

TikZ tutorial



- Manual: www.ctan.org/pkg/pgf
- Example repository: www.texample.net/tikz/
- Community for specific help: tex.stackexchange.com/ (doubtful evolution though)

Contents

1	Introduction—drawings and canvas	2
1.1	Coordinates and points—placing the pencil	3
1.2	Coordinate systems	3
1.3	Loops	4
2	Points and curves—drawing	5
2.1	Paths and their characteristics—moving the pencil	5
2.2	Line-to and relative cartesian coordinates—where to move the pencil	7
2.3	Line-to with controls—how to move the pencil. Relative polar coordinates	8
2.4	Path actions	9
3	Defining points through coordinate computations	10
3.1	Barycenters and points as images of direct similarities	11
3.2	Points as images of orthogonal projections	12
3.3	Points through intersections	13
4	Geometric transformations—path actions	14
5	Arrows—path attribute	16
6	Nodes—path operation	16
6.1	Defining nodes	17
6.2	Relative positioning of nodes	19
6.3	Connecting the nodes; the <code>edge</code> operation	22
7	Plot—path operation	23
7.1	Reading points from an external file	23
7.2	Drawing graphs of functions and parametric curves	23
7.3	<code>addplot</code> and <code>addplot3</code>	23
8	Pics—path operation	25

9	Decorations, i.e. markings on paths	26
9.1	Markings	26
9.2	Random edge	28
9.3	Text effects	29
10	Shapes	30
11	Other path actions	30
11.1	Pre and post actions	30
11.2	let ... in	32
11.3	\foreach	36
12	Using mathematics and functions	37
13	Defining keys	37
A	Various examples	40
A.1	The construction of an equilateral triangle when the circumcircle and a vertex are given	40
A.2	Dandelin's spheres for a hyperbola defined as a conic section	40
A.3	Torus in the light	42
A.4	Penrose triangle	42
B	Drawing a path of varying width	42

1. Introduction—drawings and canvas

TikZ is the frontend layer of PGF for drawing in \LaTeX . Here drawing means the use of points, curves, and shapes to convey a message. **TikZ** commands for doing it parallel natural drawing gestures: where to place the pencil, where to move the pencil, how to move the pencil, how to enhance a drawn element, and so on. The syntax is the usual \LaTeX one.

Drawing commands have to be enclosed in an *tikzpicture* environment. Among the global options, a useful one is `baseline=d`, like in $\square \square$ where `baseline=1ex` for the second rectangle (the baseline of the drawing is modified with respect to the baseline of the surrounding text).

```
\begin{tikzpicture}[<options>]
  <tikz commands>
\end{tikzpicture}
```

The basic elements of **TikZ** are

- points; `\path (a,b) coordinate (P)`; defines the point $P = (a, b)$
- paths; `\draw (P) -- (Q)`; defines the line segment $[PQ]$ where P and Q were previously defined
- rectangles, circles, and ellipses (i.e. basic shapes);
`\draw (0,0) rectangle (1em,2ex)`; draws \square as an in-text rectangle
`\draw (0,0) circle (1ex)`; draws \circ as an in-text circle of radius $2ex$
`\draw (0,0) ellipse[x radius=2em, y radius=ex]`; draws $\text{\scriptsize } \bigcirc$

1. Introduction—drawings and canvas

In this introduction I briefly present the points and the coordinate systems. Then in the following sections, I continue the presentation of the basic elements and of their main options. For an element, the presentation will be made in two steps: a simpler one first, and later, being more at ease with *TikZ*, a more detailed one.

1.1. Coordinates and points—placing the pencil

The simplest way of defining a point is shown below. The name of the point is *A* and can be used afterwards. In the example, a circle is drawn centered at *A*.

```
\coordinate (A) at (0, 0);
\draw (A) circle (2pt);
```

`\coordinate` is an abbreviation for `\path coordinate`.

It is worth noticing that on the current page there are points associated to absolute positions. The transparent antique white rectangle in the upper right corner is drawn using them. The elements appearing in this definition will be explained in the next sections:

- Apply the option `remember picture` to the `tikzpicture` environment; it enables *TikZ* to save the picture size and various segments in the picture.
- Specify the `overlay` option which puts the picture out of the text-segment. It allows it to be placed on top of text without taking up any space (otherwise the drawing is a regular text-box which is placed next to the preceding TeX-box).
- The two points defining the rectangle are `(current page.north east)` and an affine combination of it with `(current page.center)`. The rectangle is shifted using the *key=value operation* `transform canvas={shift={(-1,-1)}}`.
- Note that the layer system (see the use of `pgfsetlayers` in the code) has no effect outside the picture environment. The text already written on the page is under the antique white rectangle.

For other ways of defining points (calculated positions, intersections, ...) see §3.

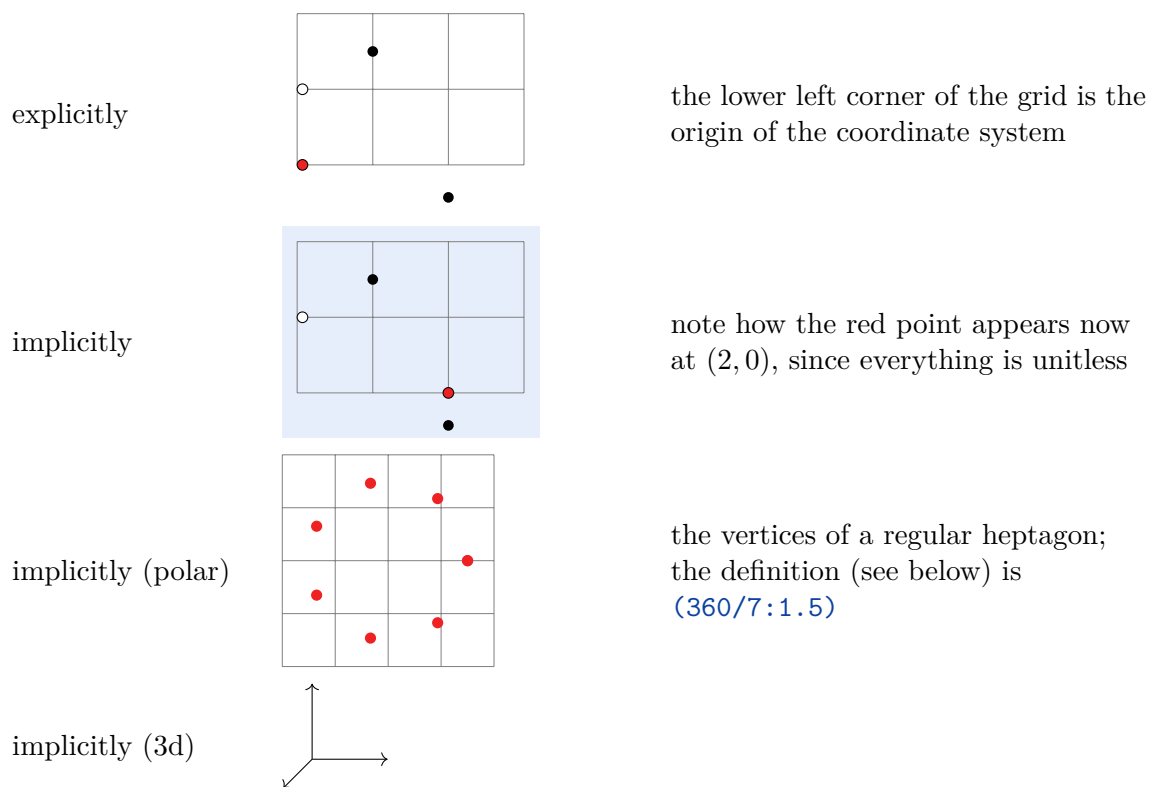
1.2. Coordinate systems

A coordinate is a position on the canvas—on which your picture is drawn. There are two ways of specifying which coordinate system should be used:

explicitly the name of the coordinate system is given at the beginning, followed by `cs:`, which stands for “coordinate system”, followed by a specification of the coordinate using the *key-value* syntax.

implicitly *TikZ* notices when you use a coordinate specified in a special syntax and chooses the correct coordinate system.

Examples. In the first example, the red point is specified by `(canvas cs:x=2, y=0)`, i.e. unit-less coordinates, while the white one by `x = 2pt`.



```
\draw[help lines] (0,0) grid (4,3);
```

creates and draws a coordinate system grid. It is a version of the path operation `rectangle`.

1.3. Loops

`\foreach \i [options] in {<list>}{<commands>}`; is a **for ... do** command. It can also be used as a *path action*, see §11.3. The `options` are introduced using the keys `count`, `evaluate`, and `remember` (see Figure 2). The former key is useful when the loop is constructed on a list having no connection with the integers (see Figure 1).

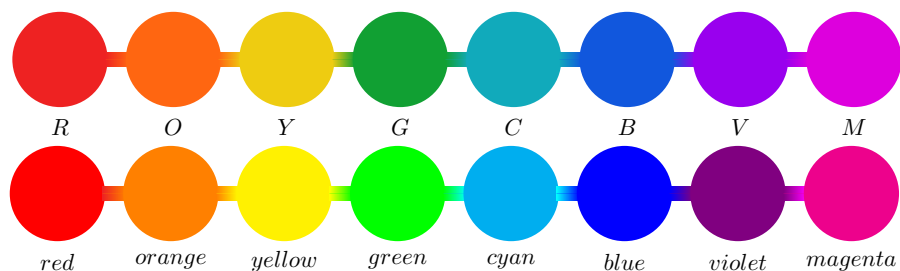


Figure 1: Name of the colors used in this presentation and the comparison with the predefined colors having the same “name”.

```
\fill[R] (-2, 0) circle (2.1) node at +(0, -3) {$R$};
\foreach \rC [count=\j from 0, remember=\rC as \lC (initially R)]
in {O, Y, G, C, B, V, M}{%
```

```

\fill[left color=\lC, right color=\rC] (5*\j, -.3) rectangle +(1, .6);
\fill[\rC] (5*\j, 0) -- ++(3, 0) circle (2.1) node at ++(0, -3) {\rC$};
}
\end{tikzpicture}

```

In the following drawing the key `evaluate` is introduced.

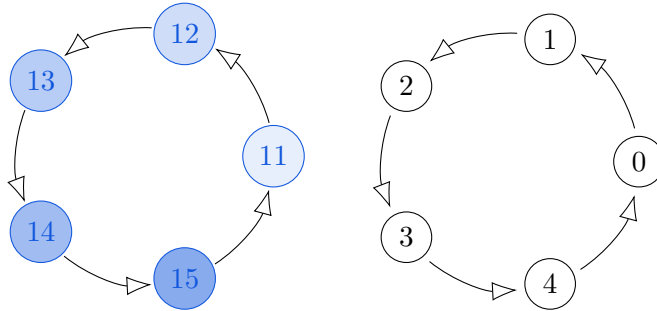


Figure 2: Using the keys `count`, `evaluate`, and `remember`.

```


\foreach \j [count=\i from 11, evaluate=\j as \a using {360/\nb*(\j -1)},
evaluate=\j as \tmp using {50*\j/\nb}]
in {1,...,\nb}{%
  \node[B, draw, circle, fill=B!\tmp!] at (\a: \radius) {\i$};
  \draw[arrows={-Latex[length=2ex, open]}]
  (\a +\margin: \radius) arc (\a +\margin: {\a +360/\nb -\margin}: \radius);
}

```

`\margin` (=15) is an angle defined according to the node's diameter; notice the difference between the two drawings in Fig. 2 introduced by the slightly smaller nodes on the right.

2. Points and curves—drawing

2.1. Paths and their characteristics—moving the pencil

A *path* (or curve) is a series of straight and curved line segments joining a series of points. The command to create it is `\path <specifications>`. To make it visible as a curve, the action `draw` must be invoked. For example  is the result of

```

\path[draw] (0, 0) -- (2em, 1ex) coordinate (A);
\path[draw, LightGray] (0, 0) rectangle (1em, 1ex);

```

The `<specifications>` is a sequence of *path operations*, *path actions*, and *characteristics* (or *attributes*) telling `TikZ` how the path is built and what it does. Some examples:

<code>-- (0,0)</code>	=	<i>line-to operation</i> meaning “continue the path from wherever you are to the origin”
<code>to (0,0)</code>	=	same as the previous
<code>let ... in</code>	=	let operation which evaluates some coordinates and numbers and assign the results to special macros (<code>\x.</code> , <code>\y.</code> , <code>\p.</code>) (see § 11.2)
<code>[draw]</code>	=	action making the path visible
<code>[color=blue]</code>	=	attribute, the color used for drawing the path—all parts of the path—
<code>[line width=2pt]</code>	=	attribute, the width of the path—all parts of the path—
<code>coordinate</code>	=	operation defining a point special <i>path operation</i> that can be used only when a normal path operation could follow
<code>node</code>	=	The effect is to typeset the node’s text and to place it at the “current location” on the path (see § 6). Note that the nodes are not part of the path . After the curve has been drawn, the nodes are added in a post-processing step.
<code>pic</code>	=	special <i>path operation</i> that introduces a short picture and can be used where a node can (see § 8)

The attributes have the `[key=value]` syntax. Some path actions also.

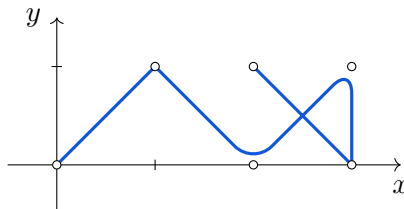
The second path below is drawn with the attribute `line width=2pt`, the third with `rounded corner`, and the fourth with `color=B`, `line width=2pt`, and `rounded corner=10pt`.



The attribute `rounded corner` can be used *in-line* during the drawing process; the line width cannot! The last value of rounded corner is used for the whole curve, or the whole scope. For example, the command

```
\draw[B] (0,0) -- (1,1) {[rounded corners=10pt] -- (2,0) -- (3,1)}
-- (3,0) -- (2,1);
```

produces the next figure.



The six points are drawn at the end, after the curve has been traced.

`\draw` is an abbreviation for `\path[draw]`.

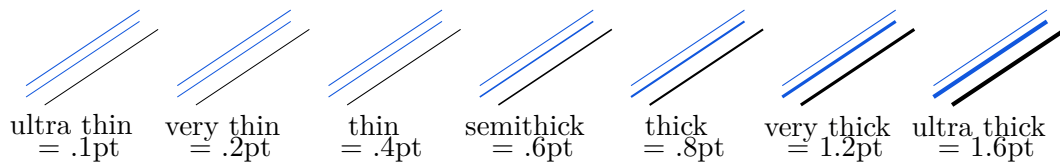


Figure 3: Some “natural” predefined line widths, the default line width being .5 pt. The top segment is drawn with the default line width.


