Tutorial on the Quantikz Package

Alastair Kay
Royal Holloway University of London, Egham, Surrey, TW20 0EX, UK[*]
(Dated: May 24, 2023)

I've always used QCircuit for typesetting quantum circuit diagrams within LaTeX, but found the Xy-pic based notation rather impenetrable and I struggled to adapt it for my needs (this is probably my failing rather than the package's). Thus, I wanted a tikz package that could do the same. That package is Quantikz. Those familiar with QCircuit will recognise much of the notation, although it has evolved a bit (hopefully simplified!).

The latest release (denoted by version numbers 1.x) is a major step forward in terms of the behind-the-scenes code. Unfortunately, this has necessitated breaking some compatibility with previous versions. Your old circuits should still work, but they might not look exactly as expected! Primarily the concept of the wires in a circuit has been modified as classical wires were only ever an after-thought, but have now been elevated to an equivalent status to the quantum wire.

CONTENTS

[*] alastair.kay@rhul.ac.uk

I. USAGE

The quantikz package is available on CTAN, and will therefore be available through most (current) TeX distributions. Once installed, simply write

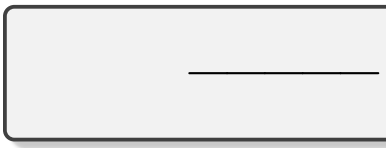
```
\usepackage{tikz}
\usetikzlibrary{quantikz2}
```

in the preamble of your document Now, each time that you want to include a quantum circuit, you just enclose it in a `tikzcd` or `quantikz` environment. (Theoretically, there is an advantange of `quantikz` over `tikzcd`, but `tikzcd` is retained for backwards compatibility. Use `quantikz` for preference.)

The current version of TeX on the arXiv is not up to date enough to provide the quantikz package. When uploading your source to the arXiv, you need to obtain the file tikzlibraryquantikz2.code.tex (you will always be able to locate it on your computer in the main tex directory if you have installed the package, but it should also accompany the source code of this file, and the most recent version is available here). Just include the file in the main directory of your source code.

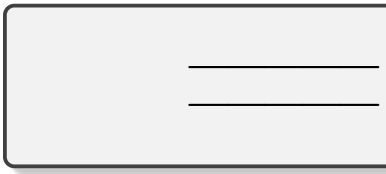
II. BASIC USAGE

Quantum circuits are laid out with a matrix notation, with cells separated by & (just like all matrices, tables etc. in LaTeX). Here, we typeset a single quantum wire.



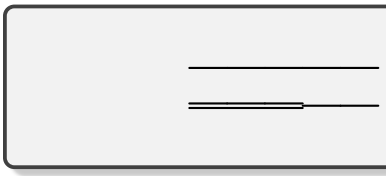
```
\begin{quantikz}
&&&&&
\end{quantikz}
```

New wires are created by the new line command, \\. By default, all wires are quantum wires, i.e. a single solid line.



```
\begin{quantikz}
&&&&& \\
&&&&&
\end{quantikz}
```

Wire types can be changed at a global level. Options are `q` (quantum), `c` (classical), `b` (bundle) and `n` (none). Here we specify that the first wire is quantum, and the second is classical. We also show how to change the classical wire to a quantum one mid-circuit (without separating gates, it looks ugly).

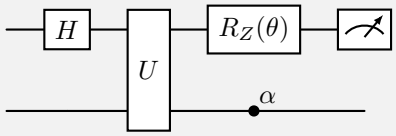


```
\begin{quantikz}[wire types={q,c}]
&&&&& \\
&&&&\setwiretype{q}&
\end{quantikz}
```

The wire change command affects the wire drawn from that cell, which is the one running back to the previous cell. This may mean changing the wire type in the column *after* the one you're expecting!

A. Gates & Measurement

Inside a cell (between two &), you can insert a gate command. Often, this will just be `\gate`, a plain box that contains a label corresponding to the parameter. The first optional parameter can be used to specify the number of qubits that the gate spans. You always put the command in the first cell in which that gate should appear. The label of the gate command is already in math mode, so you can enter arbitrary mathematical functions.




```
\begin{quantikz}
&\gate{H}&\gate[2]{U}&\gate{R_Z(\theta)}& \meter{} \\
&&&\phase{\alpha} &
\end{quantikz}
```

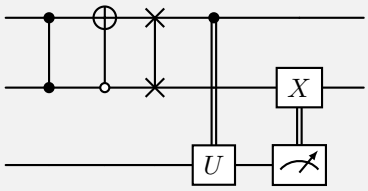
You can also see in the above example how to add a measurement gate (of which there are a number of variants) and a phase gate.

It is important that gate commands come before any other commands in a cell, such as those changing wire type. You do not typically put a gate command in the first cell (the one before the first & in a row) as there won't be an entering wire in that case.

Make sure you include the trailing & after the gate, or you won't have a wire coming out of the last gate!

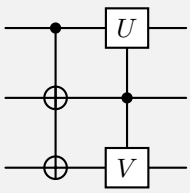
B. Controlled Gates

Controlled gates typically consist of two elements – a control and a target. The control creates a vertical wire of a specified length (may be negative). Note that the target commands, although they don't accept a parameter, need the {} after their call. Standard gates can also be targets.




```
\begin{quantikz}
& \ctrl{1} & \targ{} & \swap{1} & \ctrl[vertical
    wire=c]{2} &&\\
& \control{} & \ctrl[open]{-1} & \targX{} && \gate{X} &\\
&&&& \gate{U} & \meter{} \wire[u][1]{c}
\end{quantikz}
```

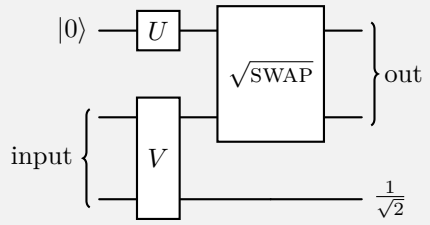
Other controls can be created simply by the addition of a vertical wire. If you want a gate that is one control and several targets, it is often good enough to just create the vertical wire so that it goes to the most distant target. You may need the command `\wire[d]{q}` (vertical quantum wire) to create vertical connections.




```
\begin{quantikz}
& \ctrl{2} & \gate{U} & \\
& \targ{} & \ctrl{1}\wire[u]{q} & \\
& \targ{} & \gate{V} &
\end{quantikz}
```

C. Labelling Circuits

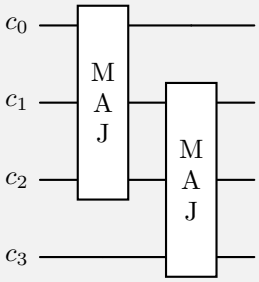
There are many ways to label the different parts of your circuit. The basic commands for labelling wires at the start/end of a circuit are `\lstick` and `\rstick` respectively. These can apply to multiple qubits; just apply the command to the first, and say how many wires it should cover. The arguments are typset in text mode, but you can of course convert to math mode.




```
\begin{quantikz}
\lstick{\ket{0}} & \gate{U} &
    \gate[2]{\sqrt{\textsc{swap}}} &\rstick[2]{out} \\
\lstick[2]{input} & \gate[2]{V} && \\
& & & \rstick{$\frac{1}{\sqrt{2}}$}
\end{quantikz}
```

You can use standard LaTeX maths expressions for your labels. Usually, spacing can be automatically adjusted just fine.

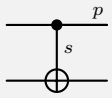
Sometimes, it might be that you want a multi-line label, and it should not be that each wire takes the height of those multiple lines. At this point, use the key `disable auto height`. By default, each row will be assigned the height that a gate with label U would be. This can be overridden by the third optional parameter of the gate command, if desired.




```
\begin{quantikz}
\lstick{$c_0$} & \gate[3,disable auto
    height]{\verticaltext{MAJ}} & & \\
\lstick{$c_1$} &  &  \gate[3,disable auto
    height]{\verticaltext{MAJ}} & \\
\lstick{$c_2$} &  &  &  \\
\lstick{$c_3$} &  &  &
\end{quantikz}
```

Note that we used the command `\verticaltext` for typesetting the text vertically.

