

The `braids` package: codebase

Andrew Stacey
loopspace@mathforge.org

v2.0 from 2019/03/20

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   2019/03/20 v2.0 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_length_int
574 \int_new:N \l__braid_strands_int
575 \int_new:N \l__braid_crossing_int
576 \int_new:N \l__braid_anchor_level_int
577 \int_new:N \l__braid_floor_int
578 \seq_new:N \l__braid_word_seq
579 \seq_new:N \l__braid_crossing_seq
580 \seq_new:N \l__braid_anchor_seq
581 \seq_new:N \l__braid_floors_seq
582 \str_new:N \l__braid_tmpa_str
583 \str_new:N \l__braid_sup_str
584 \str_set:Nn \l__braid_sup_str {^}

```

```

585 \str_new:N \l__braid_sub_str
586 \str_set:Nn \l__braid_sub_str {}
587 \str_new:N \l__braid_hyphen_str
588 \str_set:Nn \l__braid_hyphen_str {-}
589 \str_new:N \l__braid_bar_str
590 \str_set:Nn \l__braid_bar_str {}
591 \str_new:N \l__braid_one_str
592 \str_set:Nn \l__braid_one_str {1}
593 \bool_new:N \l__braid_step_level_bool
594 \bool_new:N \l__braid_swap_crossing_bool
595 \bool_new:N \l__braid_floor_bool
596 \prop_new:N \l__braid_strands_prop
597 \prop_new:N \l__braid_permutation_prop
598 \prop_new:N \l__braid_crossing_permutation_prop
599 \prop_new:N \l__braid_inverse_prop
600 \prop_new:N \l__braid_anchor_prop

```

Our interface is through a TikZ pic.

```

601 \tikzset{
602   braid/.pic={
603     code={
604       \__braid_parse_word:n {#1}
605       \__braid_count:
606       \__braid_render:
607     }
608   },
609   floor/.pic={
610     code={
611       \path[pic~ actions, draw=none] (0,0) rectangle (1,1);
612       \path[pic~ actions, fill=none] (0,0) -- (1,0) (0,1) -- (1,1);
613     }
614   },
615   /tikz/braid/.search~ also={/tikz},
616   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).

```

617   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).

```

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

```

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

```

619   height/.initial=-1cm,

```

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

```

620   width/.initial=1cm,

```

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

```

621   gap/.initial=.05,

```

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

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

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

```
623 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.

```
624 add~ floor/.code={
625   \seq_push:Nn \l__braid_floors_seq {#1}
626 },
```

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

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

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

```
628 nudge~ factor/.initial=.05
629 }
```

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

```
630 \cs_new_nopar:Npn \__braid_parse_word:n #1
631 {
632   \seq_clear:N \l__braid_word_seq
633   \tl_clear:N \l__braid_tmpa_tl
634   \tl_set:Nn \l__braid_tmpb_tl {#1}
635
636   \bool_until_do:nn { \tl_if_empty_p:N \l__braid_tmpb_tl }
637   {
```

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.

```
638   \str_set:Nx \l__braid_tmpa_str {\tl_head:N \l__braid_tmpb_tl}
639   \tl_set:Nx \l__braid_tmpb_tl {\tl_tail:N \l__braid_tmpb_tl}
640   \str_case_e:nnTF {\l__braid_tmpa_str}
641   {
```

Underscore introduces the crossing numbers

```
642   {_}
643   {
644     \tl_put_right:Nx \l__braid_tmpa_tl
645     {
646       \exp_not:N \__braid_parse_index:n {\tl_head:N \l__braid_tmpb_tl}
647     }
648     \tl_set:Nx \l__braid_tmpb_tl {\tl_tail:N \l__braid_tmpb_tl}
649   }
```

Power is used to indicate inverse.

```

650     {~}
651     {
652       \tl_put_left:Nx \l__braid_tmpa_tl
653       {
654         \exp_not:N \__braid_parse_exponent:n {\tl_head:N \l__braid_tmpb_tl}
655       }
656       \tl_set:Nx \l__braid_tmpb_tl {\tl_tail:N \l__braid_tmpb_tl}
657     }

```

Bar is for floors.

```

658     {}
659     {
660       \tl_if_empty:NF \l__braid_tmpa_tl
661       {
662         \seq_put_right:NV \l__braid_word_seq \l__braid_tmpa_tl
663         \tl_clear:N \l__braid_tmpa_tl
664       }
665
666       \tl_set:Nn \l__braid_tmpa_tl {
667         \bool_set_false:N \l__braid_step_level_bool
668         \bool_set_true:N \l__braid_floor_bool
669       }
670       \seq_put_right:NV \l__braid_word_seq \l__braid_tmpa_tl
671       \tl_clear:N \l__braid_tmpa_tl
672     }

