

The tikzquads Package

An Extension to CircuiTikZ

Version 1.2a

Alceu Frigeri*

November 2025

Abstract

This package defines a few extra shapes (single / dual port boxes) designed to be used together with the *CircuiTikZ*[5] package.

Contents

1	Introduction	2
1.1	CircuiTikZ	2
1.2	Package Requirements	2
2	Auxiliary Shapes and Basic Keys	3
2.1	Auxiliary shapes	3
2.2	General Keys	3
3	Z, Y, G, H Quadripoles	3
3.1	The Base Dual Port / Quadripole Shape	3
3.1.1	Base Keys	4
3.2	Quad	5
3.2.1	Quad Keys	5
3.2.2	Examples of <i>fit to use</i>	6
3.3	Quad Z	6
3.3.1	Quad Z keys	7
3.4	Quad Y	7
3.4.1	Quad Y keys	8
3.5	Quad G	8
3.5.1	Quad G keys	9
3.6	Quad H	9
3.6.1	Quad H keys	9
4	Thevenin, Norton single port boxes	10
4.1	The Base Single Port / Black Box Shape	10
4.1.1	Base Keys	10
4.2	Black Box	11
4.2.1	Black Box keys	11
4.2.2	Examples of <i>fit to use</i>	11
4.3	Thevenin	12
4.3.1	Thevenin keys	12
4.4	Norton	13
4.4.1	Norton keys	13
5	Pseudo-Graph Shape	13
5.1	Pseudo-Graph Keys	14
6	Parallel Connections	14

*<https://github.com/alceu-frigeri/tikzquads>

1 Introduction

In standard text books, Circuit Theory and Electronics alike, quite often one ends representing sub-circuits as either:

- a single port *black box*, or
- a dual port *black box*

This package defines a few, parameterized shapes for each case:

- for single port *black boxes*:
 - Black Box
 - Thevenin
 - Norton
- for dual port *black boxes*:
 - Quad
 - Quad Z
 - Quad Y
 - Quad G
 - Quad H

A *Pseudo-Graph load line* shape is also defined, for those moments where a true graph, like the ones `pgfplots`[1] or `tikzfxgraph`[4] enables, isn't needed. Lastly, a convenience interconnection command is defined, `\QuadParConnect` (see 6).

1.1 CircuiTikZ

Unfortunately, some implementation details of this package don't follow the code structure adopted by `CircuiTikZ`[5], and some significant part of this package's code would have to be re-written if it were to be integrated directly in `CircuiTikZ`, and that's the main reason this is, for the time being, a separate package. After all, even though this doesn't follow `CircuiTikZ` code scheme, it does work nicely with it, as is.

1.2 Package Requirements

Internally, many commands relies on `pgfkeysearch`[2] which is loaded with default options. If needed, the user is advised to use the commands `\pgfkeysearchsettings` or `\pgfkeysearch_settings:n` from `pgfkeysearch` instead of passing package options (to avoid a package options crash error). The same applies to package `tikzdotncross`[3] needed for `\QuadParConnect` (see 6), use the command `\setpindefaults` to change (if needed) it's defaults. Besides those packages, only `pkginfograb` is loaded (for package's info tracking).

So, the user is supposed to load `pgf/tikz`[6] before this, with user preferred settings and, most likely, `CircuiTikZ`[5].

```

1 \usepackage{tikz}
2 \usepackage{tikzquads}

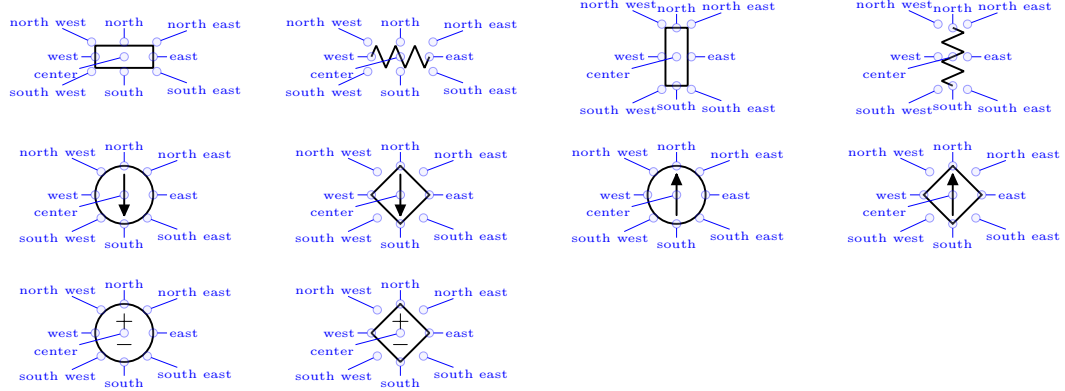
1 \usepackage[american,siunitx]{circuitikz}
2 \usepackage{tikzquads}

```

2 Auxiliary Shapes and Basic Keys

2.1 Auxiliary shapes

A set of auxiliary shapes are defined, but not meant to be used otherwise, though their anchors might be relevant:



Note: The point being that, regardless of the sub-shape orientation, the intuitive geographical coordinates applies.

2.2 General Keys

General keys to fine tuning a shape:

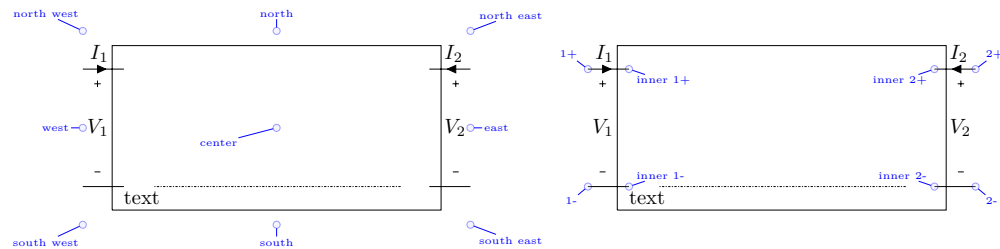
<code>outer sep</code>	Text outer separation, initial value: 1.5pt
<code>inner sep</code>	Text inner separation, initial value: 1pt
<code>thickness</code>	Components thickness (relative to the drawing thickness), initial value: 2
<code>tip len</code>	tip len (current source). initial value: 4pt
<code>tip type</code>	possible values: <i>triangle</i> and <i>bezier</i> . initial value: <i>triangle</i>
<code>minussign len</code>	Minus sign len (voltage source). initial value: <code>\pgf@circ@Rlen/14</code>
<code>plussign len</code>	Plus sign len (voltage source). initial value: <code>1.1\pgf@circ@Rlen/14</code>
<code>source radius</code>	The base radius. initial value: <code>0.3\pgf@circ@Rlen</code>
<code>round sources</code>	Sources will be round ones
<code>control sources</code>	Sources will be control/diamond ones
<code>generic, european</code>	Impedances will be generic rectangles
<code>zigzag, american</code>	Impedances will be draw as zigzags

3 Z, Y, G, H Quadripoles

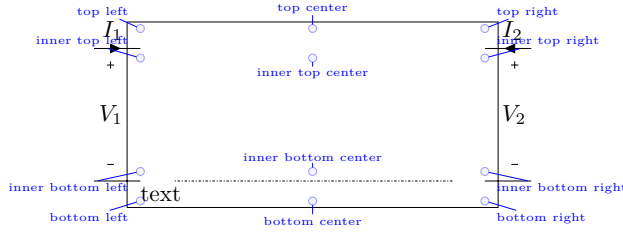
A set of configurable Quadripoles is defined, whereas quadripoles parameters (for instance Z_{11} , Z_{12} , Z_{21} and Z_{22}) are `<key-value>` parameters.

3.1 The Base Dual Port / Quadripole Shape

The base shape just draws a base box and sets some connection anchors: `1+`, `1-`, `inner 1+`, `inner 1-`, `2+`, `2-`, `inner 2+` and `inner 2-`, besides the geographic ones:



And also a set of (meant for) `text` anchors:



3.1.1 Base Keys

These applies to all Quad shapes:

<code>base width</code>	The 'box' width
<code>half base width</code>	Ditto, half width. Initial value: $2\backslash\text{pgf@circ@Rlen}$.
<code>base height</code>	The distance between $1+$ and $1-$. The 'box' full height is equal to $2*(\text{half base height} + \text{height ext} + \text{height ext+})$.
<code>half base height</code>	Ditto, half height. Initial value: $\backslash\text{pgf@circ@Rlen}/7$
<code>height ext</code>	Initial value: $2\backslash\text{pgf@circ@Rlen}/7$
<code>height ext+</code>	Initial value: 0
<code>inner ext</code>	distance between the 'box' and $\text{inner}1+/1-/2+/2-$. initial value: $\backslash\text{pgf@circ@Rlen}/7$
<code>outer ext</code>	distance between the 'box' and $1+/1-/2+/2-$. initial value: $5\backslash\text{pgf@circ@Rlen}/14$
<code>inner marks</code>	If set, the inner anchors will be marked.
<code>outer marks</code>	If set, the outer anchors will be marked.
<code>invert</code>	The shape will be inverted, more or less like 'x scale=-1'.
<code>alt, opt</code>	Case a Voltage source is zero, a series impedance will be draw vertically.
<code>outer x fit to</code>	For any Quad, this is the same as <code>outer x fit to*</code> .
<code>outer x fit to*</code>	It takes 2 arguments: <code>outer x fit*={⟨CoordA⟩}{⟨CoordB⟩}</code> . The width will be set so that $\langle 1+ \rangle$ and $\langle 2+ \rangle$ (or $\langle 1- \rangle$ and $\langle 2- \rangle$, depending on the used anchor) will fit $\langle \text{CoordA} \rangle$ and $\langle \text{CoordB} \rangle$. This might result in a shape rotation.
<code>outer x fit to!</code>	It takes 2 arguments: <code>outer x fit!={⟨CoordA⟩}{⟨CoordB⟩}</code> . The width will be set so that the distance between $\langle 1+ \rangle$ and $\langle 2+ \rangle$ (or $\langle 1- \rangle$ and $\langle 2- \rangle$, depending on the used anchor) will be the same as $\langle \text{CoordA} \rangle$ and $\langle \text{CoordB} \rangle$. This will never result in a shape rotation.
<code>inner x fit to</code>	For any Quad, this is the same as <code>inner x fit to*</code> .
<code>inner x fit to*</code>	It takes 2 arguments: <code>inner x fit*={⟨CoordA⟩}{⟨CoordB⟩}</code> . The width will be set so that $\langle \text{inner } 1+ \rangle$ and $\langle \text{inner } 2+ \rangle$ (or $\langle \text{inner } 1- \rangle$ and $\langle \text{inner } 2- \rangle$, depending on the used anchor) will fit $\langle \text{CoordA} \rangle$ and $\langle \text{CoordB} \rangle$. This might result in a shape rotation.
<code>inner x fit to!</code>	It takes 2 arguments: <code>inner x fit!={⟨CoordA⟩}{⟨CoordB⟩}</code> . The width will be set so that the distance between $\langle \text{inner } 1+ \rangle$ and $\langle \text{inner } 2+ \rangle$ (or $\langle \text{inner } 1- \rangle$ and $\langle \text{inner } 2- \rangle$, depending on the used anchor) will be the same as $\langle \text{CoordA} \rangle$ and $\langle \text{CoordB} \rangle$. This will never result in a shape rotation.
<code>y fit to</code>	For any Quad, this is the same as <code>y fit to!</code> .
<code>y fit to*</code>	It takes 2 arguments: <code>y fit*={⟨CoordA⟩}{⟨CoordB⟩}</code> . The height will be set so that $1+$ and $1-$ will fit CoordA and CoordB . This might result in a shape rotation
<code>y fit to!</code>	It takes 2 arguments: <code>y fit!={⟨CoordA⟩}{⟨CoordB⟩}</code> . The height will be set so that the distance between $\langle 1+ \rangle$ and $\langle 1- \rangle$ will be equal to the distance between $\langle \text{CoordA} \rangle$ and $\langle \text{CoordB} \rangle$. This will never result in a shape rotation.
<code>label top left</code>	It will place a label at the top left anchor
<code>label top center</code>	It will place a label at the top center anchor
<code>label top right</code>	It will place a label at the top right anchor
<code>label inner top left</code>	It will place a label at the inner top left anchor
<code>label inner top center</code>	It will place a label at the inner top center anchor
<code>label inner top right</code>	It will place a label at the inner top right anchor
<code>label bottom left</code>	It will place a label at the bottom left anchor
<code>label bottom center</code>	It will place a label at the bottom center anchor
<code>label bottom right</code>	It will place a label at the bottom right anchor

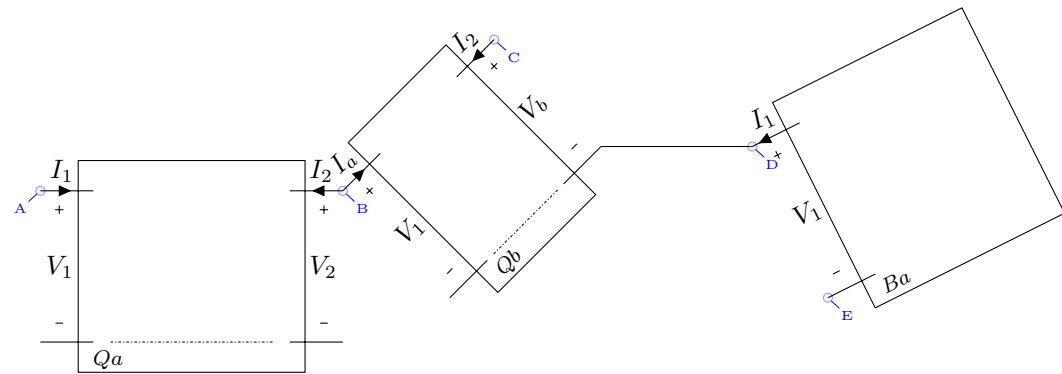
`label inner bottom left` It will place a label at the inner bottom left anchor
`label inner bottom center` It will place a label at the inner bottom center anchor
`label inner bottom right` It will place a label at the inner bottom right anchor

A small example of the *fit to* keys:

```

1 \begin{tikzpicture}
2 \draw (0,0) \pincoord(A,blue,225) ++(4,0) \pincoord(B,blue,-45) ++(2,2) \pincoord(C) ;
3
4 \draw (A) node[Quad,anchor=1+,outer x fit to={A}{B}](Qa){\footnotesize$Qa$};
5 \draw (B) node[Quad,anchor=1+,outer x fit to={B}{C},I1=$I_a$,V2=$V_b$](Qb){\footnotesize$Qb$};
6
7 \draw (Qb.2-) -- ++(2,0) \pincoord(D) ++(1,-2) \pincoord(E);
8
9 \draw (D) node[Black Box,anchor=1+,y fit to={D}{E}](Ba){\footnotesize$Ba$};
10
11 \draw (Qa.1-) ++(0,-1);
12 \end{tikzpicture}

```



3.2 Quad

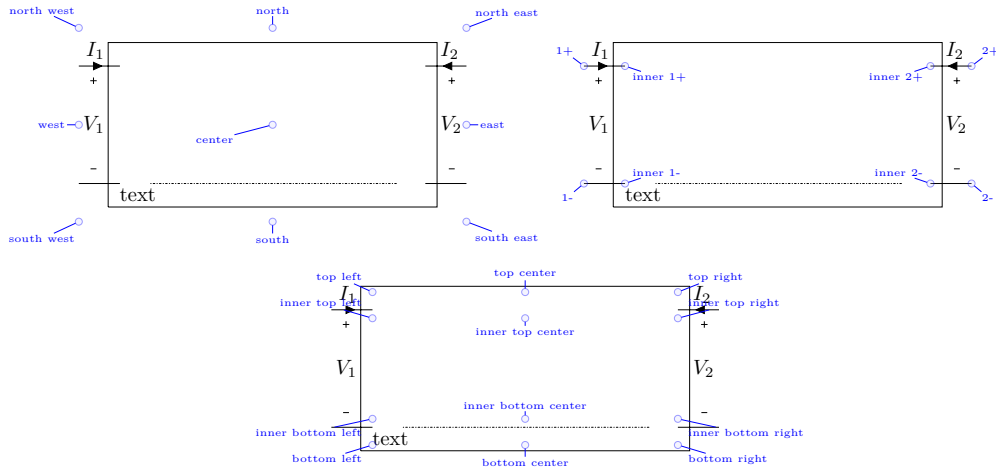
```

1 % Node use
2 node[Quad]{}
3
4 % To path use
5 (A) to[ToQuad] (B)

```

This is just the base shape, to be used in cases whereas one just want to emphasises part of a circuit (using, for instance, the *inner x fit to* key, or just mark a two port black box.

Note: There is also a *ToQuad* to be used in a *to[]* path, in which case the key *outer x fit to* style will be triggered with the starting and ending points of the *to[]* path.



3.2.1 Quad Keys

name <node-name>, when using a *to[]* path.

I_1	Initial value: $\$I_1\$$
I_2	Initial value: $\$I_2\$$
V_1	Initial value: $\$V_1\$$
V_2	Initial value: $\$V_2\$$

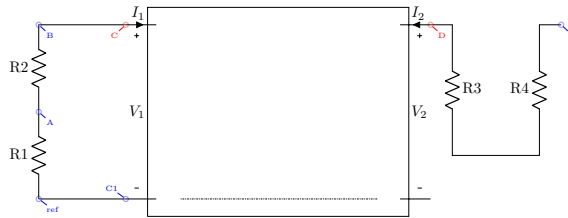
3.2.2 Examples of *fit to use*

Squeezing a Quadripole between two parts of a circuit (nodes C and D):

```

1 \begin{center}
2   \resizebox{0.5\textwidth}{!}{
3     \begin{tikzpicture}
4       \draw (0,0) \pincoord(ref) to[R=R1] ++(0,2) \pincoord(A) to[R=R2] ++(0,2) \pincoord(B)
5       -- ++(2,0) \pincoord(C,red,225) (C |- ref) \pincoord(C1,blue,135) -- (ref);
6       \draw (C) ++(7,0) \pincoord(D,red) -- ++(0.5,0) to[R=R3] ++(0,-3) -- ++(2,0) to[R=R4] ++(0,3)
7       -- ++(0.5,0) \pincoord(E);
8       \draw (C) node[Quad,anchor=1+,y fit to={C}{C1},outer x fit to={C}{D}]{};
9     \end{tikzpicture}
10  \end{center}