The key `line join` specifies how lines meet; the possible values are `round`, `bevel`, and `miter`, with the last one being the default value. The result of using the second:  There exists also the key `line cap` with three possible values: `round`, `rect`, and (default) `butt`.



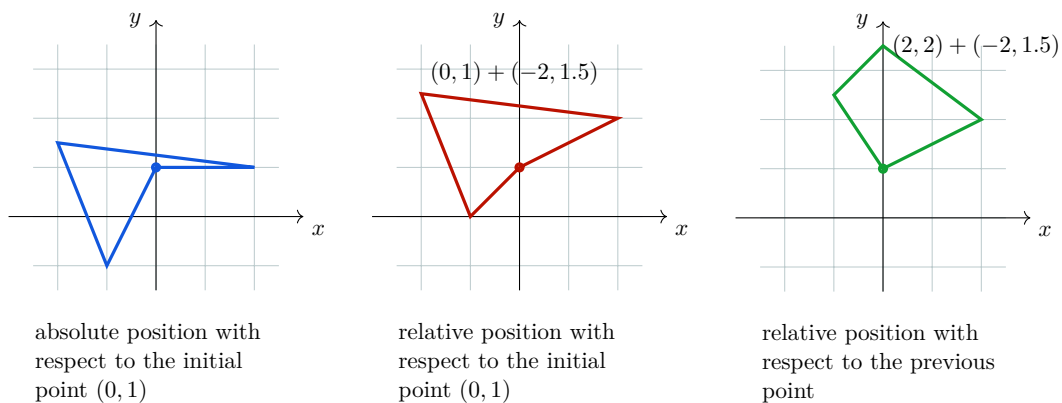
Figure 4: The three possibilities for `line cap` distinguished by colors, `rect`, `round`, `butt`

2.2. Line-to and relative cartesian coordinates—where to move the pencil

In the line-to operation, the to-point can be given using absolute coordinates relative to the canvas, relative coordinates relative to the **first preceding absolute defined** point, or relative coordinates relative to the previous point, the from-point. The three syntaxes are

`-- (B)` `-- +(B)` `-- ++(B)`

respectively. See the three closed curves below.

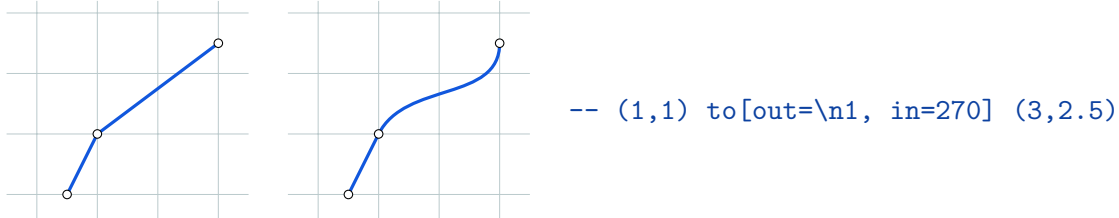


The three path commands, in order, are written below.

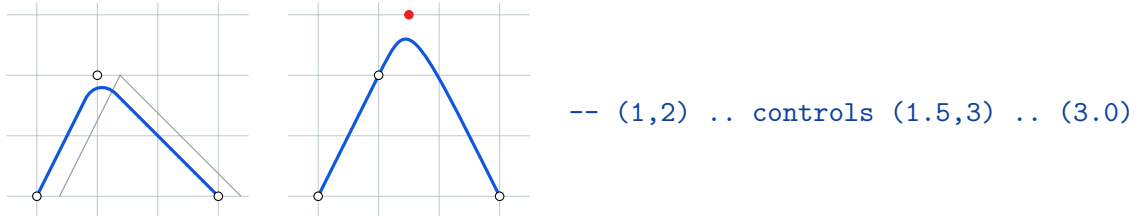
```
\draw (0,1) -- (2,1) -- (-2,1.5) -- (-1,-1) -- cycle;
\draw (0,1) -- +(2,1) -- +(-2,1.5) -- +(-1,-1) -- cycle;
\draw (0,1) -- ++(2,1) -- ++(-2,1.5) -- ++(-1,-1) -- cycle;
```

2.3. Line-to with controls—how to move the pencil. Relative polar coordinates

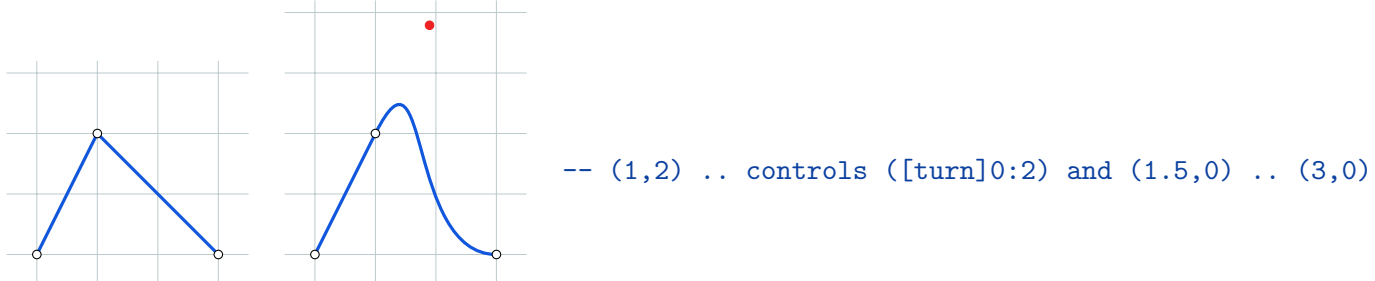
The controls (for the line-to operation) are tangent vectors indicating how to move the pencil when drawing the current piece of the path. In the next examples, various type of controls are given. They are indicated on the right; the first drawing is the reference obtained without any control, i.e. the polygonal line.



The tangent vector at $(1,1)$ makes an angle of $\n1$ degrees (which was previously computed as $\arctan(2)$). It is the angle made by the piece of curve leaving the from-point. The tangent vector at $(3, \frac{5}{2})$ is of 270 degrees—the angle of the curve entering the to-point.



The tangent vector at $(1,2)$ is $(.5, 1) = (1.5, 3) - (1, 2)$.



Now, the tangent vector at $(1,2)$ is the vector that supports the line segment from $(0,0)$ to $(1,2)$ and is of length 2. The point obtained by adding th vector to $(1,2)$ is indicated in red.

Remark. The controls are best given either in a relative form $+(\text{vector})$ through operations on coordinates, or using the relative polar coordinates as above. Explicitly, identical curves are obtained with the following three line-to operations.

- -- (A) .. controls (P) and $(\$ (B) + (w) \$)$.. (B)
Here P is the point $A + v$ for a certain vector v ; the curve leaves A with speed vector v . The second control point will be explained in §3—mathematically, it is $B + w$.
- -- (A) .. controls $+(v)$ and $+(w)$.. (B)
Here, $+(v)$ is the point $P = A + v$ and $+(w)$ is the point $B + w$.
- -- (A) .. controls ([turn]0: norm0fv)
Here, the *key* `turn` is used; inside a coordinate, it continues the curve in a direction that might not be controlled a priori.

$([\text{turn}]<\text{angle}>:<\text{norm}>)$ stands for either a point or a tangent vector. The current line-to operation for which the command $([\text{turn}]<\text{angle}>:<\text{norm}>)$ is either the to-point or the first

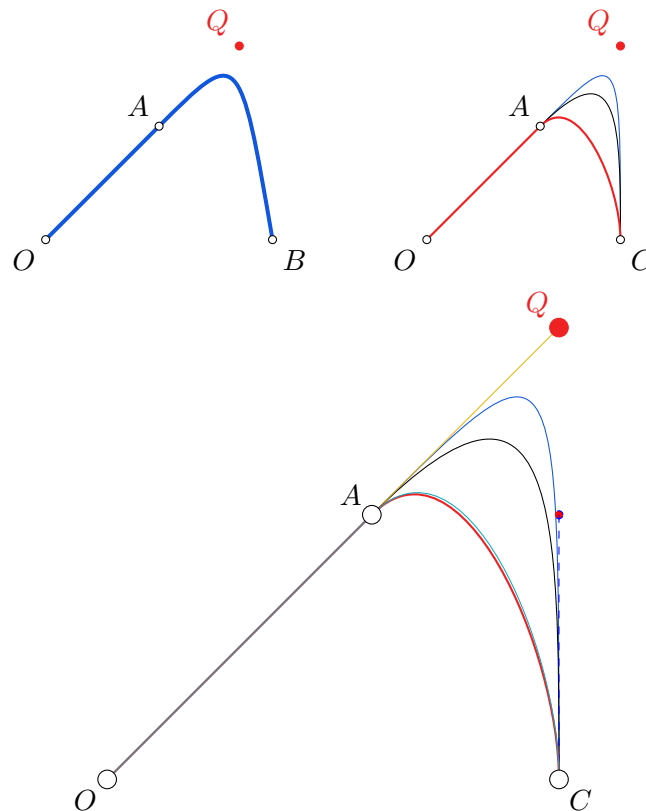


Figure 5: Different manners to define the same control (tangent vector). On the left, the three ways described previously yield curves that are identical—superposed in the drawing. On the right, the comparison with the command line `to[out=45, in=90]` (the red curve) is made. For the middle black curve, the tangent vectors are of norm 1 and both point towards Q .

control, **must** follow a `line-to` operation; if its ending tangent vector makes an angle α , then the starting tangent angle of the current `line-to` equals $\alpha + \langle \text{angle} \rangle$.

For example in `-- (A) to[bend left] (B) -- ([turn]0:2)` the curve is smoothly continued through B . But if the last `line-to` operation were `-- ([turn]90:2)`, then B would be a singular point and the two tangents (toward B and away from B) would form a 90° direct angle. See Fig. 6.

2.4. Path actions

Once a path has been constructed, different things can be done with it, such that:

- be drawn (or stroked) with a “pencil” \Leftarrow `draw`
- be filled with a color \Leftarrow `fill`
- be shaded (around some colors) \Leftarrow `shade`
(shading is a variation on filling that changes colors smoothly from one to another)
- be used for clipping subsequent drawing \Leftarrow `clip`
(all subsequent drawings up to the end of the current scope are clipped against the current path and the size of subsequent paths **will not be important for the picture size**)

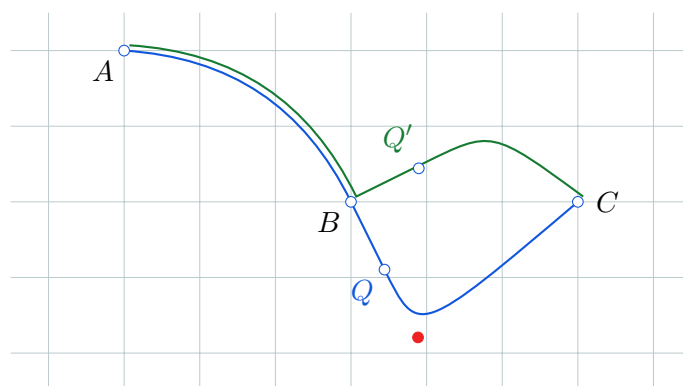
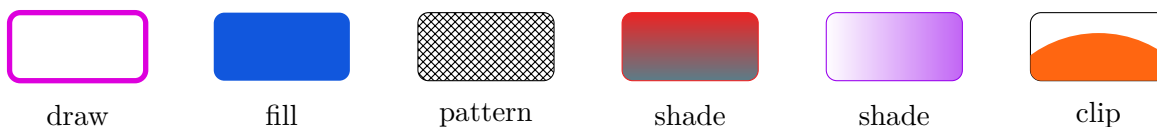


Figure 6: Using `[turn]`; for the blue curve, Q is the end point of `-- ([turn]0:1)`. For the green curve, Q' is the end of `-- ([turn]90:1)`.


- be used to specify the extent of the picture \Leftarrow `use as bounding box`
- be geometrically (affine) transformed \Leftarrow `rotate, scale, (xy)shift`
- any combination of these actions at the same time.

The actions appear as optional arguments along the `\path` command. Abbreviations exist; for example `\filldraw` abbreviates `\path[fill, draw]`.



The first shade action is given by

```
\draw[R, shade, bottom color=C, top color=R] (9,0) rectangle +(2, 1);
```

Now, to understand the bounding box action, look at the following in-text image representing a red disk.  The rectangle's edges bound the image. Note the vertical space that is introduced only. The code follows.

```
\draw[draw, use as bounding box] (0,0) rectangle (3ex, 4ex);
\fill[R, opacity=.2] (0, 0) circle (2.5ex);
```

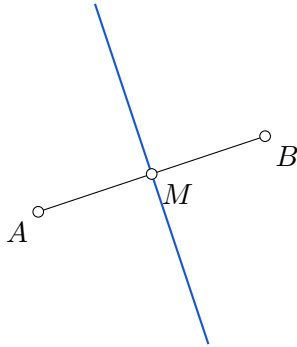
3. Defining points through coordinate computations

How can be obtained the coordinates of points issued from geometric constructions such as the intersection of two curves, the orthogonal projection of a point on a line segment, the center of the circumcircle of a triangle, the point of tangency under certain hypotheses ... ?

3.1. Barycenters and points as images of direct similarities

$(p_1)!c!(p_2)$, where c is a real number, is the barycenter $(1 - c)p_1 + cp_2$. Notice that this point can be obtained also through the heavier command that copies the mathematical formula, $(1-c)*(p_1)+c*(p_2)$. As a geometric transformation, it represents the image of p_2 through a homothety of center p_1 and ratio c .

$(p_1)!c!angle:(p_2)$ is the image of p_2 through a direct similarity of center p_1 , ratio c , and angle $angle$. This similarity is the composition of the rotation of center p_1 and angle $angle$ with the previous homothety.



The construction of the mediator of $[AB]$ is realised with the command below.

```
\draw (A) -- (B);
\draw ($(M)!2!90:(B)$) -- ($(M)!2!-90:(B)$);
```

In Fig. 7, the spiral is defined using the code below. Note that `tmp` is computed at each step of the loop using the `evaluate` key and that the radius, `\r{i}` was computed initially using again the `evaluate` key.

```
\begin{tikzpicture}[evaluate={%
  int \i;
  for \i in {0, ..., 10}{\r{i} = .1*(\i+1)};
}]
\foreach \i [evaluate=\i as \tmp using \i*10] in {0,...,10}{%
  \filldraw[white, fill=M!\tmp!C]
  ($(0, 0)!\i/12!\i*18:(5, -1)$) circle (\r{i});
}
\end{tikzpicture}
```

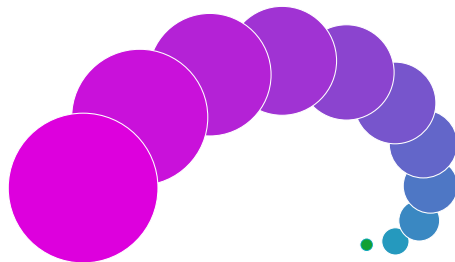


Figure 7: Using direct similarities; the green point is the origin of the similarity and the red one is the helping point in the partway modifier.

