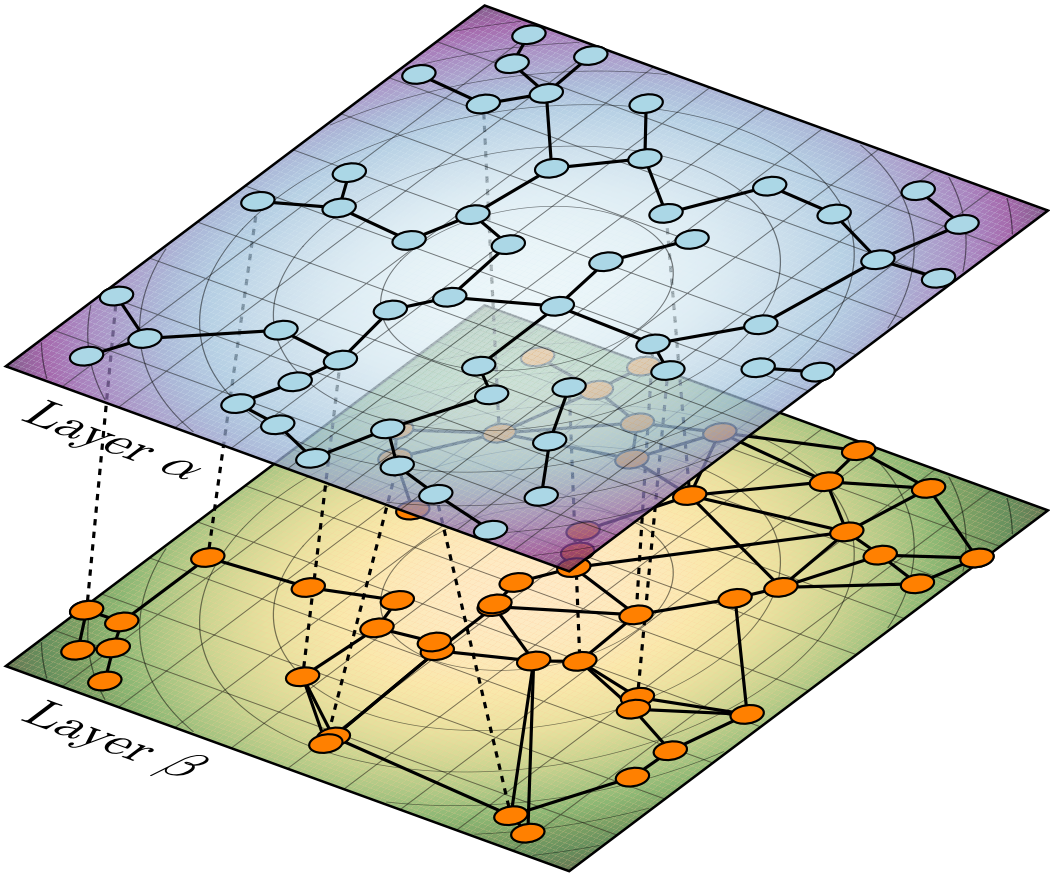

Layer α
Layer β

Released, May 2023

Contents

1 Introduction **5**
- 1.1 How to read this manual? . . . 6
  - 1.1.1 A few explanations . . . 6
  - 1.1.2 Inputs . . . 6
  - 1.1.3 Additional help . . . 7
- 1.2 Installation . . . 7
- 1.3 Additional necessary packages . . . 7

2 Simple Networks **9**
- 2.1 Vertex . . . 9
- 2.2 Edge . . . 14
- 2.3 Text . . . 20

3 Complex Networks **23**
- 3.1 Vertices . . . 23
- 3.2 Edges . . . 27

4 Multilayer Networks **31**
- 4.1 Simple Networks . . . 31
- 4.2 Complex Networks . . . 32
- 4.3 Layers and Layouts . . . 33
- 4.4 Plane . . . 34

5 Default Settings **37**
- 5.1 General Settings . . . 37
- 5.2 Vertex Style . . . 38
- 5.3 Edge Style . . . 38
- 5.4 Text Style . . . 39
- 5.5 Plane Style . . . 39

6 Troubleshooting and Support **41**
- 6.1 `tikz-network` Website . . . 41
- 6.2 Getting Help . . . 41
- 6.3 Errors, Warnings, and Informational Messages . . . 41
- 6.4 Package Dependencies . . . 41

A ToDo **43**
- A.1 Code to fix . . . 43
- A.2 Documentation . . . 43
- A.3 Features . . . 43

A.4 Add-ons 43

B Add-ons **45**
B.1 Python networks to TikZ with network2tikz 45
B.1.1 Introduction 45
B.1.2 Installation 46
B.1.3 Usage 46
B.1.4 Simple example 47
B.1.5 The plot function in detail 50

Bibliography **57**

1 Introduction

In recent years, complex network theory becomes more and more popular within the scientific community. Besides a solid mathematical base on which these theories are built on, a visual representation of the networks allow communicating complex relationships to a broad audience.

Nowadays, a variety of great visualization tools are available, which helps to structure, filter, manipulate and of course to visualize the networks. However, they come with some limitations, including the need for specific software tools, difficulties to embed the outputs properly in a LaTeX file (e.g. font type, font size, additional equations and math symbols needed, ...) and challenges in the post-processing of the graphs, without rerunning the software tools again.

In order to overcome this issues, the package `tikz-network` was created. Some of the features are:

- LaTeX is a standard for scientific publications and widely used
- beside LaTeX no other software is needed
- no programming skills are needed
- simple to use but allows 100% control over the output
- easy for post-processing (e.g. adding drawings, texts, equations,...)
- same fonts, font sizes, mathematical symbols, ...as in the document
- no quality loss of the output due to the pdf format
- networks are easy to adapt or modify for lectures or small examples
- able to visualize larger networks
- three-dimensional visualization of (multilayer) networks
- compatible with other visualization tools

1.1 How to read this manual?

The aim of this manual is to describe the use of the `tikz-network` library for visualizing networks. To ensure an easy use of the elements and to keep the clarity, this manual is structured as follows:

- In Chapter 2 the elements to create simple networks (by hand) in a plane are explained. Thereby, the use of the commands `\Vertex` and `\Edge` are shown.
- How to create complex networks from external files[1] are explained in Chapter 3. The main commands, therefore are `\Vertices` and `\Edges` which are using the same options as in the simple case.
- In Chapter 4, the visualization of multilayer networks is explained. Additional visualization tools such as `\Plane` and `Layer` are introduced.
- The default settings used and how they can be modified is explained in Chapter 5.
- Information about troubleshooting and support is given in Chapter 6
- Since this is the alpha version (0.1) of the package, features which will be probably added and commands which have to be fixed are listed in Appendix B.

[1] e.g. `igraph` or `networkx`

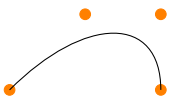
1.1.1 A few explanations

The images in this manual are created with the `tikz-network` library or Ti*k*Z . The code used for this is specified for each image.

1.1
```
\begin{tikzpicture}
  \filldraw (-.2,.2) circle (2pt) (.2,.2) circle (2pt)
      ;
  \draw (0,0) circle (5mm) (-.3,-.1) .. controls
      (0,-.3) .. (.3,-.1);
\end{tikzpicture}
```

Special additions which are needed for a better understanding are shown in orange but are not in the sample code available.

1.2
```
\begin{tikzpicture}
  \draw (0,0) .. controls (1,1) and (2,1) .. (2,0);
\end{tikzpicture}
```



1.1.2 Inputs

The commands in the `tikz-network` library (e.g. `\Vertex`, `\Edge`) always start with capital letters and DO NOT need a semicolon «;» at the end. Boolean arguments start also with capital letters (e.g. ⟨*NoLabel*⟩). Arguments which need an user input, use are written in small letters (e.g. ⟨*color*⟩).

Basically, one can distinguish between the mandatory argument { } and the optional argument [ ]. The first values must be entered compulsory. By contrast, nothing has to be entered for the optional input. Additional features (e.g. ⟨*size*⟩)) can be activated when entering optional parameters.

When entering size values the base unit is always predefined in [*cm*][2], except for line widths which are dedined in [*pt*]. Percentage values % are always specified as decimal values; for example, 100% = 1.0 and 10% corresponds to 0.1.

[2] The default unit can be changed with `\SetDefaultUnit`; see Section 5.1

1.1.3 *Additional help*

Is the manual not enough, occur some ambiguities or some Ti*k*Z commands are unclear, please have a look in the "Ti*k*Z and PGF Manual" from Till Tantau[3].

Should you have any further questions, please do not hesitate to contact me.

[3] `http://mirror.switch.ch/ftp/mirror/tex/graphics/pgf/base/doc/pgfmanual.pdf`

1.2 *Installation*

Actually, we can hardly speak of an installation since only the necessary package `tikz-network` must be loaded in the preamble of your document.

The current release of the package is available via CTAN[4]. A release candidate for the next version of `tikz-network` is available on github[5]

Is the package installed or the style file is stored in the folder of the main file, so the library can be imported, as the following example shows:

[4] `https://ctan.org/pkg/tikz-network`

[5] `https://github.com/hackl/`

1.3
```
% ------------
% header
\documentclass{scrreprt}

% ------------
% packages
\usepackage{tikz-network}
```

1.3 *Additional necessary packages*

To use all commands and options of Ti*k*Z , possibly some packages need to be reloaded. These missing files (or their names) appear in the error log when you convert the file. However, for the package described in this manual, it is sufficient to use the library and the Ti*k*Z standard commands.

2 Simple Networks

2.1 Vertex

On essential command is \Vertex, which allow placing vertices in the document and modify their appearance.

\Vertex[⟨*local options*⟩]{*Name*}

In order to be able to place a vertex, a non-empty *Name* argument is required. This argument defines the vertex's reference name, which must be unique. Mathematical symbols are not allowed as name as well as no blank spaces. The *Name* should not be confused with the ⟨*label*⟩, that is used for display; for example one may want to display A_1 while the name will be *A1*.

For a \Vertex the following options are available:

Option	Default	Type	Definition
x	0	measure	x-coordinate
y	0	measure	y-coordinate
size	{}	measure	diameter of the circle
color	{}	color	fill color of vertex
opacity	{}	number	opacity of the fill color
shape	{}	string	shape of the vertex
label	{}	string	label
fontsize	{}	string	font size of the label
fontcolor	{}	color	font color of the label
fontscale	{}	number	scale of the label
position	center	value[a]	label position
distance	0	measure	label distance from the center
style	{}	string	additional Ti*k*Z styles
layer	{}	number	assigned layer of the vertex
NoLabel	false	Boolean	delete the label
IdAsLabel	false	Boolean	uses the *Name* as label
Math	false	Boolean	displays the label in math mode
RGB	false	Boolean	allow RGB colors
Pseudo	false	Boolean	create a pseudo vertex

[a] either measure or string

Table 2.1: Local options for the \Vertex command.

The order how the options are entered does not matter. Changes to the default Vertex layout can be made with \SetVertexStyle[1]

[1] see Section 5.2

\Vertex[⟨*x*⟩=*measure*,⟨*y*⟩=*measure*]{*Name*}

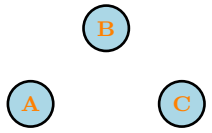
The location of the vertices are determined by Cartesian coordinates in ⟨*x*⟩ and ⟨*y*⟩. The coordinates are optional. If no coordinates are determined the vertex will be placed at the origin $(0, 0)$. The

entered *measures* are in default units (cm). Changing the unites (locally) can be done by adding the unit to the *measure*[2]. Changes to the default setting can be made with `\SetDefaultUnit`[3].

[2] e.g. x=1 in

[3] see Section 5.1

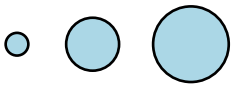
2.1
```
\begin{tikzpicture}
  \Vertex{A}
  \Vertex[x=1,y=1]{B}
  \Vertex[x=2]{C}
\end{tikzpicture}
```



`\Vertex[`⟨*size*⟩`=`*measure*`]{`*Name*`}`

The diameter of the vertex can be changed with the option ⟨*size*⟩. Per default a vertex has 0.6 cm in diameter. Also, here the default units are cm and have not to be added to the *measure*.

2.2
```
\begin{tikzpicture}
  \Vertex[size=.3]{A}
  \Vertex[x=1,size=.7]{B}
  \Vertex[x=2.3,size=1]{C}
\end{tikzpicture}
```



`\Vertex[`⟨*color*⟩`=`*color*`]{`*Name*`}`

To change the fill color of each vertex individually, the option ⟨*color*⟩ has to be used. Without the option ⟨*RGB*⟩ set, the default Ti*k*Z and LaTeX colors can be applied.

2.3
```
\begin{tikzpicture}
  \Vertex[color = blue]{A}
  \Vertex[x=1,color=red]{B}
  \Vertex[x=2,color=green!70!blue]{C}
\end{tikzpicture}
```

`\Vertex[`⟨*opacity*⟩`=`*number*`]{`*Name*`}`

With the option ⟨*opacity*⟩ the opacity of the vertex fill color can be modified. The range of the *number* lies between 0 and 1. Where 0 represents a fully transparent fill and 1 a solid fill.