```

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

```

673     {-}
674     {
675       \tl_put_right:Nn \l__braid_tmpa_tl
676       {
677         \bool_set_false:N \l__braid_step_level_bool
678       }
679     }

```

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.

```

680     {1}
681     {
682       \tl_if_empty:NF \l__braid_tmpa_tl
683       {
684         \seq_put_right:NV \l__braid_word_seq \l__braid_tmpa_tl
685         \tl_clear:N \l__braid_tmpa_tl
686       }
687       \tl_put_right:Nn \l__braid_tmpa_tl {\__braid_do_identity:}
688     }

```

Ignore spaces.

```

689     {~}
690     {
691     }
692   }
693   {
694   }
695   {

```

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

```

696     \tl_if_empty:NF \l__braid_tmpa_tl
697     {
698       \seq_put_right:NV \l__braid_word_seq \l__braid_tmpa_tl
699       \tl_clear:N \l__braid_tmpa_tl
700     }
701   }
702 }

```

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

```

703 \tl_if_empty:NF \l__braid_tmpa_tl
704 {
705   \seq_put_right:NV \l__braid_word_seq \l__braid_tmpa_tl
706   \tl_clear:N \l__braid_tmpa_tl
707 }
708 }

```

(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.

```

709 \cs_new_nopar:Npn \__braid_parse_index:n #1
710 {
711   \seq_set_from_clist:Nn \l__braid_crossing_seq {#1}
712   \int_compare:nT {\seq_count:N \l__braid_crossing_seq == 1}
713   {
714     \seq_put_right:Nx \l__braid_crossing_seq {\int_eval:n {#1 + 1} }
715   }
716   \bool_if:NT \l__braid_swap_crossing_bool
717   {
718     \seq_pop_left:NN \l__braid_crossing_seq \l__braid_tmpa_tl
719     \seq_put_right:NV \l__braid_crossing_seq \l__braid_tmpa_tl
720   }
721 }

```

(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..

```

722 \cs_new_nopar:Npn \__braid_parse_exponent:n #1
723 {
724   \int_compare:nTF {#1 == -1}
725   {
726     \bool_set_true:N \l__braid_swap_crossing_bool
727   }
728   {
729     \bool_set_false:N \l__braid_swap_crossing_bool
730   }
731 }

```

(End definition for __braid_parse_exponent:n.)

`__braid_do_identity:`

```
732 \cs_new_nopar:Npn \__braid_do_identity:
733 {
734 }
```

(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.

```
735 \cs_new_nopar:Npn \__braid_count:
736 {
737   \int_zero:N \l__braid_length_int
738   \int_set:Nn \l__braid_strands_int {\__braid_value:n {number-of~strands}}
739   \prop_clear:N \l__braid_permutation_prop
740   \prop_clear:N \l__braid_crossing_permutation_prop
741   \prop_clear:N \l__braid_anchor_prop
742   \prop_clear:N \l__braid_inverse_prop
743
744   \seq_map_inline:Nn \l__braid_word_seq
745   {
```

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

```
746   \seq_clear:N \l__braid_crossing_seq
747   \bool_set_true:N \l__braid_step_level_bool
748   \bool_set_false:N \l__braid_swap_crossing_bool
```

Run the details of this crossing.

```
749   ##1
```

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

```
750   \bool_if:NT \l__braid_step_level_bool
751   {
752     \int_incr:N \l__braid_length_int
753   }
```

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

```
754   \seq_if_empty:NF \l__braid_crossing_seq
755   {
756     \int_set:Nn \l__braid_strands_int
757     {
758       \int_max:nn
759       {
760         \int_max:nn {\l__braid_strands_int}
761         { \seq_item:Nn \l__braid_crossing_seq {1}}
762       }
763       {
764         \seq_item:Nn \l__braid_crossing_seq {2}
765       }
766     }
767   }
768 }
```

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.

```

769 \int_step_inline:nmmm {1} {1} {\l__braid_strands_int}
770 {
771   \prop_put:Nnn \l__braid_permutation_prop {##1} {##1}
772   \prop_put:Nnn \l__braid_anchor_prop {##1} {##1}
773   \prop_put:Nnn \l__braid_crossing_permutation_prop {##1} {##1}
774 }

```

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.

```

775 \tl_set:Nx \l__braid_tmpa_tl {\__braid_value:n {anchor}}
776 \seq_set_split:NnV \l__braid_anchor_seq {-} \l__braid_tmpa_tl
777
778 \tl_set:Nx \l__braid_tmpa_tl {\seq_item:Nn \l__braid_anchor_seq {1}}
779 \tl_if_eq:VnTF \l__braid_tmpa_tl {rev}
780 {
781   \tl_set:Nx \l__braid_anchor_strand_tl {\seq_item:Nn \l__braid_anchor_seq {2}}
782   \tl_set:Nx \l__braid_anchor_level_tl {\seq_item:Nn \l__braid_anchor_seq {3}}
783 }
784 {
785   \tl_set:Nx \l__braid_anchor_strand_tl {\seq_item:Nn \l__braid_anchor_seq {1}}
786   \tl_set:Nx \l__braid_anchor_level_tl {\seq_item:Nn \l__braid_anchor_seq {2}}
787 }