3.2. Points as images of orthogonal projections

$(\text{p1})!(\text{p})!(\text{p2})$ returns the projection of (p) on the line $(\text{p1}) -- (\text{p2})$. It is a projection modifier. There exists a more general form, but I can not see a reasonable situation to use it: $(\text{p1})!(\text{p})!\text{angle}:(\text{p2})$ returns the rotation with center (p1) of the given angle of the projected point.

As a first application, we consider in Fig. 8 the heights of a triangle. To obtain the common point of the three heights, we need to handle intersections. We'll do it in the next subsection.

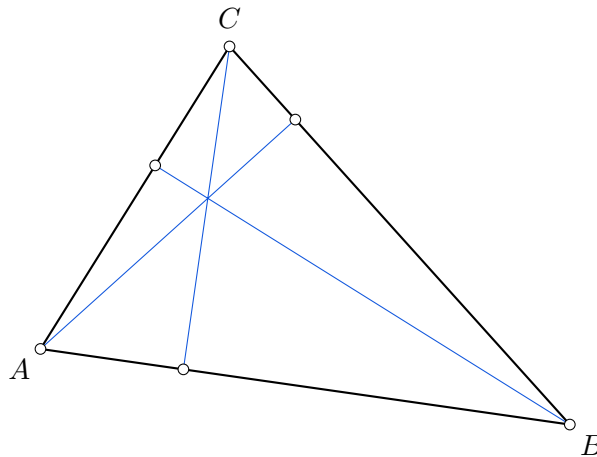


Figure 8: The heights of a triangle are meeting in a point.

The triangle and the heights have been stroked with the code below. Note the use of `-- cycle` for the triangle and of the *jump to* for the heights.

```
\draw[thick] (A) -- (B) -- (C) -- cycle;
\draw[B, thin]
(A) -- ($(B)!(A)!(C)$) coordinate (A')
(B) -- ($(A)!(B)!(C)$) coordinate (B')
(C) -- ($(A)!(C)!(B)$) coordinate (C');
```

As a second application, in Fig. 9, we construct the Simson line associated to a point P on the circumcircle of the triangle $[ABC]$ —it passes through the projections of P on the triangle's edges.

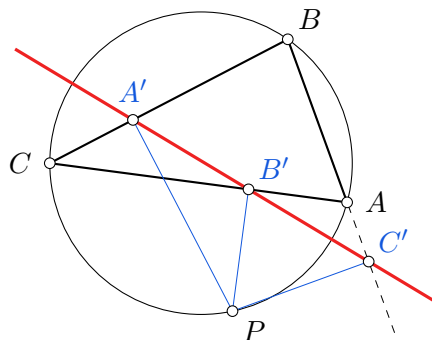


Figure 9: The Simson line associated to the point P on the $[ABC]$'s circumcircle. The points A' , B' , and C' are obtained through projection modifiers.

3.3. Points through intersections

The path action `name intersections`—with a *key=value* syntax and two arguments—returns, as names, the intersection points of two curves, its arguments. The intersecting curves must have been enriched with the attribute `name path`. I start by presenting three ways of using the `name intersections` action. For all three of them, an intersection point must be a “real” intersection of the two pieces of curve considered. There is a fourth way that will be seen in § 11.2—it can be used inside the path action `let ... in`.

Form 1) `name intersections={of=<curve1> and <curve2>, name=<prefix>}`

It defines the names’ prefix of the intersection points of the two curves. The code for the left drawing in Fig. 10 is

```
\path [name intersections={of=circle and line, name=iP}];
\foreach \j/\pos in {1/below, 2/below right}{%
  \filldraw[red] (iP-\j) circle (2pt) node[\pos, black] {$I_{\j}$};
}
```

The prefix is `iP` and the two points are `(iP-1)` and `(iP-2)`.

Form 2) `name intersections={of=<curve1> and <curve2>, by=<points>}`

It defines the individual names of the intersection points through the use of `by=`. The code for the middle drawing in Fig. 10 is

```
\path[name intersections={of=circle and line, by={A, B}}];
\foreach \P/\pos in {B/below left, A/above right}{%
  \filldraw[blue] (\P) circle (2pt) node[\pos] {$\P$};
}
```

Form 3) `name intersections={of=<curve1> and <curve2>, name=<prefix>, total=\t}`

It defines the names’ prefix of the intersection points and their cardinal through the macro `\t`. The code for the right drawing in Fig. 10 is

```
\fill[name intersections={of=circle and ellipse, name=I, total=\t}
[DarkB] \foreach \j in {1, ..., \t}{%
  (I-\j) circle (2pt) node[above right] {$I_{\j}$}
};
```

The macro `\t` in the right drawing from Fig. 10 is a **local** macro. It does not survive outside the path command. To be able to use it outside the path command, insert `\pgfextra{\xdef\total{\t}}`.

Remark. Concerning the order of the intersection points: if one of the curves is a line, then the points are ordered along the line (see Fig. 12).

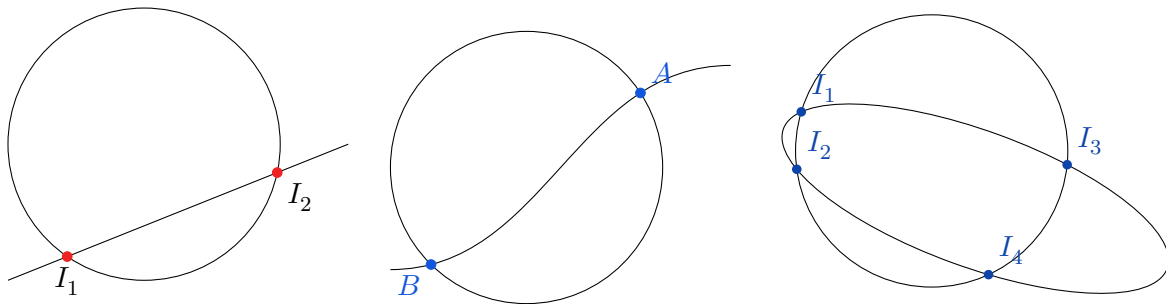


Figure 10: Using the three ways of dealing with the intersection points of two curves; note that the order of the two intersection points in the first two forms varies with the intersecting curves. In the middle drawing, the open curve is the same line as used in the left drawing, but with controls.

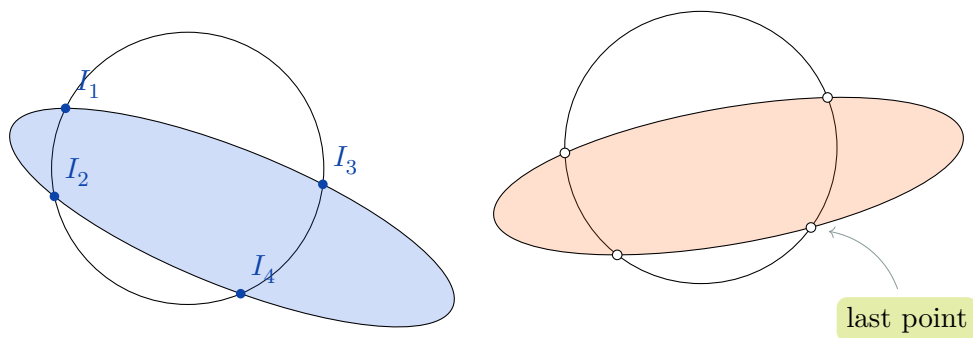


Figure 11: Around the total number of points in **intersection**; in the second drawing, the label is inserted outside the intersection command (see above).

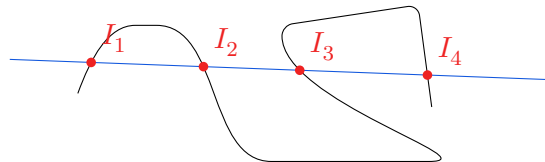


Figure 12: Intersection along a line

4. Geometric transformations—path actions

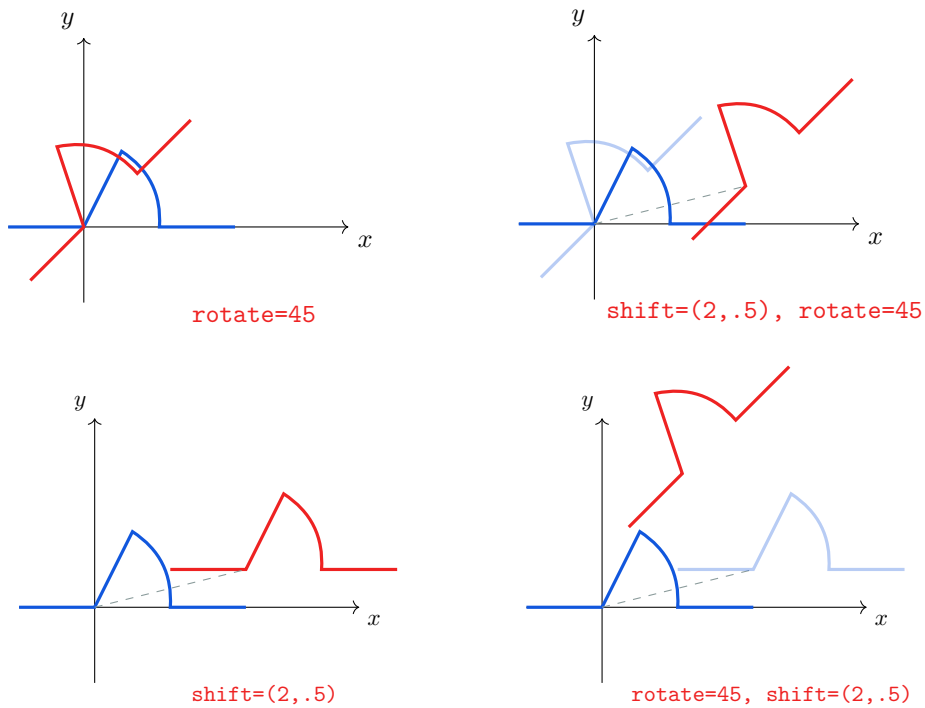
The geometric path actions are

- `rotate=<angle in degree>`
- `xshift=<length>, yshift=<length>`
- `shift=<{vector}>` (the curly brackets are needed!)
- `xscale=<factor>, yscale=<factor>`
- `scale=<factor>`

The actions' order of appearance counts as can be noticed in the following drawings. It is as if composing functions in mathematics: $(f \circ g)(x) = f(g(x))$. Explicitly, in

```
\draw[rotate=45, shift={(2,.5)}] ...
```

the translation is executed first, then the rotation.



The global options are applied at the end, see Fig. 19. It is as if they appear first in the local options.

The same non commuting behavior may be observed with scale and shift. They do not commute, the last one given being the first one executed; see Fig. 13.

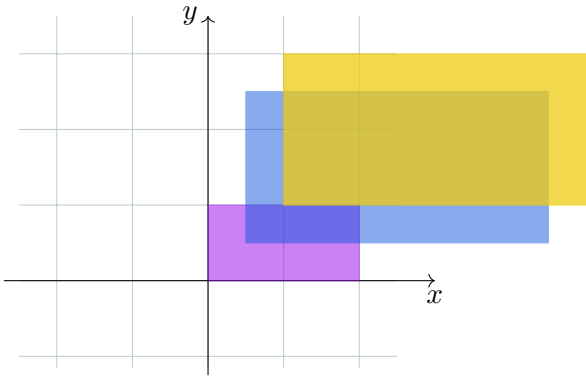


Figure 13: The operations `scale` and `shift` do not commute. The blue rectangle is obtained from the violet one through a scaling of factor 2 followed by shifting with the vector (0.5, 0.5). The order is changed for the yellow one. Note that in this case, the scaling acts on the translation vector, since everything in drawing instruction is expanded by 2.

5. Arrows—path attribute

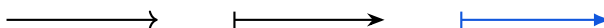
In order to add arrow tips to the lines which are drawn, the following conditions must be met:

1. the `arrow(s)` key or its short form must be specified
2. the curve doesn't realise a clip action
3. the curve is not closed.

Remark. It is better to use the library `arrows.meta`.

`arrows=<start arrow specification>-<end arrow specification>`

For example, as in `\draw[->] (0, 0) -- (2, 0);` or `\draw[|-Stealth] (0, 0) -- (2, 0);` that yield

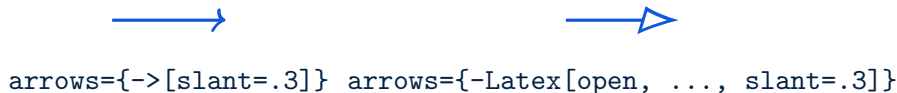


The blue arrow is obtained with `arrow=|-Latex`. Note that the tips properties can be defined independently of the curve's properties:



`slant=<factor>`

Slanting is used to create an italic effect for arrow tips; all arrow tips get “slanted” a little bit relative to the axis of the arrow.



Like for `color=` attribute, since the arrow option is so often used, the `arrows=` syntax is optional. But **it is no longer optional** if `slant` is one of the arrow's attributes.

Now, **the use of the curly braces** is different depending on whether `arrows=` is explicit.

Remark. The library `arrows.meta` is needed; the library `arrows` is deprecated.

6. Nodes—path operation

A node is typically a rectangle or circle or another simple shape with some text on it. Nodes are added to paths using the special path operation `node`. *Nodes are not part of the path itself.* Rather, they are added to the picture just before (with the attribute `behind path`) or after the path has been drawn.

Coordinate transformations do not apply to a node. Its anchor remains the same. To achieve a translation for example, the shift command must be placed inside the node's name parentheses.

6.1. Defining nodes

The path operation `...node[<options>] (nName) at (a,b) {content}...`; creates a **node** for the current path, i.e. writes down the *content* at the point (a,b) with some *node options* and name *nName*. The node options or attributes have no effect outside the node; they control its

- position with respect to the elements of the path
- anchor
- appearance
- label.

The end of the node path action is detected by the opening curly brace of the content. The name and the coordinates are optional.