```

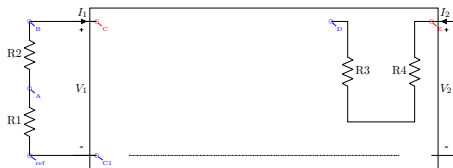


Fitting some circuit inside the Quadripole (nodes C and E):

```

1 \resizebox{0.4\textwidth}{!}{
2   \begin{tikzpicture}
3     \draw (0,0) \pincoord(ref) to[R=R1] ++(0,2) \pincoord(A) to[R=R2] ++(0,2) \pincoord(B)
4     -- ++(2,0) \pincoord(C,red) (C |- ref) \pincoord(C1) -- (ref);
5     \draw (C) ++(7,0) \pincoord(D) -- ++(0.5,0) to[R=R3] ++(0,-3) -- ++(2,0) to[R=R4] ++(0,3)
6     -- ++(0.5,0) \pincoord(E,red);
7     \draw (C) node[Quad,anchor=inner 1+,y fit to={C}{C1},inner x fit to={C}{E}]{};
8   \end{tikzpicture}

```



3.3 Quad Z

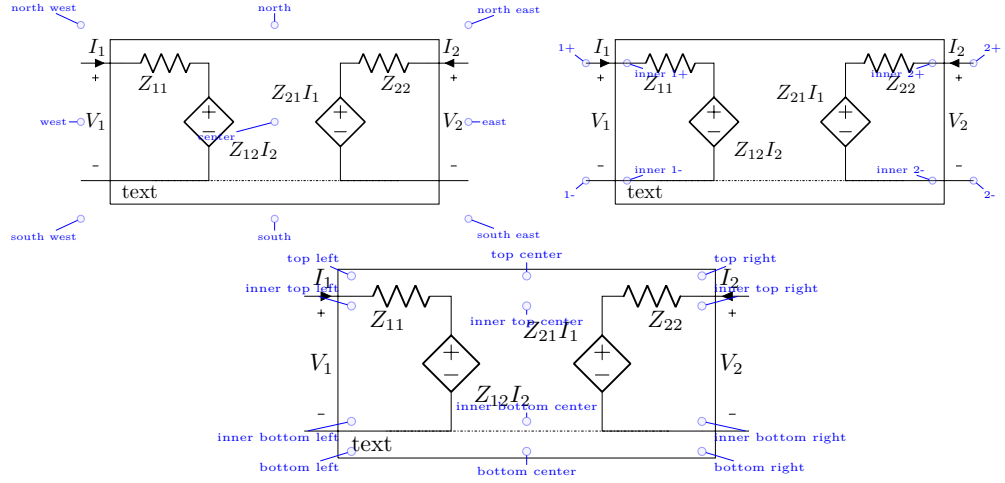
```

1 % Node use
2 node[Quad Z]{}
3
4 % To path use
5 (A) to[ToQuad Z] (B)

```

Besides the base anchors (see 3) it has 4 internal nodes: `<node>-Z11`, `<node>-Z12`, `<node>-Z21` and `<node>-Z22` and each of those sub-nodes has geographic anchors as defined at 2.1.

Note: There is also a `ToQuad Z` to be used in a `to[]` path, in which case the key `outer x fit to` style will be triggered with the starting and ending points of the `to[]` path.



3.3.1 Quad Z keys

<i>name</i>	$\langle \text{node-name} \rangle$, when using a <code>to[]</code> path.
<i>I1</i>	Initial value: $\$I_1\$$
<i>I2</i>	Initial value: $\$I_2\$$
<i>V1</i>	Initial value: $\$V_1\$$
<i>V2</i>	Initial value: $\$V_2\$$
<i>raw sources</i>	This will suppress the control variables (<i>I1</i> , <i>I2</i>) in the sources' labels
<i>Z11</i>	Initial value: $\$Z_{{11}}\$$
<i>Z12</i>	Initial value: $\$Z_{{12}}\$$
<i>Z21</i>	Initial value: $\$Z_{{21}}\$$
<i>Z22</i>	Initial value: $\$Z_{{22}}\$$
<i>Z11 label pos</i>	changes the label position. Defaults to: $\{\text{south west}\}\{\text{top left}\}$
<i>Z12 label pos</i>	changes the label position. Defaults to: $\{\text{south east}\}\{\text{top left}\}$
<i>Z21 label pos</i>	changes the label position. Defaults to: $\{\text{north west}\}\{\text{bottom right}\}$
<i>Z22 label pos</i>	changes the label position. Defaults to: $\{\text{south east}\}\{\text{top right}\}$

Note: The label pos keys expects two anchor names (\dots label pos= $\{\langle \text{anchor A} \rangle\}\{\langle \text{anchor B} \rangle\}$). The first anchors refers the sub-shape node and the second anchor is the text one.

3.4 Quad Y

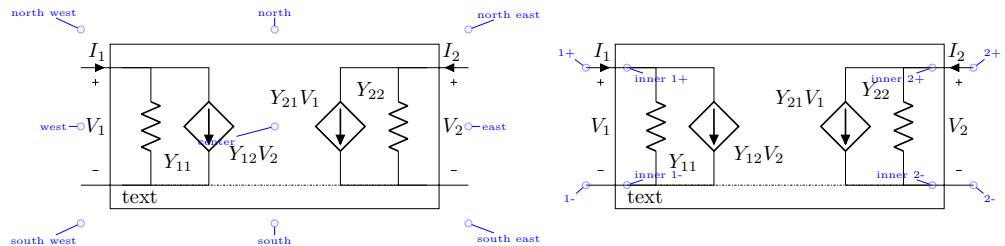
```

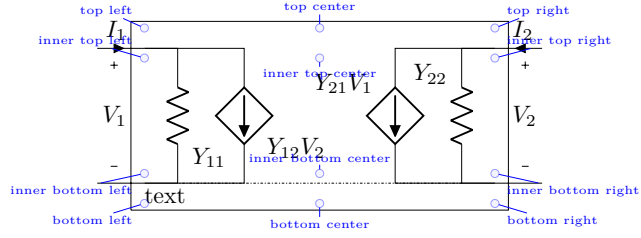
1 % Node use
2 node[Quad Y]{}
3
4 % To path use
5 (A) to[ToQuad Y] (B)

```

Besides the base anchors (see 3) it has 4 internal nodes: $\langle \text{node} \rangle\text{-Y11}$, $\langle \text{node} \rangle\text{-Y12}$, $\langle \text{node} \rangle\text{-Y21}$ and $\langle \text{node} \rangle\text{-Y22}$ and each of those sub-nodes has geographic anchors as defined at 2.1.

Note: There is also a *ToQuad Y* to be used in a `to[]` path, in which case the key *outer x fit to* style will be triggered with the starting and ending points of the `to[]` path.





3.4.1 Quad Y keys

<i>name</i>	$\langle \text{node-name} \rangle$, when using a <code>to[]</code> path.
<i>I1</i>	Initial value: $\$I_1\$$
<i>I2</i>	Initial value: $\$I_2\$$
<i>V1</i>	Initial value: $\$V_1\$$
<i>V2</i>	Initial value: $\$V_2\$$
<i>raw sources</i>	This will suppress the control variables (V1, V2) in the sources' labels
<i>Y11</i>	Initial value: $\$Y_{11}\$$
<i>Y12</i>	Initial value: $\$Y_{12}\$$
<i>Y21</i>	Initial value: $\$Y_{21}\$$
<i>Y22</i>	Initial value: $\$Y_{22}\$$
<i>Y11 label pos</i>	changes the label position. Defaults to: $\{\text{south west}\}\{\text{top left}\}$
<i>Y12 label pos</i>	changes the label position. Defaults to: $\{\text{south east}\}\{\text{top left}\}$
<i>Y21 label pos</i>	changes the label position. Defaults to: $\{\text{north west}\}\{\text{bottom right}\}$
<i>Y22 label pos</i>	changes the label position. Defaults to: $\{\text{north west}\}\{\text{bottom right}\}$

Note: The label pos keys expects two anchor names (\dots label pos= $\{\langle \text{anchor A} \rangle\}\{\langle \text{anchor B} \rangle\}$). The first anchors refers the sub-shape node and the second anchor is the text one.

3.5 Quad G

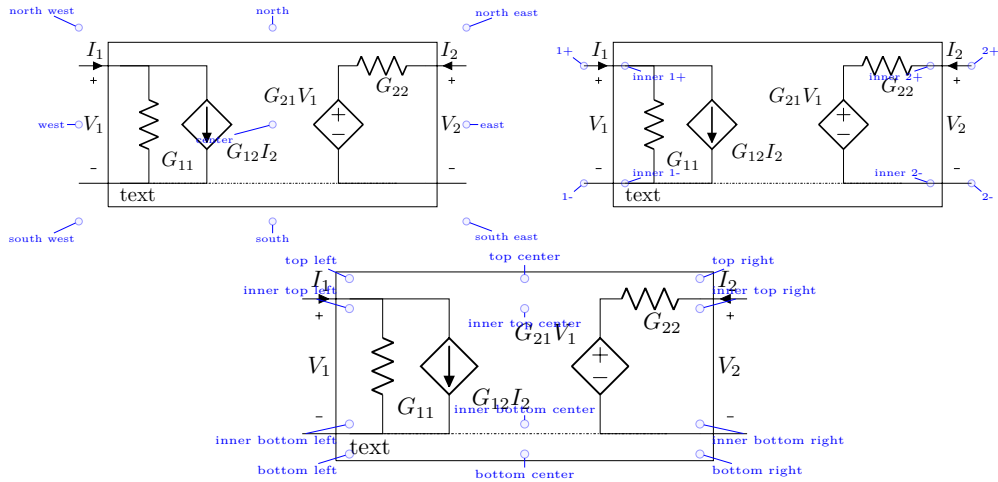
```

1 % Node use
2 node[Quad G]{}
3
4 % To path use
5 (A) to[ToQuad G] (B)

```

Besides the base anchors (see 3) it has 4 internal nodes: $\langle \text{node} \rangle\text{-G11}$, $\langle \text{node} \rangle\text{-G12}$, $\langle \text{node} \rangle\text{-G21}$ and $\langle \text{node} \rangle\text{-G22}$ and each of those sub-nodes has geographic anchors as defined at 2.1.