```

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.

```

788 \tl_if_eq:VnTF \l__braid_anchor_level_tl {s}
789 {
790   \int_set:Nn \l__braid_anchor_level_int {-1}
791 }
792 {
793   \tl_if_eq:VnTF \l__braid_anchor_level_tl {e}
794   {
795     \int_set:Nn \l__braid_anchor_level_int {-1}
796   }
797   {
798     \int_set:Nn \l__braid_anchor_level_int
799     {\tl_use:N \l__braid_anchor_level_tl}
800   }
801 }
802
803 \int_zero:N \l__braid_crossing_int
804 \int_incr:N \l__braid_crossing_int
805 \seq_map_inline:Nn \l__braid_word_seq
806 {
807   \bool_set_true:N \l__braid_step_level_bool
808   \seq_clear:N \l__braid_crossing_seq
809   \bool_set_false:N \l__braid_swap_crossing_bool
810   ##1

```

```

811 \seq_if_empty:NF \l__braid_crossing_seq
812 {
813   \prop_get:NxN \l__braid_permutation_prop
814   {
815     \seq_item:Nn \l__braid_crossing_seq {1}
816   } \l__braid_tmpa_tl
817   \prop_get:NxN \l__braid_permutation_prop
818   {
819     \seq_item:Nn \l__braid_crossing_seq {2}
820   } \l__braid_tmpb_tl
821
822   \prop_put:NxV \l__braid_permutation_prop
823   {
824     \seq_item:Nn \l__braid_crossing_seq {2}
825   } \l__braid_tmpa_tl
826   \prop_put:NxV \l__braid_permutation_prop
827   {
828     \seq_item:Nn \l__braid_crossing_seq {1}
829   } \l__braid_tmpb_tl
830 }

```

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

```

831 \int_compare:nT {\l__braid_crossing_int = \l__braid_anchor_level_int}
832 {
833   \prop_set_eq:NN \l__braid_anchor_prop \l__braid_permutation_prop
834 }
835 \bool_if:NT \l__braid_step_level_bool
836 {
837   \int_incr:N \l__braid_crossing_int
838 }
839 }

```

This inverts the anchor permutation.

```

840 \int_step_inline:nnnn {1} {1} {\l__braid_strands_int}
841 {
842   \prop_get:NnN \l__braid_anchor_prop {##1} \l__braid_tmpa_tl
843   \prop_put:NVn \l__braid_inverse_prop \l__braid_tmpa_tl {##1}
844 }
845 \prop_set_eq:NN \l__braid_anchor_prop \l__braid_inverse_prop

```

This inverts the full permutation.

```

846 \int_step_inline:nnnn {1} {1} {\l__braid_strands_int}
847 {
848   \prop_get:NnN \l__braid_permutation_prop {##1} \l__braid_tmpa_tl
849   \prop_put:NVn \l__braid_inverse_prop \l__braid_tmpa_tl {##1}
850 }

```

Now that we have the inverse, we can figure out our anchor. First, see if we requested a strand by its position at the end of the braid.

```

851 \tl_set:Nx \l__braid_tmpa_tl {\seq_item:Nn \l__braid_anchor_seq {1}}
852 \tl_if_eq:VnT \l__braid_tmpa_tl {rev}
853 {
854   \prop_get:NVN \l__braid_permutation_prop
855   \l__braid_anchor_strand_tl \l__braid_anchor_strand_tl
856 }

```



```

857 \tl_if_eq:VnF \l__braid_anchor_level_tl {s}
858 {
859   \tl_if_eq:VnTF \l__braid_anchor_level_tl {e}
860   {
861     \prop_get:NVN \l__braid_inverse_prop
862     \l__braid_anchor_strand_tl \l__braid_anchor_strand_tl
863   }
864   {
865     \prop_get:NVN \l__braid_anchor_prop
866     \l__braid_anchor_strand_tl \l__braid_anchor_strand_tl
867   }
868 }
869 }

```

(End definition for `__braid_count:NNN`.)