2.4
```
\begin{tikzpicture}
  \Vertex[opacity = 1]{A}
  \Vertex[x=1,opacity =.7]{B}
  \Vertex[x=2,opacity =.2]{C}
\end{tikzpicture}
```

`\Vertex[`⟨*shape*⟩`=`*string*`]{`*Name*`}`

With the option ⟨*shape*⟩ the shape of the vertex can be modified. Thereby the shapes implemented in Ti*k*Z can be used, including: *circle*, *rectangle*, *diamond*, *trapezium*, *semicircle*, *isosceles triangle*, ...

2.5
```
\begin{tikzpicture}
  \Vertex[shape = rectangle]{A}
  \Vertex[x=1,shape = diamond]{B}
  \Vertex[x=2,shape = isosceles triangle]{C}
\end{tikzpicture}
```

`\Vertex[`⟨*label*⟩`=`*string*`]{`*Name*`}`

In `tikz-network` there are several ways to define the labels of the vertices and edges. The common way is via the option ⟨*label*⟩. Here, any *string* argument can be used, including blank spaces. The environment $ $ can be used to display mathematical expressions.

2.6
```
\begin{tikzpicture}
  \Vertex[label=foo]{A}
  \Vertex[x=1,label=bar]{B}
  \Vertex[x=2,label=$u_1$]{C}
\end{tikzpicture}
```

`\Vertex[`⟨*label*⟩`=`*string*`,`⟨*fontsize*⟩`=`*string*`]{`*Name*`}`

The font size of the ⟨*label*⟩ can be modified with the option ⟨*fontsize*⟩. Here common LaTeX font size commands[4] can be used to change the size of the label.

[4] e.g. `\tiny`, `\scriptsize`, `\footnotesize`, `\small`, ….

2.7
```
\begin{tikzpicture}
  \Vertex[label=foo,fontsize=\normalsize]{A}
  \Vertex[x=1,label=bar,fontsize=\tiny]{B}
  \Vertex[x=2,label=$u_1$,fontsize=\large]{C}
\end{tikzpicture}
```

`\Vertex[`⟨*label*⟩`=`*string*`,`⟨*fontcolor*⟩`=`*color*`]{`*Name*`}`

The color of the ⟨*label*⟩ can be changed with the option ⟨*fontcolor*⟩. Currently, only the default Ti*k*Z and LaTeX colors are supported[5].

[5] **TODO!** Add RGB option!

2.8
```
\begin{tikzpicture}
  \Vertex[label=foo,fontcolor=blue]{A}
  \Vertex[x=1,label=bar,fontcolor=magenta]{B}
  \Vertex[x=2,label=$u_1$,fontcolor=red]{C}
\end{tikzpicture}
```

`\Vertex[`⟨*label*⟩`=`*string*`,`⟨*fontscale*⟩`=`*number*`]{`*Name*`}`

Contrary to the option ⟨*fontsize*⟩, the option ⟨*fontscale*⟩ does not change the font size itself but scales the curent font size up or down. The *number* defines the scale, where numbers between 0 and 1 down scale the font and numbers greater then 1 up scale the label. For example 0.5 reduces the size of the font to 50% of its originial size, while 1.2 scales the font to 120%.

2.9
```
\begin{tikzpicture}
  \Vertex[label=foo,fontscale=0.5]{A}
  \Vertex[x=1,label=bar,fontscale=1]{B}
  \Vertex[x=2,label=$u_1$,fontscale=2]{C}
\end{tikzpicture}
```

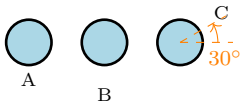
`\Vertex[`⟨*label*⟩`=`*string*`,`⟨*position*⟩`=`*value*`,`⟨*distance*⟩`=`*number*`]{`*Name*`}`

Per default the ⟨*position*⟩ of the ⟨*label*⟩ is in the *center* of the vertex. Classical Ti*k*Z commands[6] can be used to change the ⟨*position*⟩ of the ⟨*label*⟩. Instead, using such command, the position can be determined via an angle, by entering a *number* between −360 and 360. The origin (0°) is the y axis. A positive *number* change the ⟨*position*⟩ counter clockwise, while a negative *number* make changes clockwise.

[6] e.g. *above*, *below*, *left*, *right*, *above left*, *above right*,...

With the option, ⟨*distance*⟩ the distance between the vertex and the label can be changed.

2.10
```
\begin{tikzpicture}
  \Vertex[label=A,position=below]{A}
  \Vertex[x=1,label=B,position=below,distance=2mm]{B}
  \Vertex[x=2,label=C,position=30,distance=1mm]{C}
\end{tikzpicture}
```



`\Vertex[`⟨*style*⟩`={`*string*`}]{`*Name*`}`

Any other Ti*k*Z style option or command can be entered via the option ⟨*style*⟩. Most of these commands can be found in the "Ti*k*Z and PGF Manual". Contain the commands additional options (e.g.⟨*shading*⟩=*ball*), then the argument for the ⟨*style*⟩ has to be between { } brackets.

2.11
```
\begin{tikzpicture}
  \Vertex[style={color=green}]{A}
  \Vertex[x=1,style=dashed]{B}
  \Vertex[x=2,style={shading=ball}]{C}
\end{tikzpicture}
```

`\Vertex[`⟨*IdAsLabel*⟩`]{`*Name*`}`

`\Vertex[`⟨*NoLabel*⟩`,`⟨*label*⟩`=`*string*`]{`*Name*`}`

⟨*IdAsLabel*⟩ is a Boolean option which assigns the *Name* of the vertex as label. On the contrary, ⟨*NoLabel*⟩ suppress all labels.

2.12
```
\begin{tikzpicture}
  \Vertex[IdAsLabel]{A}
  \Vertex[x=1,label=B,NoLabel]{B}
  \Vertex[x=2,IdAsLabel,NoLabel]{C}
\end{tikzpicture}
```

`\Vertex[`⟨*Math*⟩`,`⟨*label*⟩`=`*string*`]{`*Name*`}`

The option ⟨*Math*⟩ allows transforming labels into mathematical expressions without using the $ $ environment. In combination with ⟨*IdAsLabel*⟩ allows this option also mathematical expressions by the definition of the vertex *Name*.

2.13
```
\begin{tikzpicture}
  \Vertex[IdAsLabel]{A1}
  \Vertex[x=1,label=B_1,Math]{B}
  \Vertex[x=2,Math,IdAsLabel]{C_1}
\end{tikzpicture}
```

`\Vertex[`⟨*RGB*⟩`,`⟨*color*⟩`=`*RGB values*`]{`*Name*`}`

In order to display RGB colors for the vertex fill color, the option ⟨*RGB*⟩ has to be entered. In combination with this option, the ⟨*color*⟩ hast to be a list with the *RGB values*, separated by «,» and within { }.[7]

[7] e.g. the RGB code for white: {255, 255, 255}

2.14
```
\begin{tikzpicture}
  \Vertex[RGB,color={127,201,127}]{A}
  \Vertex[x=1,RGB,color={190,174,212}]{B}
  \Vertex[x=2,RGB,color={253,192,134}]{C}
\end{tikzpicture}
```

`\Vertex[`⟨*Pseudo*⟩`]{`*Name*`}`

The option ⟨*Pseudo*⟩ creates a pseudo vertex, where only the vertex name and the vertex coordinate will be drawn. Edges etc, can still be assigned to this vertex.

2.15
```
\begin{tikzpicture}
  \Vertex{A}
  \Vertex[x=2,Pseudo]{B}
\end{tikzpicture}
```

`\Vertex[`⟨*layer*⟩`=`*number*`]{`*Name*`}`

With the option ⟨*layer*⟩ the vertex can be assigned to a specific layer. More about this option and the use of layers is explained in Chapter 4.

2.2 Edge

The second essential command is an \Edge, which allow connecting two vertices.

\Edge[⟨*local options*⟩](*Vertex i*)(*Vertex j*)

Edges can be generated between one or two vertices. In the first case, a self-loop will be generated. As mandatory arguments the *Names* of the vertices which should be connected must be entered between ( ) brackets. In case of a directed edge, the order is important. An edge is created from *Vertex i* (origin) to *Vertex j* (destination).

For an \Edge the following options are available:

Table 2.2: Local options for the \Edge command.

Option	Default	Type	Definition
lw	{}	measure	line width of the edge
color	{}	color	edge color
opacity	{}	number	opacity of the edge
bend	0	number	angle out/in of the vertex
label	{}	string	label
fontsize	{}	string	font size of the label
fontcolor	{}	color	font color of the label
fontscale	{}	number	scale of the label
position	{}	string	label position
distance	0.5	number	label distance from vertex i
style	{}	string	additional TikZ styles
path	{}	list	path over several vertices
loopsize	1cm	measure	size parameter of the self-loop
loopposition	0	number	orientation of the self-loop
loopshape	90	number	loop angle out/in of the vertex
Direct	false	Boolean	allow directed edges
Math	false	Boolean	displays the label in math mode
RGB	false	Boolean	allow RGB colors
NotInBG	false	Boolean	edge is not in the background layer

The options ⟨*loopsize*⟩, ⟨*loopposition*⟩, and ⟨*loopsize*⟩ are only for self-loops available.

\Edge(*Vertex i*)(*Vertex j*)

An edge is created between *Vertex i* and *Vertex j*.

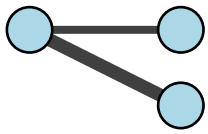
2.16
```
\begin{tikzpicture}
  \Vertex{A} \Vertex[x=2]{B}
  \Edge(A)(B)
\end{tikzpicture}
```

\Edge[⟨*lw*⟩=*measure*](*Vertex i*)(*Vertex j*)

The line width of an edge can be modified with the option ⟨*lw*⟩. Here, the unit of the *measure* has to be specified. The default value is 1.5 pt.

2.17

```
\begin{tikzpicture}
  \Vertex{A} \Vertex[x=2]{B} \Vertex[x=2,y=-1]{C}
  \Edge[lw=3pt](A)(B)
  \Edge[lw=5pt](A)(C)
\end{tikzpicture}
```

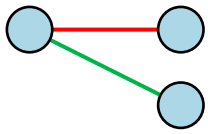


`\Edge[`⟨*color*⟩`=`*color*`](`*Vertex i*`)(`*Vertex j*`)`

To change the line color of each edge individually, the option ⟨*color*⟩ has to be used. Without the option ⟨*RGB*⟩ set, the default Ti*k*Z and LaTeX colors can be applied.

2.18

```
\begin{tikzpicture}
  \Vertex{A} \Vertex[x=2]{B} \Vertex[x=2,y=-1]{C}
  \Edge[color=red](A)(B)
  \Edge[color=green!70!blue](A)(C)
\end{tikzpicture}
```

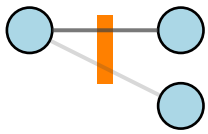


`\Edge[`⟨*opacity*⟩`=`*number*`](`*Vertex i*`)(`*Vertex j*`)`

With the option ⟨*opacity*⟩ the opacity of the edge line can be modified. The range of the *number* lies between 0 and 1. Where 0 represents a fully transparent fill and 1 a solid fill.

2.19

```
\begin{tikzpicture}
  \Vertex{A} \Vertex[x=2]{B} \Vertex[x=2,y=-1]{C}
  \Edge[opacity=.7](A)(B)
  \Edge[opacity=.2](A)(C)
\end{tikzpicture}
```

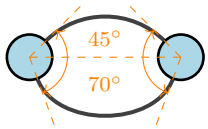


`\Edge[`⟨*bend*⟩`=`*number*`](`*Vertex i*`)(`*Vertex j*`)`

The shape of the edge can be modified with the ⟨*bend*⟩ option. If nothing is specified a straight edge, between the vertices, is drawn. The *number* defines the angle in which the edge is diverging from its straight connection. A positive *number* bend the edge counter clockwise, while a negative *number* make changes clockwise.

2.20

```
\begin{tikzpicture}
  \Vertex{A} \Vertex[x=2]{B}
  \Edge[bend=45](A)(B)
  \Edge[bend=-70](A)(B)
\end{tikzpicture}
```

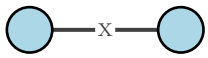


`\Edge[`⟨*label*⟩`=`*string*`](`*Vertex i*`)(`*Vertex j*`)`

An edge is labeled with the option ⟨*label*⟩. For the label any *string* argument can be used, including blank spaces. The environment $ $ can be used to display mathematical expressions.

2.21

```
\begin{tikzpicture}
  \Vertex{A} \Vertex[x=2]{B}
  \Edge[label=X](A)(B)
\end{tikzpicture}
```

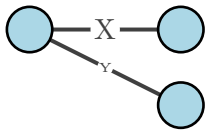


`\Edge[`⟨*label*⟩`=`*string*`,`⟨*fontsize*⟩`=`*string*`](`*Vertex i*`)(`*Vertex j*`)`

The font size of the ⟨*label*⟩ can be modified with the option ⟨*fontsize*⟩. Here common LaTeX font size commands[8] can be used to change the size of the label.

[8] e.g. `\tiny`, `\scriptsize`, `\footnotesize`, `\small`, ….