Note: There is also a `ToQuad G` to be used in a `to[]` path, in which case the key `outer x fit to` style will be triggered with the starting and ending points of the `to[]` path.



3.5.1 Quad G keys

<i>name</i>	$\langle \text{node-name} \rangle$, when using a <code>to[]</code> path.
<i>I1</i>	Initial value: $\$I_1\$$
<i>I2</i>	Initial value: $\$I_2\$$
<i>V1</i>	Initial value: $\$V_1\$$
<i>V2</i>	Initial value: $\$V_2\$$
<i>raw sources</i>	This will suppress the control variables (V1, I2) in the sources' labels
<i>G11</i>	Initial value: $\$G_{\{11\}}\$$
<i>G12</i>	Initial value: $\$G_{\{12\}}\$$
<i>G21</i>	Initial value: $\$G_{\{21\}}\$$
<i>G22</i>	Initial value: $\$G_{\{22\}}\$$
<i>G11 label pos</i>	changes the label position. Defaults to: $\{\text{south west}\}\{\text{top left}\}$
<i>G12 label pos</i>	changes the label position. Defaults to: $\{\text{south east}\}\{\text{top left}\}$
<i>G21 label pos</i>	changes the label position. Defaults to: $\{\text{north west}\}\{\text{bottom right}\}$
<i>G22 label pos</i>	changes the label position. Defaults to: $\{\text{south east}\}\{\text{top right}\}$

Note: The label pos keys expects two anchor names (\dots label pos= $\{\langle \text{anchor A} \rangle\}\{\langle \text{anchor B} \rangle\}$). The first anchors refers the sub-shape node and the second anchor is the text one.

3.6 Quad H

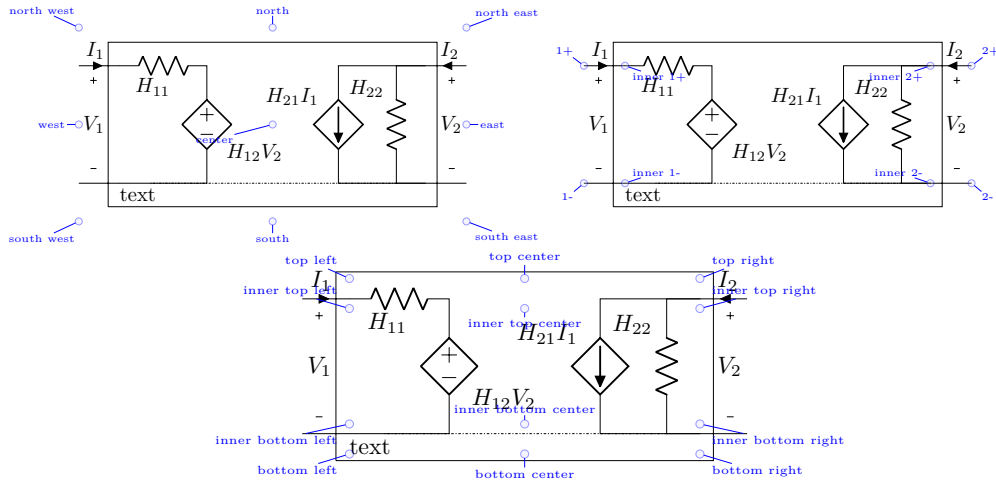
```

1 % Node use
2 node[Quad H] {}
3
4 % To path use
5 (A) to[ToQuad H] (B)

```

Besides the base anchors (see 3) it has 4 internal nodes: $\langle \text{node} \rangle\text{-H11}$, $\langle \text{node} \rangle\text{-H12}$, $\langle \text{node} \rangle\text{-H21}$ and $\langle \text{node} \rangle\text{-H22}$ and each of those sub-nodes has geographic anchors as defined at 2.1.

Note: There is also a *ToQuad H* to be used in a `to[]` path, in which case the key *outer x fit to* style will be triggered with the starting and ending points of the `to[]` path.



3.6.1 Quad H keys

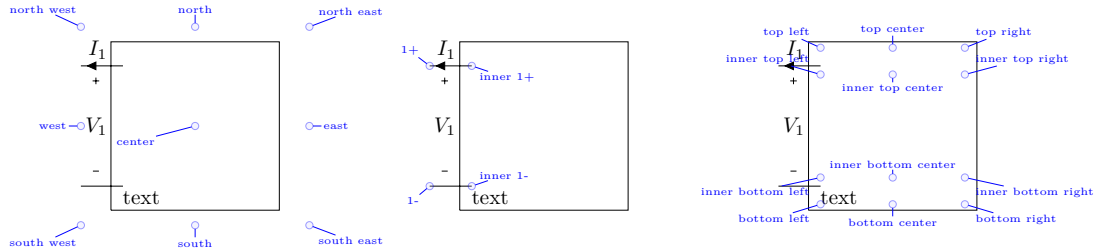
<i>name</i>	$\langle \text{node-name} \rangle$, when using a <code>to[]</code> path.
<i>I1</i>	Initial value: $\$I_1\$$
<i>I2</i>	Initial value: $\$I_2\$$
<i>V1</i>	Initial value: $\$V_1\$$
<i>V2</i>	Initial value: $\$V_2\$$
<i>raw sources</i>	This will suppress the control variables (I1, V2) in the sources' labels
<i>H11</i>	Initial value: $\$H_{\{11\}}\$$

<code>H12</code>	Initial value: $\$H_{\{12\}}\$$
<code>H21</code>	Initial value: $\$H_{\{21\}}\$$
<code>H22</code>	Initial value: $\$H_{\{22\}}\$$
<code>H11 label pos</code>	changes the label position. Defaults to: $\{south\ west\}\{top\ left\}$
<code>H12 label pos</code>	changes the label position. Defaults to: $\{south\ east\}\{top\ left\}$
<code>H21 label pos</code>	changes the label position. Defaults to: $\{north\ west\}\{bottom\ right\}$
<code>H22 label pos</code>	changes the label position. Defaults to: $\{north\ west\}\{bottom\ right\}$
Note: The label pos keys expects two anchor names (\dots label pos= $\{\langle anchor\ A\rangle\}\{\langle anchor\ B\rangle\}$). The first anchors refers the sub-shape node and the second anchor is the text one.	

4 Thevenin, Norton single port boxes

4.1 The Base Single Port / Black Box Shape

The base shape just draws a base box and sets some connection anchors: $1+$, $1-$, *inner* $1+$, *inner* $1-$, besides the geographic and text ones:



4.1.1 Base Keys

These applies to all *Black Box* shapes:

<code>base width</code>	The 'box' width
<code>half base width</code>	Ditto, half width. Initial value: $2\backslash pgf@circ@Rlen$.
<code>base height</code>	The distance between $1+$ and $1-$. The 'box' full height is equal to $2*(half\ base\ height + height\ ext + height\ ext+)$.
<code>half base height</code>	Ditto, half height. Initial value: $\backslash pgf@circ@Rlen/7$
<code>height ext</code>	Initial value: $2\backslash pgf@circ@Rlen/7$
<code>height ext+</code>	Initial value:0
<code>inner ext</code>	distance between the 'box' and <i>inner</i> $1+/1-/2+/2-$. initial value: $\backslash pgf@circ@Rlen/7$
<code>outer ext</code>	distance between the 'box' and $1+/1-/2+/2-$. initial value: $5\backslash pgf@circ@Rlen/14$
<code>inner marks</code>	If set, the inner anchors will be marked.
<code>outer marks</code>	If set, the outer anchors will be marked.
<code>invert</code>	The shape will be inverted, more or less like 'x scale=-1'.
<code>alt, opt</code>	Case a Voltage source is zero, a series impedance will be draw vertically.
<code>outer x fit to</code>	For any Black Box, this is the same as <i>outer x fit to!</i> .
<code>outer x fit to*</code>	It takes 2 arguments: <i>outer x fit*=</i> $\{\langle CoordA\rangle\}\{\langle CoordB\rangle\}$. The width will be set so that $\langle 1+\rangle$ and $\langle 2+\rangle$ (or $\langle 1-\rangle$ and $\langle 2-\rangle$, depending on the used anchor) will fit $\langle CoordA\rangle$ and $\langle CoordB\rangle$. This might result in a shape rotation.
<code>outer x fit to!</code>	It takes 2 arguments: <i>outer x fit!=</i> $\{\langle CoordA\rangle\}\{\langle CoordB\rangle\}$. The width will be set so that the distance between $\langle 1+\rangle$ and $\langle 2+\rangle$ (or $\langle 1-\rangle$ and $\langle 2-\rangle$, depending on the used anchor) will be the same as $\langle CoordA\rangle$ and $\langle CoordB\rangle$. This will never result in a shape rotation.
<code>inner x fit to</code>	For any Black Box, this is the same as <i>inner x fit to!</i> .
<code>inner x fit to*</code>	It takes 2 arguments: <i>inner x fit*=</i> $\{\langle CoordA\rangle\}\{\langle CoordB\rangle\}$. The width will be set so that $\langle inner\ 1+\rangle$ and $\langle inner\ 2+\rangle$ (or $\langle inner\ 1-\rangle$ and $\langle inner\ 2-\rangle$, depending on the used anchor) will fit $\langle CoordA\rangle$ and $\langle CoordB\rangle$. This might result in a shape rotation.

