:N \l__braid_crossing_seq}
842     {
843       \int_set:Nn \l__braid_tmpa_int {####1}
844       \int_set:Nn \l__braid_tmpb_int {####1 - 1}
845
846       \prop_get:NxN \l__braid_permutation_prop
847       {
848         \seq_item:Nn \l__braid_crossing_seq {\l__braid_tmpa_int}
849       } \l__braid_tmpa_tl
850       \prop_get:NxN \l__braid_permutation_prop
851       {
852         \seq_item:Nn \l__braid_crossing_seq {\l__braid_tmpb_int}
853       } \l__braid_tmpb_tl
854
855       \prop_put:NxV \l__braid_permutation_prop
856       {
857         \seq_item:Nn \l__braid_crossing_seq {\l__braid_tmpb_int}
858       } \l__braid_tmpa_tl
859       \prop_put:NxV \l__braid_permutation_prop
860       {
861         \seq_item:Nn \l__braid_crossing_seq {\l__braid_tmpa_int}
862       } \l__braid_tmpb_tl
863     }
864   }

```

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

```

865   \int_compare:nT {\l__braid_crossing_int = \l__braid_anchor_level_int}

```



```

866     {
867     \prop_set_eq:NN \l__braid_anchor_prop \l__braid_permutation_prop
868     }
869     \bool_if:NT \l__braid_step_level_bool
870     {
871     \int_incr:N \l__braid_crossing_int
872     }
873     }

```

This inverts the anchor permutation.

```

874 \int_step_inline:nmmn {1} {1} {\l__braid_strands_int}
875 {
876 \prop_get:NnN \l__braid_anchor_prop {##1} \l__braid_tmpa_tl
877 \prop_put:NVn \l__braid_inverse_prop \l__braid_tmpa_tl {##1}
878 }
879 \prop_set_eq:NN \l__braid_anchor_prop \l__braid_inverse_prop

```

This inverts the full permutation.

```

880 \int_step_inline:nmmn {1} {1} {\l__braid_strands_int}
881 {
882 \prop_get:NnN \l__braid_permutation_prop {##1} \l__braid_tmpa_tl
883 \prop_put:NVn \l__braid_inverse_prop \l__braid_tmpa_tl {##1}
884 }

```

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.

```

885 \tl_if_eq:VnF \l__braid_anchor_strand_tl {-1}
886 {

```

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

```

887 \tl_set:Nx \l__braid_tmpa_tl {\seq_item:Nn \l__braid_anchor_seq {1}}
888 \tl_if_eq:VnT \l__braid_tmpa_tl {rev}
889 {
890 \prop_get:NVN \l__braid_permutation_prop
891 \l__braid_anchor_strand_tl \l__braid_anchor_strand_tl
892 }
893 \tl_if_eq:VnF \l__braid_anchor_level_tl {s}
894 {
895 \tl_if_eq:VnTF \l__braid_anchor_level_tl {e}
896 {
897 \prop_get:NVN \l__braid_inverse_prop
898 \l__braid_anchor_strand_tl \l__braid_anchor_strand_tl
899 }
900 {
901 \prop_get:NVN \l__braid_anchor_prop
902 \l__braid_anchor_strand_tl \l__braid_anchor_strand_tl
903 }
904 }
905 }
906 }

```

(End definition for __braid_count:NNN.)

```


__braid_dim_value:n Extract a length or a value from a PGF key.
  __braid_value:n
    907 \cs_new_nopar:Npn __braid_dim_value:n #1
    908 {
    909   \dim_to_fp:n {\pgfkeysvalueof{/tikz/braid/#1}}
    910 }
    911 \cs_new_nopar:Npn __braid_value:n #1
    912 {
    913   \pgfkeysvalueof{/tikz/braid/#1}
    914 }


```

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

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

```


    915 \cs_generate_variant:Nn \prop_get:NnN {NxN}
    916 \cs_generate_variant:Nn \prop_put:Nnn {NxV}
    917 \cs_generate_variant:Nn \tl_if_eq:nnTF {VnTF}
    918 \cs_generate_variant:Nn \tl_if_eq:nmF {VnF}
    919 \cs_generate_variant:Nn \tl_if_eq:nnT {VnT}
    920
    921 \cs_new_nopar:Npn __braid_render:
    922 {


```

Start by figuring out our anchor.

```


    923 \tl_if_eq:VnTF \l__braid_anchor_strand_tl {-1}
    924 {


```

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

```


    925 \tl_set:cn {pgf@sh@ns@temporary braid node}{rectangle}
    926 \tl_set:cx {pgf@sh@np@temporary braid node}{%
    927   \exp_not:N\def
    928   \exp_not:N\southwest
    929   {
    930     \exp_not:N\pgfqpoint
    931     {0pt}
    932     {0pt}
    933   }
    934   \exp_not:N\def
    935   \exp_not:N\northeast
    936   {
    937     \exp_not:N\pgfqpoint
    938     {
    939       \fp_to_dim:n
    940       {
    941         (\l__braid_strands_int - 1)
    942         *
    943         abs(\__braid_dim_value:n {width})
    944       }
    945     }
    946     {
    947       \fp_to_dim:n
    948       {
    949         \l__braid_length_int * abs(\__braid_dim_value:n {height})
    950         + 2 * \__braid_dim_value:n {border~ height}
    951       }
    }


```