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

```

\__braid_value:n
870 \cs_new_nopar:Npn \__braid_dim_value:n #1
871 {
872   \dim_to_fp:n {\pgfkeysvalueof{/tikz/braid/#1}}
873 }
874 \cs_new_nopar:Npn \__braid_value:n #1
875 {
876   \pgfkeysvalueof{/tikz/braid/#1}
877 }

```

(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.

```

878 \cs_generate_variant:Nn \prop_get:NnN {NxN}
879 \cs_generate_variant:Nn \prop_put:Nnn {NxV}
880 \cs_generate_variant:Nn \tl_if_eq:nnTF {VnTF}
881 \cs_generate_variant:Nn \tl_if_eq:nnF {VnF}
882 \cs_generate_variant:Nn \tl_if_eq:nnT {VnT}
883
884 \cs_new_nopar:Npn \__braid_render:
885 {
886   \fp_set:Nn \l__braid_anchor_x_fp { - 1 * (\tl_use:N \l__braid_anchor_strand_tl - 1) * \__b
887
888   \tl_if_eq:VnTF \l__braid_anchor_level_tl {s}
889   {
890     \fp_set:Nn \l__braid_anchor_y_fp {0}
891   }
892   {
893     \tl_if_eq:VnTF \l__braid_anchor_level_tl {e}
894     {
895       \fp_set:Nn \l__braid_anchor_y_fp {
896         -1 * \l__braid_length_int * \__braid_dim_value:n {height}
897         - sign(\__braid_dim_value:n {height})
898         * 2 * \__braid_dim_value:n {border~ height}
899       }
900     }
901     {
902       \fp_set:Nn \l__braid_anchor_y_fp {

```

```

903     -1 * \l__braid_anchor_level_tl * \l__braid_dim_value:n {height}
904     - sign(\l__braid_dim_value:n {height})
905     * \l__braid_dim_value:n {border~ height}
906   }
907 }
908 }
909
910 \begin{scope}[
911   shift={
912     (\fp_to_decimal:N \l__braid_anchor_x_fp pt,
913     \fp_to_decimal:N \l__braid_anchor_y_fp pt
914     )
915   }
916 ]

```

Initialise a prop for the individual strands.

```

917 \prop_clear:N \l__braid_strands_prop

```

Initialise some lengths.

```

918 \fp_zero:N \l__braid_height_fp
919 \fp_zero:N \l__braid_nudge_fp
920 \fp_zero:N \l__braid_control_fp

```

This holds our current height of our strands.

```

921 \fp_set:Nn \l__braid_height_fp
922 {
923   sign(\l__braid_dim_value:n {height})
924   * \l__braid_dim_value:n {border~ height}
925 }

```

This holds the total width of our strands.

```

926 \fp_set:Nn \l__braid_width_fp
927 {
928   (\l__braid_strands_int - 1) * \l__braid_dim_value:n {width}
929   + 2 * sign(\l__braid_dim_value:n {width})
930   * \l__braid_dim_value:n {floor~ border}
931 }

```

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

```

932 \fp_set:Nn \l__braid_nudge_fp
933 {
934   \l__braid_value:n {nudge~ factor} * \l__braid_dim_value:n {height}
935 }

```

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

```

936 \fp_set:Nn \l__braid_control_fp
937 {
938   \l__braid_value:n {control~ factor} * \l__braid_dim_value:n {height}
939 }
940 \fp_sub:Nn \l__braid_control_fp {\l__braid_nudge_fp}

```

Initialise our strand paths with a \draw.

```

941 \int_step_inline:nnmn {1} {1} {\l__braid_strands_int}
942 {
943   \prop_get:NnN \l__braid_inverse_prop {##1} \l__braid_tmpa_tl

```

```

944 \prop_put:Nnx \l__braid_strands_prop {##1}
945 {
946   \exp_not:N \draw[
947     braid/every~ strand/.try,
948     braid/strand~ ##1/.try
949   ]
950   \exp_not:N \__braid_moveto:nn {
951     \fp_eval:n {(##1 - 1) * \__braid_dim_value:n {width} }
952   } {0}
953   \exp_not:N \__braid_lineto:nn {
954     \fp_eval:n {(##1 - 1) * \__braid_dim_value:n {width} }
955   } { \fp_to_decimal:N \l__braid_height_fp}
956 }

```

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

```

957   \__braid_coordinate:xxxx {-##1-s} {-rev-\l__braid_tmpa_tl-s}
958   {\fp_eval:n {(##1 - 1) * \__braid_dim_value:n {width} }} {0}
959
960   \__braid_coordinate:xxxx {-##1-0} {-rev-\l__braid_tmpa_tl-0}
961   {\fp_eval:n {(##1 - 1) * \__braid_dim_value:n {width} }}
962   { \fp_to_decimal:N \l__braid_height_fp}
963 }