Individual wires can also often be labelled. However, as this is typically an optional argument that appears late in the sequence, make sure you specify all proceeding parameters! For example,



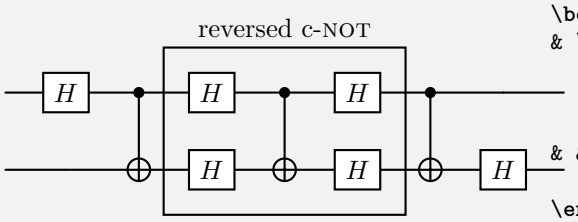

```
\begin{quantikz}
& \ctrl[wire style={"s"}]{1} &
    \wire[l][1]["p"{above,pos=0.2}]{a} \\
& \targ{} &
\end{quantikz}
```

You will observe that stylings can also be passed. The `pos` is a fraction of the distance along the line. In this case, it's being drawn right to left, hence being closer to the right-hand edge[1].

D. Boxing/Highlighting Parts of a Circuit

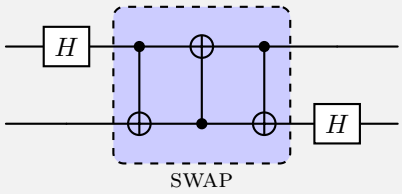
It is often useful to highlight parts of a circuit. We do this with the `\gategroup` command. The optional parameters `wires` (the default) and `steps` specify the number of rows and columns that the group spans respectively. The mandatory argument is the label for the box (although this can be empty). The top-left corner of the box coincides with the cell in which the command is placed.

[1] In the previous incarnation of quantikz, there was an accidental, undocumented method to label control wires. Internal changes have necessitated breaking that functionality.




```
\begin{quantikz}
& \gate{H} & \ctrl{1} &
    \gate{H}\gategroup[2,steps=3,style={inner
    sep=6pt}]{reversed c-{\sc not}} & \ctrl{1} &
    \gate{H} & \ctrl{1} & & \\
& & \targ{} & \gate{H} & \targ{} & \gate{H} & \targ{} &
    \gate{H} &
\end{quantikz}
```

This is probably where you want to start to tune some of the optional parameters to style the box the way you want. For more details beyond some basic examples that follow, see Sec. V. By default, this box is drawn on top of the circuit itself. If you want it to be behind (for example, should you want it to have a background colour), then use the `background` option.

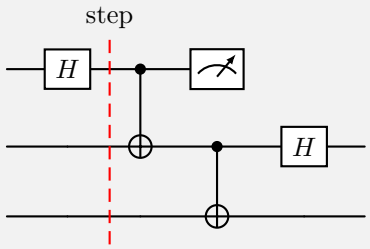



```
\begin{quantikz}
& \gate{H} &
    \ctrl{1}\gategroup[2,steps=3,style={dashed,rounded
    corners,fill=blue!20, inner
    xsep=2pt},background,label style={label
    position=below,anchor=north,yshift=-0.2cm}]{{\sc
    swap}} & \targ{} & \ctrl{1} & & \\
& & \targ{} & \ctrl{-1} & \targ{} & \gate{H} &
\end{quantikz}
```

The `label style` key can be used to tune the label's properties, including positioning. Note that it is often good to use `anchor=mid` for label anchors because if you have multiple labels, this will help get them horizontally aligned. It just means you have to use some `yshift` to move the label off the border around the gategroup.

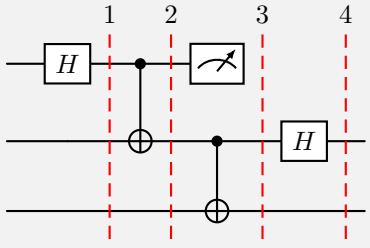
E. Slicing

It is often helpful to 'slice' up a circuit for the sake of explaining it step by step. To do this, we provide the `\slice{title}` command, which inserts a dashed vertical line after the column in which the command is added.



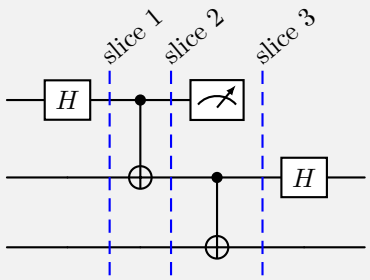

```
\begin{quantikz}
& \gate{H}\slice{step}  & \ctrl{1} & \meter{} \\
&       & \targ{} & \ctrl{1} & \gate{H} & \\
& & & \targ{} & &
\end{quantikz}
```

You can also slice every step by using option `slice all`, and the labels will be automatically numbered. This is likely to behave strangely unless you explicitly ensure that all rows have the same number of entries (i.e. short rows should have extra & characters added).




```
\begin{quantikz}[slice all]
& \gate{H} & \ctrl{1} & \meter{} &\setwiretype{n}& \\
&       & \targ{} & \ctrl{1} & \gate{H} & \\
& & & \targ{} & &
\end{quantikz}
```

If you need to adjust where the last slice is, use the optional parameter `remove end slices`, which counts the number of columns fewer to add slices to. You can also change the title of each of the slices, by setting `slice titles`. Include the macro `\col` in your specification if you want to use the step number. Note, however, that the columns won't automatically space themselves out to accommodate a very wide label. You can style the slicing lines with the `slice style` key, and the labels with `slice label style`. These can be used to rotate the labels and create a bit more space!




```
\begin{quantikz}[slice all,remove end slices=1,slice
    titles=slice \col,slice style=blue,slice label
    style={inner sep=1pt,anchor=south west,rotate=40}]
& \gate{H} & \ctrl{1} & \meter{} &\setwiretype{n}& \\
&       & \targ{} & \ctrl{1} & \gate{H} & \\
& & & \targ{} & &
\end{quantikz}
```

If you get compile errors when trying to slice, check the last line of your matrix, and make sure it doesn't end in \\.

III. COMMANDS & OPTIONS

\begin{quantikz}[opts]...\end{quantikz}:

The main environment in which you create quantum circuit diagrams. The main body of the environment is a table, with cells separated by &, and new rows started by \\.

`opts` is a comma separated list of the following options:

`wire types = {list}`	`list` is a comma separated list defining the wire type for each wire in the circuit, choosing from `q` (quantum), `c` (classical), `b` (wire bundle), `n` (none) If not specified, all wires are assumed to be quantum.
`thin lines`	Option for circuit aesthetic with thinner lines.
`transparent`	Sets entire circuit to have transparent background.
`classical gap=`	Classical wires are created by placing two wires, at centre ± classical gap. Default: 0.03cm.
`align equals at = n`	place the vertical centre of the circuit at wire number n (can be non-integer). If `wire types` is specified, this is automatically set to $(N+1)/2$ where N is the length of the list (i.e. number of wires).
`slice all`	add slices on all columns
`remove end slices = n`	does not show the last n slices
`slice titles =`	use the assigned text to label each slice. Use `\col` to convey column number.
`slice style =`	Standard tikz style commands for the lines of the slice. Enclose in {} if giving multiple commands
`slice label style =`	Standard tikz style commands for the label of the slice. Enclose in {} if giving multiple commands
`vertical slice labels`	write the text of the slices vertically instead of horizontally.

The environment also receives any standard tikzcd (and tikz) options. See that manual for more information. Particularly useful cases include:

`column sep = n`	put spacing between each column of length n (e.g. 1cm). This is the padding between columns, *not* the centres of gates, unless you add the between origins key.
`row sep = n`	put spacing between each row of length n (e.g. 1cm). This is the padding between rows, *not* the centres of gates, unless you add the between origins key.
`between origins`	makes the row/column sep measurement be between the centres of gates.

but you can also supply a colour, and that specifies the border colour of gates etc.