```

952     }
953   }
954 }%
955 \pgfgettransform\l__braid_tmpa_tl
956 \tl_set:cV {pgf@sh@nt@temporary braid node} \l__braid_tmpa_tl
957 \tl_set:cV {pgf@sh@pi@temporary braid node} \pgfpictureid
958 \pgfpointanchor{temporary braid node} {\_braid_value:n {anchor}}

```

Adjustments due to the possibility of negative widths/heights

```

959 \fp_set:Nn \l__braid_anchor_x_fp {
960   - \dim_use:c {pgf@x}
961   - (1 - sign(\_braid_dim_value:n {width})) / 2
962   * (\_braid_strands_int - 1)
963   * \_braid_dim_value:n {width}
964 }
965 \fp_set:Nn \l__braid_anchor_y_fp {
966   - \dim_use:c {pgf@y}
967   - (1 - sign(\_braid_dim_value:n {height})) / 2
968   * (
969     \_braid_length_int * abs(\_braid_dim_value:n {height})
970     + 2 * \_braid_dim_value:n {border~ height}
971     ) * sign(\_braid_dim_value:n {height})
972 }
973 }
974 {

```

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

```

975 \fp_set:Nn \l__braid_anchor_x_fp { - 1 * (\tl_use:N \l__braid_anchor_strand_tl - 1) * \_
976
977 \tl_if_eq:VnTF \l__braid_anchor_level_tl {s}
978 {
979   \fp_set:Nn \l__braid_anchor_y_fp {0}
980 }
981 {
982   \tl_if_eq:VnTF \l__braid_anchor_level_tl {e}
983   {
984     \fp_set:Nn \l__braid_anchor_y_fp {
985       -1 * \_braid_length_int * \_braid_dim_value:n {height}
986       - sign(\_braid_dim_value:n {height})
987       * 2 * \_braid_dim_value:n {border~ height}
988     }
989   }
990   {
991     \fp_set:Nn \l__braid_anchor_y_fp {
992       -1 * \_braid_anchor_level_tl * \_braid_dim_value:n {height}
993       - sign(\_braid_dim_value:n {height})
994       * \_braid_dim_value:n {border~ height}
995     }
996   }
997 }
998 }
999
1000 \begin{scope}[
1001   shift={
1002     (\fp_to_decimal:N \l__braid_anchor_x_fp pt,

```

```

1003     \fp_to_decimal:N \l__braid_anchor_y_fp pt
1004     )
1005   }
1006 ]

```

Initialise a prop for the individual strands.

```

1007 \prop_clear:N \l__braid_strands_prop

```

Initialise some lengths.

```

1008 \fp_zero:N \l__braid_height_fp
1009 \fp_zero:N \l__braid_nudge_fp
1010 \fp_zero:N \l__braid_control_fp

```

This holds our current height of our strands.

```

1011 \fp_set:Nn \l__braid_height_fp
1012 {
1013   sign(\__braid_dim_value:n {height})
1014   * \__braid_dim_value:n {border~ height}
1015 }

```

This holds the total width of our strands.

```

1016 \fp_set:Nn \l__braid_width_fp
1017 {
1018   (\l__braid_strands_int - 1) * \__braid_dim_value:n {width}
1019   + 2 * sign(\__braid_dim_value:n {width})
1020   * \__braid_dim_value:n {floor~ border}
1021 }

```

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

```

1022 \fp_set:Nn \l__braid_nudge_fp
1023 {
1024   \__braid_value:n {nudge~ factor} * \__braid_dim_value:n {height}
1025 }

```

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

```

1026 \fp_set:Nn \l__braid_control_fp
1027 {
1028   \__braid_value:n {control~ factor} * \__braid_dim_value:n {height}
1029 }
1030 \fp_sub:Nn \l__braid_control_fp {\l__braid_nudge_fp}

```

Initialise our strand paths with a \draw.

```

1031 \int_step_inline:nnnn {1} {1} {\l__braid_strands_int}
1032 {
1033   \prop_get:NnN \l__braid_inverse_prop {##1} \l__braid_tmpa_tl
1034   \prop_put:Nnx \l__braid_strands_prop {##1}
1035   {
1036     \exp_not:N \draw[
1037       braid/every~ strand/.try,
1038       braid/strand~ ##1/.try
1039     ]
1040     \exp_not:N \__braid_moveto:nn {
1041       \fp_eval:n {(##1 - 1) * \__braid_dim_value:n {width} }
1042     } {0}
1043     \exp_not:N \__braid_lineto:nn {
1044       \fp_eval:n {(##1 - 1) * \__braid_dim_value:n {width} }

```

```

1045     } { \fp_to_decimal:N \l__braid_height_fp}
1046   }

```

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

```

1047   \__braid_coordinate:xxxx {-##1-s} {-rev-\l__braid_tmpa_tl-s}
1048   {\fp_eval:n {(##1 - 1) * \__braid_dim_value:n {width} }} {0}
1049
1050   \__braid_coordinate:xxxx {-##1-0} {-rev-\l__braid_tmpa_tl-0}
1051   {\fp_eval:n {(##1 - 1) * \__braid_dim_value:n {width} }}
1052   { \fp_to_decimal:N \l__braid_height_fp}
1053 }

```

Run through any extra floors requested.