2.22
```
\begin{tikzpicture}
  \Vertex{A} \Vertex[x=2]{B} \Vertex[x=2,y=-1]{C}
  \Edge[label=X,fontsize=\large](A)(B)
  \Edge[label=Y,fontsize=\tiny](A)(C)
\end{tikzpicture}
```

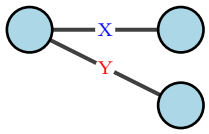


`\Edge[`⟨*label*⟩`=`*string*`,`⟨*fontcolor*⟩`=`*color*`](`*Vertex i*`)(`*Vertex j*`)`

The color of the ⟨*label*⟩ can be changed with the option ⟨*fontcolor*⟩. Currently, only the default TikZ and LaTeX colors are supported [9].

[9] **TODO!** Add RGB option!

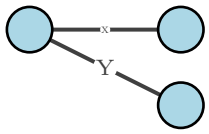
2.23
```
\begin{tikzpicture}
  \Vertex{A} \Vertex[x=2]{B} \Vertex[x=2,y=-1]{C}
  \Edge[label=X,fontcolor=blue](A)(B)
  \Edge[label=Y,fontcolor=red](A)(C)
\end{tikzpicture}
```



`\Edge[`⟨*label*⟩`=`*string*`,`⟨*fontscale*⟩`=`*color*`](`*Vertex i*`)(`*Vertex j*`)`

Contrary to the option ⟨*fontsize*⟩, the option ⟨*fontscale*⟩ does not change the font size itself but scales the curent font size up or down. The *number* defines the scale, where numbers between 0 and 1 down scale the font and numbers greater then 1 up scale the label. For example 0.5 reduces the size of the font to 50% of its originial size, while 1.2 scales the font to 120%.

2.24
```
\begin{tikzpicture}
  \Vertex{A} \Vertex[x=2]{B} \Vertex[x=2,y=-1]{C}
  \Edge[label=X,fontscale=.5](A)(B)
  \Edge[label=Y,fontscale=2](A)(C)
\end{tikzpicture}
```

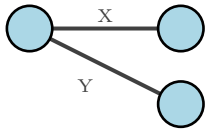


`\Edge[`⟨*label*⟩`=`*string*`,`⟨*position*⟩`=`*string*`](`*Vertex i*`)(`*Vertex j*`)`

Per default the ⟨*label*⟩ is positioned in between both vertices in the center of the line. Classical TikZ commands[10] can be used to change the ⟨*position*⟩ of the ⟨*label*⟩.

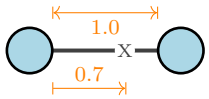
[10] e.g. *above*, *below*, *left*, *right*, *above left*, *above right*,…

2.25
```
\begin{tikzpicture}
  \Vertex{A} \Vertex[x=2]{B} \Vertex[x=2,y=-1]{C}
  \Edge[label=X,position=above](A)(B)
  \Edge[label=Y,position={below left=2mm}](A)(C)
\end{tikzpicture}
```



`\Edge[`⟨*label*⟩`=`*string*`,`⟨*distance*⟩`=`*number*`](`*Vertex i*`)(`*Vertex j*`)`

The label position between the vertices can be modified with the ⟨*distance*⟩ option. Per default the ⟨*label*⟩ is centered between both vertices. The position is expressed as the percentage of the length between the vertices, e.g. of ⟨*distance*⟩=0.7, the label is placed at 70% of the edge length away of *Vertex i*.

2.26
```
\begin{tikzpicture}
  \Vertex{A} \Vertex[x=2]{B}
  \Edge[label=X,distance=.7](A)(B)
\end{tikzpicture}
```


`\Edge[`⟨*style*⟩`=`*string*`](`*Vertex i*`)(`*Vertex j*`)`

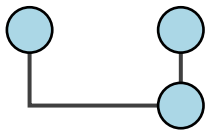
Any other Ti*k*Z style option or command can be entered via the option ⟨*style*⟩. Most of these commands can be found in the "Ti*k*Z and PGF Manual". Contain the commands additional options (e.g.⟨*shading*⟩=*ball*), then the argument for the ⟨*style*⟩ has to be between { } brackets.

2.27
```
\begin{tikzpicture}
  \Vertex{A} \Vertex[x=2]{B}
  \Edge[style={dashed}](A)(B)
\end{tikzpicture}
```


`\Edge[`⟨*path*⟩`=`*list*`](`*Vertex i*`)(`*Vertex j*`)`

In order to draw a finite sequence of edges which connect a sequence of vertices and/or coordinates, the option ⟨*path*⟩ can be used[11]. The argument for this option has to be a list element indicated by { } brackets, containing the *Names* of the intermediated vertices. New coordinates, i.e. there is no vertex located, can be insert with {⟨*x*⟩,⟨*y*⟩}. Arguments of the list, have to be seperated by commas «,».

[11] **TODO!** currently labels and bend edges are not supported!

2.28
```
\begin{tikzpicture}
  \Vertex{A} \Vertex[x=2]{B} \Vertex[x=2,y=-1]{C}
  \Edge[path={A,{0,-1},C,B}](A)(B)
\end{tikzpicture}
```


`\Edge(`*Vertex i*`)(`*Vertex i*`)`

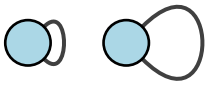
Self-loops are created by using the same vertex as origin and destination. Beside the options explained above, there are three self-loop specific options: ⟨*loopsize*⟩, ⟨*loopposition*⟩, and ⟨*loopshape*⟩.

2.29
```
\begin{tikzpicture}
  \Vertex{A}
  \Edge(A)(A)
\end{tikzpicture}
```


`\Edge[`⟨*loopsize*⟩`=`*measure*`](`*Vertex i*`)(`*Vertex i*`)`

With the option ⟨*loopsize*⟩ the length of the edge can be modified. The *measure* value has to be insert together with its units. Per default the ⟨*loopsize*⟩ is 1 cm.

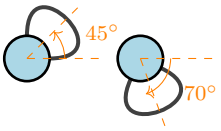
2.30
```
\begin{tikzpicture}
  \Vertex{A} \Vertex[x=1.3]{B}
  \Edge[loopsize=.5cm](A)(A)
  \Edge[loopsize=1.5cm](B)(B)
\end{tikzpicture}
```



`\Edge[`⟨*loopposition*⟩`=`*number*`](`*Vertex i*`)(`*Vertex i*`)`

The position of the self-loop is defined via the rotation angle around the vertex. The origin (0°) is the y axis. A positive *number* change the ⟨*loopposition*⟩ counter clockwise, while a negative *number* make changes clockwise.

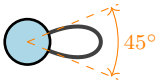
2.31
```
\begin{tikzpicture}
  \Vertex{A} \Vertex[x=1.5]{B}
  \Edge[loopposition=45](A)(A)
  \Edge[loopposition=-70](B)(B)
\end{tikzpicture}
```



`\Edge[`⟨*loopshape*⟩`=`*number*`](`*Vertex i*`)(`*Vertex i*`)`

The shape of the self-loop is defined by the enclosing angle. The shape can be changed by decreasing or increasing the argument value of the ⟨*loopshape*⟩ option.

2.32
```
\begin{tikzpicture}
  \Vertex{A}
  \Edge[loopshape=45](A)(A)
\end{tikzpicture}
```



`\Edge[`⟨*Direct*⟩`](`*Vertex i*`)(`*Vertex j*`)`

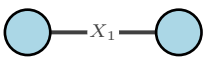
Directed edges are created by enabling the option ⟨*Direct*⟩. The arrow is drawn from *Vertex i* to *Vertex j*.

2.33
```
\begin{tikzpicture}
  \Vertex{A} \Vertex[x=2]{B}
  \Edge[Direct](A)(B)
\end{tikzpicture}
```

`\Edge[Math, label=`⟨*string*⟩`](`*Vertex i*`)(`*Vertex j*`)`

The option ⟨*Math*⟩ allows transforming labels into mathematical expressions without using the $ $ environment.

2.34
```
\begin{tikzpicture}
  \Vertex{A} \Vertex[x=2]{B}
  \Edge[Math,label=X_1](A)(B)
\end{tikzpicture}
```



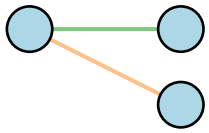
`\Edge[RBG, color=`⟨*RGB value*⟩`](`*Vertex i*`)(`*Vertex j*`)`

In order to display RGB colors for the line color of the edge, the option ⟨*RGB*⟩ has to be entered. In combination with this option, the ⟨*color*⟩ hast to be a list with the *RGB values*, separated by «,» and within { }.[12]

[12] e.g. the RGB code for white: {255, 255, 255}

2.35

```
\begin{tikzpicture}
  \Vertex{A} \Vertex[x=2]{B} \Vertex[x=2,y=-1]{C}
  \Edge[RGB,color={127,201,127}](A)(B)
  \Edge[RGB,color={253,192,134}](A)(C)
\end{tikzpicture}
```



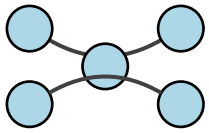
`\Edge[`⟨*NotInBG*⟩`]{`*filename*`}`

Per default, the edge is drawn on the background layer of the *tikzpicture*. I.e. objects which are created after the edges appear also on top of them. To turn this off, the option ⟨*NotInBG*⟩ has to be enabled. Changes to the default setting can be made with `\EdgesNotInBG` or `\EdgesInBG`[13].

[13] See Section 5.3

2.36

```
\begin{tikzpicture}
  \Vertex{A} \Vertex[x=2]{B} \Vertex[x=1,y=-.5]{C}
  \Vertex[y=-1]{D} \Vertex[x=2,y=-1]{E}
  \Edge[bend=-30](A)(B)
  \Edge[bend=30,NotInBG](D)(E)
\end{tikzpicture}
```



2.3 Text

While Ti*k*Z offers multiple ways to label objects and create text elements, a simplified command `\Text` is implemented, which allow placing and modifying text to the networks.

`\Text[`⟨*local options*⟩`]{`*string*`}`

In order to be able to create a text, a non-empty *string* argument is required. This argument is the actual text added to the figure. Mathematical symbols are entered in the same way as in a normal LaTeX document, i.e. between $ $.

For a `\Text` the following options are available:

Option	Default	Type	Definition
x	0	measure	x-coordinate
y	0	measure	y-coordinate
fontsize	{}	fontsize	font size of the text
color	{}	color	color of the text
opacity	{}	number	opacity of the text
position	center	string	position of the text to the origin
distance	0 cm	measure	distance from the origin
rotation	0	number	rotation of the text
anchor	{}	string	anchor of the text
width	{}	number	width of the text box
style	{}	string	additional Ti*k*Z styles
layer	{}	number	assigned layer of the text
RGB	false	Boolean	allow RGB colors

Table 2.3: Local options for the `\Text` command.

The order how the options are entered does not matter. Changes to the default Text layout can be made with `\SetTextStyle`[14]

[14] see Section 5.4

`\Text[`⟨*x*⟩`=`*measure*`,`⟨*y*⟩`=`*measure*`]{`*string*`}`

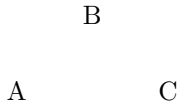
The location of the text is determined by Cartesian coordinates in ⟨*x*⟩ and ⟨*y*⟩. The coordinates are optional. If no coordinates are determined the text will be placed at the origin $(0, 0)$. The entered *measures* are in default units (cm). Changing the unites (locally) can be done by adding the unite to the *measure*[15]. Changes to the default setting can be made with `\SetDefaultUnit`[16].

[15] e.g. x=1 in

[16] see Section 5.1

2.37

```
\begin{tikzpicture}
  \Text{A}
  \Text[x=1,y=1]{B}
  \Text[x=2]{C}
\end{tikzpicture}
```



`\Text[`⟨*fontsize*⟩`=`*font size*`]{`*string*`}`

The font size of the text can be changed with the option ⟨*fontsize*⟩. Per default the font size of the text is defined as `\normalsize`.

2.38

```
\begin{tikzpicture}
  \Text[fontsize=\small]{A}
  \Text[x=1,fontsize=\LARGE]{B}
  \Text[x=2,fontsize=\Huge]{C}
\end{tikzpicture}
```

`\Text[`⟨*color*⟩`=`*color*`]{`*string*`}`

To change the text color individually, the option ⟨*color*⟩ has to be used. Without the option ⟨*RGB*⟩ set, the default Ti*k*Z and LaTeX colors can be applied.

2.39
```
\begin{tikzpicture}
  \Text[color = blue]{A}
  \Text[x=1,color=red]{B}
  \Text[x=2,color=green!70!blue]{C}
\end{tikzpicture}
```

`\Text[`⟨*opacity*⟩`=`*number*`]{`*string*`}`

With the option ⟨*opacity*⟩ the opacity of the text can be modified. The range of the *number* lies between 0 and 1. Where 0 represents a fully transparent text and 1 a solid text.

2.40
```
\begin{tikzpicture}
  \Text[opacity = 1]{A}
  \Text[x=1,opacity =.7]{B}
  \Text[x=2,opacity =.2]{C}
\end{tikzpicture}
```

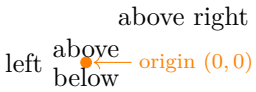
`\Text[`⟨*position*⟩`=`*string*`,`⟨*distance*⟩`=`*measure*`]{`*string*`}`

Per default the ⟨*position*⟩ of the text is in the *center* of the origin. Classical Ti*k*Z commands[17] can be used to change the ⟨*position*⟩ of the text.

With the option, ⟨*distance*⟩ the distance between the text and the origin can be changed.

[17] e.g. *above*, *below*, *left*, *right*, *above left*, *above right*,...