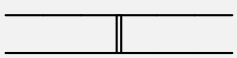
\setwiretype[n]{t}:

Sets wire n (default: current row, from that column onwards) to being of type t, which must be one of: `q` (quantum), `c` (classical) or `b` (bundle). Note that this will only affect the rendering from that point on, letting you change the wire type mid circuit. Should always come after any gate drawing command in a cell.


```
\begin{quantikz}
&&&\setwiretype{c}&&&
\end{quantikz}
```

\wire[d][n][s]{t}:

Draw a wire starting from the current cell of type `t` which can be one of `a` (automatic), `q`, `c`, `b`, going in direction `d` (`u,d,l,r` for up/down/left/right) for a number of cells n. Automatic uses the current wire type of that wire. Note that the standard horizontal wire for that cell will still be drawn. Some styling can be specified using `s`. Largely unnecessary. Should always come after any gate drawing command in a cell.



```
\begin{quantikz}
&&&\wire[d][1]{c}&&&\\
&&&&&&
\end{quantikz}
```

\wireoverride{t}:

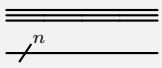
Sets the wire type for the current cell (i.e. the wire starting in this cell and going backwards) to `t` without changing the wire type for the rest of the row. Should always come after any gate drawing command in a cell.


```
\begin{quantikz}
%of the 6 wire sections (count the &), knock out the
    third.
&&&\wireoverride{n}&&&
\end{quantikz}
```

\qwbundle[s]{n}:

The standard typesetting of a wire bundle is 3 horizontal wires. Alternatively, you can use a single quantum wire, adding \qwbundle to put a slash through the wire, labelled by `n`, typically the number of qubits that single wire represents. Should always come after any gate drawing command in a cell. The size of the strike can be altered by altering the parameter `s`:

`style=`	Set the style of the line through a comma-separated list of tikz style commands.
`Strike Height=`	Set the height of the strike through
`Strike Width=`	Set the width of the strike through.




```
\begin{quantikz}[wire types={b,q},classical
    gap=0.07cm]
&&&& \\
& \qwbundle{n} &&&
\end{quantikz}
```

\permute{list}:

Permutation of wires. Takes the comma separated `list` of positive integers and connects wire n to the wire specified in the n^{th} element of the list. Counting starts such that wire 1 is the current wire. All wires are assumed to be quantum wires.

For example, `\permute{3,1,2}` connects the first qubit to the third (relative to the location of the command, not absolute row number), the second to the first and the third to the second.

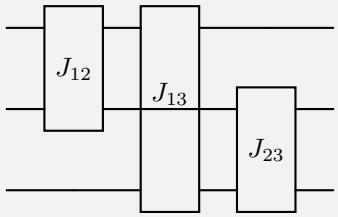
```
\begin{quantikz}[background color=black!5!white]
 &\permute{3,1,2} & \\
 &  &  \\
 &  &
 \end{quantikz}
```

Note that the wiring code is not sophisticated – it will probably look quite ugly for anything complicated! Also, it assumes that the wires are quantum (not classical or bundled). Overlapping wires are given a gap by using a thicker line of the colour `background color` (default: white), so if your circuit sits on top of something non-white (as here), you'll need to change that.

\linethrough:

Objects (e.g. gates) in cells take up their own space that, by default, wires do not cross. This command puts a quantum wire across the current cell. Note that this will appear underneath multi-qubit gates, and therefore invisible unless those gates are transparent.

One place where this might be useful is as a "pass-through" on a gate, such as

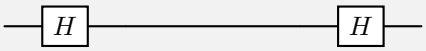



```
\begin{quantikz}[transparent]
& \gate[2]{J_{12}} & \gate[3,label
    style={yshift=0.2cm}]{J_{13}} & &   \\
& & \linethrough &\gate[2]{J_{23}} & \\
&&&&
\end{quantikz}
```

\push{t}:
Places text `t` on the quantum wire, with no spacing or gate command around it.

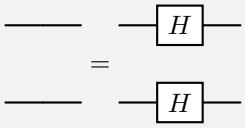

```
\begin{quantikz}
&&\push{X}&&
\end{quantikz}
```

\phantomgate{s}, \hphantomgate{s}:
Places a quantum wire that occupies that same space as a single-qubit gate with label `s` would. The h version only occupies horizontal space, not vertical.




```
\begin{quantikz}
& \gate{H} & \phantomgate{really wide gate} &
    \gate{H} &
\end{quantikz}
```

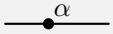
\ghost[w][h]{l}:
Creates an invisible quantum gate that has the same height as `\gate[][w][h]{l}`. Just like any other gate command, this should come before any other commands in a cell, and cannot be in the same cell as another gate command.




```
\begin{quantikz}
%give the wires the same vertical space as if they
    had H gates on them
&&\ghost{H} \\
&& \ghost{H}
\end{quantikz}=\begin{quantikz}
 & \gate{H} & \\
 & \gate{H} &
\end{quantikz}
```

\phase[s]{l}:
Creates a phase gate (black circle) with label `l`. The optional parameter `s` controls the styling via the parameters

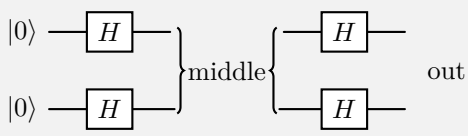
`style=`	Set the style of the line through a comma-separated list of tikz style commands.
`label style=`	Set the style of the label through a comma-separated list of tikz style commands.




```
\begin{quantikz}
& \phase{\alpha} &
\end{quantikz}
```

\lstick[s]{t}, \midstick[s]{t}, \rstick[s]{t}:
Text placement for labelling a wire to left/middle/right using text `t` (in text mode, not math mode) and styled using comma separated list of commands `s` from

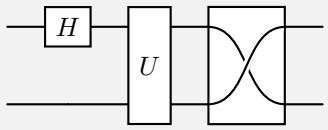
`n`	If n is a positive integer value, this is interpreted as the number of wires to use (default is 1). By default, introduces curly braces to span multiple wires.
`wires=n`	long-hand form of the above.
`label style=`	Styles the text using standard tikz commands. If using multiple commands, or anything involving a space, enclose in {}.
`brackets=`	Choose from `none/left/right/both` to switch on/off appropriate bracketing. (left won't affect \lstick, and right won't affect \rstick).
`braces=`	Styles the braces using standard tikz commands.




```
\begin{quantikz}
\lstick{\ket{0}} & \gate{H} & \midstick[2]{middle} &
    \gate{H}& \rstick[2,brackets=none]{out} \\
\lstick{\ket{0}} & \gate{H} &&\gate{H}&
\end{quantikz}
```

\gate[opts][w][h]{l}:
The standard quantum gate command. Creates a box containing the label `l`. `w` and `h` are optional minimum width and minimum height parameters to override the automatic sizing. Gate is styled using the optional parameter `opts`, which is a comma separated list of commands:

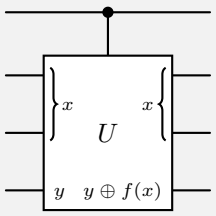
`n`	If n is a positive integer value, this is interpreted as the number of wires to use (default is 1).
`wires=n`	long-hand form of the above.
`style=`	Styles the gate (box) using standard tikz commands. If using multiple commands, or anything involving a space, enclose in {}.
`label style=`	Styles the gate text using standard tikz commands. If using multiple commands, or anything involving a space, enclose in {}.
`disable auto height`	pretty self-explanatory
`swap`	Fixes the gate to be a 2-qubit gate, depicting a swap between the two wires.




```
\begin{quantikz}
& \gate{H} & \gate[2]{U} & \gate[2,swap]{} & \\
&&&&
\end{quantikz}
```