```

1054 \seq_map_inline:Nn \l__braid_floors_seq
1055 {
1056   \tl_set:Nx \l__braid_tmpa_tl {\clist_item:nn {##1} {5}}
1057   \__braid_do_floor:Vxxxx \l__braid_tmpa_tl
1058   {\fp_eval:n
1059     {
1060       -1*sign(\__braid_dim_value:n{width})
1061       * \__braid_dim_value:n {floor~ border}
1062       + (\__braid_dim_value:n {width}) * (\clist_item:nn {##1} {1} - 1)
1063     }
1064     pt
1065   }
1066   {\fp_eval:n
1067     {
1068       \l__braid_height_fp + ( \__braid_dim_value:n {height} ) * (\clist_item:nn {##1} {2})
1069     }
1070     pt
1071   }
1072   {\fp_eval:n {
1073     ( (\clist_item:nn {##1} {3}) * \__braid_dim_value:n {width}
1074     + 2 * sign(\__braid_dim_value:n{width})
1075     * \__braid_dim_value:n {floor~ border} ) / \dim_to_fp:n {1cm}
1076   }
1077   }
1078   {\fp_eval:n {
1079     (\clist_item:nn {##1} {4}) * ( \__braid_dim_value:n {height} ) / \dim_to_fp:n {1cm}
1080   }
1081   }
1082 }

```

Keep track of the crossing level for the floor.

```

1083 \int_zero:N \l__braid_crossing_int
1084 \int_incr:N \l__braid_crossing_int
1085
1086 \seq_map_inline:Nn \l__braid_word_seq
1087 {

```

Clear the flags for this segment of the braid word

```

1088   \seq_clear:N \l__braid_crossing_seq
1089   \bool_set_true:N \l__braid_step_level_bool
1090   \bool_set_false:N \l__braid_floor_bool
1091   \bool_set_false:N \l__braid_swap_crossing_bool

```

```

1092     ##1
If we're drawing a floor, do so straightaway.
1093     \bool_if:NT \l__braid_floor_bool
1094     {
1095         \__braid_do_floor:Vxxxx \l__braid_crossing_int
1096         {\fp_eval:n
1097             {
1098                 -1*sign(\__braid_dim_value:n{width})
1099                 * \__braid_dim_value:n {floor~ border}
1100             }
1101             pt
1102         }
1103         {\fp_to_decimal:N \l__braid_height_fp pt}
1104         {\fp_eval:n { \l__braid_width_fp / \dim_to_fp:n {1cm} }}
1105         {\fp_eval:n { ( \__braid_dim_value:n {height} ) / \dim_to_fp:n {1cm}}}
1106     }

```

If we have a crossing, process it.

```

1107     \seq_if_empty:NF \l__braid_crossing_seq
1108     {
1109         \int_set:Nn \l__braid_crossing_long_int
1110         {
1111             % \seq_item:Nn \l__braid_crossing_seq {\seq_count:N \l__braid_crossing_seq}
1112             \seq_item:Nn \l__braid_crossing_seq {1}
1113         }
1114         \int_set:Nn \l__braid_crossing_start_int
1115         {
1116             \int_min:nn
1117             {
1118                 \seq_item:Nn \l__braid_crossing_seq {1}
1119             }
1120             {
1121                 \seq_item:Nn \l__braid_crossing_seq {\seq_count:N \l__braid_crossing_seq}
1122             }
1123         }
1124         \int_set:Nn \l__braid_crossing_end_int
1125         {
1126             \int_max:nn
1127             {
1128                 \seq_item:Nn \l__braid_crossing_seq {1}
1129             }
1130             {
1131                 \seq_item:Nn \l__braid_crossing_seq {\seq_count:N \l__braid_crossing_seq}
1132             }
1133         }
1134     }
1135     \int_set:Nn \l__braid_crossing_width_int
1136     {
1137         \l__braid_crossing_end_int
1138         -
1139         \l__braid_crossing_start_int
1140     }

```

Step through the crossing

```

1141 \int_step_inline:nnn {2} {\seq_count:N \l__braid_crossing_seq}
1142 {
1143   \int_set:Nn \l__braid_tmpa_int {####1}
1144   \int_set:Nn \l__braid_tmpb_int {####1 - 1}

```

Keep track of the current permutation.

```

1145 \prop_get:NxN \l__braid_crossing_permutation_prop
1146 {\seq_item:Nn \l__braid_crossing_seq {\l__braid_tmpa_int}} \l__braid_tmpa_tl
1147 \prop_get:NxN \l__braid_crossing_permutation_prop
1148 {\seq_item:Nn \l__braid_crossing_seq {\l__braid_tmpb_int}} \l__braid_tmpb_tl
1149
1150 \prop_put:NxV \l__braid_crossing_permutation_prop
1151 {\seq_item:Nn \l__braid_crossing_seq {\l__braid_tmpb_int}} \l__braid_tmpa_tl
1152 \prop_put:NxV \l__braid_crossing_permutation_prop
1153 {\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.

```

1154 \prop_get:NxN \l__braid_strands_prop
1155 {\seq_item:Nn \l__braid_crossing_seq {\l__braid_tmpa_int}} \l__braid_tmpa_tl
1156 \prop_get:NxN \l__braid_strands_prop
1157 {\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.