2.41
```
\begin{tikzpicture}
  \Text[position=above]{above}
  \Text[position=below]{below}
  \Text[position=left,distance=5mm]{left}
  \Text[position=above right,distance=5mm]{above right
      }
\end{tikzpicture}
```



`\Text[`⟨*rotation*⟩`=`*number*`]{`*string*`}`

With the ⟨*rotation*⟩, the text can be rotated by entering a *number* between −360 and 360. The origin (0°) is the *y* axis. A positive *number* change the ⟨*position*⟩ counter clockwise, while a negative *number* make changes clockwise.

2.42
```
\begin{tikzpicture}
  \Text[rotation=30]{A}
  \Text[x=1,rotation=45]{B}
  \Text[x=2,rotation=75]{C}
\end{tikzpicture}
```

`\Text[`⟨*anchor*⟩`=`*string*`]{`*string*`}`

With the option ⟨*anchor*⟩ the alignment of the text can be changed. Per default the text will be aligned centered. Classical Ti*k*Z commands[18] can be used to change the alignment of the text.

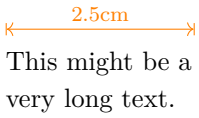
[18] e.g. *north*, *east*, *south*, *west*, *north east*, *north west*,...

2.43
```
\begin{tikzpicture}
  \Text[anchor=north east]{NE}
  \Text[x=1,anchor = south]{S}
  \Text[x=2,anchor =south west]{SW}
\end{tikzpicture}
```

`\Text[`⟨*width*⟩`=`*measure*`]{`*string*`}`

With the option ⟨*width*⟩ enabled, the text will break after the entered *measure*.

2.44
```
\begin{tikzpicture}
  \Text[width=2.5cm]{This might be a very long text.}
\end{tikzpicture}
```



`\Text[`⟨*style*⟩`={`*string*`}]{`*string*`}`

Any other Ti*k*Z style option or command can be entered via the option ⟨*style*⟩. Most of these commands can be found in the "Ti*k*Z and PGF Manual". Contain the commands additional options (e.g.⟨*fill*⟩=*red*), then the argument for the ⟨*style*⟩ has to be between { } brackets.

2.45
```
\begin{tikzpicture}
  \Text[style={draw,rectangle}]{A}
  \Text[x=1,style={fill=red}]{B}
  \Text[x=2,style={fill=blue,circle,opacity=.3}]{C}
\end{tikzpicture}
```

`\Text[`⟨*RGB*⟩`,`⟨*color*⟩`=`*RGB values*`]{`*string*`}`

In order to display RGB colors for the text color, the option ⟨*RGB*⟩ has to be entered. In combination with this option, the ⟨*color*⟩ hast to be a list with the *RGB values*, separated by «,» and within { }.[19]

[19] e.g. the RGB code for white: {255, 255, 255}

2.46
```
\begin{tikzpicture}
  \Text[RGB,color={127,201,127}]{A}
  \Text[x=1,RGB,color={190,174,212}]{B}
  \Text[x=2,RGB,color={253,192,134}]{C}
\end{tikzpicture}
```

`\layer[`⟨*layer*⟩`=`*number*`]{`*string*`}`

With the option ⟨*layer*⟩ the text can be assigned to a specific layer. More about this option and the use of layers is explained in Chapter 4.

3 Complex Networks

While in Chapter 2 the building blocks of the networks are introduced, here the main strength of the `tikz-network` package is explained. This includes creating networks based on data, obtained from other sources (e.g. Python, R, GIS). The idea is that the layout will be done by this external sources and `tikz-network` is used make some changes and to recreate the networks in LaTeX.

3.1 Vertices

The `\Vertices` command is the extension of the `\Vertex` command. Instead of a single vertex, a set of vertices will be drawn. This set of vertices is defined in an external file but can be modified with `\Vertices`.

`\Vertices[`⟨*global options*⟩`]{`*filename*`}`

The vertices have to be stored in a clear text file[1], preferentially in a `.csv` format. The first row should contain the headings, which are equal to the options defined in Table 2.1. Option are separated by a comma «,». Each new row is corresponds to a new vertex.

[1] e.g. .txt, .tex, .csv, .dat, ...

Example 3.1

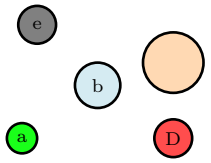
File: vertices.csv

```
id, x, y ,size,color ,opacity,label,IdAsLabel,NoLabel
 A, 0,  0, .4 ,green ,  .9    ,  a  ,  false   , false
 B, 1, .7, .6 ,      ,  .5    ,  b  ,  false   , false
 C, 2,  1, .8 ,orange,  .3    ,  c  ,  false   , true
 D, 2,  0, .5 ,red   ,  .7    ,  d  ,  true    , false
 E,.2,1.5, .5 ,gray  ,        ,  e  ,  false   , false
```

Only the ⟨*id*⟩ value is mandatory for a vertex and corresponds to the *Name* argument of a single `\Vertex`. Therefore, the same rules and naming conventions apply as for the *Name* argument: no mathematical expressions, no blank spaces, and the ⟨*id*⟩ must be unique! All other options are optional. No specific order of the options must be maintained. If no value is entered for an option, the default value will be chosen[2]. The *filename* should not contain blank spaces or special characters. The vertices are drawn by the command `\Vertex` with the *filename* plus file format (e.g. `.csv`). If the vertices file is not in the same directory as the main LaTeX file, also the path has to be specified.

[2] **TODO!** This is NOT valid for Boolean options, here values for all vertices have to be entered.

3.2
```
\begin{tikzpicture}
  \Vertices{data/vertices.csv}
\end{tikzpicture}
```



Predefined `\Vertex` options can be overruled by the ⟨*global options*⟩ of the `\Vertices` command; I.e. these options apply for all vertices in the file. For the `\Vertices` the following options are available:

Option	Default	Type	Definition
size	{}	measure	diameter of the circles
color	{}	color	fillcolor of vertices
opacity	{}	number	opacity of the fill color
style	{}	string	additional Ti*k*Z styles
layer	{}	number	assigned layer of the vertices
NoLabel	false	Boolean	delete the labels
IdAsLabel	false	Boolean	uses the *Names* as labels
Math	false	Boolean	displays the labels in math mode
RGB	false	Boolean	allow RGB colors
Pseudo	false	Boolean	create a pseudo vertices

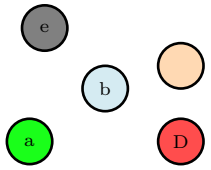
Table 3.1: Global options for the `\Vertices` command.

The use of these options are similar to the options for a single `\Vertex` defined in Section 2.1.

`\Vertices[`⟨*size*⟩`=`*measure*`]{`*filename*`}`

The diameter of the vertices can be changed with the option ⟨*size*⟩. Per default a vertex has 0.6 cm in diameter. Also, here the default units are cm and have not to be added to the *measure*.

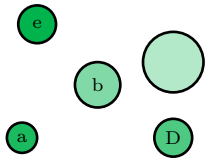
3.3
```
\begin{tikzpicture}
  \Vertices[size=.6]{data/vertices.csv}
\end{tikzpicture}
```



`\Vertices[`⟨*color*⟩`=`*color*`]{`*filename*`}`

To change the fill color for all vertices, the option ⟨*color*⟩ has to be used. Without the option ⟨*RGB*⟩ set, the default Ti*k*Z and LaTeX colors can be applied.

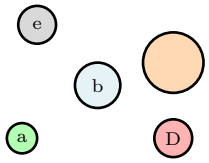
3.4
```
\begin{tikzpicture}
  \Vertices[color=green!70!blue]{data/vertices.csv}
\end{tikzpicture}
```



`\Vertices[`⟨*opacity*⟩`=`*number*`]{`*filename*`}`

With the option ⟨*opacity*⟩ the opacity of all vertices fills colors can be modified. The range of the *number* lies between 0 and 1. Where 0 represents a fully transparent fill and 1 a solid fill.

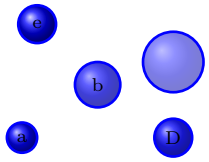
3.5
```
\begin{tikzpicture}
  \Vertices[opacity=.3]{data/vertices.csv}
\end{tikzpicture}
```



`\Vertices[`⟨*style*⟩`=`*string*`]{`*filename*`}`

Any other TikZ style option or command can be entered via the option ⟨*style*⟩. Most of these commands can be found in the "TikZ and PGF Manual". Contain the commands additional options (e.g.⟨*shading*⟩=*ball*), then the argument for the ⟨*style*⟩ has to be between { } brackets.

3.6
```
\begin{tikzpicture}
  \Vertices[style={shading=ball,blue}]{data/vertices.
      csv}
\end{tikzpicture}
```

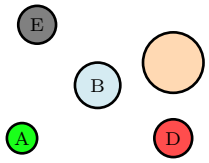


`\Vertices[`⟨*IdAsLabel*⟩`]{`*filename*`}`

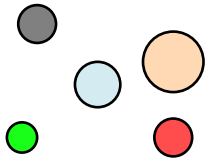
`\Vertices[`⟨*NoLabel*⟩`]{`*filename*`}`

⟨*IdAsLabel*⟩ is a Boolean option which assigns the ⟨*id*⟩ of the single vertices as labels. On the contrary, ⟨*NoLabel*⟩ suppress all labels.

3.7
```
\begin{tikzpicture}
  \Vertices[IdAsLabel]{data/vertices.csv}
\end{tikzpicture}
```



3.8
```
\begin{tikzpicture}
  \Vertices[NoLabel]{data/vertices.csv}
\end{tikzpicture}
```



`\Vertices[`⟨*RGB*⟩`]{`*filename*`}`

In order to display RGB colors for the vertex fill colors, the option ⟨*RGB*⟩ has to be entered. Additionally, the RGB values have to be specified in the file where the vertices are stored. Each value has its own column with the caption ⟨*R*⟩, ⟨*G*⟩, and ⟨*B*⟩.

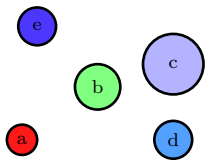
3.9
```
id, x, y ,size, color,opacity,label, R , G , B
 A, 0,  0, .4 , green,  .9   ,  a  ,255,  0,  0
 B, 1, .7, .6 ,      ,  .5   ,  b  ,  0,255,  0
 C, 2,  1, .8 ,orange,  .3   ,  c  ,  0,  0,255
 D, 2,  0, .5 ,   red,  .7   ,  d  , 10,120,255
 E,.2,1.5, .5 ,  gray,       ,  e  , 76, 55,255
```

File: vertices_RGB.csv

The "normal" color definition can also be part of the vertex definition. If the option ⟨*RGB*⟩ is not set, then the colors under ⟨*color*⟩ are applied.

3.10

```
\begin{tikzpicture}
  \Vertices[RGB]{data/vertices_RGB.csv}
\end{tikzpicture}
```



`\Vertices[`⟨*Pseudo*⟩`]{`*filename*`}`

The option ⟨*Pseudo*⟩ creates a pseudo vertices, where only the names and the coordinates of the vertices wil be drawn. Edges etc, can still be assigned to these vertices.

`\Vertices[`⟨*layer*⟩`=`*number*`]{`*filename*`}`

With the option ⟨*layer*⟩, only the vertices on the selected layer are plotted. More about this option and the use of layers is explained in Chapter 4.

3.2 Edges

The \Edges command is the extension of the \Edge command. Instead of a single edge, a set of edges will be drawn. This set of edges is defined in an external file but can be modified with \Edges.

\Edges[⟨*global options*⟩]{*filename*}

Like the vertices, the edges have to be stored in a clear text file[3], preferentially in a .csv format. The first row should contain the headings, which are equal to the options defined in Table 2.2. Option are separated by a comma «,». Each new row is corresponds to a new edge.

[3] e.g. .txt, .tex, .csv, .dat, ...

3.11

File: edges.csv

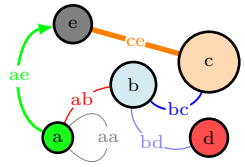
```
u,v,label,lw,color ,opacity,bend, R , G , B ,Direct
A,B, ab  ,.5,red   ,   1   ,  30,  0,120,255,false
B,C, bc  ,.7,blue  ,   1   , -60, 76, 55,255,false
B,D, bd  ,.5,blue  ,  .5   , -60, 76, 55,255,false
A,E, ae  , 1,green ,   1   ,  75,255,  0,  0,true
C,E, ce  , 2,orange,   1   ,   0,150,150,150,false
A,A, aa  ,.3,black ,  .5   ,  75,255,  0  ,0,false
```

The mandatory values are the ⟨*u*⟩ and ⟨*v*⟩ argument, which corresponds to the *Vertex i* and *Vertex j* arguments of a single \Edge. Edges can only create if a vertex exists with the same *Name*. All other options are optional. No specific order of the options must be maintained. If no value is entered for an option, the default value will be chosen[4]. The *filename* should not contain blank spaces or special characters. The edges are drawn by the command \Edges with the *filename* plus file format (e.g. .csv). If the edges file is not in the same directory as the main LaTeX file, also the path has to be specified. In order to draw edges, first, the vertices have to be generated. Only then, edges can be assigned.

[4] **TODO!** This is NOT valid for Boolean options, here values for all vertices have to be entered.