<code>inner x fit to!</code>	It takes 2 arguments: <code>inner x fit!=\{CoordA\}\{CoordB\}</code> . The width will be set so that the distance between <code>\inner 1+</code> and <code>\inner 2+</code> (or <code>\inner 1-</code> and <code>\inner 2-</code> , depending on the used anchor) will be the same as <code>\CoordA</code> and <code>\CoordB</code> . This will never result in a shape rotation.
<code>y fit to</code>	For any Black Box, this is the same as <code>y fit to*</code> .
<code>y fit to*</code>	It takes 2 arguments: <code>y fit!=\{CoordA\}\{CoordB\}</code> . The height will be set so that <code>1+</code> and <code>1-</code> will fit <code>CoordA</code> and <code>CoordB</code> . This might result in a shape rotation
<code>y fit to!</code>	It takes 2 arguments: <code>y fit!=\{CoordA\}\{CoordB\}</code> . The height will be set so that the distance between <code>\inner 1+</code> and <code>\inner 1-</code> will be equal to the distance between <code>\CoordA</code> and <code>\CoordB</code> . This will never result in a shape rotation.
<code>label top left</code>	It will place a label at the top left anchor
<code>label top center</code>	It will place a label at the top center anchor
<code>label top right</code>	It will place a label at the top right anchor
<code>label inner top left</code>	It will place a label at the inner top left anchor
<code>label inner top center</code>	It will place a label at the inner top center anchor
<code>label inner top right</code>	It will place a label at the inner top right anchor
<code>label bottom left</code>	It will place a label at the bottom left anchor
<code>label bottom center</code>	It will place a label at the bottom center anchor
<code>label bottom right</code>	It will place a label at the bottom right anchor
<code>label inner bottom left</code>	It will place a label at the inner bottom left anchor
<code>label inner bottom center</code>	It will place a label at the inner bottom center anchor
<code>label inner bottom right</code>	It will place a label at the inner bottom right anchor

4.2 Black Box

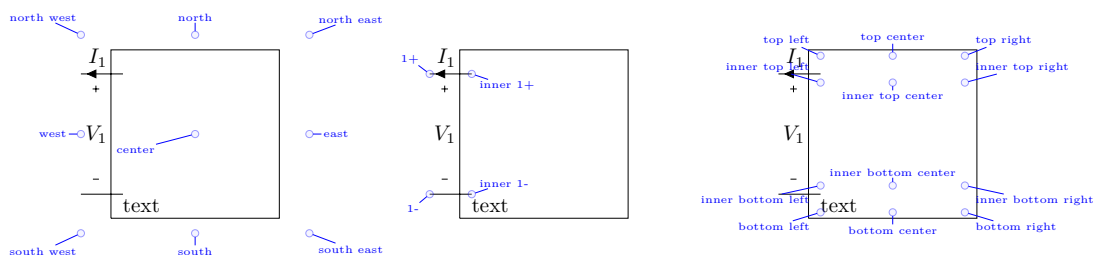
```

1 % Node use
2 node[Black Box]{}
3
4 % To path use
5 (A) to[ToBlack Box] (B)

```

This is just the base shape, to be used in cases whereas one just want to emphasises part of a circuit (using, for instance, the `inner x fit to` key, or just mark a single port black box.

Note: There is also a `ToBlack Box` to be used in a `to[]` path, in which case the key `y fit to` style will be triggered with the starting and ending points of the `to[]` path.



4.2.1 Black Box keys

<code>name</code>	<code>\node-name</code> , when using a <code>to[]</code> path.
<code>I1</code>	Initial value: <code>\I_1\$</code>
<code>V1</code>	Initial value: <code>\V_1\$</code>

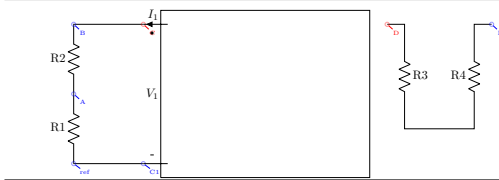
4.2.2 Examples of `fit to` use

Squeezing a Black Box between two parts of a circuit (nodes C and D):

```

1 \resizebox{0.4\textwidth}{!}{
2 \begin{tikzpicture}
3   \draw (0,0) \pincoord(ref) to[R=R1] ++(0,2) \pincoord(A) to[R=R2] ++(0,2) \pincoord(B)
4   -- ++(2,0) \pincoord(C,red) (C |- ref) \pincoord(C1) -- (ref);
5   \draw (C) ++(7,0) \pincoord(D,red) -- ++(0.5,0) to[R=R3] ++(0,-3) -- ++(2,0) to[R=R4] ++(0,3) --
6   ++(0.5,0) \pincoord(E);
7   \draw (C) node[Black Box,anchor=1+,y fit to={C}{C1},outer x fit to={C}{D}]{};
8 \end{tikzpicture}
9 }

```

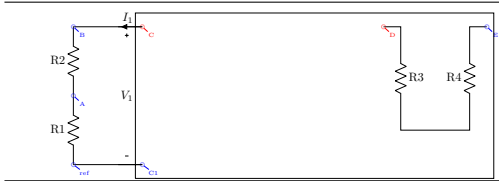


Fitting some circuit inside the Black Box (nodes C and E):

```

1 \resizebox{0.4\textwidth}{!}{
2 \begin{tikzpicture}
3   \draw (0,0) \pincoord(ref) to[R=R1] ++(0,2) \pincoord(A) to[R=R2] ++(0,2) \pincoord(B)
4   -- ++(2,0) \pincoord(C,red) (C |- ref) \pincoord(C1) -- (ref);
5   \draw (C) ++(7,0) \pincoord(D,red) -- ++(0.5,0) to[R=R3] ++(0,-3) -- ++(2,0) to[R=R4] ++(0,3) --
6   ++(0.5,0) \pincoord(E);
7   \draw (C) node[Black Box,anchor=inner 1+,y fit to={C}{C1},inner x fit to={C}{E}]{};
8 \end{tikzpicture}
9 }

```



4.3 Thevenin

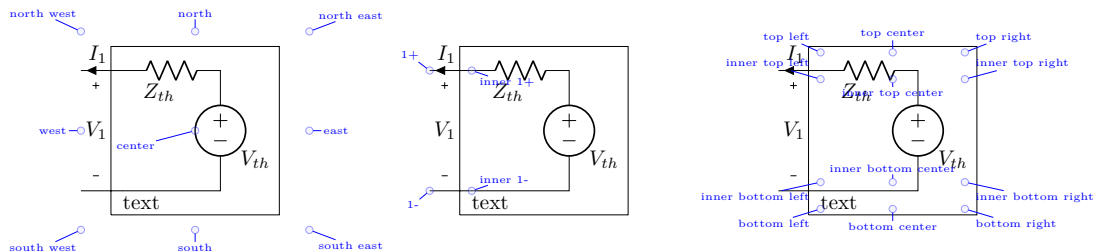
```

1 % Node use
2 node[Thevenin]{}
3
4 % To path use
5 (A) to[ToThevenin] (B)

```

This is the classical Thevenin circuit. Besides the base anchors (see 4.1) it has 2 internal nodes: `<node>-Zth` and `<node>-Vth` and each of those sub-nodes has geographic anchors as defined at 2.1.

Note: There is also a `ToThevenin` to be used in a `to[]` path, in which case the key `y fit to` style will be triggered with the starting and ending points of the `to[]` path.



4.3.1 Thevenin keys

<i>name</i>	<code><node-name></code> , when using a <code>to[]</code> path.
<i>I1</i>	Initial value: I_1
<i>V1</i>	Initial value: V_1
<i>Zth</i>	Initial value: Z_{th}
<i>Vth</i>	Initial value: V_{th}
<i>Zth label pos</i>	changes the label position. Defaults to: <code>{south west}{top left}</code>

`Vth label pos` changes the label position. Defaults to: `{south east}{top left}`

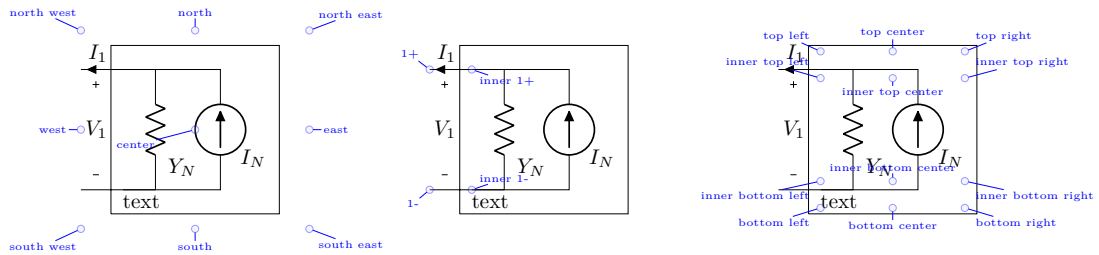
Note: The label pos keys expects two anchor names (... label pos= `{(anchor A)}{(anchor B)}`). The first anchors refers the sub-shape node and the second anchor is the text one.