```

1158 % \int_set:Nn \l__braid_crossing_start_factor_int {1}
1159 % \int_set:Nn \l__braid_crossing_end_factor_int {1}
1160 % \int_compare:nT {
1161 %   \seq_item:Nn \l__braid_crossing_seq {\l__braid_tmpa_int}
1162 %   =
1163 %   \l__braid_crossing_long_int
1164 % }
1165 % {
1166 %   \int_set:Nn \l__braid_crossing_start_factor_int {0}
1167 %   \int_set:Nn \l__braid_crossing_end_factor_int {0}
1168 %
1169 %   \int_compare:nT {
1170 %     ####1 = \seq_count:N \l__braid_crossing_seq
1171 %   }
1172 %   {
1173 %     \int_set:Nn \l__braid_crossing_end_factor_int {1}
1174 %   }
1175 %   \int_compare:nT {
1176 %     ####1 = 2
1177 %   }
1178 %   {
1179 %     \int_set:Nn \l__braid_crossing_start_factor_int {1}
1180 %   }
1181 % }
1182
1183 \tl_put_right:Nx \l__braid_tmpa_tl
1184 {
1185   \exp_not:N \__braid_lineto:nn
1186
1187   {\fp_eval:n
1188     {
1189       (\seq_item:Nn \l__braid_crossing_seq {\l__braid_tmpa_int} - 1)

```

```

1190         * \_braid_dim_value:n {width}
1191     }
1192 }
1193 {\fp_eval:n { \l__braid_height_fp + \l__braid_nudge_fp * \l__braid_crossing_start_fa
1194     + \_braid_dim_value:n {height} * (###1 - 2)/(\seq_count:N \l__braid_crossing_s
1195 } }
1196
1197 \exp_not:N \_braid_curveto:nnnnnn
1198
1199 {0}
1200 {\fp_eval:n { \l__braid_control_fp
1201 %     * \l__braid_crossing_start_factor_int
1202     * 1/(\seq_count:N \l__braid_crossing_seq - 1)}}
1203
1204 {0}
1205 {\fp_eval:n {- \l__braid_control_fp
1206 %     * \l__braid_crossing_end_factor_int
1207     * 1/(\seq_count:N \l__braid_crossing_seq - 1)}}
1208
1209 {\fp_eval:n
1210 {
1211     (\seq_item:Nn \l__braid_crossing_seq {\l__braid_tmpb_int} - 1)
1212     * \_braid_dim_value:n {width}
1213 }
1214 }
1215 {\fp_eval:n
1216 {
1217     \l__braid_height_fp
1218     + \_braid_dim_value:n {height} * (###1 - 1)/(\seq_count:N \l__braid_crossing_s
1219     - \l__braid_nudge_fp * \l__braid_crossing_end_factor_int
1220 }
1221 }
1222 }

```

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

```

1223 % \int_set:Nn \l__braid_crossing_start_factor_int {1}
1224 % \int_set:Nn \l__braid_crossing_end_factor_int {1}
1225 % \int_compare:nT {
1226 %     \seq_item:Nn \l__braid_crossing_seq {\l__braid_tmpb_int}
1227 %     =
1228 %     \l__braid_crossing_long_int
1229 % }
1230 % {
1231 %     \int_set:Nn \l__braid_crossing_start_factor_int {0}
1232 %     \int_set:Nn \l__braid_crossing_end_factor_int {0}
1233 %
1234 %     \int_compare:nT {
1235 %         ###1 = \seq_count:N \l__braid_crossing_seq
1236 %     }
1237 %     {
1238 %         \int_set:Nn \l__braid_crossing_end_factor_int {1}
1239 %     }
1240 %     \int_compare:nT {
1241 %         ###1 = 2
1242 %     }

```



```

1243     {
1244     \int_set:Nn \l__braid_crossing_start_factor_int {1}
1245     }
1246 % }
1247
1248 \tl_put_right:Nx \l__braid_tmpb_tl
1249 {
1250 \exp_not:N \__braid_lineto:nn
1251
1252 \fp_eval:n
1253 {
1254 (\seq_item:Nn \l__braid_crossing_seq {\l__braid_tmpb_int} - 1)
1255 * \__braid_dim_value:n {width}
1256 }
1257 }
1258 \fp_eval:n { \l__braid_height_fp + \l__braid_nudge_fp * \l__braid_crossing_start_fa
1259 + \__braid_dim_value:n {height} * (###1 - 2)/(\seq_count:N \l__braid_crossing_s
1260
1261 } }
1262
1263 \exp_not:N \__braid_curveto:nnnnnn
1264
1265 {0}
1266 {
1267 \fp_eval:n {
1268 \l__braid_control_fp * (.5 - \__braid_value:n {gap} * (\seq_count:N \l__braid_c
1269 % * \l__braid_crossing_start_factor_int
1270 }
1271 }
1272
1273 {
1274 \fp_eval:n {
1275 - (.5 - \__braid_value:n {gap} * (\seq_count:N \l__braid_crossing_seq - 1) ) / 3
1276 \__braid_bezier_tangent:nnnnn
1277 {.5 - \__braid_value:n {gap} * (\seq_count:N \l__braid_crossing_seq - 1) }
1278 {0}
1279 {0}
1280 {
1281 (\seq_item:Nn \l__braid_crossing_seq {\l__braid_tmpa_int}
1282 - \seq_item:Nn \l__braid_crossing_seq {\l__braid_tmpb_int})
1283 * \__braid_dim_value:n {width}
1284 }
1285 {
1286 (\seq_item:Nn \l__braid_crossing_seq {\l__braid_tmpa_int}
1287 - \seq_item:Nn \l__braid_crossing_seq {\l__braid_tmpb_int})
1288 * \__braid_dim_value:n {width}
1289 }
1290 }
1291 }
1292 {
1293 \fp_eval:n {
1294 -( .5 - \__braid_value:n {gap} * (\seq_count:N \l__braid_crossing_seq - 1) ) / 3
1295 \__braid_bezier_tangent:nnnnn
1296 {.5 - \__braid_value:n {gap} * (\seq_count:N \l__braid_crossing_seq - 1) }

```