When a node is typeset, all the text given in the braces is put in one long line—in an `\hbox`—and the node will become as wide as necessary (p. 224). The use of `\\` **must be accompanied** by the option `align` which acts on the node’s content.

this is a node with a line break inside its content
and `align=right` as one of its options

The `align` option must appear, otherwise the double backslash has no effect. But the same effect (almost) might be achieved through the node’s option `text width=width`—a better choice since the text properties, in the former solution, do not descend after the line break; see the second solution in the drawing below.

this is a node without a line break inside its content and `align=right`
as one of its options ; note the balanced lengths for the two lines

ABSOLUTE POSITION OF A NODE. The path action introducing a node can appear either inside a path operation or after a coordinate. In the former case, the node position is defined by the attribute `pos=t`, where $t \in [0,1]$. In the latter case, the node is placed at the last position mentioned before the `node` path action. Now, if `at (a,b)` is added to the node specification, then the node is placed at (a,b) instead.

The code of the left drawing in Fig. 14.

```
\path (0, 0) coordinate (A) node[below left] {$A$};
\path (3, 2) coordinate (B) node[below right] {$B$};
\draw (A) to[out=-30, in=180]
  node[draw, red, pos=.5, above left] {content}
  node[draw, blue, pos=.5] {content} (B);
\draw[fill=white] (A) circle (2pt) (B) circle (2pt);
```

Remark. The left drawing in Fig. 14 can be stroked in a single `\path` command. I present one possibility here for three reasons: it illustrates the general philosophy of *TikZ*, it opens

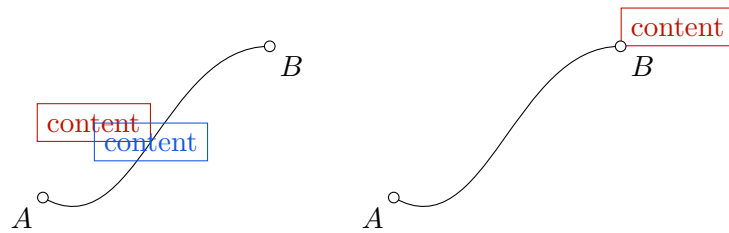
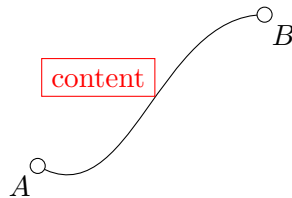


Figure 14: Nodes appearing either inside a path action (the left drawing) or after a coordinate (the right drawing). In the left drawing, the position of the blue and red nodes is initially the same; the red one is eventually modified by the attribute `above left`. In the right drawing, the red node is placed at B and is modified by the same attribute.

the discussion about the attributes and the labels of nodes, and finally, dealing with labels, it leads to the relative position of nodes.



```
\begin{tikzpicture}[every label/.style={inner sep=1pt},
  point2d/.style={circle, draw, fill=white, inner sep=2pt}]
  \draw
    (0,0) node[point2d, label={240:$A$}] (A) {}
    to[out=-30, in=180] node[draw, red, pos=.55, above left] {content}
    (3,2) node[point2d, label={-60:$B$}] (B) {};
\end{tikzpicture}
```

Note that the two white circles indicating the extremities of the curve are nodes now—the *shape* is circle having the radius controlled by the attribute `inner sep`—with no content. Their names are A and B as before, but this time, these names refer to nodes and not to coordinates. The A and B written down in Fig. 14 are the *labels* of the nodes. The labels are nodes also, hence the discussion about the relative positioning of nodes below, in § 6.2.

ANCHOR OF A NODE. We have already met the anchor of a node: in the left drawing of Fig. 14 the blue and red nodes have initially the same position, the red one being slightly moved through the attribute `above left`. The same characteristic can be obtained through the equivalent attribute `anchor=south east`.

`above left` \iff `anchor=south east`

These equivalent attributes correspond to different points of view on how the node is placed in the drawing: the coordinate system's for the former and the node's for the latter.

The node occupies in the drawing a certain space corresponding to its shape. The attribute `draw` makes this shape visible. The shape has nine anchoring points; the anchor places one of

them at (a, b) . See Fig. 15. I am referring to the path command

```
... (a,b) node[<options>] (nName) {content}...;
```

By default, the anchor is set at the center of the shape.

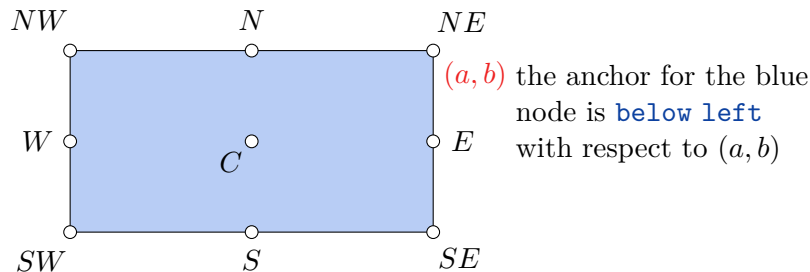


Figure 15: The nine anchoring points of a node; if the point $NE = (a, b)$ is the node's position, then the anchor used is `below left` with respect to (from the point of view of) the coordinate (a, b) .

APPEARANCE OF A NODE. There are many attributes that control the appearance of a node. Some of them are listed below.

- `draw`, `fill`, `fill color=...`, `opacity=...`, `fill opacity=...`
- `shape=...`, `inner sep=...`, `outer sep=...`
- `text color=...`, `text width=...`, `align=right/center/left`
- `rotate=angle`, `anchor=...`

LABEL OF A NODE. When the option `label` is given to a node operation, it causes another node to be added to the path after the current node has been finished. This extra node will have the text defined by `lText` and is placed, in the direction `lAngle` relative to the main node (on the border of it):

```
label=<lAngle>:<lText>.
```

The distance between a node and its label is controlled through the `[key=value]` syntax corresponding to the key `label={ [label distance=...] angle:text }` (see the “white” label and the violet one in Fig. 16).

`every label/.style` defines the style used in every node created by the label option. The default values are `draw=none`, `fill=none`.

Remark. It would be natural to use the nodes' label when drawing points in geometry. The point would be represented in this case by the main node. The philosophy would be different though, since the points would not be coordinates anymore, but nodes.

But coordinates support labels too! See the cyan point in Fig. 16 with the label “a point”.

6.2. Relative positioning of nodes

With the library `\usetikzlibrary{positioning}`, relative positioning can be used in placing the nodes. For example, a part of the code for the Fig. 16 is

```
\node[label={ [label distance=3em] 60:something}] (Main) {Main node};
\node[label={120:something else},
label={ [fill=WBuff, text width=7em, align=center,
```

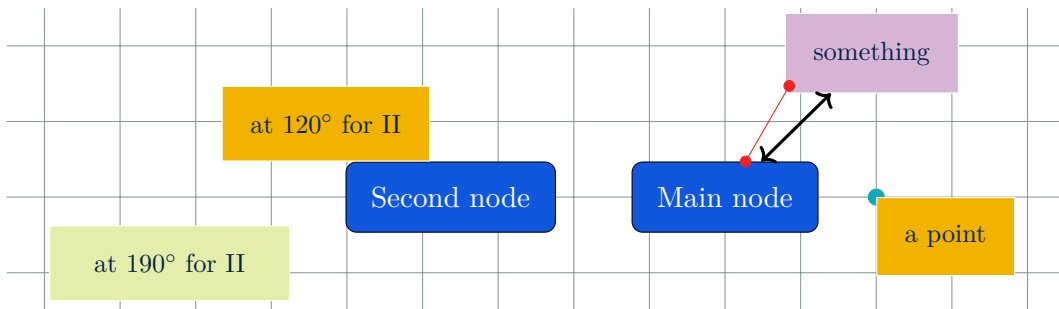


Figure 16: Labels and their position; in red are shown and connected the reference point on the border of the main node and the anchoring position chosen on the border of the label— `[label distance=3em] 60:something` was used. The anchoring point is determined in such a way that the label node will “face away” from the border of the main node. In black, the usual line-to is used, which points toward the center of both nodes (main node and its label).

```
label distance=5ex] 188:second label of Second node}]
(Second) [left=of Main] {Second node};
```

Note the option `left=of Main` which controls the position of the node `Second` with respect to the node `Main`.

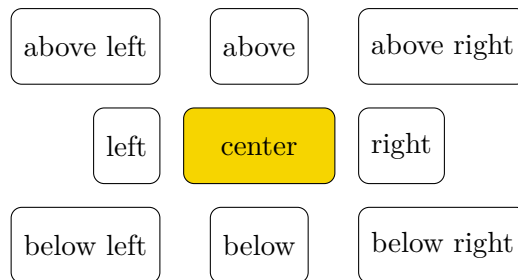


Figure 17: Relative positioning; note how the relative positioning induces the anchoring of the exterior nodes with respect to the node `(center)`.

In what follows, the relative position is inspected from the point of view of the vertical alignment—the height of each node equals 1cm .

- In Fig. 18, on the left, the position is defined using `below=of nodename`. The vertical and horizontal distances are the default ones between the borders of the nodes, 1cm . On the right, the position is defined using `below=of nodename1.west`. The vertical and horizontal distances are the default ones between the border (here `nodename2.north`) and the `nodename1.west` of the previous node.
- In Fig. 19, the positioning of the nodes is controlled through two instructions: the anchoring `anchor=west` and the vertical alignment, as in Fig. 18. The result depends on the order of these instructions. On the left, the vertical alignment appears first, and the anchoring second; on the right, it is the opposite. Note that the result of the drawings on the right in both figures are the same, except for a translation to the right of the second.

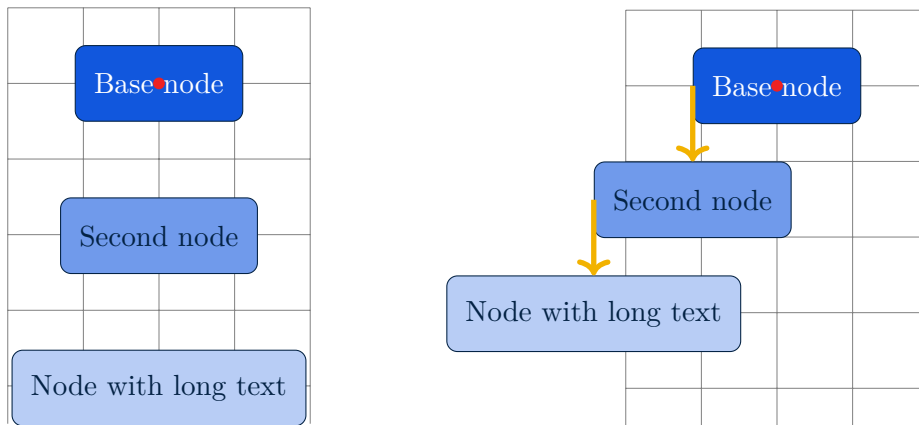


Figure 18: Vertical alignment using the library `positioning`. The distance between the nodes depends on the reference point given, if explicit. On the right, the reference point is set through `below=of nodename.west`.

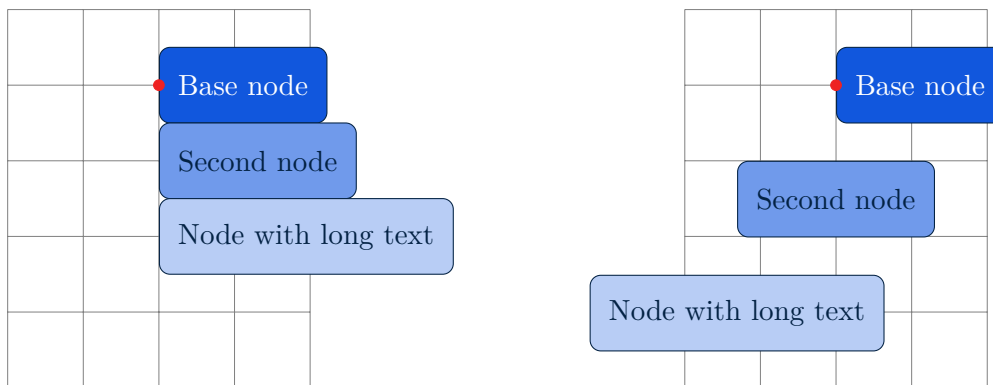
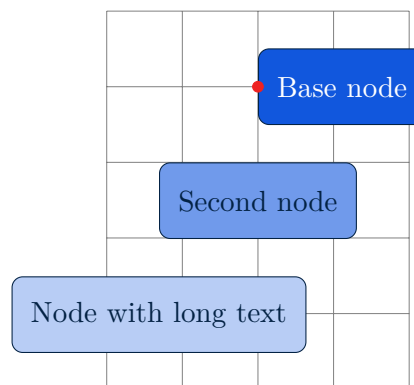


Figure 19: Vertical alignment; the red dot is the origin. On the left `below=of base.west` appears first and then `anchor=west`; on the the roles are exchanged.

In the next drawing `anchor=west` is given globally. The result is identical with the right image in Fig. 19, so a *global definition is equivalent to first position in a local definition* (see page 15 also).

Note that there are three actions in the placement of the second and third node. Explicitly for `second`:

1. `below=of base.west` \rightsquigarrow vertical alignment using `second.north`
2. the distance between `base.west` and `second.north` equals 1 *cm*
3. anchoring point is set to `second.west`—used for the next positioning.



Setting the anchor of the current node yields the reference point for its positioning (vertical and/or horizontal). The first instruction given is the last executed (see also Fig. 13).

6.3. Connecting the nodes; the `edge` operation

The `edge` operation has similar effect as the `node` operation in that it adds something after the main path has been drawn. However, it works like the `line-to` operation, that is, it adds a to-path to the picture after the main path has been drawn.

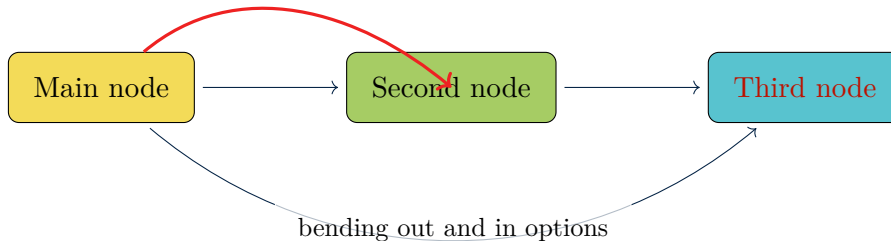


Figure 20: Connecting curves or arrows between nodes. The red arrow is a `to` operation; the others are `edge` operations.

As the `node` operation, an `edge` temporarily suspends the construction of the current path and a new path is constructed. (This new path will be drawn after the main path has been drawn.)

The `edge` operation doesn't move the pencil!