4.4 Norton

```
1 % Node use
2 node[Norton]{}
3
4 % To path use
5 (A) to[ToNorton] (B)
```

This is the classical Norton circuit. Besides the base anchors (see 4.1) it has 2 internal nodes: `<node>-Yn` and `<node>-In` and each of those sub-nodes has geographic anchors as defined at 2.1.

Note: There is also a `ToNorton` to be used in a `to[]` path, in which case the key `y fit to` style will be triggered with the starting and ending points of the `to[]` path.



4.4.1 Norton keys

<code>name</code>	<code><node-name></code> , when using a <code>to[]</code> path.
<code>I1</code>	Initial value: <code>\$I_1\$</code>
<code>V1</code>	Initial value: <code>\$V_1\$</code>
<code>Yn</code>	Initial value: <code>\$Y_{N}\$</code>
<code>In</code>	Initial value: <code>\$I_{N}\$</code>
<code>Yn label pos</code>	changes the label position. Defaults to: <code>{south west}{top left}</code>
<code>In label pos</code>	changes the label position. Defaults to: <code>{south east}{top left}</code>

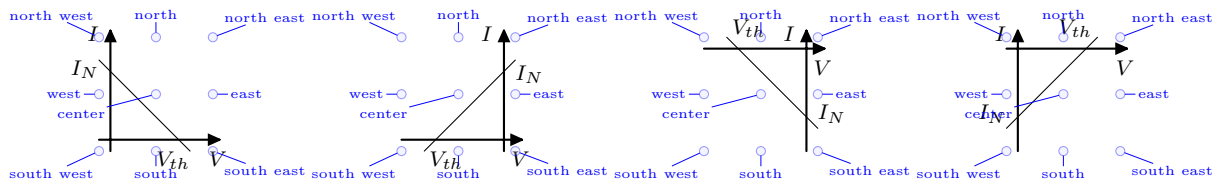
Note: The label pos keys expects two anchor names (... label pos= `{(anchor A)}{(anchor B)}`). The first anchors refers the sub-shape node and the second anchor is the text one.

5 Pseudo-Graph Shape

```
1 % Node use
2 node[PG load line]{}
3
4 node[PG linear load line]
```

Sometimes when representing a single port sub-circuit, one might use a X-Y graph, for which `gnuplot`[7] and `pgfplots`[1] are excellent choices, but a bit overkill if all you want is a crude representation of a linear load line.

This shape is just that, a X-Y graph mockup, that nicely fits inside a black box, and nothing else.



5.1 Pseudo-Graph Keys

These are the keys to fine tuning a shape:

<code>x axis</code>	X axis <i>name</i> . Initial value: V
<code>x val</code>	X axis <i>val</i> at the crossing point. Initial value: V_{th}
<code>y axis</code>	Y axis <i>name</i> . Initial value: I
<code>y val</code>	Y axis <i>val</i> at the crossing point. Initial value: I_N
<code>first quadrant</code>	First quadrant mock up. (which is also the default).
<code>second quadrant</code>	Second quadrant mock up.
<code>third quadrant</code>	Third quadrant mock up.
<code>fourth quadrant</code>	Fourth quadrant mock up.
<code>base width</code>	The <i>graph</i> width
<code>half base width</code>	Ditto, half width. Initial value: $0.5\pgf@circ@Rlen$.
<code>base height</code>	The <i>graph</i> height
<code>half base height</code>	Ditto, half height. Initial value: $0.5\pgf@circ@Rlen$.

Note: Besides these, one can also use the keys presented at 2.2.

6 Parallel Connections

Albeit simple, the association of quadripoles in parallel can be tedious if one has to do it many times over in a circuit. That for, an auxiliary, single command, is provided to easy it a bit.

`\QuadParConnect` [`<options>`] {`<Quad-A>`} {`<Quad-B>`}

new: 2025/03/07

This will interconnect one side of `<Quad-A>` and `<Quad-B>` *in parallel*. It is assumed that `<Quad-A>` is above `<Quad-B>`. New coordinates, “`c<Quad-Ref> <terminals>`” (note the blank), are created at the *connection terminals*, whereas `<Quad-Ref>` is either `<Quad-A>` (up) or `<Quad-B>` (down), `<terminals>` are the usual `1+` and `1-` (left) or `2+` and `2-` (right).

`<options>` can be any combination of:

<code>left</code>	The parallel connection will be done at the left side (default, assuming, terminals <code>1+</code> and <code>1-</code>)
<code>right</code>	The parallel connection will be done at the right side (default, assuming, terminals <code>2+</code> and <code>2-</code>)
<code>up</code>	The <i>connection terminals</i> will be at the top quadripole.
<code>down</code>	The <i>connection terminals</i> will be at the bottom quadripole.
<code>dots</code>	Dots will be added at the <i>connection terminals</i> .
<code>spacing</code>	To increase/decrease the spacing between the quadripoles terminals and the connection. The default distance being the natural <i>out ext</i> .
<code>swapped sides</code>	If the quadripoles are inverted, and the terminals <code>1+</code> and <code>1-</code> are at the right side.
<code>default sides</code>	Thats the default. <code>1+</code> and <code>1-</code> are at the left side.

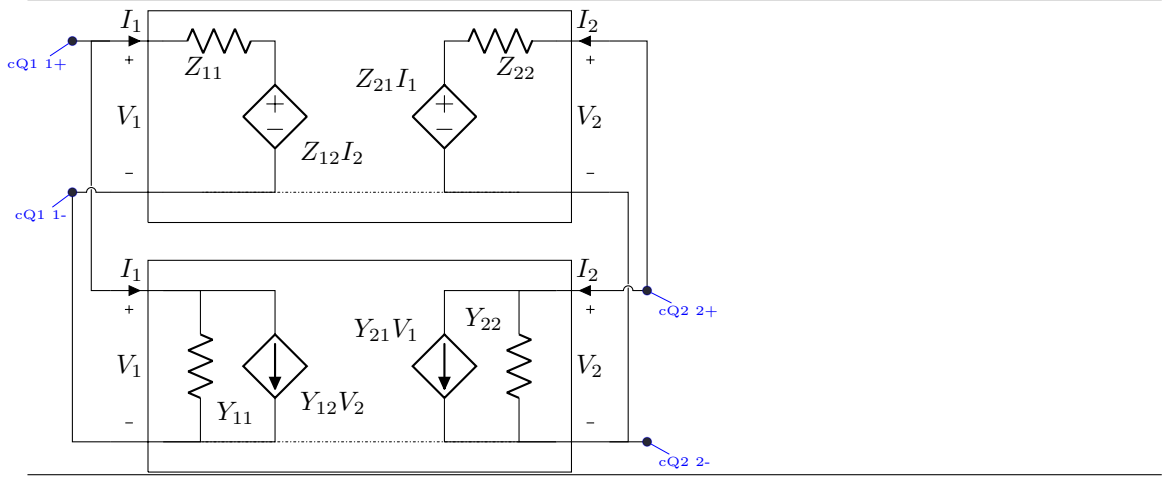
Note: It is assumed that the left/right directions are along the X axis, likewise, up/down are along the Y axis.

Note: This relies on the `\pathcross` command from the `tikzdotncross` package, which has to be loaded by the user.

```

1 \begin{tikzpicture}
2   \draw (0,0) node[Quad Z, anchor=1+](Q1){}
3   (Q1.south) node[Quad Y, anchor=north](Q2){};
4   \QuadParConnect[left,up,dots]{Q1}{Q2}
5   \QuadParConnect[right,down,dots]{Q1}{Q2}
6 \end{tikzpicture}

```



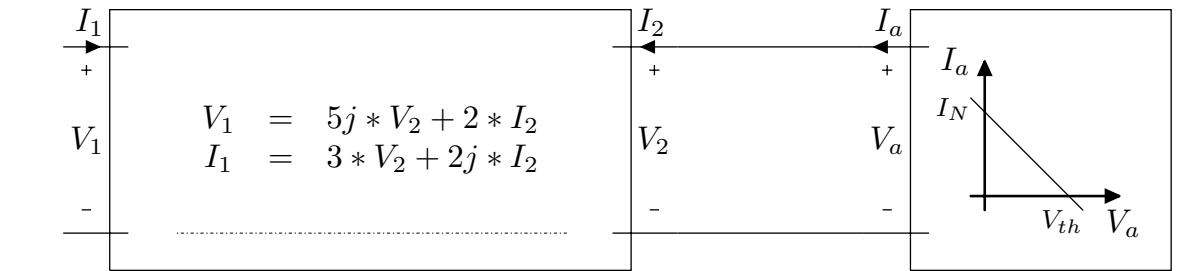
7 Examples of use

First of, a simple case of combining a generic Quad with equations and a generic Black Box with a Pseudo-Graph:

```

1 \resizebox{\textwidth}{!}{
2 \begin{tikzpicture}
3   \draw (0,0) \ncoord(ref) node[Quad,anchor=1+](Q1){}
4   (Q1.2+) -- ++(1,0) \ncoord(X) -- ++(1,0) node[Black Box,anchor=1+,V1=$V_a$,I1=$I_a$](B1){}
5   (Q1.2-) -- (B1.1-)
6   (B1.center) node[PG linear load line,x axis=$V_a$,y axis=$I_a$]{}
7   (Q1.center) node{$\begin{matrix} V_1 & \& 5j \cdot V_2 + 2 \cdot I_2 \\ I_1 & \& 3 \cdot V_2 + 2j \cdot I_2 \end{matrix}$}
8   \end{tikzpicture}
9 }