```

1297     {0}
1298     {
1299         \l__braid_control_fp * 1/(\seq_count:N \l__braid_crossing_seq - 1)
1300 %      * \l__braid_crossing_start_factor_int
1301     }
1302     {
1303         \__braid_dim_value:n {height} * 1/(\seq_count:N \l__braid_crossing_seq - 1)
1304         - \l__braid_nudge_fp * \l__braid_crossing_start_factor_int
1305         - \l__braid_nudge_fp * \l__braid_crossing_end_factor_int
1306         - \l__braid_control_fp * 1/(\seq_count:N \l__braid_crossing_seq - 1)
1307 %      * \l__braid_crossing_end_factor_int
1308     }
1309     {
1310         \__braid_dim_value:n {height} * 1/(\seq_count:N \l__braid_crossing_seq - 1)
1311         - \l__braid_nudge_fp * \l__braid_crossing_start_factor_int
1312         - \l__braid_nudge_fp * \l__braid_crossing_end_factor_int
1313     }
1314 }
1315 }
1316 {
1317     {
1318         \fp_eval:n {
1319             (\seq_item:Nn \l__braid_crossing_seq {\l__braid_tmpb_int} - 1)
1320             * \__braid_dim_value:n {width} +
1321             \__braid_bezier_point:nnnnn
1322             {.5 - \__braid_value:n {gap} * (\seq_count:N \l__braid_crossing_seq - 1) }
1323             {0}
1324             {0}
1325             {
1326                 (\seq_item:Nn \l__braid_crossing_seq {\l__braid_tmpa_int}
1327                 - \seq_item:Nn \l__braid_crossing_seq {\l__braid_tmpb_int})
1328                 * \__braid_dim_value:n {width}
1329             }
1330             {
1331                 (\seq_item:Nn \l__braid_crossing_seq {\l__braid_tmpa_int}
1332                 - \seq_item:Nn \l__braid_crossing_seq {\l__braid_tmpb_int})
1333                 * \__braid_dim_value:n {width}
1334             }
1335         }
1336     }
1337     {
1338         \fp_eval:n {
1339             \l__braid_height_fp
1340             + \l__braid_nudge_fp * \l__braid_crossing_start_factor_int
1341             + \__braid_dim_value:n {height} * (###1 - 2)/(\seq_count:N \l__braid_crossing_s
1342             +
1343             \__braid_bezier_point:nnnnn
1344             {.5 - \__braid_value:n {gap} * (\seq_count:N \l__braid_crossing_seq - 1) }
1345             {0}
1346             {
1347                 \l__braid_control_fp * 1/(\seq_count:N \l__braid_crossing_seq - 1)
1348 %      * \l__braid_crossing_start_factor_int
1349             }
1350         }

```

```

1351         \l__braid_dim_value:n {height} * 1/(\seq_count:N \l__braid_crossing_seq - 1)
1352         - \l__braid_nudge_fp * \l__braid_crossing_start_factor_int
1353         - \l__braid_nudge_fp * \l__braid_crossing_end_factor_int
1354         - \l__braid_control_fp * 1/(\seq_count:N \l__braid_crossing_seq - 1)
1355 %         * \l__braid_crossing_end_factor_int
1356     }
1357     {\l__braid_dim_value:n {height} * 1/(\seq_count:N \l__braid_crossing_seq - 1)
1358     - \l__braid_nudge_fp * \l__braid_crossing_start_factor_int
1359     - \l__braid_nudge_fp * \l__braid_crossing_end_factor_int
1360     }
1361 }
1362 }
1363
1364 \exp_not:N \l__braid_moveto:nn
1365 {
1366     \fp_eval:n {
1367         (\seq_item:Nn \l__braid_crossing_seq {\l__braid_tmpb_int} - 1)
1368         * \l__braid_dim_value:n {width} +
1369         \l__braid_bezier_point:nnnnn
1370         { .5 + \l__braid_value:n {gap} * (\seq_count:N \l__braid_crossing_seq - 1) }
1371         { 0 }
1372         { 0 }
1373         {
1374             (\seq_item:Nn \l__braid_crossing_seq {\l__braid_tmpa_int}
1375             - \seq_item:Nn \l__braid_crossing_seq {\l__braid_tmpb_int})
1376             * \l__braid_dim_value:n {width}
1377         }
1378         {
1379             (\seq_item:Nn \l__braid_crossing_seq {\l__braid_tmpa_int}
1380             - \seq_item:Nn \l__braid_crossing_seq {\l__braid_tmpb_int})
1381             * \l__braid_dim_value:n {width}
1382         }
1383     }
1384 }
1385 {
1386     \fp_eval:n {
1387         \l__braid_height_fp
1388         + \l__braid_nudge_fp * \l__braid_crossing_start_factor_int
1389         + \l__braid_dim_value:n {height} * (####1 - 2)/(\seq_count:N \l__braid_crossing_s
1390         +
1391         \l__braid_bezier_point:nnnnn
1392         { .5 + \l__braid_value:n {gap} * (\seq_count:N \l__braid_crossing_seq - 1) }
1393         { 0 }
1394         {
1395             \l__braid_control_fp * 1/(\seq_count:N \l__braid_crossing_seq - 1)
1396 %             * \l__braid_crossing_start_factor_int
1397         }
1398         {
1399             \l__braid_dim_value:n {height} * 1/(\seq_count:N \l__braid_crossing_seq - 1)
1400             - \l__braid_nudge_fp * \l__braid_crossing_start_factor_int
1401             - \l__braid_nudge_fp * \l__braid_crossing_end_factor_int
1402             - \l__braid_control_fp * 1/(\seq_count:N \l__braid_crossing_seq - 1)
1403 %             * \l__braid_crossing_end_factor_int
1404         }

```