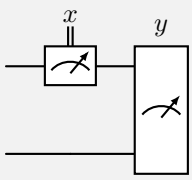
\gateinput[s]{l}, \gateoutput[s]{l}:
Put a label `l` inside the current gate command, starting on the current row, on either the input or output. If spanning multiple rows, will group the wires using curly braces by default. The width of the containing gate does not automatically adjust to the contents of these extra labels, so you will have to add it with the second optional parameter of `\gate`. Style with parameter `s` using comma separated list of commands from:

`n`	If n is a positive integer value, this is interpreted as the number of wires to span (default is 1).
`wires=n`	long-hand form of the above.
`label style=`	Styles the text using standard tikz commands. If using multiple commands, or anything involving a space, enclose in {}.
`braces=`	Styles the braces using standard tikz commands.




```
\begin{quantikz}
&\ctrl{1} & \\
&\gate[3][1.7cm]{U}\gateinput[2]{$x$}\gateoutput[2]{$x$}
    & \\
& & \\
&\gateinput{$y$}\gateoutput{$y\oplus f(x)$}
&
\end{quantikz}
```

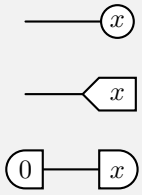
\meter[opts][w][h]{l}, \metercw[opts][w][h]{l}:
Measurement gates of different styles. Measurement is labelled by label `l`. Styling specified by optional parameter `opts`, a comma separated list of tikz commands. (Note the similarity to the gate command.) Can span multiple wires.




```
\begin{quantikz}
& \metercw[label style={inner sep=1pt}]{x} &
    \meter[2]{y} \\
&&
\end{quantikz}
```

`n`	If n is a positive integer value, this is interpreted as the number of wires to use (default is 1).
`wires=n`	long-hand form of the above.
`style=`	Styles the gate (box) using standard tikz commands. If using multiple commands, or anything involving a space, enclose in {}.
`label style=`	Styles the gate text using standard tikz commands. If using multiple commands, or anything involving a space, enclose in {}.
`disable auto height`	pretty self-explanatory

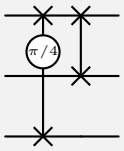
\measure[s]{l}, \measuretab[s]{l}, \meterD[s]{l}, \inputD[s]{l}:
Single-qubit measurement gates of different styles. Measurement is labelled by label `l`. Styling specified by optional parameter `s`, a comma separated list of tikz style commands. Note that this is provided as `[s]` where most other gates would expect `[style=s]`. `\inputD` is the equivalent for inputs of `\meterD`. For `\inputD`, incoming wire is disabled, overriding global style.




```
\begin{quantikz}
& \measure{x} \\
& \measuretab{x} \\
\inputD{0} & \meterD{x\vphantom{0}}
%vphantom makes the input and output shapes the same
    height, because x and 0 are different heights
\end{quantikz}
```

\swap[s]{n}:
Create an X shape on the current wire and a single vertical wire going downwards n rows (may be negative). Used as a swap gate, typically paired with `\targX`. Styled using parameter `s` as comma separated list of commands, some of which provide access to a partial swap:

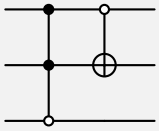
`partial swap=`	The text to place inside a circle
`partial position=`	Fractional position of the circle along the vertical line. Default: 0.5 (i.e. middle).
`style=`	Tikz styling parameters for the gate.
`label style=`	Tikz styling parameters for the label, overriding those of the whole gate.
`vertical wire=`	Specifies the type of vertical wire: `q` (quantum, default), `c` (classical) or `b` (bundle)




```
\begin{quantikz}[row sep=0.8cm]
& \swap[partial swap={\pi/4},partial position=0.3]{2}
    & \swap{1} &  \\
& & \targX{} & \\
& \targX{} &&
 \end{quantikz}
```

\ctrl[s]{n}, \octrl[s]{n}:
Starting point of a controlled gate, using filled circle (\ctrl) or open circle (\octrl). By default, single vertical line is added to go down n rows (may be negative). Settings are controlled via the optional parameter `s`

`style=`	Tikz styling parameters for the gate (both the circle and the vertical wire).
`wire style=`	Styling parameters for the wire only, overriding anything set by `style`.
`vertical wire=`	Specifies the type of vertical wire: `q` (quantum, default), `c` (classical) or `b` (bundle)
`open`	Makes the circle an open circle, i.e. `\ctrl[open]{1}` is a synonym for `\octrl{1}`.



```
\begin{quantikz}
& \ctrl{2} & \octrl{1} & \\
& \control{} & \targ{} & \\
& \ocontrol{} &&
\end{quantikz}
```

\control[s]{}, \ocontrol[s]{}:
Target equivalents of the above commands, but with no vertical wire. See previous two entries for usage examples.

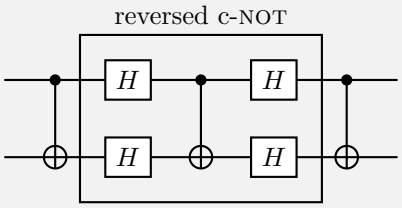
`style=`	Tikz styling parameters for the gate (both the circle and the vertical wire).
`open`	Makes the circle an open circle, i.e. `\control[open]{}` is a synonym for `\ocontrol{}`.

\targ[s]{}, \targX[s]{}:
Target elements of controlled-not and controlled-swap (no vertical wire). `s` provides styling parameters. See previous entries for usage examples.

`style=`	Tikz styling parameters for the gate.

\gategroup[opts]{l}:
Create a large box with top-left-hand corner positioned (roughly) at the top-left of the current cell. The box has label `l` and is styled with the options `opts`:

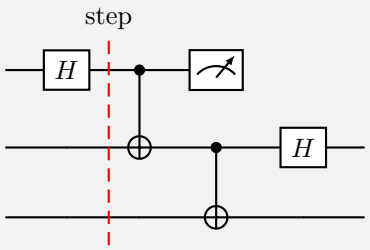
`n`	If n is a positive integer value, this is interpreted as the number of rows (wires) to cover (default is 1).
`wires=n`	long-hand form of the above.
`steps=m`	This is the number of columns (time steps) that the box covers. Default is 1.
`style=`	Styles the box using standard tikz commands. If using multiple commands, or anything involving a space, enclose in {}.
`label style=`	Styles the label text using standard tikz commands. If using multiple commands, or anything involving a space, enclose in {}.
`background`	Draw the gategroup behind the circuit (useful if the box has a background colour).




```
\begin{quantikz}
 & \ctrl{1} &
    \gate{H}\gategroup[2,steps=3,style={inner
    sep=6pt}]{reversed c-{\sc not}} & \ctrl{1} &
    \gate{H} & \ctrl{1} & \\
 & \targ{} & \gate{H} & \targ{} & \gate{H} & \targ{} &
\end{quantikz}
```

\slice[s]{l}:
Insert a slice between the current column and the next, with a title of `l`. Styles can be applied via the optional parameter `s`:

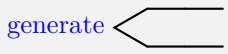
`style=`	Tikz styling parameters for the slice.
`label style=`	Tikz styling parameters for the title/label.




```
\begin{quantikz}
& \gate{H}\slice{step}  & \ctrl{1} & \meter{} \\
&       & \targ{} & \ctrl{1} & \gate{H} & \\
& & & \targ{} & &
\end{quantikz}
```

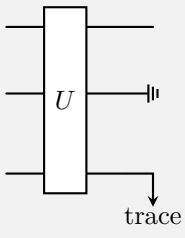
\makeebit[s]{l}:
Create an e-bit. This places the label `l` halfway between the current row and the next row. Two lines come off this label, leading to the current wire and the one below.