```

Run through any extra floors requested.

```

964 \seq_map_inline:Nn \l__braid_floors_seq
965 {
966   \tl_set:Nx \l__braid_tmpa_tl {\clist_item:nn {##1} {5}}
967   \__braid_do_floor:Vxxxx \l__braid_tmpa_tl
968   {\fp_eval:n
969     {
970       -1*sign(\__braid_dim_value:n{width})
971       * \__braid_dim_value:n {floor~ border}
972       + (\__braid_dim_value:n {width}) * (\clist_item:nn {##1} {1} - 1)
973     }
974     pt
975   }
976   {\fp_eval:n
977     {
978       \l__braid_height_fp + ( \__braid_dim_value:n {height} ) * (\clist_item:nn {##1} {2})
979     }
980     pt
981   }
982   {\fp_eval:n {
983     ( (\clist_item:nn {##1} {3}) * \__braid_dim_value:n {width}
984     + 2 * sign(\__braid_dim_value:n{width})
985     * \__braid_dim_value:n {floor~ border} ) / \dim_to_fp:n {1cm}
986   }
987   }
988   {\fp_eval:n {
989     (\clist_item:nn {##1} {4}) * ( \__braid_dim_value:n {height} ) / \dim_to_fp:n {1cm}
990   }
991   }
992 }

```

Keep track of the crossing level for the floor.

```

993 \int_zero:N \l__braid_crossing_int
994 \int_incr:N \l__braid_crossing_int
995
996 \seq_map_inline:Nn \l__braid_word_seq
997 {

```

Clear the flags for this segment of the braid word

```

998 \seq_clear:N \l__braid_crossing_seq
999 \bool_set_true:N \l__braid_step_level_bool
1000 \bool_set_false:N \l__braid_floor_bool
1001 \bool_set_false:N \l__braid_swap_crossing_bool
1002 ##1

```

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

```

1003 \bool_if:NT \l__braid_floor_bool
1004 {
1005   \__braid_do_floor:Vxxxx \l__braid_crossing_int
1006   {\fp_eval:n
1007     {
1008       -1*sign(\__braid_dim_value:n{width})
1009       * \__braid_dim_value:n {floor~ border}
1010     }
1011     pt
1012   }
1013   {\fp_to_decimal:N \l__braid_height_fp pt}
1014   {\fp_eval:n { \l__braid_width_fp / \dim_to_fp:n {1cm} }}
1015   {\fp_eval:n { ( \__braid_dim_value:n {height} ) / \dim_to_fp:n {1cm}}}
1016 }

```

If we have a crossing, process it.

```

1017 \seq_if_empty:NF \l__braid_crossing_seq
1018 {

```

Keep track of the current permutation.

```

1019 \prop_get:NxN \l__braid_crossing_permutation_prop
1020 {\seq_item:Nn \l__braid_crossing_seq {1}} \l__braid_tmpa_tl
1021 \prop_get:NxN \l__braid_crossing_permutation_prop
1022 {\seq_item:Nn \l__braid_crossing_seq {2}} \l__braid_tmpb_tl
1023
1024 \prop_put:NxV \l__braid_crossing_permutation_prop
1025 {\seq_item:Nn \l__braid_crossing_seq {2}} \l__braid_tmpa_tl
1026 \prop_put:NxV \l__braid_crossing_permutation_prop
1027 {\seq_item:Nn \l__braid_crossing_seq {1}} \l__braid_tmpb_tl

```

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

```

1028 \prop_get:NxN \l__braid_strands_prop
1029 {\seq_item:Nn \l__braid_crossing_seq {1}} \l__braid_tmpa_tl
1030 \prop_get:NxN \l__braid_strands_prop
1031 {\seq_item:Nn \l__braid_crossing_seq {2}} \l__braid_tmpb_tl

```

The over-strand is easy as that's a single curve.

```

1032 \tl_put_right:Nx \l__braid_tmpa_tl
1033 {
1034   \exp_not:N \__braid_lineto:nn
1035 }

```

```

1036     {\fp_eval:n
1037     {
1038         (\seq_item:Nn \l__braid_crossing_seq {1} - 1)
1039         * \__braid_dim_value:n {width}
1040     }
1041     }
1042     {\fp_eval:n { \l__braid_height_fp + \l__braid_nudge_fp } }
1043
1044     \exp_not:N \__braid_curveto:nnnnnn
1045
1046     {0}
1047     {\fp_eval:n { \l__braid_control_fp}}
1048
1049     {0}
1050     {\fp_eval:n {- \l__braid_control_fp}}
1051
1052     {\fp_eval:n
1053     {
1054         (\seq_item:Nn \l__braid_crossing_seq {2} - 1)
1055         * \__braid_dim_value:n {width}
1056     }
1057     }
1058     {\fp_eval:n
1059     {
1060         \l__braid_height_fp
1061         + \__braid_dim_value:n {height}
1062         - \l__braid_nudge_fp
1063     }
1064     }
1065     }