The code `spear/.style={->, BK, thin, shorten <=3pt, shorten >=3pt}` defines `spear`, the common style of the connecting arrows in Fig. 20. See also the *TikZ-PGF* manual, § 16.3, *Arrow Keys: Configuring the Appearance of a Single Arrow Tip*.

Remark. The `edge` operation creates a curve from the previous element of the path, to the following one. The `\tikztostart` is the last coordinate on the path just before the `edge` operation, just as for the `node` or `path-to` operations. However, there is one exception to this rule: *If the edge operation is directly preceded by a node operation, then this just-declared node is the start coordinate (and not, as would normally be the case, the coordinate where this just-declared node is placed—a small, but subtle difference).* Below, on the first row `to` is used; on the second `edge`.

```

----->
----->
first----->
first----->

```

```

\draw[R, ->] (0, 0) node {first} to (4,0);
\draw (0, 0) node {first} edge[->] (4,0);

```

Note that were the nodes previously defined, then `to` and `edge` would result in identical graphical behaviors; see below.

```

first-----to----->second
first-----edge----->second

```

```

\path
(0, 0) node[draw] (first) {first}
(4, 0) node[draw] (second) {second};
\path[->] (first) edge node[pos=.5, above] {edge} (second);

```

7. Plot—path operation

The `plot` path operation constructs a curve that passes through a large number of points given

- either given in a list of coordinates `plot[attributes] coordinates {A, B, ...}`
- read from a file `plot[..] file {filename}`
- or computed through a formula (yielding a parametric curve; see below).

Among the attributes, one can use `const plot mark right, mark=*`, see the blue graph in Fig. 21, which connects the points on the path using piecewise constant—horizontal and vertical—segments. The `mark right, mark=*` adds marks on the right ends of the horizontal segments.

7.1. Reading points from an external file

In the file format that `TikZ` allows each line either starts with two numbers separated by a space, or is empty (really empty or starting with `#` or `%` symbols).

- An empty line starts a *new data set* resulting in a new curve being plotted.
- A line starting with two numbers provides a new coordinate; they may be followed by arbitrary text, which is ignored (it might be a good habit to use `%`).

7.2. Drawing graphs of functions and parametric curves

The `plot` path operation can be used to draw graphs of functions, i.e. the coordinates are computed through a mathematical formula. This is the most “basic” way of drawing a graph or parametric curve; it leaves us with the construction of the environment (axes, ticks, ...).

The computations of the coordinates are influenced by the following options: `variable=`, `domain=`, `samples=`.

The curve $y^2 = x^3 + x^2$ was drawn using the command

```

\draw[variable=\t, domain=-1.6:1.6, samples=50]
plot (\t*\t-1, {\t*(\t*\t-1)}) node[above] {$y^2=x^3+x^2$};

```

7.3. `addplot` and `addplot3`

These commands are provided by the `pgfplots` package. Each axis in `pgfplots` is written into a separate environment, for example `\begin{axis}... \end{axis}`—for a normal set of axes.

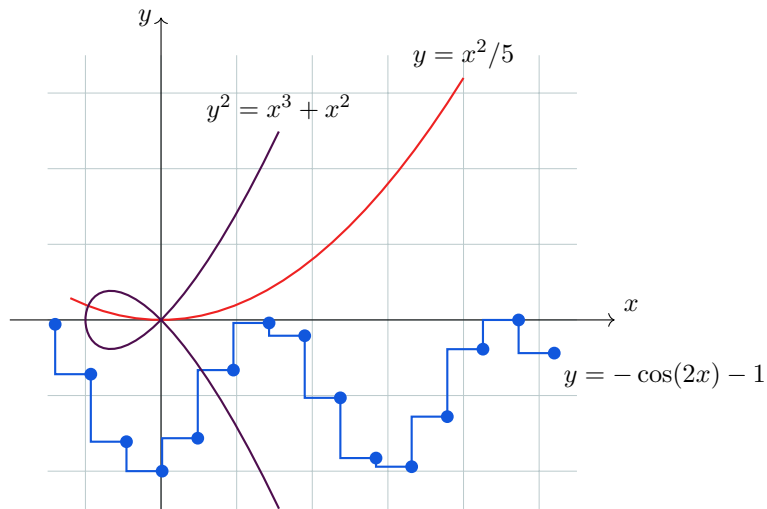


Figure 21: Using `plot`; note that expressions containing powers of the variable (written as in a \LaTeX mathematical expression, *e.g.* t^2) are badly handled—use `pow(t, 2)` instead.

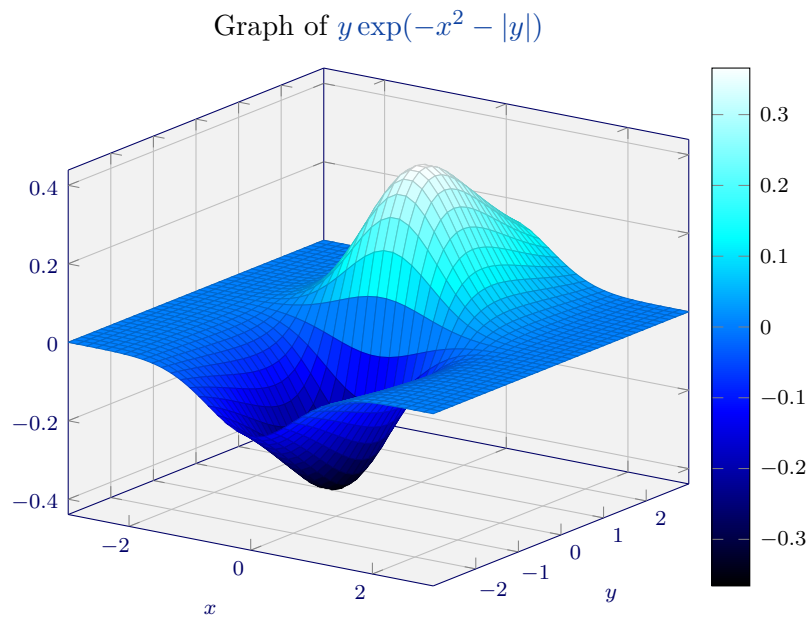


Figure 22: Using `\addplot3` with the key `view={35}{20}`.

Note that the code contains two sets of keys: the first is provided after `\begin{axis}[...]`; the second after `\addplot[...]` or `\addplot3[...]`. The former and the latter, in Fig. 22, are given below.


```
view={35}{20},
title={Graph of \textttc{\$y\exp(-x^2-|y|)\$}},
title style={black, font=\normalsize},
font=\scriptsize,
blue!40!black, % axes color and text
axis background style={fill=gray!10},
grid=major,
grid style={color=gray!50},
xlabel=$x$, ylabel=$y$,
ytick={-2,-1,...,2},
colormap/cold, colorbar,
colorbar style={font=\scriptsize},
scale=1.2
```

```
surf,
domain=-3:3,
domain y=-3:3,
samples=40
```

see `axis equal image` for axes' coherence

8. Pics—path operation

The path operation `pic` adds some drawing to the current path where a node might also be inserted. There are two main differences between nodes and pics:

- (i) Unlike nodes, pics cannot be referenced later on. A node inside a pic can be referenced but not “the pic itself”. In general, if some parts of the drawing must be connected to one another, it is better to use a node rather than a pic.
- (ii) A pic used to emulate the full power of a node will be slower to construct and take up more memory than a node achieving the same effect.

A first example of pic is given by the code bellow. It might produce



```
pics/seagull/.style={%
  code={%
    \filldraw[#1, very thin] (-2ex, .5ex)
    to[out=10, in=140] (0, 0) to[out=40, in=170] (2ex, .7ex)
    to[bend right] (0, -.3ex) to[bend right] (-2ex, .5ex) -- cycle;
  }
}
[...]
\path (-.1, .3) pic[rotate=30, scale=1.9] {seagull=C}
(0, 0) pic[rotate=-10, scale=2.4] {seagull=M};
```

As for nodes,

- a pic is placed either at the path's last position or, when `at` is used, at the corresponding specified position; the coordinate system is translated at the position (inside the pic's code, any mentioning of the origin refers either to the last position used on the path or to the specified `at`)
- the current path is not modified by the pic command, all drawings produced by the pic'code being “external” to the path

- a given option applies only to the pic and have no effect outside of it
- most “outside” options apply to the pic, but not the path actions like `draw` or `fill` which must be given in the pic’s code.

Example 8.1 (The whole game). In Fig. 23 a node is embedded in the pic named `young` (Young tableau); the name of the node, for further reference, is formed by concatenation from the name given to each calling of the pic and the initial name of the node.

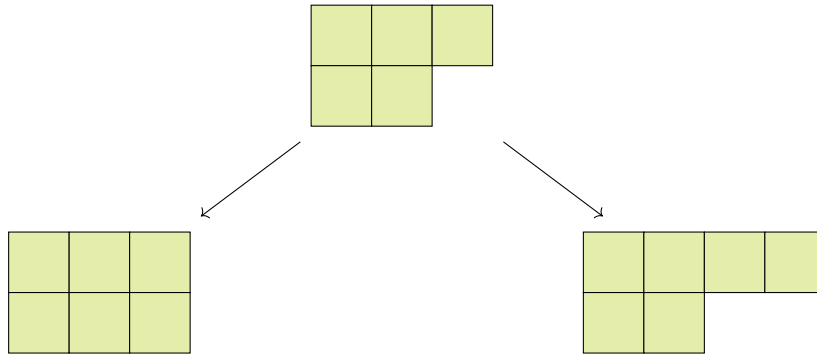


Figure 23: The pic `young` contains an embedded node and behaves as such for further reference. Note that **the relative positioning does not work well in this context.**

9. Decorations, i.e. markings on paths

A marking on a path is a *path operation* that places at a specific position a certain “small drawing”. For example, an arrow tip can be placed at the middle of a path to indicate a flowing direction, or information can be placed at certain positions of a path, or the path can be suggested by the “small drawings”.

9.1. Markings

Remark. `\usetikzlibrary{decorations.markings}` needs to be called for arbitrary markings.

The blue Frenet-Serret frames in Fig. 24 are introduced through the code below. The command `use frenet` allows us to draw using the Frenet coordiante system.

```
show frenet/.style 2 args={ % position and opacity
  decoration={
    markings, % switch on markings
    mark = at position #1 with {
      \path (0, 0) coordinate
      (origin-\pgfkeysvalueof{/pgf/decoration/mark info/sequence number});
      \path (1, 0) coordinate (tangent unit vector-\%
      \pgfkeysvalueof{/pgf/decoration/mark info/sequence number});
      \path (0, 1) coordinate (normal unit vector-\%
```

```

\pgfkeysvalueof{/pgf/decoration/mark info/sequence number});
\filldraw[B] (0, 0) circle (1.5pt) ++(0, -1.5ex);
\draw[thick, arrows={-Latex[width=7pt, open]}, opacity=#2] (0, 0) -- (1, 0);
\draw[thick, arrows={-Latex[width=7pt, open]}, opacity=#2] (0, 0) -- (0, 1);
}
}, postaction=decorate
}

```

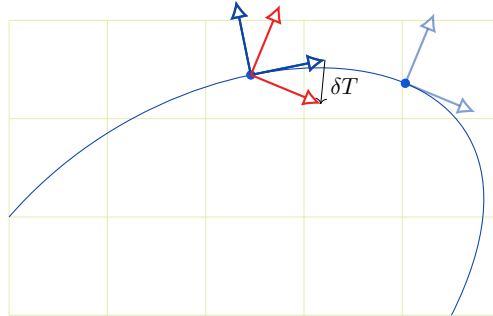


Figure 24: The Frenet-Serret frame, i.e. the T and N vectors, is represented at two points on a plane curve. A translated version of the second frame (in red) is indicated yielding the change in T denoted δT . Accordingly, δs is the distance between the points. When δs goes to 0, the vector $\frac{dT}{ds}$ is proportional to N . The curvature describes the speed of rotation of the frame.

`mark=at position ... with {<drawing commands>}` induces the followings:

- `pgf` determines the corresponding position on the path
- the coordinate system is translated to this position and is rotated such that the positive x -axis is tangent to the path
- a temporary `scope` is created, inside which the code is executed.

The `markings` decoration allows us to place one or more markings on a path. The decoration destroys the input path. To keep the path intact, the markings must be added through the use of either a pre or a postaction.

A second way to use the `mark` key is to place multiple marks from a starting position till an ending one and being spaced apart by a given step. These positions and step may be given in the same way as for the `at position` version—using units or no units and also using positive or negative values.

The decoration is defined as follows (`Omega` being the origin defined in the drawing):

```

counting/.style={%
  decorate,
  decoration={markings,
    mark=between positions 0 and 1 step{#1} with {%
      \draw (Omega) -- (0, 0) coordinate[
        name=mark-\pgfkeysvalueof{/pgf/decoration/mark info/sequence number}
      ] node[circle, draw, fill=Sand, transform shape]
      {\pgfkeysvalueof{/pgf/decoration/mark info/sequence number}};
    }
  }
}

```

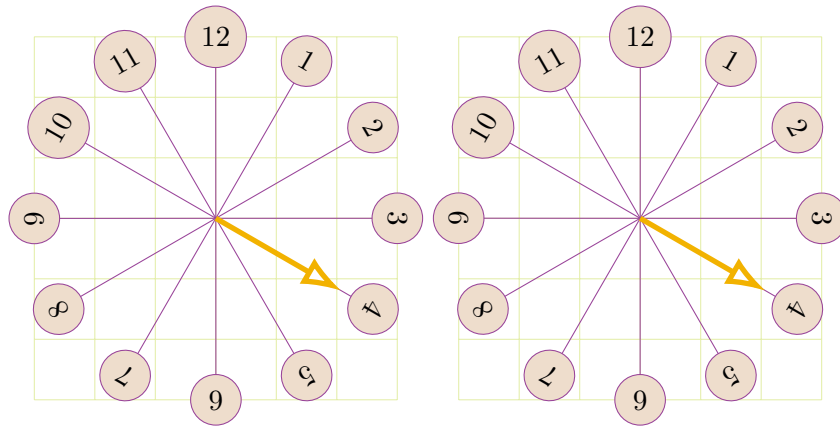


Figure 25: The final image is the same, but the construction is different. On the left an arc negatively parametrized is used; on the right a full circle but the image is reflected in the Oy axis. So the difference for the markings come from the open vs closed character of the curves.

```

}
}
}

```

Remark 9.1. For arrow tip markings, two special commands are available when the *code of a mark option* is executed:

- `\arrow[... , options]{arrow end tip}`
- `\arrowreversed[... , options]{arrow end tip}`

These commands draw the arrow end tip at the origin (of the marking), pointing right and left, respectively.