```

1405         {\_braid_dim_value:n {height} * 1/(\seq_count:N \l__braid_crossing_seq - 1)
1406         - \l__braid_nudge_fp * \l__braid_crossing_start_factor_int
1407         - \l__braid_nudge_fp * \l__braid_crossing_end_factor_int
1408     }
1409 }
1410 }
1411
1412 \exp_not:N \_braid_curveto:nnnnnn
1413
1414 {
1415     \fp_eval:n {
1416         (.5 - \_braid_value:n {gap} * (\seq_count:N \l__braid_crossing_seq - 1) ) / 3 *
1417         \_braid_bezier_tangent:nnnnn
1418         {.5 + \_braid_value:n {gap} * (\seq_count:N \l__braid_crossing_seq - 1) }
1419         {0}
1420         {0}
1421         {
1422             (\seq_item:Nn \l__braid_crossing_seq {\l__braid_tmpa_int}
1423             - \seq_item:Nn \l__braid_crossing_seq {\l__braid_tmpb_int})
1424             * \_braid_dim_value:n {width}
1425         }
1426         {
1427             (\seq_item:Nn \l__braid_crossing_seq {\l__braid_tmpa_int}
1428             - \seq_item:Nn \l__braid_crossing_seq {\l__braid_tmpb_int})
1429             * \_braid_dim_value:n {width}
1430         }
1431     }
1432 }
1433 {
1434     \fp_eval:n {
1435         (.5 - \_braid_value:n {gap} * (\seq_count:N \l__braid_crossing_seq - 1) ) / 3 *
1436         \_braid_bezier_tangent:nnnnn
1437         {.5 + \_braid_value:n {gap} * (\seq_count:N \l__braid_crossing_seq - 1) }
1438         {0}
1439         {
1440             \l__braid_control_fp * 1/(\seq_count:N \l__braid_crossing_seq - 1)
1441             * \l__braid_crossing_start_factor_int
1442         }
1443         {
1444             \_braid_dim_value:n {height} * 1/(\seq_count:N \l__braid_crossing_seq - 1)
1445             - \l__braid_nudge_fp * \l__braid_crossing_start_factor_int
1446             - \l__braid_nudge_fp * \l__braid_crossing_end_factor_int
1447             - \l__braid_control_fp * 1/(\seq_count:N \l__braid_crossing_seq - 1)
1448             * \l__braid_crossing_end_factor_int
1449         }
1450         {\_braid_dim_value:n {height} * 1/(\seq_count:N \l__braid_crossing_seq - 1)
1451         - \l__braid_nudge_fp * \l__braid_crossing_start_factor_int
1452         - \l__braid_nudge_fp * \l__braid_crossing_end_factor_int
1453     }
1454 }
1455 }
1456
1457 {0}
1458 {\fp_eval:n {

```

```

1459     - \l__braid_control_fp * (.5 - \__braid_value:n {gap} * (\seq_count:N \l__braid
1460 %     * \l__braid_crossing_end_factor_int
1461     * 1/(\seq_count:N \l__braid_crossing_seq - 1))
1462   }
1463
1464   {\fp_eval:n
1465   {
1466     (\seq_item:Nn \l__braid_crossing_seq {\l__braid_tmpa_int} - 1)
1467     * \__braid_dim_value:n {width}
1468   }
1469   }
1470   {\fp_eval:n
1471   {
1472     \l__braid_height_fp
1473     + \__braid_dim_value:n {height} * (####1 - 1)/(\seq_count:N \l__braid_crossing_s
1474     - \l__braid_nudge_fp * \l__braid_crossing_end_factor_int
1475   }
1476   }
1477
1478   }

```

Now put those new strands back in the prop.