`style=`	Tikz styling parameters for the wires.
`label style=`	Tikz styling parameters for the label.
`angle=`	The angle of the two lines drawn, in degrees. Default: -45



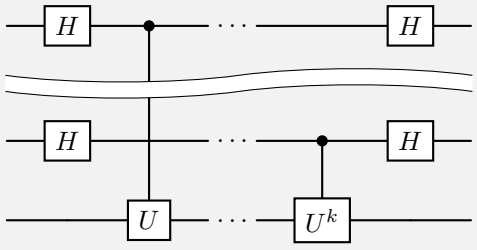

```
\begin{quantikz}
\makeebit[angle=-60,label style=blue]{generate} & & \\
 & &
\end{quantikz}
```

\trash[s]{l}, \ground[s]{l}:
Two different ways of denoting the termination of a wire (e.g. tracing out). Label text `l`, styling options directly supplied via `s`.




```
\begin{quantikz}
&\gate[3]{U} & \\
&& \ground{} \\
&& \trash{\text{trace}}
\end{quantikz}
```

\wave[s]{}:
Draw a wave along an entire row. `s` is standard tikz formatting commands for additional control over the style. By default, there is no wire drawn on this row. To have one, you should run \setwiretype{q} immediately after the \wave command.




```
\begin{quantikz}
& \gate{H} & \ctrl{3} & \ \ldots\  & & \gate{H} & \\
\wave&&&&&\\
& \gate{H} &  & \ \ldots\  & \ctrl{1} & \gate{H} & \\
& & \gate{U} & \ \ldots\  & \gate{U^k} &  &
\end{quantikz}
```

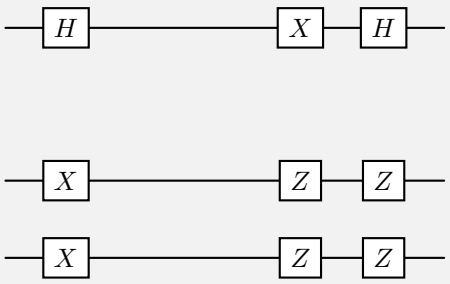
\verticaltext{l}:
Present the text in label `l` as vertically stacked. Can be helpful for slices.

\ket{l}, \bra{l}, \proj{l}, \braket{l}{m}:
Typeset Dirac notation $|l\rangle$, $\langle l|$, $|l\rangle\langle l|$ and $\langle l|m\rangle$ respectively. These commands do not require math mode, and the braces will automatically resize to the argument. They are defined to behave well with other packages (e.g. physics) that may define the same commands. You may need to be careful of the order in which you load those packages: load quantikz *after* the other package — if quantikz sees that those commands are already defined, it does not redefine them. If you wish to ensure that you are using the version that this package defines, run \forceredefine at the end of your preamble (after all packages have loaded).

IV. SPACING

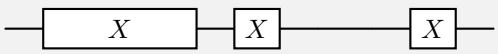
A. Local Adjustment

There are several different ways in which we can manipulate the spacing of a diagram. Adding space to an individual row or column can be done in the standard way of tables in LaTeX. Here we add 2cm of space to the column between the H and X gates, and 1cm of space between the top two rows.




```
\begin{quantikz}
& \gate{H} &[2cm] \gate{X} & \gate{H} & \\[1cm]
& \gate{X} & \gate{Z} & \gate{Z} &\\
& \gate{X} & \gate{Z} & \gate{Z} &
\end{quantikz}
```

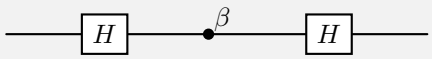
If you don't know how much space you need, but it should be determined by the size of some text, you can use `\hphantom{}` (widens the gate, in a similar way to `\gate[1cm]{}`) or `\hphantomgate{}` (increases the length of a wire) for horizontal spacing, and `\ghost{}` for vertical spacing.




```
\begin{quantikz}
& \gate{X} \hphantom{very wide} & \gate{X} &
    \hphantomgate{wide} & \gate{X} &
\end{quantikz}
```

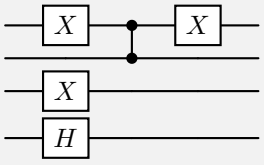
B. Global Adjustment

Standard tikz commands facilitate a global adjustment of row and column spacing. For example, a ridiculous horizontal spacing:



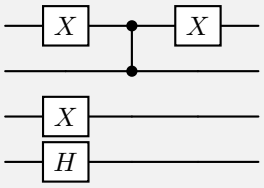

```
\begin{quantikz}[column sep=1cm]
& \gate{H} & \phase{\beta} & \gate{H} &
\end{quantikz}
```

This specifically adjusts the *gap* between the rows and columns, not the distance between the centres of the rows and columns. Depending on what gates you have on each wire, the spacing may not be the same between each wire. Sometimes this is desirable, particularly if a gate in one particular row is much larger than anything in the other rows. At other times, it just makes your diagram look a little odd. For example, look at the gap between the top two wires and the bottom two wires:




```
\begin{quantikz}[row sep=0.1cm]
& \gate{X} & \ctrl{1} & \gate{X} & \\
& & \control{} &  & \\
& \gate{X} & & & \\
& \gate{H} & & &
\end{quantikz}
```

If you want to make sure that every quantum wire is equally spaced, do the following to `row sep`:



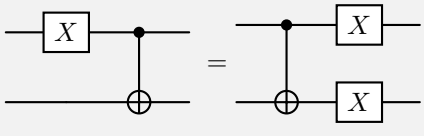

```
\begin{quantikz}[row sep={0.6cm,between origins}]
& \gate{X} & \ctrl{1} & \gate{X} & \\
& & \control{} & & \\
& \gate{X} & & & \\
& \gate{H} & & &
\end{quantikz}
```

This is particularly useful to achieve alignment of several circuits, as in IV C.

C. Alignment

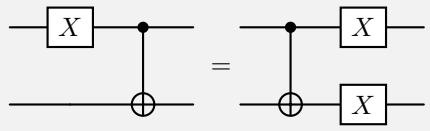
How do we centre a circuit diagram? Simply surround it with `\begin{center}` and `\end{center}` commands, or within any standard equation environment.

Vertical alignment between different circuits can be more fiddly, depending on how much of a perfectionist you are. Sometimes, they work immediately, but the wires don't always align perfectly with each other. Generally the problem is that the highest gate in each row is different (here, the LHS is missing an X gate on the second row)



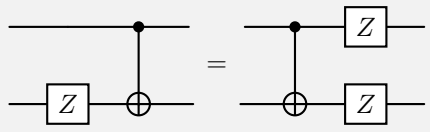

```
\begin{quantikz}
& \gate{X} & \ctrl{1} & \\
& & \targ{} &
\end{quantikz}
=\begin{quantikz}
& \ctrl{1} & \gate{X} & \\
& \targ{}  & \gate{X} &
\end{quantikz}
```

Ensuring an even spacing between rows, as described in Sec. IV, can help (but is not always appropriate). Often the easiest is to fudge it using the `\ghost` command which will add a 0-width gate of the height corresponding to its argument. So, having identified the problem with the above circuit, we can replace it with:



```
\begin{quantikz}
& \gate{X} & \ctrl{1} & \\
& \ghost{X} & \targ{} &
\end{quantikz}
=\begin{quantikz}
& \ctrl{1} & \gate{X} & \\
& \targ{}  & \gate{X} &
\end{quantikz}
```

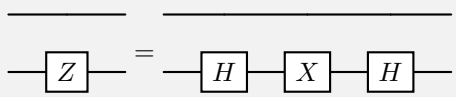
If you cannot identify the offending gate, and particularly if the operation is not a standard `\gate` command, you might be better off combining the two circuits in a single circuit with no wires joining the two parts. You can use a `midstick` command here. By default, it places braces both before and after, but these can be replaced using the optional argument `brackets=none|left|right|both`. Thus,



```
\begin{quantikz}
& & \ctrl{1} & \midstick[2,brackets=none]{=}& \ctrl{1} &
    \gate{Z} & \\
& \gate{Z} & \targ{} & & \targ{}  & \gate{Z} &
\end{quantikz}
```

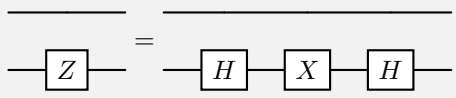
1. *Perfecting Vertical Alignment*

Sometimes when you're typesetting circuit identities as multiple separate circuits, the vertical alignment of the equals sign doesn't appear quite right (and can really niggle). Here, for example, the equals seems a bit low (because the baseline is the middle of the diagram by default, and here, the rows are not equal heights):



```
\begin{quantikz}
&& \\
& \gate{Z} &
 \end{quantikz}=\begin{quantikz}
&&&& \\
& \gate{H} & \gate{X} & \gate{H} &
 \end{quantikz}
```

To that end, we have added the key `align equals at=` option for the quantikz environment. This specifies which wire should be aligned with the equals sign. You can even use a non-integer. For instance, 1.5 will set it half way between wires 1 and 2.