3.12

```
\begin{tikzpicture}
  \Vertices{data/vertices.csv}
  \Edges{data/edges.csv}
\end{tikzpicture}
```



Predefined \Edge options can be overruled by the ⟨*global options*⟩ of the \Edges command; I.e. these options apply for all edges in the file. For the \Edges the following options are available:

Option	Default	Type	Definition
lw	{}	measure	line width of the edge
color	{}	color	edge color
opacity	{}	number	opacity of the edge
style	{}	string	additional Ti*k*Z styles
vertices	{}	file	vertices were the edges are assigned to
layer	{}	number	edges in specific layers
Direct	false	Boolean	allow directed edges
Math	false	Boolean	displays the labels in math mode
NoLabel	false	Boolean	delete the labels
RGB	false	Boolean	allow RGB colors
NotInBG	false	Boolean	edges are not in the background layer

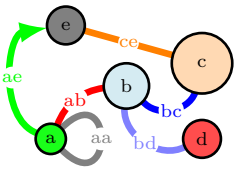
Table 3.2: Global options for the `\Edges` command.

The use of these options are similar to the options for a single `\Edge` defined in Section 2.2.

`\Edges[`⟨*lw*⟩`=`*measure*`]{`*filename*`}`

The line width of the edges can be modified with the option ⟨*lw*⟩. Here, the unit of the *measure* can be specified, otherwise, it is in pt.

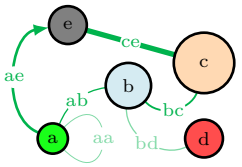
3.13
```
\begin{tikzpicture}
  \Vertices{data/vertices.csv}
  \Edges[lw=2.5]{data/edges.csv}
\end{tikzpicture}
```



`\Edges[`⟨*color*⟩`=`*color*`]{`*filename*`}`

To change the line color of all edges, the option ⟨*color*⟩ has to be used. Without the option ⟨*RGB*⟩ set, the default Ti*k*Z and LaTeX colors can be applied.

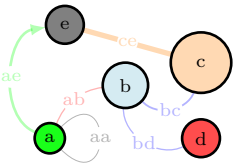
3.14
```
\begin{tikzpicture}
  \Vertices{data/vertices.csv}
  \Edges[color=green!70!blue]{data/edges.csv}
\end{tikzpicture}
```



`\Edges[`⟨*opacity*⟩`=`*number*`]{`*filename*`}`

With the option ⟨*opacity*⟩ the opacity of all edge lines can be modified. The range of the *number* lies between 0 and 1. Where 0 represents a fully transparent fill and 1 a solid fill.

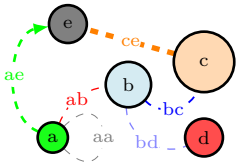
3.15
```
\begin{tikzpicture}
  \Vertices{data/vertices.csv}
  \Edges[opacity=0.3]{data/edges.csv}
\end{tikzpicture}
```



`\Edges[`⟨*style*⟩`=`*string*`]{`*filename*`}`

Any other TikZ style option or command can be entered via the option ⟨*style*⟩. Most of these commands can be found in the "TikZ and PGF Manual".

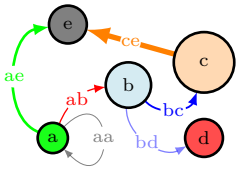
3.16
```
\begin{tikzpicture}
  \Vertices{data/vertices.csv}
  \Edges[style={dashed}]{data/edges.csv}
\end{tikzpicture}
```



`\Edges[`⟨*Direct*⟩`]{`*filename*`}`

Directed edges are created by enabling the option ⟨*Direct*⟩. The arrow is drawn from ⟨*u*⟩ to ⟨*v*⟩.

3.17
```
\begin{tikzpicture}
  \Vertices{data/vertices.csv}
  \Edges[Direct]{data/edges.csv}
\end{tikzpicture}
```



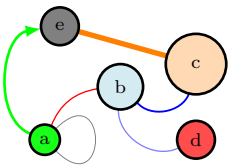
`\Edges[Math]{`*filename*`}`

The option ⟨*Math*⟩ allows transforming labels into mathematical expressions without using the $ $ environment.

`\Edges[`⟨*NoLabel*⟩`]{`*filename*`}`

The option ⟨*NoLabel*⟩ suppress all edge labels.

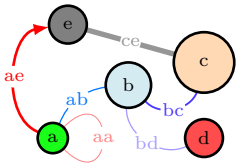
3.18
```
\begin{tikzpicture}
  \Vertices{data/vertices.csv}
  \Edges[NoLabel]{data/edges.csv}
\end{tikzpicture}
```



`\Edges[`⟨*RGB*⟩`]{`*filename*`}`

In order to display RGB colors for the edge line colors, the option ⟨*RGB*⟩ has to be entered. Additionally, the RGB values have to be specified in the file where the vertices are stored. Each value has its own column with the caption ⟨*R*⟩, ⟨*G*⟩, and ⟨*B*⟩. The "normal" color definition can also be part of the vertex definition. If the option ⟨*RGB*⟩ is not set, then the colors under ⟨*color*⟩ are applied.

3.19
```
\begin{tikzpicture}
  \Vertices{data/vertices.csv}
  \Edges[RGB]{data/edges.csv}
\end{tikzpicture}
```



`\Edges[`⟨*NotInBG*⟩`]{`*filename*`}`

Per default, the edges are drawn on the background layer of the *tikzpicture*. I.e. objects which are created after the edges appear also on top of them. To turn this off, the option ⟨*NotInBG*⟩ has to be enabled.

`\Edges[`⟨*vertices*⟩`=`*filename*`]{`*filename*`}`

Edges can be assigned to a specific set of `\Vertices` with the option ⟨*vertices*⟩. Thereby the argument *filename* is the same as used for the `\Vertices` command. This option might be necessary if multiple `\Vertices` are created and edges are assigned at the end.

`\Edges[`⟨*layer*⟩`={`*layer* α`,`*layer* β`}]{`*filename*`}`

With the option ⟨*layer*⟩ only the edges between layer α and β are plotted. The argument is a tuple of both layers indicated by { , }. More about this option and the use of layers is explained in Chapter 4.

4 Multilayer Networks

One of the main purposes of the `tikz-network` package is the illustration of multilayer network structures. Thereby, all the previous commands can be used. A multilayer network is represented as a three-dimensional object, where each layer is located at a different z plane. In order to enable this functionality, the option ⟨*multilayer*⟩ has to be used at the beginning of the `tikzpicture`.

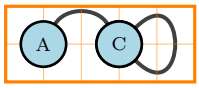
4.1 Simple Networks

`\Vertex[`⟨*layer*⟩`=`*number*`]{`*Name*`}`

With the option ⟨*layer*⟩ the vertex can be assigned to a specific layer. Layers are defined by numbers (e.g. 1, 2, 3,…). Working with the ⟨*multilayer*⟩ option, each `\Vertex` has to be assigned to a specific layer. For the edge assignment no additional information is needed.

4.1

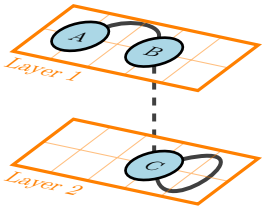
```
\begin{tikzpicture}[multilayer]
  \Vertex[x=0.5,IdAsLabel,layer=1]{A}
  \Vertex[x=1.5,IdAsLabel,layer=1]{B}
  \Vertex[x=1.5,IdAsLabel,layer=2]{C}
  \Edge[bend=60](A)(B)
  \Edge[style=dashed](B)(C)
  \Edge(C)(C)
\end{tikzpicture}
```



Enabling the option ⟨*multilayer*⟩, returns the network in a two-dimensional plane, like the networks discussed before. Setting the argument ⟨*multilayer*⟩=*3d*, the network is rendered in a three-dimensional representation. Per default, the layer with the lowest number is on the top. This and the spacing between the layers can be changed with the command `\SetLayerDistance`.

4.2

```
\begin{tikzpicture}[multilayer=3d]
  \Vertex[x=0.5,IdAsLabel,layer=1]{A}
  \Vertex[x=1.5,IdAsLabel,layer=1]{B}
  \Vertex[x=1.5,IdAsLabel,layer=2]{C}
  \Edge[bend=60](A)(B)
  \Edge[style=dashed](B)(C)
  \Edge(C)(C)
\end{tikzpicture}
```



4.2 Complex Networks

Similar as in Chapter 3 introduced, layers can be assigned to the vertices by adding a column ⟨*layer*⟩ to the file where the vertices are stored.

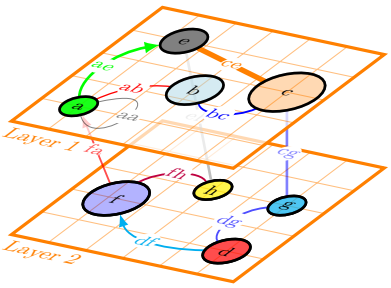
4.3 File: ml_vertices.csv

```
id, x, y ,size, color,opacity,label,layer
 A, 0,  0, .4 , green,   .9   ,  a  ,  1
 B, 1, .7, .6 ,      ,   .5   ,  b  ,  1
 C, 2,  1, .8 ,orange,   .3   ,  c  ,  1
 D, 2,  0, .5 ,   red,   .7   ,  d  ,  2
 E,.2,1.5, .5 ,  gray,        ,  e  ,  1
 F,.1, .5, .7 ,  blue,   .3   ,  f  ,  2
 G, 2,  1, .4 ,  cyan,   .7   ,  g  ,  2
 H, 1,  1, .4 ,yellow,   .7   ,  h  ,  2
```

4.4 File: ml_edges.csv

```
u,v,label,lw,color  ,opacity,bend,Direct
A,B, ab  ,.5,red    ,   1    ,  30,false
B,C, bc  ,.7,blue   ,   1    , -60,false
A,E, ae  , 1,green  ,   1    ,  45,true
C,E, ce  , 2,orange ,   1    ,   0,false
A,A, aa  ,.3,black  ,  .5    ,  75,false
C,G, cg  , 1,blue   ,  .5    ,   0,false
E,H, eh  , 1,gray   ,  .5    ,   0,false
F,A, fa  ,.7,red    ,  .7    ,   0,true
D,F, df  ,.7,cyan   ,   1    ,  30,true
F,H, fh  ,.7,purple ,   1    ,  60,false
D,G, dg  ,.7,blue   ,  .7    ,  60,false
```

With the commands `\Vertices` and `\Edges`, the network can be created automatically. Again the `\Vertices` vertices should be performed first and then the command `\Edges`.



4.5

```
\begin{tikzpicture}[multilayer=3d]
  \Vertices{data/ml_vertices.csv}
  \Edges{data/ml_edges.csv}
\end{tikzpicture}
```

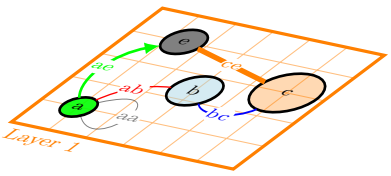
`\Vertices[`⟨*layer*⟩`=`*number*`]{`*filename*`}`

`\Edges[`⟨*layer*⟩`={`*layer* α`,`*layer* β`}]{`*filename*`}`

With the `\Vertices` option ⟨*layer*⟩ only the vertices on the selected layer are plotted. While, with the `\Edges` option ⟨*layer*⟩, the edges between layer α and β are plotted. The argument is a tuple of both layers indicated by { , }.

4.6

```
\begin{tikzpicture}[multilayer=3d]
  \Vertices[layer=1]{data/ml_vertices.csv}
  \Edges[layer={1,1}]{data/ml_edges.csv}
\end{tikzpicture}
```



Plotting edges without defining first the vertices can be done with the `\Edges` option ⟨*vertices*⟩. This allows modifying specific sets of Edges.

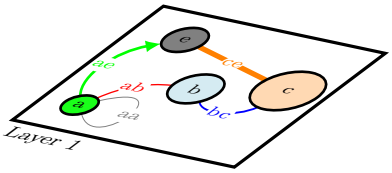
4.7
```
\begin{tikzpicture}[multilayer=3d]
  \Edges[vertices=data/ml_vertices.csv,
         layer={1,2},style=dashed]{data/ml_edges.csv}
\end{tikzpicture}
```

4.3 Layers and Layouts

Besides adding vertices and edges to specific layers, every other Ti*k*Z object can be drawn on such a layer using the `Layer` environment. With the option ⟨*layer*⟩=*layer* α, the position of the canvas can be assigned to the specific layer.

`\begin{Layer}[`⟨*layer*⟩=*layer* α`]`
`\end{Layer}`

4.8
```
\begin{tikzpicture}[multilayer=3d]
  \begin{Layer}[layer=1]
    \draw[very thick] (-.5,-.5) rectangle (2.5,2);
    \node at (-.5,-.5)[below right]{Layer 1};
  \end{Layer}
  \Vertices[layer=1]{data/ml_vertices.csv}
  \Edges[layer={1,1}]{data/ml_edges.csv}
\end{tikzpicture}
```



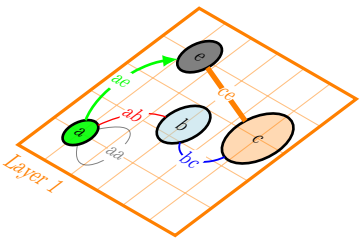
`\SetLayerDistance{`*measure*`}`

With the command `\SetLayerDistance` the distance between the layers and their orientation can be modified. Per default the distance is set to −2`\DefaultUnit` (here cm). A negative number implies that layers with a higher number will be stacked below layers with a smaller number.