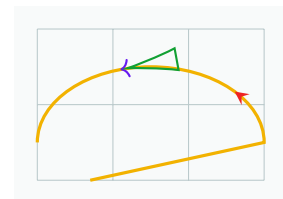
For example, in the drawing on the right, the red and green tips are obtained through

```

\arrow[R, very thick]{stealth}
\arrow[G, thick]{Latex[width=2ex, open]}

```

respectively. Note that the origin of the marking coincides with the summit of the arrow tip. The blue and green markings both occur at position 0.75.



9.2. Random edge

The following example needs the library `decorations.pathmorphing`. The name of the decoration is `random steps`. It has two arguments:

- `segment length` that determines the length of each step in the drawn process of the path
- `amplitude` that determines the interval from which the perturbation values in the x and y directions are both drawn.

```

\draw[decoration={random steps, segment length=1ex, amplitude=1ex},
decorate] (-.2, 0) -- (3, 1) arc (0: 180: 1.5 and 1) -- cycle;

```

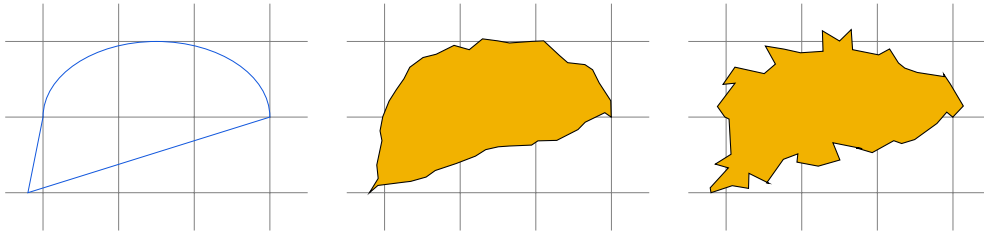


Figure 26: Random steps decoration; the last drawing uses `segment length=1ex` and `amplitude=1ex`.

9.3. Text effects

The examples in Fig. 27 use the library `decorations.text`. The philosophy is different for the “red” effect and for the other two .

First, `text along path` decorates the path with text; the drawing of the text is a “side effect” of the decoration.

- Each character is positioned using the center of its baseline. To move the text vertically (relative to the path), the additional key `raise` should be used.
- It is only possible to typeset text in math mode under considerable restrictions, say simple math expressions. Math subscripts and superscripts need to be contained within braces (e.g. $\{y_i^2\}$) as do commands like `\times` or `\cdot`.
- The text’s options can be given in a concatenation style; for example

```
text={It is a |\color{R}|ground breaking |+\bf|experience||, isn't it.}
```

Second, `text effects along path` is similar to `text along path` except that each character is inserted into the picture as a `TikZ` node. Thus, node options (such as `text`, `scale`, `opacity`) may be used to create *text effects*. Note the following differences with respect to `text along path`:

- Formatting (e.g. font, color) cannot be specified in the decoration text, but only through the node’s options.
- Due to the number of computations involved, this is quite a slow decoration.

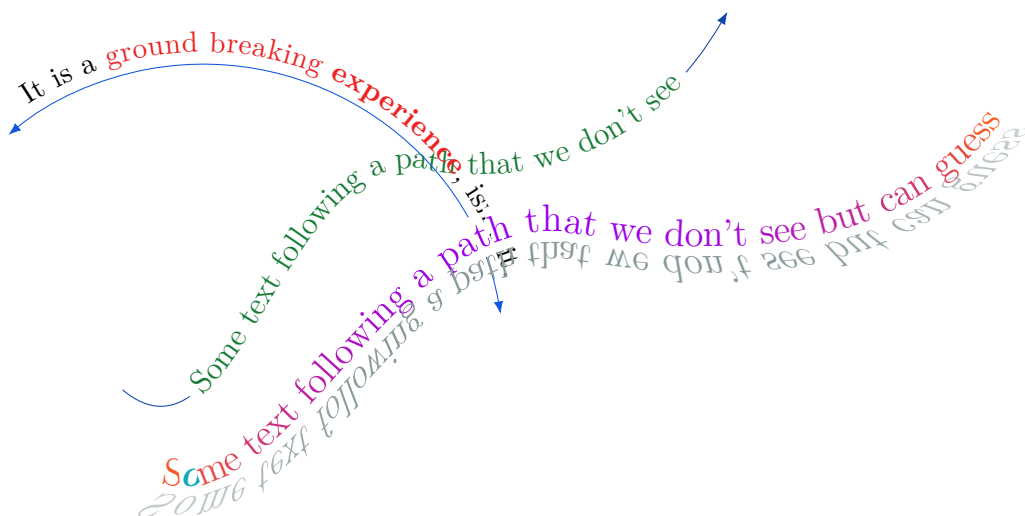
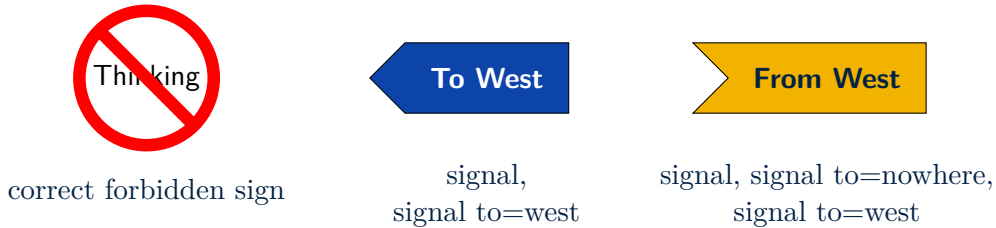


Figure 27: Using `text along path` and `text effects along path`.

10. Shapes

For the shapes below, we need `\usetikzlibrary{shapes.symbols}`.



The code for the second and third nodes are below; note the need to invoke `signal to=nowhere` to achieve the third one.

```
\path (0, 0) node[draw, fill=DarkB, text=W,
signal, signal to=west, inner sep=2ex] {To West};
\path (5, 0)
node[draw, fill=Gold, text=BK,
signal, signal to=nowhere, signal from=west, inner sep=2ex] {From West};
```

Next, we consider closely the shape `signal to`; we increase the `inner sep` dimension—an additional (invisible) separation space of added inside the shape, between the text and the shape’s background path—to better see the point. See Fig. 28.

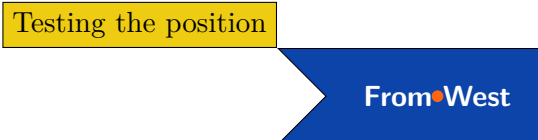


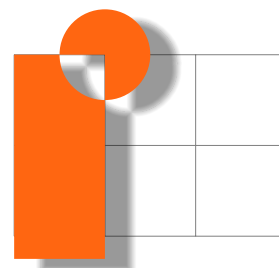
Figure 28: In the case of `signal` shapes, the center is considered with respect to the initial form (the orange circle), not to the border; by comparison, note the position of the node “Testing the position” set as `above left` with respect to the shape node.

11. Other path actions

11.1. Pre and post actions

```
\fill[preaction={%
fill=BK, opacity=.4, transform canvas={xshift=10pt, yshift=-5pt}
}, fill=0, even odd rule] (0, -.25) rectangle (1, 2) (1, 2) circle (.5);
```

We consider the same figure but construct the cast shadow differently. More precisely, we use the library `shadows.blur`. The `preaction` is not needed; it is contained in `blur shadow=...`



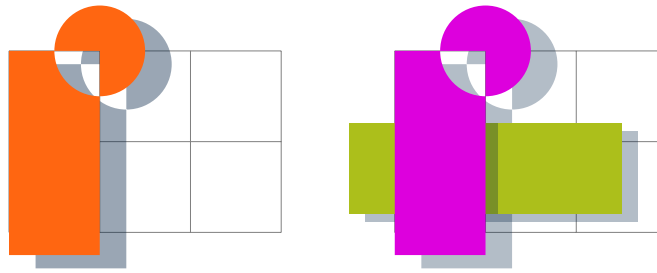
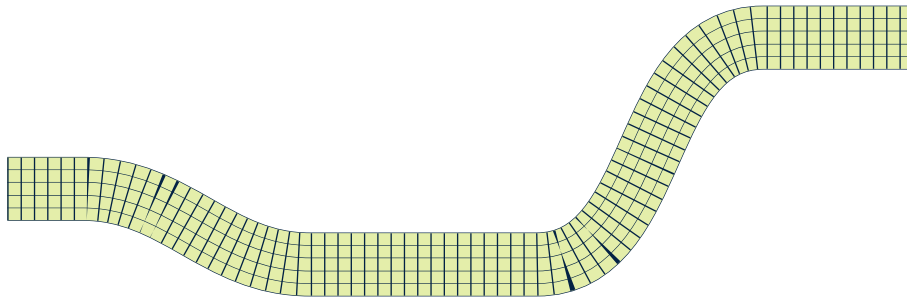


Figure 29: Using `postaction` and `even odd rule` for graphical effects.

A DIFFERENT EXAMPLE—MULTIPLE POSTACTIONS. In the figure below, the construction uses multiple postactions which create the horizontal lines as well as the vertical ones; the letters are the outcome of the last postaction,

```
postaction={draw=BK, line width=4.98ex, dash pattern=on .5 off 4.5}.
```



EXAMPLE—APPROACHING THE COLOR VARIATIONS OF A PARCHMENT. Starting with the left image in Fig. 30, we try to reproduce the color variations, margin, and texture as the background and border of a node.

- We use the library `shadings`; there are two shadings (a vertical—horizontal stripes—and a radial one) that are created. The former with opacity 0.9, the latter 0.7. See below the definition of the radial one. They are applied through two `postaction` commands.
- The drop shadow is created with the `blur shadow=...` command as in the first example of this subsection.
- The border is obtained through the random step decoration, see Fig. 26.
- I don't know how to handle the texture. One of the main reasons is the fact that this is the background of a node.

```
\pgfdeclareradialshading{rparchment}{\pgfpoint{0cm}{0cm}}{%
  color(0bp)=(pylight);
  color(13bp)=(pylight);
  color(20bp)=(py);
  color(40bp)=(pydark);
  color(60bp)=(pydark!50!DarkR);
  color(100bp)=(RK)%
}
```

Below are some examples of using the “parchment node”. The second uses the *rotunda* fonts.

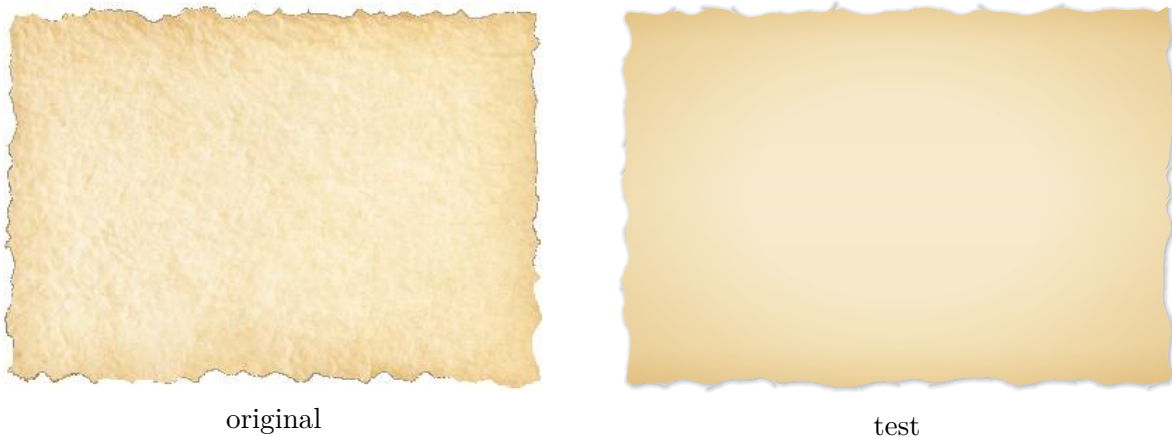


Figure 30: Reproducing an old parchment. Well, not really, since it resembles more to a biscuit. Some irregularities would be most needed.

The brown fox jumps over the lazy dog following the joyful rabbit. "Rabbit wait!"

The brown fox jumps over the lazy dog following the joyful rabbit.

The brown fox jumps over the lazy dog following the joyful rabbit.

11.2. *let ... in*

We give two examples; this action was already used on page 8 to construct the angle `\n1..`

EXAMPLE (EUCLID). How to construct an equilateral triangle on a given finite straight line, i.e. a segment.

If the segment is determined by the points A and B , the idea is to construct the two circles centered at A and B and of radius AB . Each point of intersection of the circles gives rise to an equilateral triangle together with A and B . The only remaining problem is to access the x - and y -coordinate of the vector \overrightarrow{AB} . For this, we need a new concept: the `let ... in` operation. A `let` operation can be given anywhere on a path where a normal path operation like a `line-to` or a `move-to` is expected. The effect of a `let` operation is to evaluate some coordinates and numbers and to assign the results to special macros. These macros make it easy to access the x - and y -coordinates of the points and/or vectors (i.e. coordinates):

- `\p<digit> = (a, b)` creates the point (a, b) . Note that, say `\p1`, is the string formed with the coordinates of the point (a, b) .

- `(\p<digit>)` yields the resulting point
- `\x<digit>` or `\y<digit>` yields the x - or y -coordinate of the resulting point
- `\n<digit> = ...` or `\n{symbols} = ...` creates a number.

Given $a, b \in \mathbb{R}$, `veclen(a,b)` inside a mathematical expression, i.e. **curly brackets**, yields $\sqrt{a^2 + b^2}$.

The result of the code below is used in Fig. 31 to obtain the vertex C of the equilateral triangle.

```
\path ($(B)-(A)$) coordinate (v);
\draw let
\p1 = (v),
\n1 = {veclen(\x1, \y1)}
in [name path=circleA] (A) circle (\n1);
\draw let
\p1 = (v),
\n1 = {veclen(\x1, \y1)}
in [name path=circleB] (B) circle (\n1);
\path[name intersections={of=circleA and circleB, by={C, D}}];
\draw[thick, fill=orange] (A) -- (B) -- (C) -- cycle;
```

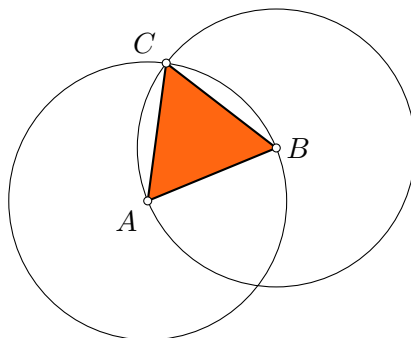


Figure 31: An equilateral triangle constructed on the segment $[AB]$

Remark. A different way of solving the problem is to compute the radius only once in a `tikzmath` declaration.

```
\begin{tikzpicture}
\path (0, 0) coordinate (A) (1.7, .7) coordinate (B);
\tikzmath{%
coordinate \v;
real \r;
\v = ($(B)-(A)$);
\r = veclen(\vx, \vy);
}
\draw[name path=circleA] (A) circle (\r pt);
\draw[name path=circleB] (B) circle (\r pt);
\path[name intersections={of=circleA and circleB, by={C, D}}];
```

```

\draw[thick, fill=0] (A) -- (B) -- (C) -- cycle;
\foreach \P/\pos in {A/below left, B/right, C/above left}{%
  \draw[fill=white] (\P) circle (1.5pt) node[\pos] {$\P$};
}
\end{tikzpicture}

```

Note the need of specifying the units in `\draw (A) circle (\r pt)`; since, even if the macros `\vx` and `\vy` return coordinates converted in points, the use of `real \r` yields a dimensionless number which is big!