```

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

```

1066     \tl_put_right:Nx \l__braid_tmpb_tl
1067     {
1068         \exp_not:N \__braid_lineto:nn
1069
1070         {\fp_eval:n
1071         {
1072             (\seq_item:Nn \l__braid_crossing_seq {2} - 1)
1073             * \__braid_dim_value:n {width}
1074         }
1075         }
1076         {\fp_eval:n { \l__braid_height_fp + \l__braid_nudge_fp } }
1077
1078         \exp_not:N \__braid_curveto:nnnnnn
1079
1080         {0}
1081         {
1082             \fp_eval:n {
1083                 \l__braid_control_fp * (.5 - \__braid_value:n {gap} )
1084             }
1085         }
1086
1087         {
1088             \fp_eval:n {

```

```

1089 - (.5 - \_braid_value:n {gap} ) / 3 *
1090 \_braid_bezier_tangent:nnnnn
1091 {.5 - \_braid_value:n {gap} }
1092 {0}
1093 {0}
1094 {
1095   (\seq_item:Nn \l__braid_crossing_seq {1}
1096   - \seq_item:Nn \l__braid_crossing_seq {2})
1097   * \_braid_dim_value:n {width}
1098 }
1099 {
1100   (\seq_item:Nn \l__braid_crossing_seq {1}
1101   - \seq_item:Nn \l__braid_crossing_seq {2})
1102   * \_braid_dim_value:n {width}
1103 }
1104 }
1105 }
1106 {
1107   \fp_eval:n {
1108     -( .5 - \_braid_value:n {gap} ) / 3 *
1109     \_braid_bezier_tangent:nnnnn
1110     { .5 - \_braid_value:n {gap} }
1111     {0}
1112     {\l__braid_control_fp}
1113     {
1114       \_braid_dim_value:n {height}
1115       - 2* \l__braid_nudge_fp
1116       - \l__braid_control_fp
1117     }
1118     {\_braid_dim_value:n {height} - 2* \l__braid_nudge_fp}
1119   }
1120 }
1121
1122 {
1123   \fp_eval:n {
1124     (\seq_item:Nn \l__braid_crossing_seq {2} - 1)
1125     * \_braid_dim_value:n {width} +
1126     \_braid_bezier_point:nnnnn
1127     { .5 - \_braid_value:n {gap} }
1128     {0}
1129     {0}
1130     {
1131       (\seq_item:Nn \l__braid_crossing_seq {1}
1132       - \seq_item:Nn \l__braid_crossing_seq {2})
1133       * \_braid_dim_value:n {width}
1134     }
1135     {
1136       (\seq_item:Nn \l__braid_crossing_seq {1}
1137       - \seq_item:Nn \l__braid_crossing_seq {2})
1138       * \_braid_dim_value:n {width}
1139     }
1140   }
1141 }
1142 {

```

```

1143     \fp_eval:n {
1144     \l__braid_height_fp + \l__braid_nudge_fp +
1145     \__braid_bezier_point:nnnnn
1146     {.5 - \__braid_value:n {gap} }
1147     {0}
1148     {\l__braid_control_fp}
1149     {
1150     \__braid_dim_value:n {height}
1151     - 2* \l__braid_nudge_fp
1152     - \l__braid_control_fp
1153     }
1154     {\__braid_dim_value:n {height} - 2* \l__braid_nudge_fp}
1155     }
1156   }
1157
1158   \exp_not:N \__braid_moveto:nn
1159   {
1160     \fp_eval:n {
1161     (\seq_item:Nn \l__braid_crossing_seq {2} - 1)
1162     * \__braid_dim_value:n {width} +
1163     \__braid_bezier_point:nnnnn
1164     {.5 + \__braid_value:n {gap} }
1165     {0}
1166     {0}
1167     {
1168     (\seq_item:Nn \l__braid_crossing_seq {1}
1169     - \seq_item:Nn \l__braid_crossing_seq {2})
1170     * \__braid_dim_value:n {width}
1171     }
1172     {
1173     (\seq_item:Nn \l__braid_crossing_seq {1}
1174     - \seq_item:Nn \l__braid_crossing_seq {2})
1175     * \__braid_dim_value:n {width}
1176     }
1177     }
1178   }
1179   {
1180     \fp_eval:n {
1181     \l__braid_height_fp + \l__braid_nudge_fp +
1182     \__braid_bezier_point:nnnnn
1183     {.5 + \__braid_value:n {gap} }
1184     {0}
1185     {\l__braid_control_fp}
1186     {
1187     \__braid_dim_value:n {height} - 2* \l__braid_nudge_fp
1188     - \l__braid_control_fp
1189     }
1190     {\__braid_dim_value:n {height} - 2* \l__braid_nudge_fp}
1191     }
1192   }
1193
1194   \exp_not:N \__braid_curveto:nnnnnn
1195   {
1196