`\SetCoordinates[`⟨*xAngle*⟩=*number*`,`⟨*yAngle*⟩=*number*`,`⟨*zAngle*⟩=*number*`,`
⟨*xLength*⟩=*number*`,`⟨*yLength*⟩=*number*`,`⟨*zLength*⟩=*number*`]`

The perspective of the three-dimensional plot can be modified by changing the orientation of the coordinate system, which is done with the command `\SetCoordinates`. Here the angle and the length of each axis can be modified. Angles are defined as a *number* in the range between −360 and 360. Per default, the lengths of the axes are defined by the identity matrix, i.e. no distortion. If the length ratio is changed x, y, and/or z values are distorted. The `\SetCoordinates` command has to be entered before the ⟨*multilayer*⟩ option is called!

4.9
```
\SetCoordinates[xAngle=-30,yLength=1.2,xLength=.8]
\begin{tikzpicture}[multilayer=3d]
  \Vertices[layer=1]{data/ml_vertices.csv}
  \Edges[layer={1,1}]{data/ml_edges.csv}
\end{tikzpicture}
```



4.4 Plane

To support the illustration of multilayer networks, the background of the layer can be simply visualized with the command `\Plane`, which allow to draw boundaries, grids and include images to the layer.

`\Plane[`⟨*options*⟩`]`

No obligatory arguments are needed. For a `\Plane` the following options are available:

Option	Default	Type	Definition
x	0	measure	x-coordinate of the origin
y	0	measure	y-coordinate of the origin
width	5 cm	measure	width of the plane
height	5 cm	measure	height of the plane
color	vertexfill	color	fill color of the plane
opacity	0.3	number	opacity of the fill color
grid	{}	measure	spacing of the grid
image	{}	file	path to the image file
style	{}	string	additional Ti*k*Z styles
layer	1	number	layer where the plane is located
RGB	false	Boolean	allow RGB colors
NoFill	false	Boolean	disable fill color
NoBorder	false	Boolean	disable border line
ImageAndFill	false	Boolean	allow image and fill color
InBG	false	Boolean	plane is in the background layer

[a] either measure or string

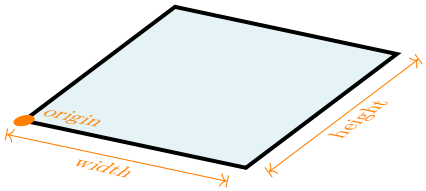
Table 4.1: Options for the `\Plane` command.

`\Plane[`⟨*x*⟩`=`*measure*`,`⟨*y*⟩`=`*measure*`,`⟨*width*⟩`=`*measure*`,`⟨*height*⟩`=`*measure*`]`

A `\Plane` is a rectangle with origin (⟨*x*⟩,⟨*y*⟩), a given ⟨*width*⟩ and ⟨*height*⟩. The origin is defined in the left lower corner and per default (0, 0). The plane is default 5 cm (width) by 5 cm (height). This default options can be changed with `\SetPlaneWidth` and `\SetPlaneHeight`[1]

[1] See Section 5.5.

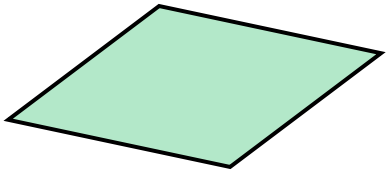
4.10
```
\begin{tikzpicture}[multilayer=3d]
  \Plane[x=-.5,y=-.5,width=3,height=2.5]
\end{tikzpicture}
```



`\Plane[`⟨*color*⟩`=`*color*`]`

To change the fill color of each plane individually, the option ⟨*color*⟩ has to be used. Without the option ⟨*RGB*⟩ set, the default Ti*k*Z and LaTeX colors can be applied. Per default the default vertex color is used.

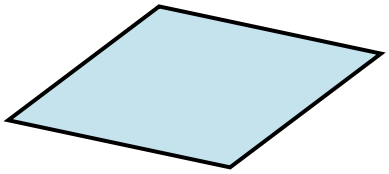
4.11
```
\begin{tikzpicture}[multilayer=3d]
  \Plane[x=-.5,y=-.5,width=3,height=2.5,color=green
      !70!blue]
\end{tikzpicture}
```



`\Plane[`⟨*opacity*⟩`=`*number*`]`

With the option ⟨*opacity*⟩ the opacity of the plane fill color can be modified. The range of the *number* lies between 0 and 1. Where 0 represents a fully transparent fill and 1 a solid fill. Per default the opacity is set to 0.3.

4.12
```
\begin{tikzpicture}[multilayer=3d]
  \Plane[x=-.5,y=-.5,width=3,height=2.5,opacity=.7]
\end{tikzpicture}
```



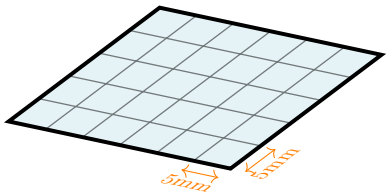
`\Plane[`⟨*grid*⟩`=`*measure*`]`

With the option ⟨*grid*⟩ a grid will be drawn on top of the plane. The argument of this option defines the spacing between the grid lines. The entered *measures* are in default units (cm). Changing the unites (locally) can be done by adding the unit to the *measure*[2]. Changes to the default setting can be made with `\SetDefaultUnit`[3].

[2] e.g. x=5 mm

[3] see Section 5.1

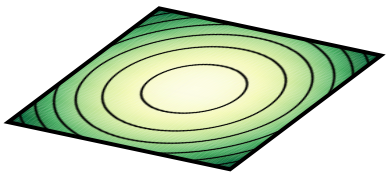
4.13
```
\begin{tikzpicture}[multilayer=3d]
  \Plane[x=-.5,y=-.5,width=3,height=2.5,grid=5mm]
\end{tikzpicture}
```



`\Plane[`⟨*image*⟩`=`*file*`]`

An image can be assigned to a plane with the option ⟨*image*⟩. The argument is the file name and the folder where the image is stored. The width and height of the figure is scaled to the size of the plane. Without the option ⟨*ImageAndFill*⟩ the image overwrite the color options.

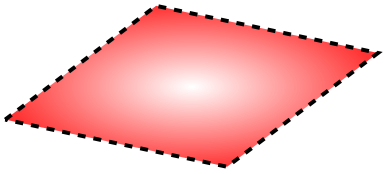
4.14
```
\begin{tikzpicture}[multilayer=3d]
  \Plane[x=-.5,y=-.5,width=3,height=2.5,image=data/
      plane.png]
\end{tikzpicture}
```



`\Plane[`⟨*style*⟩`=`*string*`]`

Any other Ti*k*Z style option or command can be entered via the option ⟨*style*⟩. Most of these commands can be found in the "Ti*k*Z and PGF Manual". Contain the commands additional options (e.g.⟨*inner color*⟩=*color*), then the argument for the ⟨*style*⟩ has to be between { } brackets.

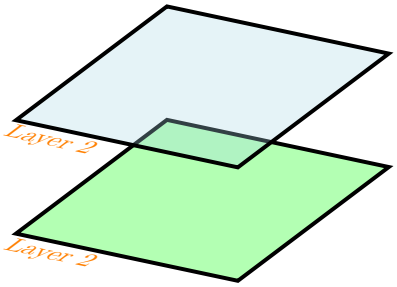
4.15
```
\begin{tikzpicture}[multilayer=3d]
  \Plane[x=-.5,y=-.5,width=3,height=2.5,style={dashed,
      inner color=white,outer color=red!80}]
\end{tikzpicture}
```



`\Plane[`⟨*layer*⟩`=`*number*`]`

With the option ⟨*layer*⟩=*layer* α, the position of the plane can be assigned to a specific layer. Per default the plane is drawn on layer 1.

4.16
```
\begin{tikzpicture}[multilayer=3d]
  \SetLayerDistance{-1.5}
  \Plane[x=-.5,y=-.5,width=3,height=2.5,color=green,
      layer=2]
  \Plane[x=-.5,y=-.5,width=3,height=2.5]
\end{tikzpicture}
```

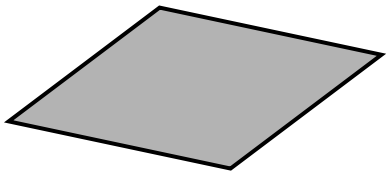


`\Plane[`⟨*RGB*⟩`,`⟨*color*⟩`=`*RGB values*`]`

In order to display RGB colors for the plane fill color, the option ⟨*RGB*⟩ has to be entered. In combination with this option, the ⟨*color*⟩ hast to be a list with the *RGB values*, separated by «,» and within { }.[4]

[4] e.g. the RGB code for white: {255, 255, 255}

4.17
```
\begin{tikzpicture}[multilayer=3d]
  \Plane[x=-.5,y=-.5,width=3,height=2.5,RGB,color
      ={0,0,0}]
\end{tikzpicture}
```

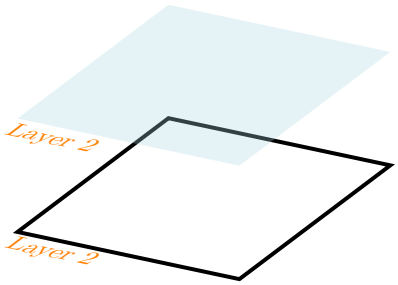


`\Plane[`⟨*NoFill*⟩`]`

`\Plane[`⟨*NoBorder*⟩`]`

⟨*NoFill*⟩ is a Boolean option which disables the fill color of the plane and ⟨*NoBorder*⟩ is a Boolean option which suppress the border line of the plane.

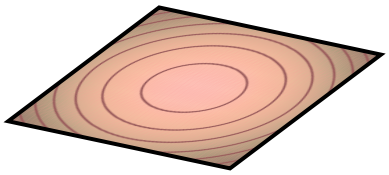
4.18
```
\begin{tikzpicture}[multilayer=3d]
  \SetLayerDistance{-1.5}
  \Plane[x=-.5,y=-.5,width=3,height=2.5,layer=2,NoFill
      ]
  \Plane[x=-.5,y=-.5,width=3,height=2.5,NoBorder]
\end{tikzpicture}
```



`\Plane[`⟨*ImageAndFill*⟩`]`

With the option ⟨*ImageAndFill*⟩ both, image and fill color can be drawn on a plane. The option ⟨*opacity*⟩ is applied to both objects.

4.19
```
\begin{tikzpicture}[multilayer=3d]
 \Plane[x=-.5,y=-.5,width=3,height=2.5,image=data/
     plane.png,color=red,opacity=.4,ImageAndFill]
\end{tikzpicture}
```



`\Plane[`⟨*InBG*⟩`]`

A plane is drawn on the current layer of the *tikzpicture*. I.e. objects which are created after the plane appear on top of it and objects created before below of it. With the option ⟨*InBG*⟩ enabled, the plane is drawn on the background layer of the *tikzpicture*.

5 Default Settings

In order to customize the look of the networks, each layout setting used can be modified and adapted. There are three categories: General settings, vertex style, and edge style.

5.1 General Settings

With the general settings mainly the sizes, distances and measures of the networks can be modified.

`\SetDefaultUnit{`*unit*`}`

The command `\SetDefaultUnit` allows to change the units used for drawing the network[1], including diameters of the vertices, x and y coordinates or the distance between the layers. The default unit is cm.

[1] Except the line width, which are defined in pt.

`\SetDistanceScale{`*number*`}`

With the command `\SetDistanceScale`, the distance between the vertices can be scaled. Per default 1 cm entered corresponds to 1 cm drawn, i.e. `\SetDistanceScale{1}`. Decreasing or increasing the scale changes the drawing distances between the vertices.

`\SetLayerDistance{`*measure*`}`

With the command `\SetLayerDistance` the distance between the layers and their orientation can be modified. Per default, the distance is set to -2. A negative number implies that layers with a higher number will be stacked below layers with a smaller number.

`\SetCoordinates[`⟨*xAngle*⟩`=`*number*`,`⟨*yAngle*⟩`=`*number*`,`⟨*zAngle*⟩`=`*number*`,`
⟨*xLength*⟩`=`*number*`,`⟨*yLength*⟩`=`*number*`,`⟨*zLength*⟩`=`*number*`]`

The perspective of the three-dimensional plot can be modified by changing the orientation of the coordinate system, which is done with the command `\SetCoordinates`. Here the angle and the length of each axis can be modified. Angles are defined as a *number* in the range between -360 and 360. Per default, the length of the axes are defined by the identity matrix, i.e. no distortion. If the length ratio is changed x, y, and/or z values are distorted. The `\SetCoordinates` command has to be entered before the ⟨*multilayer*⟩ option is called!

5.2 Vertex Style

The appearance of the vertices can be modified with the command `\SetVertexStyle`. This command will change the default settings of the vertices in the network.

`\SetVertexStyle[`*document options*`]`

The following options are available:

Option	Default	Type	Definition
Shape	circle	text	shape of the vertex
InnerSep	2pt	measure	separation space which will be added inside the shape
OuterSep	0pt	measure	separation space outside the background path
MinSize	0.6`\DefaultUnit`	measure	diameter (size) of the vertex
FillColor	vertexfill	color	color of the vertex
FillOpacity	1	number	opacity of the vertex
LineWidth	1pt	measure	line width of the vertex boundary
LineColor	black	color	line color of the vertex boundary
LineOpacity	1	number	line opacity of the vertex boundary
TextFont	`\scriptsize`	fontsize	font size of the vertex label
TextColor	black	color	color of the vertex label
TextOpacity	1	number	opacity of the vertex label
TextRotation	0	number	initial rotation of the vertex

Table 5.1: Document style options for the vertices.