```

1479   \prop_put:NxV \l__braid_strands_prop
1480   {\seq_item:Nn \l__braid_crossing_seq {\l__braid_tmpb_int}} \l__braid_tmpa_tl
1481   \prop_put:NxV \l__braid_strands_prop
1482   {\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.

```

1483   \int_compare:nT
1484   {
1485     \int_max:nn
1486     {
1487       \seq_item:Nn \l__braid_crossing_seq {####1 - 1}
1488     }
1489     {
1490       \seq_item:Nn \l__braid_crossing_seq {####1}
1491     }
1492     -
1493     \int_min:nn
1494     {
1495       \seq_item:Nn \l__braid_crossing_seq {####1 - 1}
1496     }
1497     {
1498       \seq_item:Nn \l__braid_crossing_seq {####1}
1499     }
1500     > 1
1501   }
1502   {
1503     \int_step_inline:nnnn
1504     {
1505       \int_min:nn
1506       {
1507         \seq_item:Nn \l__braid_crossing_seq {####1 - 1}
1508       }

```

```

1509     {
1510         \seq_item:Nn \l__braid_crossing_seq {####1}
1511     }
1512     + 1}
1513 {1}
1514 {
1515     \int_max:nn
1516     {
1517         \seq_item:Nn \l__braid_crossing_seq {####1 - 1}
1518     }
1519     {
1520         \seq_item:Nn \l__braid_crossing_seq {####1}
1521     }
1522     - 1
1523 }
1524 {
1525
1526     \prop_get:Nn \l__braid_strands_prop {#####1} \l__braid_tmpa_tl
1527     \tl_put_right:Nx \l__braid_tmpa_tl
1528     {
1529         \exp_not:N \l__braid_lineto:nn
1530         {\fp_eval:n {(#####1 - 1) * \l__braid_dim_value:n {width} }}
1531         {\fp_eval:n
1532             {
1533                 \l__braid_height_fp + \l__braid_nudge_fp
1534                 + .5 * \l__braid_control_fp / (\seq_count:N \l__braid_crossing_seq - 1)
1535                 + \l__braid_dim_value:n {height} * (####1 - 2)/(\seq_count:N \l__braid_crossing_seq - 1)
1536             }
1537         }
1538         \exp_not:N \l__braid_moveto:nn
1539         {\fp_eval:n {(#####1 - 1) * \l__braid_dim_value:n {width} }}
1540         {\fp_eval:n
1541             {
1542                 \l__braid_height_fp
1543                 - \l__braid_nudge_fp - .5 * \l__braid_control_fp / (\seq_count:N \l__braid_crossing_seq - 1)
1544                 + \l__braid_dim_value:n {height} * (####1 - 1)/(\seq_count:N \l__braid_crossing_seq - 1)
1545             }
1546         }
1547     }
1548
1549     \prop_put:NnV \l__braid_strands_prop {#####1} \l__braid_tmpa_tl
1550 }
1551 }

```

Reset the current long

```

1552     \int_compare:nTF
1553     {
1554         \seq_item:Nn \l__braid_crossing_seq {\l__braid_tmpa_int}
1555         =
1556         \l__braid_crossing_long_int
1557     }
1558     {
1559         \int_set:Nn \l__braid_crossing_long_int {\seq_item:Nn \l__braid_crossing_seq {\l__braid_tmpa_int}}
1560     }
1561     {

```

```

1562     \int_compare:nT
1563     {
1564       \seq_item:Nn \l__braid_crossing_seq {\l__braid_tmpb_int}
1565       =
1566       \l__braid_crossing_long_int
1567     }
1568     {
1569       \int_set:Nn \l__braid_crossing_long_int {\seq_item:Nn \l__braid_crossing_seq {\l__br
1570     }
1571
1572     }
1573   }
1574 }

```

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

```

1575     \bool_if:NT \l__braid_step_level_bool
1576     {
1577       \fp_add:Nn \l__braid_height_fp { \__braid_dim_value:n {height} }
1578
1579       \int_step_inline:nmmn {1} {1} {\l__braid_strands_int}
1580       {
1581         \prop_get:NnN \l__braid_crossing_permutation_prop
1582         {###1} \l__braid_tmpb_tl
1583         \prop_get:NVN \l__braid_inverse_prop
1584         \l__braid_tmpb_tl \l__braid_tmpa_tl
1585
1586         \__braid_coordinate:xxxx
1587         {-\l__braid_tmpb_tl-\int_use:N \l__braid_crossing_int}
1588         {-rev-\l__braid_tmpa_tl-\int_use:N \l__braid_crossing_int }
1589         {\fp_eval:n { (###1 - 1) * \__braid_dim_value:n {width} }}
1590         {\fp_to_decimal:N \l__braid_height_fp}
1591       }
1592
1593       \int_incr:N \l__braid_crossing_int
1594     }
1595 }
1596
1597 \fp_add:Nn \l__braid_height_fp
1598 {
1599   sign(\__braid_dim_value:n {height})
1600   * \__braid_dim_value:n {border~ height}
1601 }

```

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

```

1602     \int_step_inline:nmmn {1} {1} {\l__braid_strands_int}
1603     {
1604       \prop_get:NxN \l__braid_strands_prop {##1} \l__braid_tmpa_tl
1605       \prop_get:NxN \l__braid_permutation_prop {##1} \l__braid_tmpb_tl
1606
1607       \tl_put_right:Nx \l__braid_tmpa_tl {
1608         \exp_not:N \__braid_lineto:nn
1609         {\fp_eval:n { (##1 - 1) * \__braid_dim_value:n {width} }}
1610         {\fp_to_decimal:N \l__braid_height_fp}
1611         coordinate (-rev-##1-e)

```

```

1612     coordinate (-\l__braid_tmpb_tl-e)
1613     ;
1614   }
1615
1616   \prop_put:NnV \l__braid_strands_prop {##1} \l__braid_tmpa_tl
1617 }