```

```

1197     \fp_eval:n {
1198       (.5 - \__braid_value:n {gap} ) / 3 *
1199       \__braid_bezier_tangent:nnnnn
1200       {.5 + \__braid_value:n {gap} }
1201       {0}
1202       {0}
1203       {
1204         (\seq_item:Nn \l__braid_crossing_seq {1}
1205         - \seq_item:Nn \l__braid_crossing_seq {2})
1206         * \__braid_dim_value:n {width}
1207       }
1208       {
1209         (\seq_item:Nn \l__braid_crossing_seq {1}
1210         - \seq_item:Nn \l__braid_crossing_seq {2})
1211         * \__braid_dim_value:n {width}
1212       }
1213     }
1214   }
1215   {
1216     \fp_eval:n {
1217       (.5 - \__braid_value:n {gap} ) / 3 *
1218       \__braid_bezier_tangent:nnnnn
1219       {.5 + \__braid_value:n {gap} }
1220       {0}
1221       {\l__braid_control_fp}
1222       {
1223         \__braid_dim_value:n {height} - 2* \l__braid_nudge_fp
1224         - \l__braid_control_fp
1225       }
1226       {\__braid_dim_value:n {height} - 2* \l__braid_nudge_fp}
1227     }
1228   }
1229
1230   {0}
1231   { \fp_eval:n {
1232     -\l__braid_control_fp * (.5 - \__braid_value:n {gap} )
1233   }
1234 }
1235
1236
1237   {\fp_eval:n
1238     {
1239       (\seq_item:Nn \l__braid_crossing_seq {1} - 1)
1240       * \__braid_dim_value:n {width}
1241     }
1242   }
1243   {\fp_eval:n
1244     {
1245       \l__braid_height_fp + \__braid_dim_value:n {height}
1246       - \l__braid_nudge_fp
1247     }
1248   }
1249
1250 }

```


Now put those new strands back in the prop.

```

1251     \prop_put:NxV \l__braid_strands_prop
1252     {\seq_item:Nn \l__braid_crossing_seq {2}} \l__braid_tmpa_tl
1253     \prop_put:NxV \l__braid_strands_prop
1254     {\seq_item:Nn \l__braid_crossing_seq {1}} \l__braid_tmpb_tl

```

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

```

1255     \int_compare:nT
1256     {
1257         \int_max:nn
1258         {
1259             \seq_item:Nn \l__braid_crossing_seq {1}
1260         }
1261         {
1262             \seq_item:Nn \l__braid_crossing_seq {2}
1263         }
1264         -
1265         \int_min:nn
1266         {
1267             \seq_item:Nn \l__braid_crossing_seq {1}
1268         }
1269         {
1270             \seq_item:Nn \l__braid_crossing_seq {2}
1271         }
1272         > 1
1273     }
1274     {
1275         \int_step_inline:nmmn
1276         {
1277             \int_min:nn
1278             {
1279                 \seq_item:Nn \l__braid_crossing_seq {1}
1280             }
1281             {
1282                 \seq_item:Nn \l__braid_crossing_seq {2}
1283             }
1284             + 1}
1285         {1}
1286         {
1287             \int_max:nn
1288             {
1289                 \seq_item:Nn \l__braid_crossing_seq {1}
1290             }
1291             {
1292                 \seq_item:Nn \l__braid_crossing_seq {2}
1293             }
1294             - 1
1295         }
1296     }
1297
1298     \prop_get:NnN \l__braid_strands_prop {###1} \l__braid_tmpa_tl
1299     \tl_put_right:Nx \l__braid_tmpa_tl
1300     {

```

```

1301     \exp_not:N \__braid_lineto:nn
1302     {\fp_eval:n {(###1 - 1) * \__braid_dim_value:n {width} }}
1303     {\fp_eval:n
1304     {
1305         \l__braid_height_fp + \l__braid_nudge_fp
1306         + .5 * \l__braid_control_fp
1307     }
1308     }
1309     \exp_not:N \__braid_moveto:nn
1310     {\fp_eval:n {(###1 - 1) * \__braid_dim_value:n {width} }}
1311     {\fp_eval:n
1312     {
1313         \l__braid_height_fp + \__braid_dim_value:n {height}
1314         - \l__braid_nudge_fp - .5 * \l__braid_control_fp
1315     }
1316     }
1317     }
1318
1319     \prop_put:NnV \l__braid_strands_prop {###1} \l__braid_tmpa_tl
1320     }
1321     }
1322     }