5.3 Edge Style

The appearance of the edges can be modified with the command `\SetEdgeStyle`. This command will change the default settings of the edges in the network.

`\SetEdgeStyle[`*document options*`]`

The following options are available:

Option	Default	Type	Definition
LineWidth	1.5pt	measure	width of the edge
Color	black!75	color	color of the edge
Opacity	1	number	opacity of the edge
Arrow	-latex	text	arrow shape of the directed edge
TextFont	`\scriptsize`	fontsize	font size of the edge label
TextOpacity	1	number	opacity of the edge label
TextFillColor	white	color	fill color of the edge label
TextFillOpacity	1	number	fill opacity of the edge label
InnerSep	0pt	measure	separation space which will be added inside the shape
OuterSep	1pt	measure	separation space outside the background path
TextRotation	0	number	initial rotation of the edge label

Table 5.2: Document style options for the edges.

`\EdgesNotInBG`

`\EdgesInBG`

Per default edges are drawn on the background layer, with the command `\EdgesNotInBG` this can be disabled, while the command `\EdgesInBG` restores the default setting.

5.4 Text Style

The appearance of the text can be modified with the command `\SetTextStyle`. This command will change the default settings of the text.

`\SetTextStyle[`*document options*`]`

The following options are available:

Option	Default	Type	Definition
TextFont	`\normalsize`	fontsize	font size of the text
TextOpacity	1	number	opacity of the text
TextColor	black	color	color of the text
TextOpacity	1	number	opacity of the text
InnerSep	2pt	measure	separation space which will be added inside the shape
OuterSep	0pt	measure	separation space outside the background path
TextRotation	0	number	initial rotation of the text

Table 5.3: Document style options for the planes.

5.5 Plane Style

The appearance of the planes can be modified with the command `\SetPlaneStyle`. This command will change the default settings of the planes.

`\SetPlaneStyle[`*document options*`]`

The following options are available:

Option	Default	Type	Definition
LineWidth	1.5pt	measure	width of the border line
LineColor	black	color	color of the border line
LineOpacity	1	number	opacity of the border line
FillColor	vertexfill	color	fill color of the plane
FillOpacity	0.3	number	fill opacity of the plane
GridLineWidth	0.5pt	measure	width of the grid lines
GridColor	black	color	color of the grid lines
GridOpacity	0.5	number	opacity of the grid lines

Table 5.4: Document style options for the planes.

`\SetPlaneWidth{`*measure*`}`

`\SetPlaneHeight{`*measure*`}`

With the commands `\SetPlaneWidth` and `\SetPlaneHeight` the default size of the planes can be modified.

6 Troubleshooting and Support

6.1 `tikz-network` Website

The website for the `tikz-network` packages is located at https://github.com/hackl/tikz-network. There, you'll find the actual version of the source code, a bug tracker, and the documentation.

6.2 Getting Help

If you've encountered a problem with one of the `tikz-network` commands, have a question, or would like to report a bug, please send an email to me or visit our website.

To help me troubleshoot the problem more quickly, please try to compile your document using the `debug` class option and send the generated `.log` file to the mailing list with a brief description of the problem.

6.3 Errors, Warnings, and Informational Messages

The following is a list of all of the errors, warnings, and other messages generated by the `tikz-network` classes and a brief description of their meanings.

```
Error: ! TeX capacity exceeded, sorry [main memory size=5000000].
```

The considered network is to large and `pdflatex` runs out of memory. This problem can be solved by using `lualatex` or `xetex` instead.

6.4 Package Dependencies

The following is a list of packages that the `tikz-network` package rely upon. Packages marked with an asterisk are optional.

- etex
- xifthen
- xkeyval
- datatool
- tikz
  - arrows
  - positioning
  - 3d
  - fit
  - calc
  - backgrounds

A ToDo

A.1 Code to fix

- change default entries for Boolean options in the vertices file.

A.2 Documentation

- add indices to the manual.
- extended tutorial/example to the document.
- clean-up and document the .sty file.

A.3 Features

- add a spherical coordinate system

A.4 Add-ons

- add QGIS to tikz-network compiler

B Add-ons

B.1 Python networks to TikZ with network2tikz

B.1.1 Introduction

network2tikz is a Python tool for converting network visualizations into `tikz-network` figures, for native inclusion into your LaTeX documents.

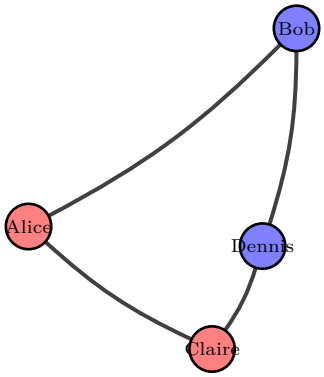
`network2tikz` works with Python 3 and supports (currently) the following Python network modules:

- cnet
- python-igraph
- networkx
- pathpy
- default node/edge lists

The output of `network2tikz` is a `tikz-network` figure. Because you are not only getting an image of your network, but also the LaTeX source file, you can easily post-process the figures (e.g. adding drawings, texts, equations,...).

Since *a picture is worth a thousand words* a small example:

```
#!/usr/bin/python -tt
# -*- coding: utf-8 -*-

nodes = ['a','b','c','d']
edges = [('a','b'), ('a','c'), ('c','d'),('d','b')]
gender = ['f', 'm', 'f', 'm']
colors = {'m': 'blue', 'f': 'red'}

style = {}
style['node_label'] = ['Alice', 'Bob', 'Claire', 'Dennis
    ']
style['node_color'] = [colors[g] for g in gender]
style['node_opacity'] = .5
style['edge_curved'] = .1

from network2tikz import plot
plot((nodes,edges),'network.tex',**style)
```



(see above) gives

```
\documentclass{standalone}
\usepackage{tikz-network}
\begin{document}
\begin{tikzpicture}
\clip (0,0) rectangle (6,6);
\Vertex[x=0.785,y=2.375,color=red,opacity=0.5,label=
    Alice]{a}
\Vertex[x=5.215,y=5.650,color=blue,opacity=0.5,label=Bob
    ]{b}
\Vertex[x=3.819,y=0.350,color=red,opacity=0.5,label=
    Claire]{c}
\Vertex[x=4.654,y=2.051,color=blue,opacity=0.5,label=
    Dennis]{d}
\Edge[,bend=-8.531](a)(c)
\Edge[,bend=-8.531](c)(d)
\Edge[,bend=-8.531](d)(b)
\Edge[,bend=-8.531](a)(b)
\end{tikzpicture}
\end{document}
```

Tweaking the plot is straightforward and can be done as part of your LaTeX workflow.

B.1.2 Installation

`network2tikz` is available from the Python Package Index, so simply type

```
pip install -U network2tikz
```

to install/update. If your are intersted in the development version of the module check out the github repository.

B.1.3 Usage

1. Generate, manipulation, and study of the structure, dynamics, and functions of your complex networks as usual, with your preferred python module.

2. Instead of the default plot functions (e.g. `igraph.plot()` or `networkx.draw()`) invoke `network2tikz` by

```
plot(G,'mytikz.tex')
```

to store your network visualisation as the TikZ file `mytikz.tex`. Load the module with:

```
from network2tikz import plot
```

Advanced usage:

Of course, you always can improve your plot by manipulating the generated LaTeX file, but why not do it directly in Python? To do so, all visualization options available in `tikz-network` are also implemented in `network2tikz`. The appearance of the plot can be modified by keyword arguments.[1]

[1] For a detailed explanation, please see Section B.1.5.

```
my_style = {}
plot(G,'mytikz.tex',**my_style)
```

The arguments follow the options described above in the manual.

Additionally, if you are more interested in the final output and not only the `.tex` file, used

```
plot(G,'mypdf.pdf')
```

to save your plot as a pdf, or

```
plot(G)
```

to create a temporal plot and directly show the result, i.e. similar to the matplotlib function `show()`. Finally, you can also create a node and edge list, which can be read and easily modified (in a post-processing step) as showd above.

```
plot(G,'mycsv.csv')
```

3. Compile the figure or add the contents of `mytikz.tex` into your LaTeX source code. With the option ⟨*standalone*⟩=*false* only the Ti*k*Z figure will be saved, which can then be easily included in your LaTeX document via `\input{/path/to/mytikz.tex}`.

B.1.4 Simple example

For illustration purpose, a similar network as in the python-igraph tutorial is used. If you are using another Python network module, and like to follow this example, please have a look at the provided examples.

Create network object and add some edges.

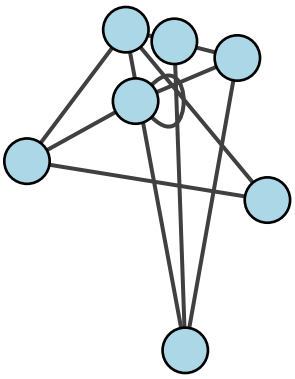
```
#!/usr/bin/python -tt
# -*- coding: utf-8 -*-

import igraph
from network2tikz import plot

net = igraph.Graph([(0,1), (0,2), (2,3), (3,4), (4,2),
    (2,5), (5,0), (6,3),
                    (5,6), (6,6)],directed=True)
```

Adding node and edge properties.

```
net.vs["name"] = ["Alice", "Bob", "Claire", "Dennis", "
    Esther", "Frank", "George"]
net.vs["age"] = [25, 31, 18, 47, 22, 23, 50]
net.vs["gender"] = ["f", "m", "f", "m", "f", "m", "m"]
net.es["is_formal"] = [False, False, True, True, True,
    False, True, False,
                        False, False]
```

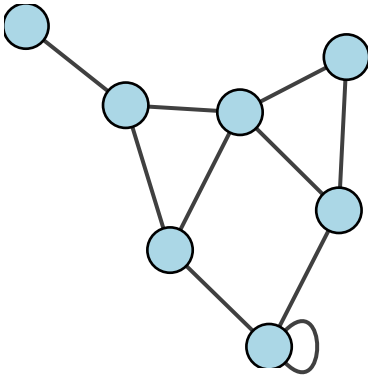
Already now the network can be plotted.

```
plot(net)
```



Per default, the node positions are assigned uniform random. In order to create a layout, the layout methods of the network packages can be used. Or the position of the nodes can be directly assigned, in form of a dictionary, where the key is the node id and the value is a tuple of the node position in x and y.

```
layout = {0: (4.3191, -3.5352), 1: (0.5292, -0.5292),
          2: (8.6559, -3.8008), 3: (12.4117, -7.5239),
          4: (12.7, -1.7069), 5: (6.0022, -9.0323),
          6: (9.7608, -12.7)}
plot(net,layout=layout)
```

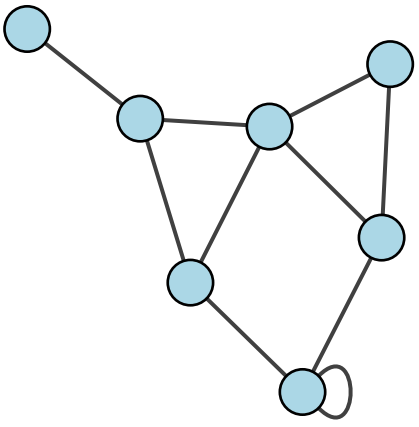


This should open an external pdf viewer showing a visual representation of the network, something like the one on the following figure:

We can simply re-using the previous layout object here, but we also specified that we need a bigger plot (8×8 cm) and a larger margin around the graph to fit the self loop and potential labels (1 cm).[2]

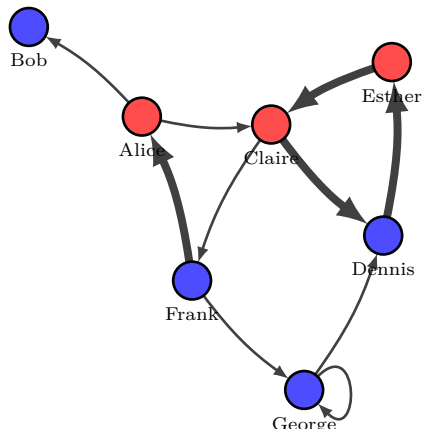
```
plot(net, layout=layout, canvas=(8,8), margin=1)
```

[2] Per default, all size values are based on cm, and all line widths are defined in pt units. With the general option ⟨*units*⟩ this can be changed, see Section B.1.5.