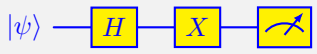
```
\begin{quantikz}[align equals at=1.5]
&& \\
& \gate{Z} &
 \end{quantikz}=\begin{quantikz}[align equals at=1.5]
&&&& \\
& \gate{H} & \gate{X} & \gate{H} &
 \end{quantikz}
```

If you use the `wire types` global option, this happens automatically. You can still override it provided the `align equals at` comes *after*.

V. STYLING & CUSTOMISING

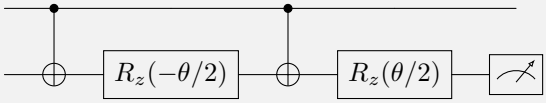
A. Global Styling

If you want to change the properties of an entire circuit such that all the typically black elements are a different colour, and the backgrounds of cells are another, you can supply the quantikz command with two keys: `color` and `background color`. The second of these works well alongside the `\pagecolor` command for making the rest of the page a particular colour.



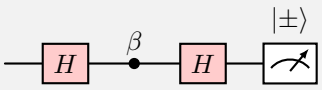

```
\begin{quantikz}[color=blue,background color=yellow]
\lstick{\ket{\psi}} & \gate{H} & \gate{X} & \meter{}
\end{quantikz}
```

There are two further keys that change styles globally: 'thin lines' to make the lines thin, more in keeping with what QCircuit produced, and 'transparent', should you want the background of all the gates to be transparent:




```
\begin{quantikz}[thin lines,transparent]
& \ctrl{1} & & \ctrl{1} & & \\
& \targ{} & \gate{R_z(-\theta/2)} & \targ{} &
    \gate{R_z(\theta/2)} & \meter{}
\end{quantikz}
```

Global properties that affect all circuit elements of a given type can be affected through `tikzset`.



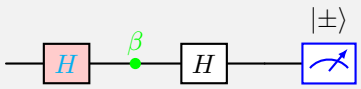

```
\tikzset{
 operator/.append style={fill=red!20},
 my label/.append style={above right,xshift=0.3cm},
 phase label/.append style={label position=above}
}
\begin{quantikz}
& \gate{H} & \phase{\beta} & \gate{H} & \meter{\ket{\pm}}
\end{quantikz}
```

The global styles are:

Style Name	Affected Command(s)
operator	`\gate`
meter	`\meter`
slice	`\slice`
wave	`\wave`
leftinternal	`\gateinput`
rightinternal	`\gateoutput`
dm	left braces (`\gateoutput`,`\lstick`)
dd	right braces (`\gateinput`,`\rstick`)
phase	`\phase`, `\control`, `\ophase`, `\ocontrol`
circlewc	`\targ`
crossx2	`\swap`,`\targX`
my label	measurement bases in `\meter`
phase label	phases in `\phase`
gg label	main gate label in `\gate`
group label	label in `\gategroup`

B. Per-Gate Styling

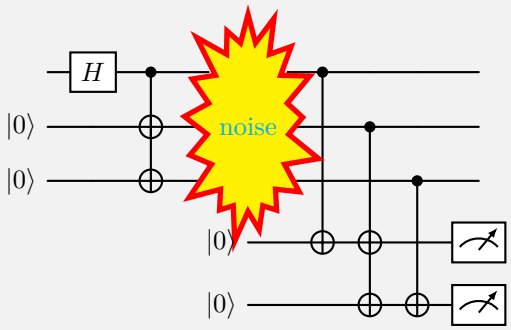
Individual gates can be modified using optional arguments of the calling function.




```
\begin{quantikz}
& \gate[style={fill=red!20},label style=cyan]{H} &
    \phase[style={green},label style={label
    position=above}]{\beta} & \gate{H} & &
    \meter[style={draw=blue}]{\ket{\pm}}
\end{quantikz}
```

The specific syntax varies a little depending on the type of gate. See the commands list for the specific cases. If you want to input several styling parameters with one of the keys, just group them together in a set of curly braces, {}. Typical styling parameters include `draw=` specifying line colour, `fill=`, specifying fill colour, `inner xsep=` and `inner ysep=` specifying horizontal and vertical margins respectively, `xshift=` and `yshift=` for adjusting horizontal and vertical positioning. Beyond that, the tikz manual is your friend!

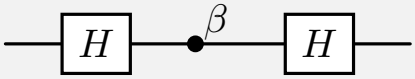
This styling is really quite flexible, as we can override the default shapes with anything that we want:




```
\tikzset{
 noisy/.style={starburst,fill=yellow,draw=red,line
    width=2pt,inner xsep=-4pt,inner ysep=-5pt}
}
\begin{quantikz}[row sep=0.3cm,column sep=0.3cm,wire
    types={q,q,q,n,n}]%
& \gate{H} & \ctrl{2}& & \gate[3,style={noisy},label
    style=cyan]{\text{noise}} & \ctrl{3} &  & & \\
\lstick{\ket{0}} & & \targ{} & & & & \ctrl{3} & & \\
\lstick{\ket{0}} & & \targ{}& & & & & \ctrl{2} & \\
&&&&\lstick{\ket{0}} & \targ{}\setwiretype{q} & \targ{}
    & & \meter{} \\
&&&&\lstick{\ket{0}} & \setwiretype{q} & \targ{} &
    \targ{} & \meter{}
\end{quantikz}
```

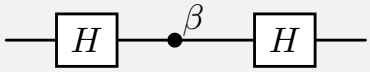
C. Scaling

If you want to override the standard size of a circuit (gate elements, text and spacing), you can make it a node inside a `tikzpicture`:




```
\begin{tikzpicture}
\node[scale=1.5] {
\begin{quantikz}
& \gate{H} & \phase{\beta} & \gate{H} &
\end{quantikz}
};
\end{tikzpicture}
```

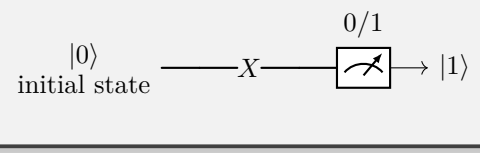
It's also possible to rescale to a fixed width, so long as you declare the `adjustbox` package in the document preamble.




```
\begin{adjustbox}{width=0.8\textwidth}
\begin{quantikz}
& \gate{H} & \phase{\beta} & \gate{H} &
\end{quantikz}
\end{adjustbox}
```

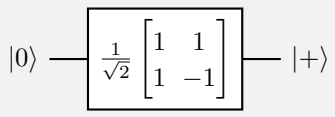
VI. BELLS AND WHISTLES

Since we have built quantikz on top of tikzcd, any of the standard arrow commands will work (don't forget to turn off the default wire!). For example, after a measurement, you might want to use an arrow to report a particular measurement outcome using `\arrow[r]`. The `r` conveys that the arrow should head one cell to the right. You can use combinations of up (u), down (d), left (l) and right as you wish. For more styling options, see the tikzcd manual.




```
\begin{quantikz}
\lstick{\ket{0}\\initial state} & & \push{X} & &
    \meter{0/1} \arrow[r] &
    \rstick{\ket{1}}\setwiretype{n}
\end{quantikz}
```

It's perhaps worth mentioning that gate commands can include matrices.




```
\begin{quantikz}
\lstick{\ket{0}} &
    \gate{\frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1
    & -1 \end{bmatrix}} & \rstick{\ket{+}}
\end{quantikz}
```

A. Externalization

A large document with many quantum circuits in it can be very slow to compile. This is where externalization comes in, producing images for each circuit. (It's also helpful if you have to produce accessible versions of lecture notes!) Quantikz *should* work with the externalization routines of tikz. Turn it on just as you normally would,

```
\usetikzlibrary{external}
\tikzexternalize % activate!
```

Note that for any circuits within an amsmath environment such as align, since these are run twice, you may end up with two copies of a given image.