For example,

- **28.45274pt** is the result of

```

\tikzmath{
  coordinate \v;
  \v=(1, 0);
}
\path node[fill=0, text=W, scale=.9] {\vx};

```

- **28.45274** is the result of

```

\tikzmath{
  coordinate \v;
  real \tmp;
  \v=(1, 0);
  \tmp = \vx;
}
\path node[fill=0, text=W, scale=.9] {\tmp};

```

BISECTING AN ANGLE. Suppose that A , B and C are known and that we want to construct the bisector of the angle \widehat{BAC} . Some solutions are given by the use of `let ... in` action.

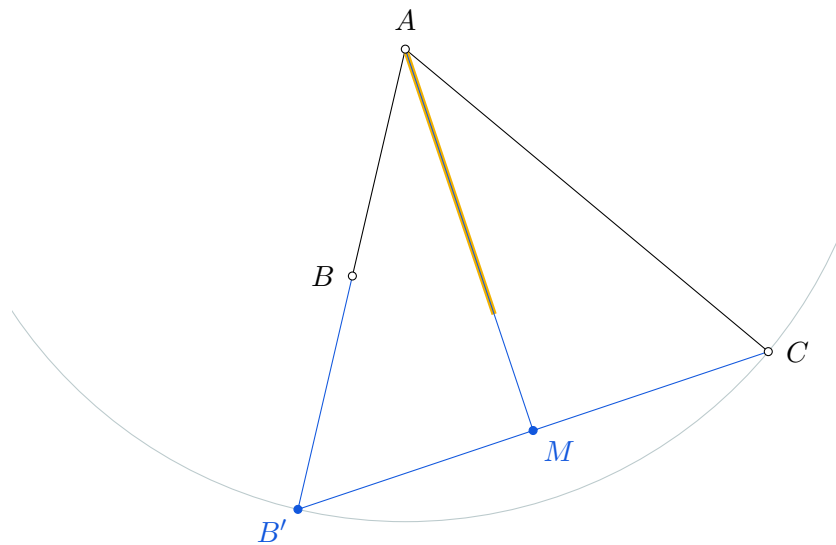
For example, it is possible to construct the measure of the angle and then to draw the bisector based on this measure. Of course, a decision must be taken concerning the sign of the angle of rotation in the direct similitude considered; see the blue (B) ray.

The red elements in the figure below are constructed as follows: the point B' is constructed on the ray $[AB)$ such that $AB' = AC$, and then the bisector is drawn using the middle point of $[B'C]$.

```

(B) \draw[B, thick] let
      \p1 = ($(B)-(A)$), \p2 = ($(C)-(A)$),
      \n1 = {veclen(\x1, \y1)}, \n2 = {veclen(\x2, \y2)},
      \n{angle} = {acos((\x1*\x2+\y1*\y2)/(\n1*\n2))/2}
    in (A) -- ($(A)!1.2!\n{angle}:(B)$);

```

Figure 32: Bisecting the angle \widehat{BAC} .

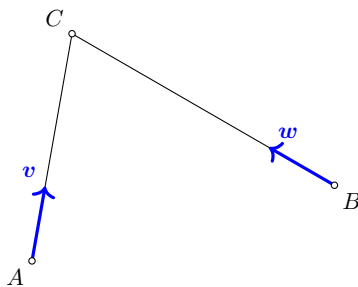
```
(R) \draw[R, thick] let
      \p1 = ($(B)-(A)$), \p2 = ($(C)-(A)$),
      \n1 = {veclen(\x1, \y1)}, \n2 = {veclen(\x2, \y2)},
      \n3 = {\n2/\n1},
      \p3 = ($(A)!\strippt(\n3)!(B)$)
    in (A) -- ($(\p3)!.5!(C)$);
```

In the preceding code, the length `\n3` (expressed in pt) is transformed into a dimensionless number through the command `\strippt` which is defined in the preamble.

INTERSECTION POINT OF TWO LINES. outside let

```
\coordinate (D) at (intersection of leftD--rightD and C--c); ?
```

This is the fourth way an intersection point can be found. We were referring to it on page 13. One advantage is that the lines are defined mathematically, i.e. as drawn elements on the canvas they may not have a point in common.



```
\p5 = (intersection of \p1--\p2 and \p3--\p4)
```

is used inside the `let ... in` action to produce the intersection point.

11.3. `\foreach`

A pattern created by rotating and scaling a rectangle multiple times. Note the use of color fading obtained by superposition only. Moreover, the `\tikzmath` environment is embedded among the other `TikZ` commands.

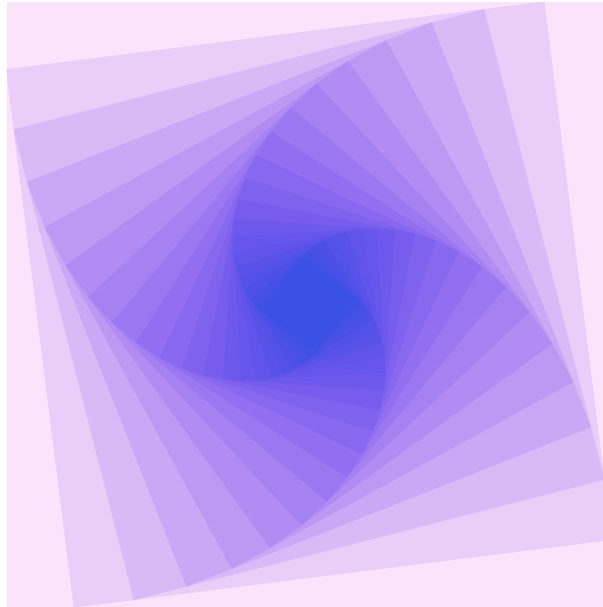


Figure 33: Fading rectangle with with 20 steps—it uses `tikzmath`

```

\tikzmath{
  coordinate \A, \B, \C, \D, \M, \N, \P, \Q;
  \A = (0, 0); \B = (10, 0); \C = (10, 10); \D = (0, 10);
  {
    \fill[VB, opacity=\oIndex] (\A) -- (\B) -- (\C) -- (\D) -- cycle;
  };
  for \i in {1,2,...,\nbPoints}{%
    \M = (\A)!\oIndex!(\B);
    \N = (\B)!\oIndex!(\C);
    \P = (\C)!\oIndex!(\D);
    \Q = (\D)!\oIndex!(\A);
    \A = (\M); \B = (\N); \C = (\P); \D = (\Q);
    {
      \fill[fill=B, opacity=\oIndex]
        (\A) -- (\B) -- (\C) -- (\D) -- cycle;
    };
  };
}

```

12. Using mathematics and functions

THE FORD CIRCLES. Given an irreducible fraction $\frac{p}{q}$, the circle at $(\frac{p}{q}, \frac{1}{2q^2})$ with radius $r = \frac{1}{2q^2}$ is tangent to (Ox) . If all such circles are drawn, they do not intersect but are at most tangent.

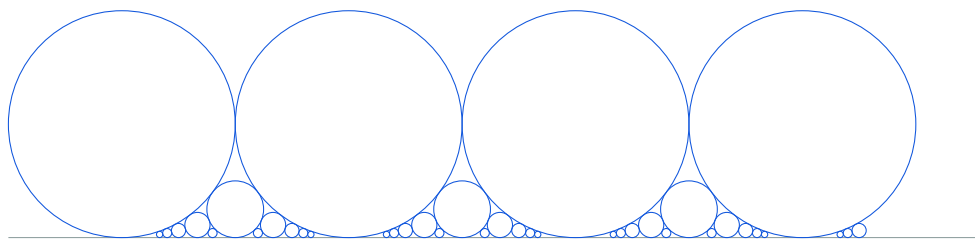


Figure 34: Giving the interval $[0, b]$ and the integer n , the Ford circles corresponding to $\frac{p}{q} \in [a, b]$ such that $0 < q \leq n$ are drawn. We have $b = 3.3$ and $n = 6$.

The outer loop introduces the upper bound of the inner loop through `evaluate`:

```
\foreach \q [evaluate=\q as \pend using {\b*\q}] in {1, ..., \n}{%
  \foreach \p in {0, ..., \pend}{%
```

A NOT SO RANDOM PATH. The construction is based on a loop and the use of the function `rand` which produces a random number in $[0, 1]$.



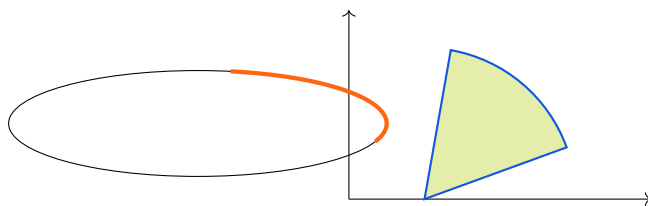
Figure 35: A random path with control points chosen not so randomly composed of 45 steps. It uses the modifier of a coordinate (a vector here) (`[scale=-.7]old velocity`).

Remark. See page 928 of *TikZ&PGF manual for version 3.0* for the list of existing functions, section **89.3 Syntax for Mathematical Expressions: Functions**.

13. Defining keys

See Fig. 5 where the key `show control point` has been defined. Below, we introduce some other user defined keys.

Example (variant for drawing an arc). This is not really a key, but rather a \TeX command.

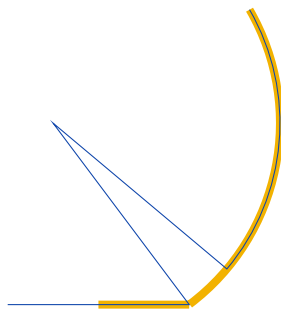


```
\def\arc at#1(#2)#3(#4:#5:#6){%(center) (initial angle:final angle:radius)
  ($(#2) +(#4: #6)$) arc (#4: #5: #6)
}
```

Note the use of $(\$A+B\$)$ instead of $(A)+(B)$, say; there is no unintentional jump to the point (B) in the drawing process.

The same effect is obtained with the following two keys—more *TikZ* oriented:

- `c arc/.style args={#1:#2:#3}{insert path={++(#1:#3) arc (#1:#2:#3)}}`
- `c-arc/.style args={#1:#2:#3}{insert path={--++(#1:#3) arc (#1:#2:#3)}}`



Example (stretched segment). The key `longer` which takes two arguments, allows to draw a segment whose extremities are prolonged with lengths corresponding to the two arguments and measured in centimeters.

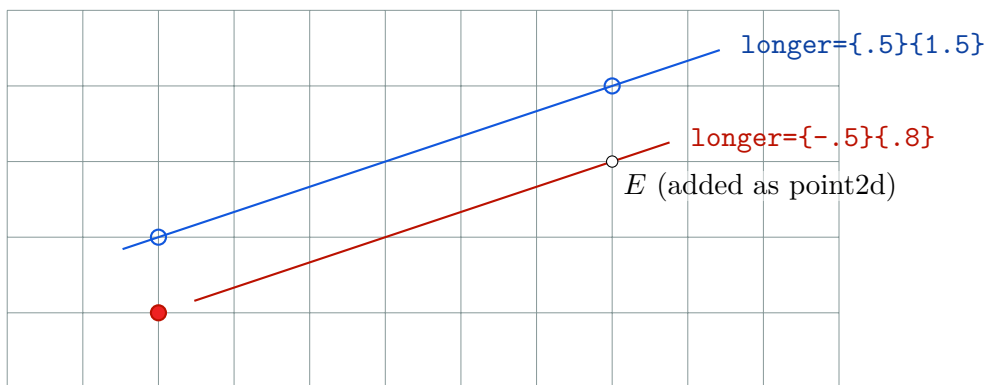


Figure 36: Using the key `longer`. Note that for the second segment, the first argument is negative..

The definition of the key uses the key `shorten <` and `shorten >`.

```
longer/.style 2 args={
  shorten <= {-#1 cm}, shorten >= {-#2 cm}
}
```

Example (longitude and latitude). The key `view` sets the longitude and latitude corresponding to the observer's point of view and defines the unitary vector pointing towards the point of view.

1. The point of view is defined by the unitary vector \vec{w} that points towards the observer. Its components are `\tox`, `\toy`, and `\toz` equal to

$$\begin{aligned}\backslash\text{tox} &= x_{\vec{w}} = \sin \mu \cos \lambda \\ \backslash\text{toy} &= y_{\vec{w}} = \sin \lambda \\ \backslash\text{toz} &= z_{\vec{w}} = \cos \mu \cos \lambda.\end{aligned}$$

The angles μ and λ represent the longitude and the latitude, respectively—the arguments of `view`.

2. The screen—plane on which the image is drawn or plane of the paper—is the plane passing through the origin and orthogonal to \vec{w} . The orthonormal basis that induces the coordinate system of the screen is $(\vec{u}, \vec{v}, \vec{w})$, where

$$\vec{u} = (\cos \mu, 0, -\sin \mu), \quad \vec{v} = (-\sin \mu \sin \lambda, \cos \lambda, -\cos \mu \sin \lambda).$$

Note that the initial coordinate system is $Oxyz$, such that, when $\mu = \lambda = 0$, Oz is horizontal and points towards the observer and Ox is horizontal and points to the right (seen by the observer). Consequently, \vec{u} is parallel to Oxz and so, $\lambda \neq 90^\circ \pmod{180^\circ}$ for a 3D image—otherwise we arrive at a 2D image representing the Oxz plane.