```

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

```

1323     \bool_if:NT \l__braid_step_level_bool
1324     {
1325         \fp_add:Nn \l__braid_height_fp { \__braid_dim_value:n {height} }
1326
1327         \int_step_inline:nnnn {1} {1} {\l__braid_strands_int}
1328         {
1329             \prop_get:NnN \l__braid_crossing_permutation_prop
1330             {###1} \l__braid_tmpb_tl
1331             \prop_get:NVN \l__braid_inverse_prop
1332             \l__braid_tmpb_tl \l__braid_tmpa_tl
1333
1334             \__braid_coordinate:xxxx
1335             {-\l__braid_tmpb_tl-\int_use:N \l__braid_crossing_int}
1336             {-rev-\l__braid_tmpa_tl-\int_use:N \l__braid_crossing_int }
1337             {\fp_eval:n { (###1 - 1) * \__braid_dim_value:n {width} }}
1338             {\fp_to_decimal:N \l__braid_height_fp}
1339         }
1340
1341         \int_incr:N \l__braid_crossing_int
1342     }
1343     }
1344
1345     \fp_add:Nn \l__braid_height_fp
1346     {
1347         sign(\__braid_dim_value:n {height})
1348         * \__braid_dim_value:n {border~ height}
1349     }

```

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

```

1350     \int_step_inline:nnnn {1} {1} {\l__braid_strands_int}
1351     {

```

```

1352 \prop_get:NxN \l__braid_strands_prop {##1} \l__braid_tmpa_tl
1353 \prop_get:NxN \l__braid_permutation_prop {##1} \l__braid_tmpb_tl
1354
1355 \tl_put_right:Nx \l__braid_tmpa_tl {
1356   \exp_not:N \__braid_lineto:nn
1357   {\fp_eval:n { (##1 - 1) * \__braid_dim_value:n {width} }}
1358   {\fp_to_decimal:N \l__braid_height_fp}
1359   coordinate (-rev-##1-e)
1360   coordinate (-\l__braid_tmpb_tl-e)
1361   ;
1362 }
1363
1364 \prop_put:NnV \l__braid_strands_prop {##1} \l__braid_tmpa_tl
1365 }

```

This is where we actually carry out the drawing commands.

```

1366 \int_step_inline:nnnn {1} {1} {\l__braid_strands_int}
1367 {
1368   \prop_get:NnN \l__braid_strands_prop {##1} \l__braid_tmpa_tl
1369   \tl_use:N \l__braid_tmpa_tl
1370 }
1371 \end{scope}
1372 }

```

(End definition for __braid_render:.)

```

\__braid_moveto:nn These are our interfaces to the TikZ code.
\__braid_lineto:nn 1373 \cs_new_nopar:Npn \__braid_moveto:nn #1#2
\__braid_curveto:nnnnnn 1374 {
\__braid_coordinate:nnnn 1375   (#1 pt, #2 pt)
1376 }
1377 \cs_new_nopar:Npn \__braid_lineto:nn #1#2
1378 {
1379   -- (#1 pt, #2 pt)
1380 }
1381 \cs_new_nopar:Npn \__braid_curveto:nnnnnn #1#2#3#4#5#6
1382 {
1383   .. controls +(#1 pt, #2 pt) and +(#3 pt, #4 pt)
1384   .. (#5 pt, #6 pt)
1385 }
1386 \cs_new_nopar:Npn \__braid_coordinate:nnnn #1#2#3#4
1387 {
1388   \coordinate[alias=#2] (#1) at (#3 pt,#4 pt);
1389 }
1390 \cs_generate_variant:Nn \__braid_coordinate:nnnn {xxxx}

```

(End definition for __braid_moveto:nn and others.)

```

\__braid_bezier_point:nnnn Used to calculate intermediate points and tangents on a bezier curve.
\__braid_bezier_tangent:nnnn 1391 \cs_new_nopar:Npn \__braid_bezier_point:nnnn #1#2#3#4#5
1392 {
1393   \fp_eval:n
1394   {
1395     (1 - (#1)) * (1 - (#1)) * (1 - (#1)) * (#2)
1396     +

```

```

1397     3 * (1 - (#1)) * (1 - (#1)) * (#1) * (#3)
1398     +
1399     3 * (1 - (#1)) * (#1) * (#1) * (#4)
1400     +
1401     (#1) * (#1) * (#1) * (#5)
1402   }
1403 }
1404 \cs_new_nopar:Npn \__braid_bezier_tangent:nnnnn #1#2#3#4#5
1405 {
1406   \fp_eval:n
1407   {
1408     3 * (1 - (#1)) * (1 - (#1)) * (#3 - (#2))
1409     +
1410     6 * (1 - (#1)) * (#1) * (#4 - (#3))
1411     +
1412     3 * (#1) * (#1) * (#5 - (#4))
1413   }
1414 }
1415 \cs_new_nopar:Npn \__braid_do_floor:nnnnn #1#2#3#4#5
1416 {
1417   \pic[pic~ type=floor,
1418     xscale=#4,
1419     yscale=#5,
1420     at={#2,#3},
1421     braid/every~ floor/.try,
1422     braid/floor~#1/.try,
1423   ];
1424 }
1425 \cs_generate_variant:Nn \__braid_do_floor:nnnnn {Vxxxx}
(End definition for \__braid_bezier_point:nnnnn and \__braid_bezier_tangent:nnnnn.)
1426 \ExplSyntaxOff

```