```

This is where we actually carry out the drawing commands.

```

1618 \int_step_inline:nmmn {1} {1} {\l__braid_strands_int}
1619 {
1620   \prop_get:NnN \l__braid_strands_prop {##1} \l__braid_tmpa_tl
1621   \tl_use:N \l__braid_tmpa_tl
1622 }

```

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

```

1623 \tl_if_empty:cF {tikz@fig@name}
1624 {
1625   \tl_gset:cn {pgf@sh@ns@ \tl_use:c{tikz@fig@name} }{rectangle}
1626   \tl_gset:cx {pgf@sh@np@ \tl_use:c{tikz@fig@name} }{%
1627     \exp_not:N\def
1628     \exp_not:N\southwest
1629     {
1630       \exp_not:N\pgfqpoint
1631       {
1632         \fp_to_dim:n
1633         {
1634           min(0,
1635             (\l__braid_strands_int - 1)
1636             *
1637             (\__braid_dim_value:n {width})
1638           )
1639         }
1640       }
1641       {
1642         \fp_to_dim:n
1643         {
1644           min(0,
1645             \l__braid_length_int * (\__braid_dim_value:n {height})
1646             + 2 * sign(\__braid_dim_value:n {height}) *
1647             \__braid_dim_value:n {border~ height}
1648           )
1649         }
1650       }
1651     }
1652     \exp_not:N\def
1653     \exp_not:N\northeast
1654     {
1655       \exp_not:N\pgfqpoint
1656       {
1657         \fp_to_dim:n
1658         {
1659           max(0,
1660             (\l__braid_strands_int - 1)
1661             *
1662             (\__braid_dim_value:n {width})

```



```

1663         )
1664     }
1665 }
1666 {
1667     \fp_to_dim:n
1668     {
1669         max(0,
1670         \l__braid_length_int * (\__braid_dim_value:n {height})
1671         + 2 * sign(\__braid_dim_value:n {height}) *
1672         \__braid_dim_value:n {border~ height}
1673         )
1674     }
1675 }
1676 }
1677 }%
1678 \pgfgettransform\l__braid_tmpa_tl
1679 \tl_gset:cV {pgf@sh@nt@ \tl_use:c{tikz@fig@name} } \l__braid_tmpa_tl
1680 \tl_gset:cV {pgf@sh@pi@ \tl_use:c{tikz@fig@name} } \pgfpictureid
1681 }
1682 \end{scope}
1683 }

```

(End definition for __braid_render:.)

```

\__braid_moveto:nnn These are our interfaces to the TikZ code.
\__braid_lineto:nn 1684 \cs_new_nopar:Npn \__braid_moveto:nn #1#2
\__braid_curveto:nnnnnn 1685 {
\__braid_coordinate:nnnn 1686     (#1 pt, #2 pt)
1687 }
1688 \cs_new_nopar:Npn \__braid_lineto:nn #1#2
1689 {
1690     -- (#1 pt, #2 pt)
1691 }
1692 \cs_new_nopar:Npn \__braid_curveto:nnnnnn #1#2#3#4#5#6
1693 {
1694     % -- +(5 pt, 0) -- +(0 pt, 0pt)
1695     % -- +( #1 pt, #2 pt) -- (#5 pt + #3 pt, #6 pt + #4 pt) -- (#5 pt, #6 pt)
1696     .. controls +( #1 pt, #2 pt) and +( #3 pt, #4 pt)
1697     .. (#5 pt, #6 pt)
1698 }
1699 \cs_new_nopar:Npn \__braid_coordinate:nnnn #1#2#3#4
1700 {
1701     \coordinate[alias=#2] (#1) at (#3 pt,#4 pt);
1702 }
1703 \cs_generate_variant:Nn \__braid_coordinate:nnnn {xxxx}

```

(End definition for __braid_moveto:nn and others.)

```

\__braid_bezier_point:nnnnn Used to calculate intermediate points and tangents on a bezier curve.
\__braid_bezier_tangent:nnnnn 1704 \cs_new_nopar:Npn \__braid_bezier_point:nnnnn #1#2#3#4#5
1705 {
1706     \fp_eval:n
1707     {
1708         (1 - (#1)) * (1 - (#1)) * (1 - (#1)) * (#2)
1709         +

```

```

1710     3 * (1 - (#1)) * (1 - (#1)) * (#1) * (#3)
1711     +
1712     3 * (1 - (#1)) * (#1) * (#1) * (#4)
1713     +
1714     (#1) * (#1) * (#1) * (#5)
1715   }
1716 }
1717 \cs_new_nopar:Npn \__braid_bezier_tangent:nnnnn #1#2#3#4#5
1718 {
1719   \fp_eval:n
1720   {
1721     3 * (1 - (#1)) * (1 - (#1)) * (#3 - (#2))
1722     +
1723     6 * (1 - (#1)) * (#1) * (#4 - (#3))
1724     +
1725     3 * (#1) * (#1) * (#5 - (#4))
1726   }
1727 }
1728 \cs_new_nopar:Npn \__braid_do_floor:nnnnn #1#2#3#4#5
1729 {
1730   \pic[pic~ type=floor,
1731     xscale=#4,
1732     yscale=#5,
1733     at={#2,#3},
1734     braid/every~ floor/.try,
1735     braid/floor~#1/.try,
1736   ];
1737 }
1738 \cs_generate_variant:Nn \__braid_do_floor:nnnnn {Vxxxx}

```

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

```

1739 \ExplSyntaxOff

```