```

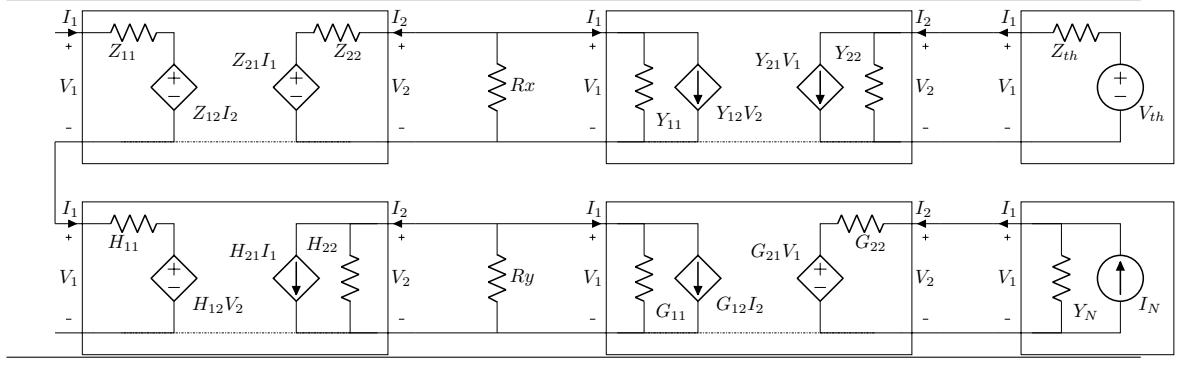


All default Quadripoles and Thevenin/Norton.

```

1 \resizebox{\textwidth}{!}{
2 \begin{tikzpicture}
3   \draw (0,0) \ncoord(ref) node[Quad Z,anchor=1+](Qz1){}
4   (Qz1.2+) -- ++(1.5,0) \ncoord(X) -- ++(1.5,0) node[Quad Y,anchor=1+](Qy1){}
5   (Qy1.2+) -- ++(1,0) node[Thevenin,anchor=1+](th1){}
6   (Qz1.1-) -- ++(0,-1.5) node[Quad H,anchor=1+](Qh1){}
7   (Qh1.2+) -- ++(1.5,0) \ncoord(Y) -- ++(1.5,0) node[Quad G,anchor=1+](Qg1){}
8   (Qg1.2+) -- ++(1,0) node[Norton,anchor=1+](nr1){}
9   (Qz1.2-) -- (Qy1.1-) (Qy1.2-) -- (th1.1-)
10  (Qh1.2-) -- (Qg1.1-) (Qg1.2-) -- (nr1.1-)
11  ;
12  \draw (X) to[R=$R_x$] (X |- Qz1.2-)
13  (Y) to[R=$R_y$] (Y |- Qh1.2-)
14  ;
15 \end{tikzpicture}
16 }

```

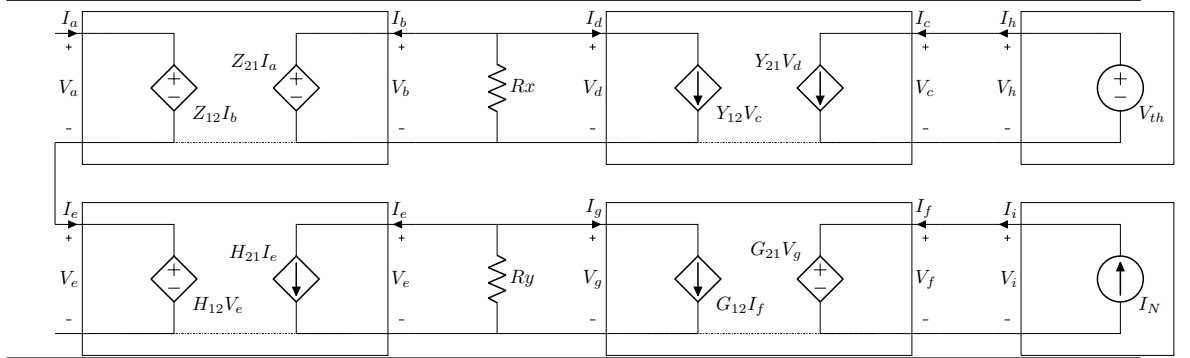


The same demo but with all parameter 11 and 22 zeroed, and changing the “control sources”

```

1 \resizebox{\textwidth}{!}{
2 \begin{tikzpicture}
3   \draw (0,0) \ncoord(ref) node[Quad Z,anchor=1+,Z11=0,Z22=0,I1=$I_a$,V1=$V_a$,I2=$I_b$,V2=$V_b$] (
4     Qz1){}
5     (Qz1.2+) -- ++(1.5,0) \ncoord(X) -- ++(1.5,0) node[Quad Y,anchor=1+,Y11=0,Y22=0,I1=$I_d$,V1=
6       $V_d$,I2=$I_c$,V2=$V_c$] (Qy1){}
7     (Qy1.2+) -- ++(1,0) node[Thevenin,anchor=1+,Zth=0,I1=$I_h$,V1=$V_h$] (th1){}
8     (Qz1.1-) -- ++(0,-1.5) node[Quad H,anchor=1+,H11=0,H22=0,I1=$I_e$,V1=$V_e$,I2=$I_e$,V2=$V_e$] (
9       Qh1){}
10    (Qh1.2+) -- ++(1.5,0) \ncoord(Y) -- ++(1.5,0) node[Quad G,anchor=1+,G11=0,G22=0,I1=$I_g$,V1=
11      $V_g$,I2=$I_f$,V2=$V_f$] (Qg1){}
12    (Qg1.2+) -- ++(1,0) node[Norton,anchor=1+,Yn=0,I1=$I_i$,V1=$V_i$] (nr1){}
13    (Qz1.2-) -- (Qy1.1-) (Qy1.2-) -- (th1.1-)
14    (Qh1.2-) -- (Qg1.1-) (Qg1.2-) -- (nr1.1-)
15  ;
16  \draw (X) to[R=$R_x$] (X |- Qz1.2-)
17        (Y) to[R=$R_y$] (Y |- Qh1.2-)
18  ;
19 \end{tikzpicture}
20 }

```

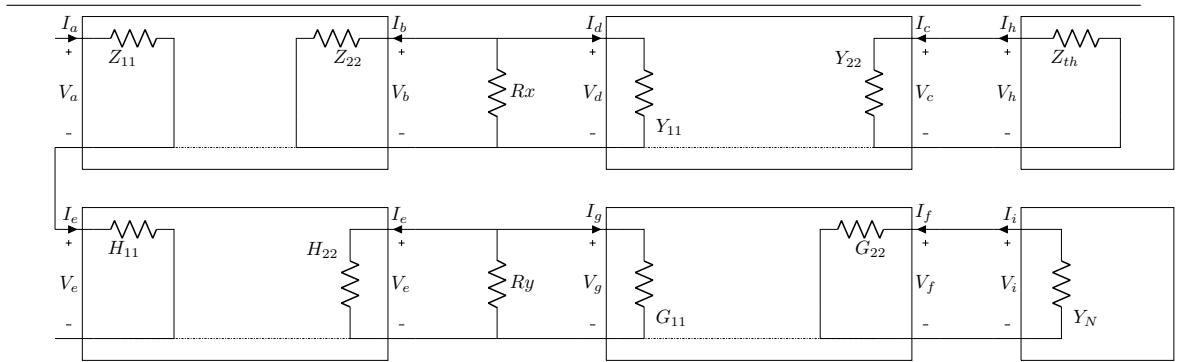


Now with the 12 and 21 parameters zeroed, normal form:


```

1 \resizebox{\textwidth}{!}{
2 \begin{tikzpicture}
3   \draw (0,0) \ncoord(ref) node[Quad Z,anchor=1+,Z12=0,Z21=0,I1=$I_a$,V1=$V_a$,I2=$I_b$,V2=$V_b$] (
4     Qz1){}
5     (Qz1.2+) -- ++(1.5,0) \ncoord(X) -- ++(1.5,0) node[Quad Y,anchor=1+,Y12=0,Y21=0,I1=$I_d$,V1=
6       $V_d$,I2=$I_c$,V2=$V_c$] (Qy1){}
7     (Qy1.2+) -- ++(1,0) node[Thevenin,anchor=1+,Vth=0,I1=$I_h$,V1=$V_h$] (th1){}
8     (Qz1.1-) -- ++(0,-1.5) node[Quad H,anchor=1+,H12=0,H21=0,I1=$I_e$,V1=$V_e$,I2=$I_e$,V2=$V_e$] (
9       Qh1){}
10    (Qh1.2+) -- ++(1.5,0) \ncoord(Y) -- ++(1.5,0) node[Quad G,anchor=1+,G12=0,G21=0,I1=$I_g$,V1=
11      $V_g$,I2=$I_f$,V2=$V_f$] (Qg1){}
12    (Qg1.2+) -- ++(1,0) node[Norton,anchor=1+,In=0,I1=$I_i$,V1=$V_i$] (nr1){}
13    (Qz1.2-) -- (Qy1.1-) (Qy1.2-) -- (th1.1-)
14    (Qh1.2-) -- (Qg1.1-) (Qg1.2-) -- (nr1.1-)
15    ;
16    \draw (X) to[R=$R_x$] (X |- Qz1.2-)
17          (Y) to[R=$R_y$] (Y |- Qh1.2-)
18    ;
19 \end{tikzpicture}
20 }