B. New Gate Types

One of the most common types of email I get about quantikz goes along the lines of "Thank you for this brilliant package. I really want to use it, but I need *this* specific gate shape that nobody else has ever heard of before for my own personal reasons and I can't believe you haven't implemented it already. Fix it now!". This is impractical. However, with the latest version of quantikz, you can now use standard tikz commands in order to define your own shapes for single-qubit gates. Be warned: it can be quite fiddly.

The key to the process is defining your own shape (or appropriating an existing shape from the shapes library). As an example, we will use the `isosceles triangle` shape.

Then we need to ensure that wires will join on to your shape at the correct places. There are two different mechanisms for this. Quantum wires use the standard shape border, i.e., if you draw a straight line between the centre of two cells, the point of the join should be where this line intersects the edge of your shape. For standard shapes, this will typically just work If you're specifying your own shape, you'll need to ensure that the function `\anchorborder` function works correctly, at least at the north/south/east/west points. Classical wires (and bundles) attach to different anchors on the edge of the shape. We need 16 of them (but really, 8 are just aliases of the first 8). For example, a left-going classical wire will involve drawing two horizontal lines. One starts at `lstartone` and ends at the `lendone` of the target node. The other starts at `lstarttwo` and ends at `lendtwo`. The `lstartone` anchor should be on the left-most edge of your shape, at a height 0.03cm above the center, but then you will have to calculate the correct horizontal position. Similarly, `lstarttwo` should be on the left-most edge of your shape, at a height

0.03cm below the center. The end points are the equivalent points on the right-hand edge, and will also correspond to `rstartone` and `rstarttwo`.

```
\usetikzlibrary{shapes.geometric}
\makeatletter

%add new anchors
\pgfaddtoshape{isosceles triangle}{
%define the anchor lstartone.
  \anchor{lstartone}{%
    \trianglepoints %built-in function for this shape. Defines certain macros with position info
    %get the position of the lower left corner. Stored in macros \pgf@x and \pgf@y
    \pgf@process{\pgfpointadd{\centerpoint}{\pgfmathrotatepointaround{\lowerleftanchor}{\
        pgfpointorigin}{\rotate}}}
    \pgf@ya=.5\pgf@y
    \pgf@process{\pgfpointadd{\centerpoint}{\pgfmathrotatepointaround{\lowerrightanchor}{\
        pgfpointorigin}{\rotate}}}
    \pgf@y=.5\pgf@y
    \advance\pgf@y by \pgf@ya%
    \advance\pgf@y by -\pgfkeysvalueof{/tikz/commutative diagrams/classical gap}%
    %by end of function, position of anchor stored in \pgf@x and \pgf@y
  }
  \anchor{lstarttwo}{%
    \trianglepoints
    \pgf@process{\pgfpointadd{\centerpoint}{\pgfmathrotatepointaround{\lowerleftanchor}{\
        pgfpointorigin}{\rotate}}}
    \pgf@ya=.5\pgf@y
    \pgf@process{\pgfpointadd{\centerpoint}{\pgfmathrotatepointaround{\lowerrightanchor}{\
        pgfpointorigin}{\rotate}}}
    \pgf@y=.5\pgf@y
    \advance\pgf@y by \pgf@ya%
    \advance\pgf@y by \pgfkeysvalueof{/tikz/commutative diagrams/classical gap}%
  }
  \anchor{lendone}{%
    \trianglepoints
    \pgf@process{\pgfpointintersectionoflines%
      {\pgfpointadd{\centerpoint}{\pgfmathrotatepointaround{\apexanchor}{\pgfpointorigin}{\rotate}}}%
      {\pgfpointadd{\centerpoint}{\pgfmathrotatepointaround{\lowerrightanchor}{\pgfpointorigin}{\
          rotate}}}%
      {\pgfpointadd{\centerpoint}{\pgfpoint{0cm}{-\pgfkeysvalueof{/tikz/commutative diagrams/
          classical gap}}}}%
      {\pgfpointadd{\centerpoint}{\pgfpoint{0.5cm}{-\pgfkeysvalueof{/tikz/commutative diagrams/
          classical gap}}}}%
    }%
  }
  \anchor{lendtwo}{%
    \trianglepoints
    \pgf@process{\pgfpointintersectionoflines%
      {\pgfpointadd{\centerpoint}{\pgfmathrotatepointaround{\apexanchor}{\pgfpointorigin}{\rotate}}}%
      {\pgfpointadd{\centerpoint}{\pgfmathrotatepointaround{\lowerleftanchor}{\pgfpointorigin}{\
          rotate}}}%
      {\pgfpointadd{\centerpoint}{\pgfpoint{0cm}{\pgfkeysvalueof{/tikz/commutative diagrams/classical
           gap}}}}%
      {\pgfpointadd{\centerpoint}{\pgfpoint{0.5cm}{\pgfkeysvalueof{/tikz/commutative diagrams/
          classical gap}}}}%
    }%
  }
```

Similarly, we need 4 anchors for the down wire.

```
  %now do the anchors for a down wire
  \anchor{dstarttwo}{%
    \trianglepoints
    \pgf@process{\pgfpointintersectionoflines%
      {\pgfpointadd{\centerpoint}{\pgfmathrotatepointaround{\apexanchor}{\pgfpointorigin}{\rotate}}}%
      {\pgfpointadd{\centerpoint}{\pgfmathrotatepointaround{\lowerrightanchor}{\pgfpointorigin}{\
          rotate}}}%
      {\pgfpointadd{\centerpoint}{\pgfpoint{\pgfkeysvalueof{/tikz/commutative diagrams/classical gap
          }}{0cm}}}%
      {\pgfpointadd{\centerpoint}{\pgfpoint{\pgfkeysvalueof{/tikz/commutative diagrams/classical gap
          }}{0.5cm}}}%
    }%
  }
  \anchor{dstartone}{%
    \trianglepoints
    \pgf@process{\pgfpointintersectionoflines%
      {\pgfpointadd{\centerpoint}{\pgfmathrotatepointaround{\apexanchor}{\pgfpointorigin}{\rotate}}}%
      {\pgfpointadd{\centerpoint}{\pgfmathrotatepointaround{\lowerrightanchor}{\pgfpointorigin}{\
          rotate}}}%
      {\pgfpointadd{\centerpoint}{\pgfpoint{-\pgfkeysvalueof{/tikz/commutative diagrams/classical gap
          }}{0cm}}}%
      {\pgfpointadd{\centerpoint}{\pgfpoint{-\pgfkeysvalueof{/tikz/commutative diagrams/classical gap
          }}{0.5cm}}}%
    }%
  }
    \anchor{dendtwo}{%
    \trianglepoints
    \pgf@process{\pgfpointintersectionoflines%
      {\pgfpointadd{\centerpoint}{\pgfmathrotatepointaround{\apexanchor}{\pgfpointorigin}{\rotate}}}%
      {\pgfpointadd{\centerpoint}{\pgfmathrotatepointaround{\lowerleftanchor}{\pgfpointorigin}{\
          rotate}}}%
      {\pgfpointadd{\centerpoint}{\pgfpoint{\pgfkeysvalueof{/tikz/commutative diagrams/classical gap
          }}{0cm}}}%
      {\pgfpointadd{\centerpoint}{\pgfpoint{\pgfkeysvalueof{/tikz/commutative diagrams/classical gap
          }}{0.5cm}}}%
    }%
  }
  \anchor{dendone}{%
    \trianglepoints
    \pgf@process{\pgfpointintersectionoflines%
      {\pgfpointadd{\centerpoint}{\pgfmathrotatepointaround{\apexanchor}{\pgfpointorigin}{\rotate}}}%
      {\pgfpointadd{\centerpoint}{\pgfmathrotatepointaround{\lowerleftanchor}{\pgfpointorigin}{\
          rotate}}}%
      {\pgfpointadd{\centerpoint}{\pgfpoint{-\pgfkeysvalueof{/tikz/commutative diagrams/classical gap
          }}{0cm}}}%
      {\pgfpointadd{\centerpoint}{\pgfpoint{-\pgfkeysvalueof{/tikz/commutative diagrams/classical gap
          }}{0.5cm}}}%
    }%
  }
}
```