In to keep the properties of the visual representation of your network separate from the network itself. You can simply set up a Python dictionary containing the keyword arguments you would pass to `plot` and then use the double asterisk (`**`) operator to pass your specific styling attributes to `plot`:

```
color_dict = {'m': 'blue', 'f': 'red'}
visual_style = {}

# Node options
visual_style['vertex_size'] = .5
visual_style['vertex_color'] = [color_dict[g] for g in
    net.vs['gender']]
visual_style['vertex_opacity'] = .7
visual_style['vertex_label'] = net.vs['name']
visual_style['vertex_label_position'] = 'below'

# Edge options
visual_style['edge_width'] = [1 + 2 * int(f) for f in
    net.es('is_formal')]
visual_style['edge_curved'] = 0.1

# General options and plot command.
visual_style['layout'] = layout
visual_style['canvas'] = (8,8)
visual_style['margin'] = 1

# Plot command
plot(net,**visual_style)
```



Beside showing the network, we can also generate the latex source file, which can be used and modified later on. This is done by adding the output file name with the ending `'.tex'`.

```
plot(net,'network.tex',**visual_style)
```

produces

```
\documentclass{standalone}
\usepackage{tikz-network}
\begin{document}
\begin{tikzpicture}
\clip (0,0) rectangle (8.0,8.0);
\Vertex[x=2.868,y=5.518,size=0.5,color=red,opacity=0.7,label=Alice,position=below]{a}
\Vertex[x=1.000,y=7.000,size=0.5,color=blue,opacity=0.7,label=Bob,position=below]{b}
\Vertex[x=5.006,y=5.387,size=0.5,color=red,opacity=0.7,label=Claire,position=below]{c}
\Vertex[x=6.858,y=3.552,size=0.5,color=blue,opacity=0.7,label=Dennis,position=below]{d}
\Vertex[x=7.000,y=6.419,size=0.5,color=red,opacity=0.7,label=Esther,position=below]{e}
\Vertex[x=3.698,y=2.808,size=0.5,color=blue,opacity=0.7,label=Frank,position=below]{f}
\Vertex[x=5.551,y=1.000,size=0.5,color=blue,opacity=0.7,label=George,position=below]{g}
\Edge[,lw=1.0,bend=-8.531,Direct](a)(b)
\Edge[,lw=1.0,bend=-8.531,Direct](a)(c)
\Edge[,lw=3.0,bend=-8.531,Direct](c)(d)
\Edge[,lw=3.0,bend=-8.531,Direct](d)(e)
\Edge[,lw=3.0,bend=-8.531,Direct](e)(c)
\Edge[,lw=1.0,bend=-8.531,Direct](c)(f)
\Edge[,lw=3.0,bend=-8.531,Direct](f)(a)
\Edge[,lw=1.0,bend=-8.531,Direct](f)(g)
\Edge[,lw=1.0,bend=-8.531,Direct](g)(g)
\Edge[,lw=1.0,bend=-8.531,Direct](g)(d)
\end{tikzpicture}
\end{document}
```

Instead of the tex file, a node and edge list can be generates, which can also be used with the library.

```
plot(net,'network.csv',**visual_style)
```

The node list `network_nodes.csv`.

```
id,x,y,size,color,opacity,label,position
a,2.868,5.518,0.5,red,0.7,Alice,below
b,1.000,7.000,0.5,blue,0.7,Bob,below
c,5.006,5.387,0.5,red,0.7,Claire,below
d,6.858,3.552,0.5,blue,0.7,Dennis,below
e,7.000,6.419,0.5,red,0.7,Esther,below
f,3.698,2.808,0.5,blue,0.7,Frank,below
g,5.551,1.000,0.5,blue,0.7,George,below
```

The edge list `network_edges.csv`.

```
u,v,lw,bend,Direct
a,b,1.0,-8.531,true
a,c,1.0,-8.531,true
c,d,3.0,-8.531,true
d,e,3.0,-8.531,true
e,c,3.0,-8.531,true
c,f,1.0,-8.531,true
f,a,3.0,-8.531,true
f,g,1.0,-8.531,true
g,g,1.0,-8.531,true
g,d,1.0,-8.531,true
```

B.1.5 The plot function in detail

`network2tikz.plot(`*network*`,` ⟨*filename*⟩`=`*None*`,` ⟨*type*⟩`=` *None*`,` ⟨***kwds*⟩`)`

Parameters

network : network object

Network to be drawn. The network can be a `cnet`, `networkx`, `igraph`, `pathpy` object, or a tuple of a node list and edge list.

filename : file, string or None, optional (default = None)

File or filename to save. The file ending specifies the output. i.e. is the file ending with `.tex` a tex file will be created; if the file ends with '.pdf' a pdf is created; if the file ends with `.csv`, two csv files are generated `filename_nodes.csv` and `filename_edges.csv`. If the filename is a tuple of strings, the first entry will be used to name the node list and the second entry for the edge list; and if no ending and no type is defined a temporary pdf file is compiled and shown.

type : str or None, optional (default = None)

Type of the output file. If no ending is defined trough the filename, the type of the output file can be specified by the type option. Currently the following output types are supported: `'tex'`, `'pdf'`, `'csv'` and `'dat'`.

kwds : keyword arguments, optional (default= no attributes)

Attributes used to modify the appearance of the plot. For details see below.

Keyword arguments for node styles

node_size : size of the node. The default is 0.6 cm.

node_color : color of the nodes. The default is light blue. Colors can be specified either by common color names, or by 3-tuples of floats (ranging between 0 and 255 for the R, G and B components).

node_opacity : opacity of the nodes. The default is 1. The range of the number lies between 0 and 1. Where 0 represents a fully transparent fill and 1 a solid fill.

node_label : labels drawn next to the nodes.

node_label_position : Per default the position of the label is in the center of the node. Classical Ti*k*Z commands can be used to change the position of the label. Instead, using such command, the position can be determined via an angle, by entering a number between -360 and 360. The origin (0) is the y axis. A positive number change the position counter clockwise, while a negative number make changes clockwise.

node_label_distance : distance between the node and the label.

node_label_color : color of the label.

node_label_size : font size of the label.

node_shape : shape of the vertices. Possibilities are: 'circle', 'rectangle', 'triangle', and any other Tikz shape

node_style : Any other Tikz style option or command can be entered via the option style. Most of these commands can be found in the "TikZ and PGF Manual". Contain the commands additional options (e.g. shading = ball), then the argument for the style has to be between { } brackets.

node_layer : the node can be assigned to a specific layer.

node_label_off : is Boolean option which suppress all labels.

node_label_as_id : is a Boolean option which assigns the node id as label.

node_math_mode : is a Boolean option which transforms the labels into mathematical expressions without using the $ $ environment.

node_pseudo : is a Boolean option which creates a pseudo node, where only the node name and the node coordinate will be provided.

Keyword arguments for edge styles

edge_width : width of the edges. The default unit is point (pt).

edge_color : color of the edges. The default is gray. Colors can be specified either by common color names, or by 3-tuples of floats (ranging between 0 and 255 for the R, G and B components).

edge_opacity : opacity of the edges. The default is 1. The range of the number lies between 0 and 1. Where 0 represents a fully transparent fill and 1 a solid fill.

edge_curved : whether the edges should be curved. Positive numbers correspond to edges curved in a counter-clockwise direction, negative numbers correspond to edges curved in a clockwise direction. Zero represents straight edges.

edge_label : labels drawn next to the edges.

edge_label_position : Per default the label is positioned in between both nodes in the center of the line. Classical Tikz commands can be used to change the position of the label.

edge_label_distance : The label position between the nodes can be modified with the distance option. Per default the label is centered between both nodes. The position is expressed as the percentage of the length between the nodes, e.g. of distance = 0.7, the label is placed at 70% of the edge length away of Vertex i.

edge_label_color : color of the label.

edge_label_size : font size of the label.

edge_style : Any other Tikz style option or command can be entered via the option style. Most of these commands can be found in the "TikZ and PGF Manual". Contain the commands additional options (e.g. shading = ball), then the argument for the style has to be between { } brackets.

edge_arrow_size : arrow size of the edges.

edge_arrow_width : width of the arrowhead on the edge.

edge_loop_size : modifies the length of the edge. The measure value has to be insert together with its units. Per default the loop size is 1 cm.

edge_loop_position : The position of the self-loop is defined via the rotation angle around the node. The origin (0) is the y axis. A positive number change the loop position counter clockwise, while a negative number make changes clockwise.

edge_loop_shape : The shape of the self-loop is defined by the enclosing angle. The shape can be changed by decreasing or increasing the argument value of the loop shape option.

edge_directed : is a Boolean option which transform edges to directed arrows. If the network is already defined as directed network this option is not needed, except to turn off the direction for one or more edges.

edge_math_mode : is a Boolean option which transforms the labels into mathematical expressions without using the $ $ environment.

edge_not_in_bg : Per default, the edge is drawn on the background layer of the tikz picture. I.e. objects which are created after the edges appear also on top of them. To turn this off, the option edge_not_in_bg has to be enabled.

Keyword arguments for layout styles

layout : dict or string, optional (default = None) A dictionary with the node positions on a 2-dimensional plane. The key value of the dict represents the node id while the value represents a tuple of coordinates (e.g. n = (x,y)). The initial layout can be placed anywhere on the 2-dimensional plane.

Instead of a dictionary, the algorithm used for the layout can be defined via a string value. Currently, supported are:

Random layout , where the nodes are uniformly at random placed in the unit square. This algorithm can be enabled with the keywords: "Random", "random", "rand", or `None`.

Fruchterman-Reingold force-directed algorithm . In this algorithm, the nodes are represented by steel rings and the edges are springs between them. The attractive force is analogous to the spring force and the repulsive force is analogous to the electrical force. The basic idea is to minimize the energy of the system by moving the nodes and changing the forces between them. This algorithm can be enabled with the keywords: “Fruchterman-Reingold”, “fruchterman_reingold”, “fr”, “spring_layout”, “spring layout”, “FR”.

Keys	Other valid keys
Random	Random, random, rand, None
Fruchterman-Reingold	Fruchterman-Reingold, fruchterman_reingold, fr spring_layout, spring layout, FR

Table B.1: Algorithms keyword naming convention.

force : float, optional (default = None) Optimal distance between nodes. If None the distance is set to $1/\sqrt{n}$ where n is the number of nodes. Increase this value to move nodes farther apart.

positions : dict or None optional (default = None) Initial positions for nodes as a dictionary with node as keys and values as a coordinate list or tuple. If None, then use random initial positions.

fixed : list or None, optional (default = None) Nodes to keep fixed at initial position.

iterations : int, optional (default = 50) Maximum number of iterations taken

threshold : float, optional (default = 1e-4) Threshold for relative error in node position changes. The iteration stops if the error is below this threshold.

weight : string or None, optional (default = None) The edge attribute that holds the numerical value used for the edge weight. If None, then all edge weights are 1.

dimension : int, optional (default = 2) Dimension of layout. Currently, only plots in 2 dimension are supported.

seed : int or None, optional (default = None) Set the random state for deterministic node layouts. If int, `seed` is the seed used by the random number generator, if None, the a random seed by created by the numpy random number generator is used.

Keyword arguments for general options

units : string or tuple of strings, optional (default = (’cm’,’pt’)) Per default, all size values are based on cm, and all line widths are defined in pt units. Whit this option the input units can be changed. Currently supported are: Pixel ’px’, Points ’pt’, Millimeters ’mm’, and Centimeters ’cm’. If a single value is entered

as unit all inputs have to be defined using this unit. If a tuple of units is given, the sizes are defined with the first entry the line widths with the second entry.

layout : dict A dictionary with the node positions on a 2-dimensional plane. The key value of the dict represents the node id while the value represents a tuple of coordinates (e.g. n = (x,y)). The initial layout can be placed anywhere on the 2-dimensional plane.

margins : None, int, float or dict, optional (default = None) The margins define the 'empty' space from the canvas border. If no margins are defined, the margin will be calculated based on the maximum node size, to avoid clipping of the nodes. If a single int or float is defined all margins using this distances. To define different the margin sizes for all size a dictionary with in the form of `{'top':2,'left':1,'bottom':2,'right':.5}` has to be used.

canvas : None, tuple of int or floats, optional (default = (6,6)) Canvas or figure_size defines the size of the plot. The values entered as a tuple of numbers where the first number is width of the figure and the second number is the height of the figure. If the option `units` is not used the size is specified in cm. Per default the canvas is 6×6 cm.

keep_aspect_ratio : bool, optional (default = True) Defines whether to keep the aspect ratio of the current layout. If `False`, the layout will be rescaled to fit exactly into the available area in the canvas (i.e. removed margins). If `True`, the original aspect ratio of the layout will be kept and it will be centered within the canvas.

standalone : bool, optional (default = True) If this option is true, a standalone latex file will be created. i.e. the figure can be compiled from this output file. If standalone is false, only the tikz environment is stored in the tex file, and can be imported in an existing tex file.

clean : bool, optional (default = True) Whether non-pdf files created that are created during compilation should be removed.

clean_tex : bool, optional (default = True) Also remove the generated tex file.

compiler : str or None, optional (default = None) The name of the LaTeX compiler to use. If it is None, cnet will choose a fitting one on its own. Starting with `latexmk` and then `pdflatex`.

compiler_args : list or None, optional (default = None) Extra arguments that should be passed to the LaTeX compiler. If this is None it defaults to an empty list.

silent : bool, optional (default = True) Whether to hide compiler output or not.

Keyword naming convention

In the style dictionary multiple keywords can be used to address attributes. These keywords will be converted to an unique key word, used in the remaining code. This allows to keep the keywords used in `igraph`.

Table B.2: Keyword naming convention.

Keys	Other valid keys
node	vertex, v, n
edge	link, l, e
margins	margin
canvas	bbox, figure_size
units	unit
fixed	fixed_nodes, fixed_vertices, fixed_n, fixed_v
positions	initial_positions, node_positions, vertex_positions, n_positions, v_positions

Bibliography