```

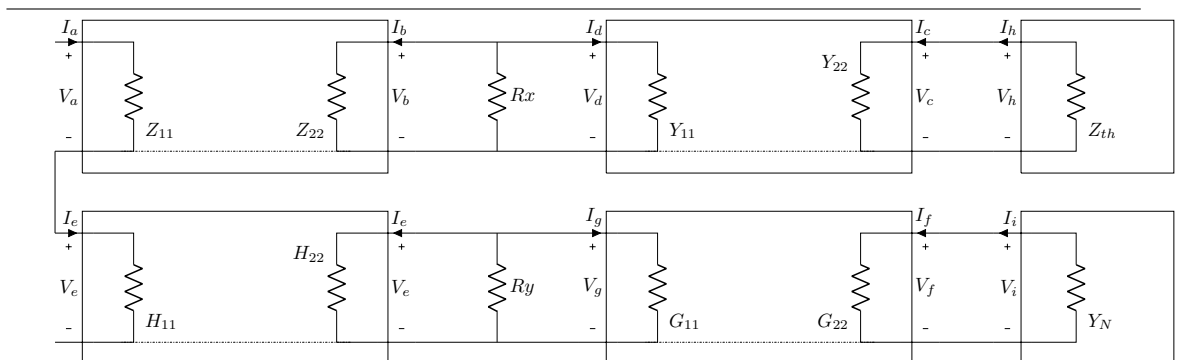


Same as last one, but with an alternate form:

```

1 \resizebox{\textwidth}{!}{
2 \begin{tikzpicture}
3   \draw (0,0) \ncoord(ref) node[Quad Z,anchor=1+,Z12=0,Z21=0,I1=$I_a$,V1=$V_a$,I2=$I_b$,V2=$V_b$] (
4     Qz1){}
5     (Qz1.2+) -- ++(1.5,0) \ncoord(X) -- ++(1.5,0) node[Quad Y,anchor=1+,Y12=0,Y21=0,I1=$I_d$,V1=
6       $V_d$,I2=$I_c$,V2=$V_c$] (Qy1){}
7     (Qy1.2+) -- ++(1,0) node[Thevenin,anchor=1+,Vth=0,I1=$I_h$,V1=$V_h$] (th1){}
8     (Qz1.1-) -- ++(0,-1.5) node[Quad H,anchor=1+,H12=0,H21=0,I1=$I_e$,V1=$V_e$,I2=$I_e$,V2=$V_e$] (
9       Qh1){}
10    (Qh1.2+) -- ++(1.5,0) \ncoord(Y) -- ++(1.5,0) node[Quad G,anchor=1+,G12=0,G21=0,I1=$I_g$,V1=
11      $V_g$,I2=$I_f$,V2=$V_f$] (Qg1){}
12    (Qg1.2+) -- ++(1,0) node[Norton,anchor=1+,In=0,I1=$I_i$,V1=$V_i$] (nr1){}
13    (Qz1.2-) -- (Qy1.1-) (Qy1.2-) -- (th1.1-)
14    (Qh1.2-) -- (Qg1.1-) (Qg1.2-) -- (nr1.1-)
15    ;
16    \draw (X) to[R=$R_x$] (X |- Qz1.2-)
17          (Y) to[R=$R_y$] (Y |- Qh1.2-)
18    ;
19 \end{tikzpicture}
20 }

```

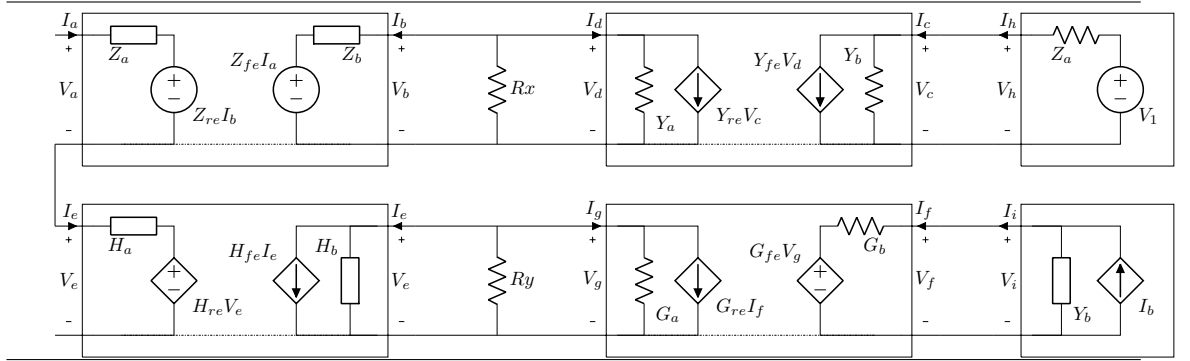


Setting all parameters, some impedances as zig-zag, others as generic, per quadripole:

```

1 \resizebox{\textwidth}{!}{
2 \begin{tikzpicture}
3   \draw (0,0) \ncoord(ref) node[Quad Z,alt,round sources,european,anchor=1+,Z11=$Z_a$,Z22=$Z_b$,Z12=
4     $Z_{re}$,Z21=$Z_{fe}$,I1=$I_a$,V1=$V_a$,I2=$I_b$,V2=$V_b$] (Qz1){}
5     (Qz1.2+) -- ++(1.5,0) \ncoord(X) -- ++(1.5,0) node[Quad Y,alt,anchor=1+,Y11=$Y_a$,Y22=$Y_b$,Y12=
6     $Y_{re}$,Y21=$Y_{fe}$,I1=$I_d$,V1=$V_d$,I2=$I_c$,V2=$V_c$] (Qy1){}
7     (Qy1.2+) -- ++(1,0) node[Thevenin,alt,anchor=1+,Vth=$V_1$,Zth=$Z_a$,I1=$I_h$,V1=$V_h$] (th1){}
8     (Qz1.1-) -- ++(0,-1.5) node[Quad H,european,alt,anchor=1+,H11=$H_a$,H22=$H_b$,H12=$H_{re}$,H21=
9     $H_{fe}$,I1=$I_e$,V1=$V_e$,I2=$I_e$,V2=$V_e$] (Qh1){}
10    (Qh1.2+) -- ++(1.5,0) \ncoord(Y) -- ++(1.5,0) node[Quad G,alt,anchor=1+,G11=$G_a$,G22=$G_b$,G12=
11    $G_{re}$,G21=$G_{fe}$,I1=$I_g$,V1=$V_g$,I2=$I_f$,V2=$V_f$] (Qg1){}
12    (Qg1.2+) -- ++(1,0) node[Norton,alt,control sources,european,anchor=1+,In=$I_b$,Yn=$Y_b$,I1=
13    $I_i$,V1=$V_i$] (nr1){}
14    (Qz1.2-) -- (Qy1.1-) (Qy1.2-) -- (th1.1-)
15    (Qh1.2-) -- (Qg1.1-) (Qg1.2-) -- (nr1.1-)
16  ;
17  \draw (X) to[R=$R_x$] (X |- Qz1.2-)
18        (Y) to[R=$R_y$] (Y |- Qh1.2-)
19  ;
20 \end{tikzpicture}
21 }

```



References

- [1] Christian Feuersänger. *The PGFPLOTS Package*. 2021, p. 573. URL: <http://mirrors.ctan.org/graphics/pgf/contrib/pgfplots/doc/pgfplots.pdf> (visited on 03/10/2025).
- [2] Alceu Frigeri. *The pgfkeysearch Package*. 2025, p. 3. URL: <https://mirrors.ctan.org/macros/latex/contrib/pgfkeysearch/doc/pgfkeysearch.pdf> (visited on 11/09/2025).
- [3] Alceu Frigeri. *The tikzdotncross Package*. 2025, p. 4. URL: <https://mirrors.ctan.org/graphics/pgf/contrib/tikzdotncross/doc/tikzdotncross.pdf> (visited on 11/09/2025).
- [4] Alceu Frigeri. *The tikzfxgraph Package*. 2025, p. 8. URL: <https://mirrors.ctan.org/graphics/pgf/contrib/tikzfxgraph/doc/tikzfxgraph.pdf> (visited on 11/09/2025).
- [5] Massimo A. Redaelli et al. *CircuitikZ*. 2025, p. 293. URL: <http://mirrors.ctan.org/graphics/pgf/contrib/circuitikz/doc/circuitikzmanual.pdf> (visited on 11/09/2025).
- [6] Till Tantau, Mark Wibrow, and Christian Feuersänger. *The TikZ and PGF Packages*. Institut für Theoretische Informatik / Universität zu Lübeck. 2023, p. 1321. URL: <http://mirrors.ctan.org/graphics/pgf/base/doc/pgfmanual.pdf> (visited on 03/10/2025).
- [7] Thomas Williams and Colin Kelley. *gnuplot 5.4*. 2022, p. 316. URL: https://gnuplot.sourceforge.net/docs_5.4/Gnuplot_5_4.pdf (visited on 03/10/2025).