The `r` and `u` wires just go in the opposite direction to the `l` and `d` wires. So we don't need to calculate anything else, just equate them.

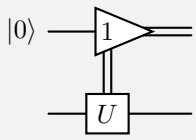
```
%aliases for r and u wires
\pgfdeclareanchoralias{isosceles triangle}{dendtwo}{ustartwo}
\pgfdeclareanchoralias{isosceles triangle}{dendone}{ustartone}
\pgfdeclareanchoralias{isosceles triangle}{dstarttwo}{uendtwo}
\pgfdeclareanchoralias{isosceles triangle}{dstartone}{uendone}
```

```
\pgfdeclareanchoralias{isosceles triangle}{lendtwo}{rstartwo}
\pgfdeclareanchoralias{isosceles triangle}{lendone}{rstartone}
\pgfdeclareanchoralias{isosceles triangle}{lstarttwo}{rendtwo}
\pgfdeclareanchoralias{isosceles triangle}{lstartone}{rendone}
\makeatother
```

We now have a shape that we can use as the base of a gate. It probably isn't quite styled how you might want it, so it's worth setting up a style.

```
\tikzset{
  trimeter/.style={
      thickness, % gets the line width correct
      filling, %inherits filling. Typically background colour (default: white), but could be
          transparent.
      shape=isosceles triangle, %this is your shape name
      draw=black, %don't forget this, or you won't see much!
      inner sep=2pt
  }}
```

Finally, define the code that will let you run the gate.




```
\DeclareExpandableDocumentCommand{\mygate}{O{}{m}}{%
|[trimeter,#1]| {#2}}

\begin{quantikz}[classical gap=0.05cm]
\lstick{\ket{0}} & \mygate{1} & \setwiretype{c} \\
&\gate{U}\wire[u]{c} &
\end{quantikz}
```

Your function will work just like any of the single-qubit measurement commands: per-gate styling can be supplied via the optional argument. If your gate does not require the text label, you can remove the `#2` from definition, but you must not remove the `{m}` from the argument specification.

It's possible to define a multi-qubit version in a very similar way, although they often end up looking a bit ugly, and non-straight edges are unlikely to fit correctly with incoming wires (which will be most noticeable if you activate transparency). It's why we don't officially have a multi-qubit version of `\meterD`, although you may be able to get a good enough version working for your purposes.

```
\DeclareExpandableDocumentCommand{\mygate}{O{}O{2em}O{1.5em}m}{%
\gate[#1,ps=trimeter,disable auto height][#2][#3]{#4}
}
```

VII. CONVERTING FROM QCIRCUIT

I've updated all of my existing teaching materials from QCircuit to the original Quantikz with very little trouble. There are a few standard replacements:

QCircuit notation	Quantikz notation
`\QCircuit @C=n @R=m {#}`	`\begin{quantikz}[row sep=m,col sep=n]#\end{quantikz}`
`\multigate{n}`	`\gate[n+1]`
`\targ`	`\targ{}`
`\control`	`\control{}`
`\meter`	`\meter{}`
`\measureD`	`\meterD{}`

Updating the `\gategroup` command requires a little more care since the first two arguments have to be removed, and the command placed in the correct cell, at which point `\gategroup{i}{j}{k}{l}{m}` becomes

`\gategroup[k+1-i,steps=l+1-j]`.

My primary use of gategroup was to achieve the effect now achieved with `\lstick[k+1-i]`.

It should not be necessary to use `\ghost` commands in the way they were used in QCircuit.

A. Converting from Quantikz to Quantikz2

Quantikz was originally written aiming to maintain compatibility with QCircuit. However, over time, it has proved desirable to add a number of new features. Some of these have necessitated breaking this backwards compatibility. This makes upgrading a little more fiddly in (probably) rare cases.

- You should always use that quantikz environment in preference to the tikzcd environment.
- Individual qw/cw commands are unnecessary. However, if there were places where you deliberately left a part of the circuit without a wire, that will need to be updated as everything has a qw by default.
- Labels on e.g. `\meter` are now in math mode where before they were in text mode. You'll get an error if your label now starts with $.
- `\ctrlbundle` is obsolete. It will be automatically selected if appropriate. Just use `\ctrl`.
- `ampersand replacement` is unlikely to work (at least in the quantikz environment), but shouldn't be necessary.
- Many of the styling options have been changed – rather than inputting many of them directly like `[s]`, they can be accessed with a key `[style=s]`. This allows for much greater capacity for variation in the future. If you didn't use the options, then no changes are needed.
- There are some instances where the original quantikz let you get away with a call such as `\targ` instead of `\targ{}`. Quantikz2 tends to b less forgiving.

VIII. TROUBLESHOOTING

- Have you checked that all commands that need them are followed by an empty argument, {}? Things like `meter`, `\control` (basically, those that can accept an optional styling parameter) look like they don't take any parameters, but they have to be followed by the pair of braces or you'll get very odd effects.
- If you get a whole bunch of unexpected text in one of your cells instead of a gate, make sure that the gate command is the first command in the cell, and that other commands (such as `\qwbundle`) appear after.
- If you're getting errors about cells not being found (and especially if you're doing any slicing, or gategroups), check that your last row doesn't end with \\, and make sure that your last row contains as many cells (even if they're empty) as there are columns in the circuit.
- I am not aware of any specific compatibility problems with beamer, tabular, align etc. in the current version. If you do have any problems you might try using the tikzcd environment instead of quantikz, and applying the `ampersand replacement=\&` option, followed by separating all cells using \& instead of &.

- If you're using transparency, and the width of gates seems to be greater than you expected, it may be worthwhile removing the .aux file and recompiling. If your tex editor isn't good at resetting the .aux file, the system may be remembering older widths.
- Package load order: I've had reports that if you load certain packages in the wrong order it can create weird errors. For example, if you load the package cleveref after quantikz, and then use a split environment, it can lead to the error "Only one # is allowed per tab.". Change the load order and it goes away. I have no idea why this happens.
- If you are trying to submit to a Springer journal, they seem to actively prevent you from using tikz. You have to produce a separate image for each circuit. The externalisation options in tikz may be very helpful here to take your existing code and produce a pdf output of each circuit.

For any bug reports (please make sure you've checked the above list first!) or feature requests, please contact alastair.kay@rhul.ac.uk.

IX. CITATION

If you found this package useful, please consider citing the arXiv version of this document, arXiv:1809.03842.