The points $(1, 0, 0)$, $(0, 1, 0)$, and $(0, 0, 1)$ projects onto points described in the global options of the drawing by

$$\begin{aligned}x &= (\langle (1, 0, 0), \vec{u} \rangle, \langle (1, 0, 0), \vec{v} \rangle) \\ y &= (\langle (0, 1, 0), \vec{u} \rangle, \langle (0, 1, 0), \vec{v} \rangle) \\ z &= (\langle (0, 0, 1), \vec{u} \rangle, \langle (0, 0, 1), \vec{v} \rangle)\end{aligned}$$

Due to the way the `pgf`-coordinate system function, z **must be modified first**.

```
\tikzset{%
  view/.style 2 args={%
    z={({-sin(#1)}, {-cos(#1)*sin(#2)}}),
    x={({cos(#1)}, {-sin(#1)*sin(#2)}}),
    y={(0, {cos(#2)}}),
    evaluate={%
      \tox={sin(#1)*cos(#2)};
      \toy={sin(#2)};
      \toz={cos(#1)*cos(#2)};
    }
  }
}
```

We apply it to draw the cubes below.

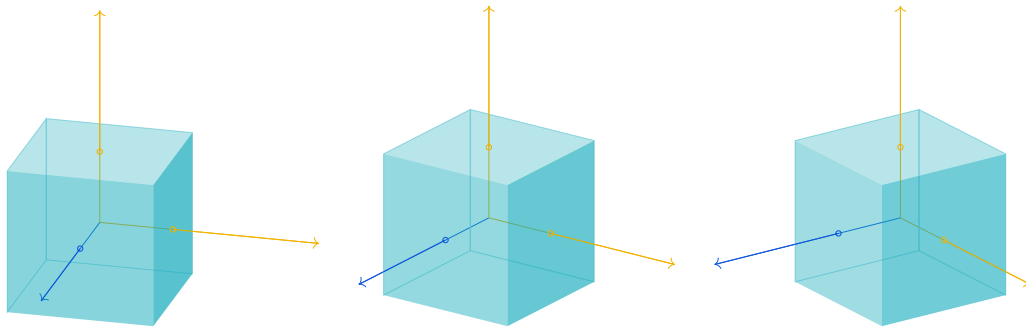
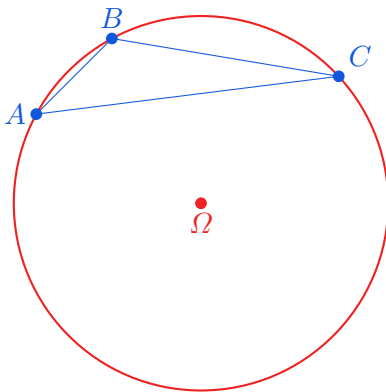


Figure 37: Using the key `view`. The latitude equals 21 and the longitude equals 15, 35, and 55, respectively. The axis Oz is represented in blue.

Example (circle through three points). This key draws the circumscribed circle to a given triangle, i.e. the unique circle through three non aligned points. As a side effect, it stores the center as (the point) `ccenter`.



The circle is firstly drawn, the center being placed afterward. The construction is based on the fact that the center is the intersection of two edge bisectors.

A. Various examples

A.1. The construction of an equilateral triangle when the circumcircle and a vertex are given

The construction is based on the following fact. Consider the configuration in Fig. 39. The two cercles centered in at A and J have the same radius and intersect at E and F —the triangles $[AJE]$ and $[AJF]$ are equilateral. The arbitrary point I belongs to the circle of center A . If the points M and N bisects the segments $[EI]$ and $[FI]$ respectively, then the measure of the angle MAN is either 60° or 120° —since the points $A, M, N,$ and I determine an inscribed quadrilateral.

A.2. Dandelin's spheres for a hyperbola defined as a conic section

The point of view is given by the unitary vector pointing toward the observer defined by the angles φ (longitude) and θ (latitude) in the coordinate system $Ozxy$. Almost all computations are performed in the coordinate plane $z = 0$.

- The cone is defined by the equation $y^2 = b^2(x^2 + z^2)$, $b > 0$.

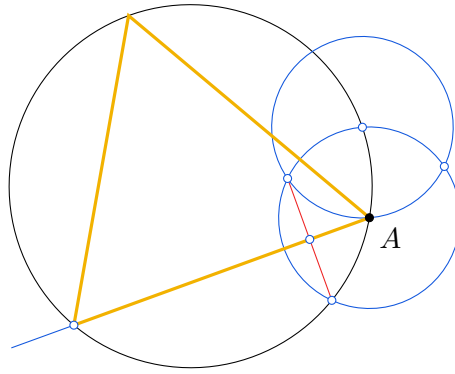


Figure 38: The construction of an equilateral triangle when the circumcircle and a vertex are given (drawn in black). It is based on the geometrical property described in Fig. 39.

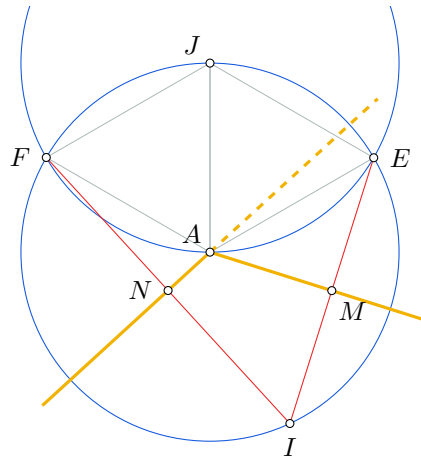


Figure 39: The golden lines are perpendicular on (EI) and (FI) respectively; they subtend an angle of 60° or 120° .

- The plane of the hyperbola is defined by $ax + y = c$ with $a > b$ and $c > 0$. The vertices of the hyperbola are defined in the plane $z = 0$ by the intersections

$$ax + y = c \quad \text{and} \quad y = bx, \quad \text{i.e.} \quad V_+ = \left(\frac{c}{a+b}, \frac{bc}{a+b} \right)$$

and

$$ax + y = c \quad \text{and} \quad y = -bx, \quad \text{i.e.} \quad V_- = \left(\frac{c}{a-b}, \frac{bc}{a-b} \right).$$

- The Dandelin spheres are centered at Ω_\pm and have radii r_\pm . Solving a distance problem, we obtain

$$\Omega_\pm = \left(0, \pm \frac{c\sqrt{b^2+1}}{\sqrt{a^2+1} \pm \sqrt{b^2+1}} \right) \quad \text{and} \quad r_\pm = \frac{c}{\sqrt{a^2+1} \pm \sqrt{b^2+1}}.$$

- In the plane $ax + y = c$, the hyperbola is defined by the equation

$$\frac{\left(y + \frac{b^2c}{a^2 - b^2}\right)^2}{\left(\frac{abc}{a^2 - b^2}\right)^2} - \frac{z^2}{\left(\frac{c}{\sqrt{a^2 - b^2}}\right)^2} = 1.$$

(So we can use parametrizations in cosh and sinh to represent it graphically.)

- To determine the tangency circles of the spheres and the cone, two points are sufficient. They are granted by the distance problem solved previously. The same reasoning gives the hyperbola's foci. Setting

$$h_{\pm} = \frac{c\sqrt{b^2 + 1}}{\sqrt{a^2 + 1} \pm \sqrt{b^2 + 1}} > 0,$$

we have

$$Q_{\pm} = \left(\frac{bh_{\pm}}{b^2 + 1}, \pm \frac{b^2h_{\pm}}{b^2 + 1}\right) \quad \text{and} \quad F_{\pm} = \left(\frac{a(c \mp h_{\pm})}{a^2 + 1}, \frac{c \pm a^2h_{\pm}}{a^2 + 1}\right).$$

Remark. The code is commented (see below in the file); it uses constants and functions defined in a `tikzmath` environment, and the 3D view given by latitude and longitude angles introduced with respect to the $Ozxy$ coordinate system. The image in Fig. 40 is included from a `standalone` file.

A.3. Torus in the light

Using the same elements introduced to draw Dandelin's spheres, and adding the shade (computed by projection), we get the torus below. The coordinate axes are shadowless.

A.4. Penrose triangle

The Penrose triangle is an *optical illusion* consisting of an object which can be depicted in a perspective drawing, but cannot exist as a solid object. It was first created by the Swedish artist Oscar Reutersvärd in 1934.

There also exist three-dimensional solid shapes each of which, when viewed from a certain angle, appears the same as the 2-dimensional depiction of the Penrose triangle (see Fig. 42). The term *Penrose triangle* can refer to the 2-dimensional depiction or the impossible object itself.

If a line is traced around the Penrose triangle, a 4-loop Möbius strip is formed.

B. Drawing a path of varying width

This is based on Alain Matthes's idea; see his answer at

<https://tex.stackexchange.com/questions/14283/stroke-with-variable-thickness>

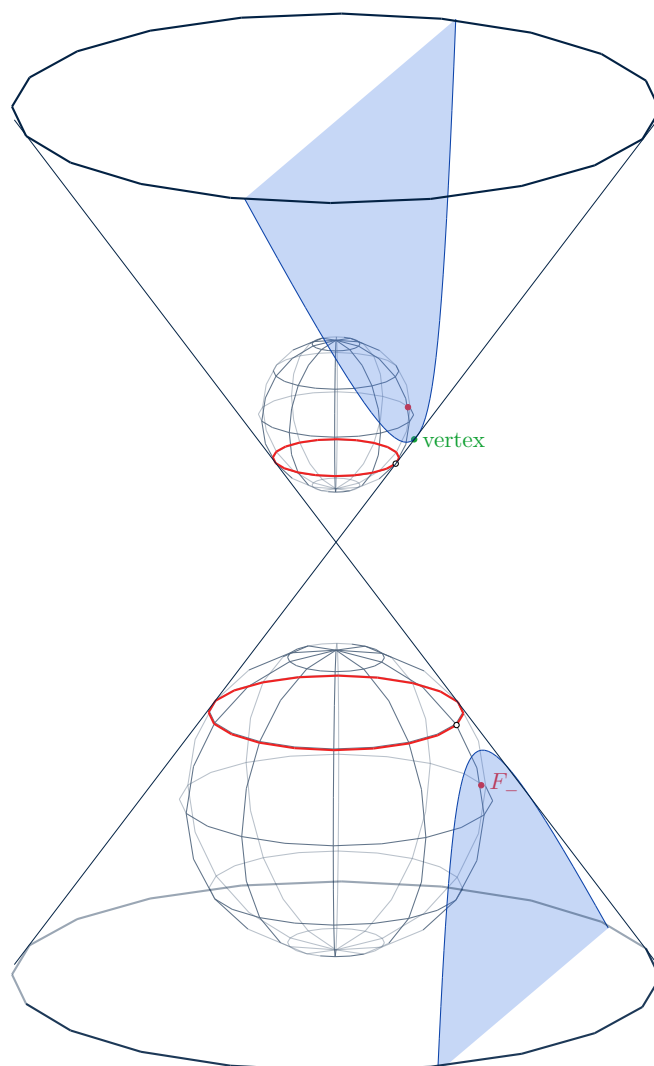


Figure 40: Dandelin's spheres of a hyperbola

```

\pgfkeys{/pgf/decoration/.cd,
  width factor/.store in =\wfactor,
  start color/.store in =\startcolor,
  end color/.store in =\endcolor
}

```

```

\pgfdeclaredecoration{width and color change}{initial}{%
  \state{initial}[width=0pt, next state=line, persistent precomputation={%
    \pgfmathdivide{50}{\pgfdecoratedpathlength}%
    \let\increment=\pgfmathresult%
    \def\x{0}%
  ]}{}
  \state{line}[width=.5pt, persistent postcomputation={%
    \pgfmathadd@\x{\increment}%
    \let\x=\pgfmathresult%
  ]}{}
}

```

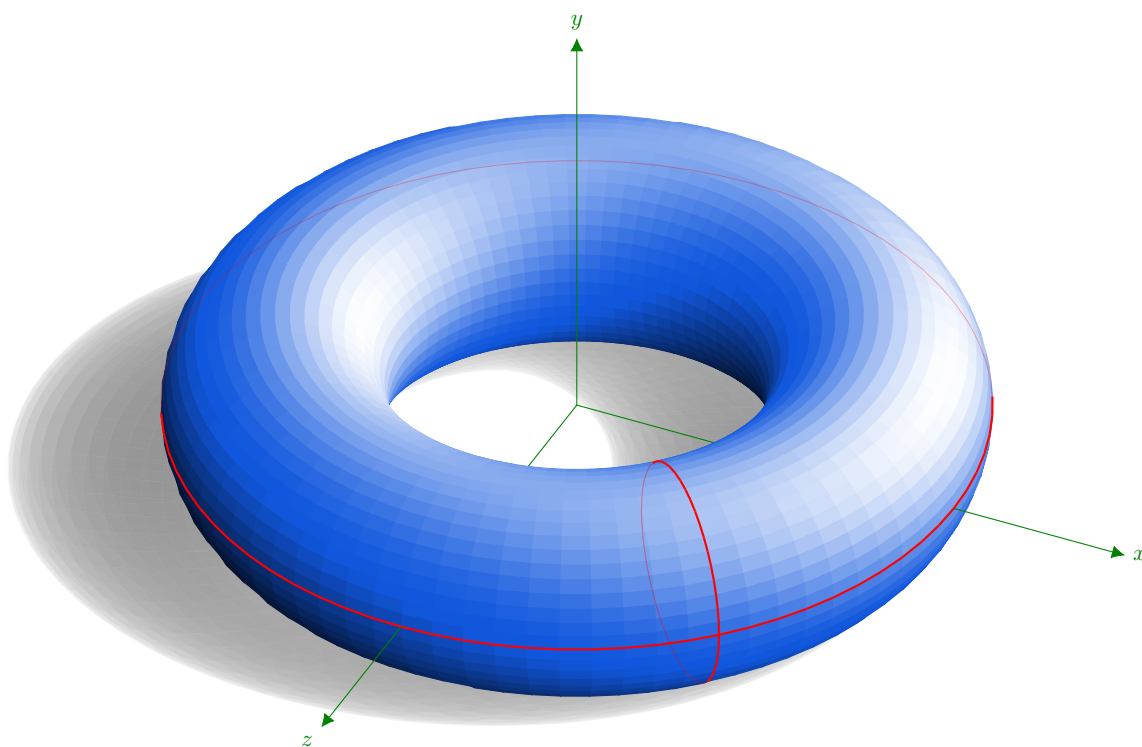


Figure 41: Torus in the light



Figure 42: The construction of the Penrose triangle: the triangle is constructed by performing on the turquoise element (the right drawing), two rotations of 60 and 120 degrees respectively, centered at O . The red triangle can be seen as the building brick of the whole figure.

```

}}{%
\pgfsetlinewidth{\wfactor*\x/50*0.075pt+\pgflinewidth}%
\pgfsetarrows{-}%
\pgfpathmoveto{\pgfpointorigin}%
\pgfpathlineto{\pgfqpoint{.75pt}{0pt}}%
\pgfsetstrokecolor{\endcolor!\x!\startcolor}%

```

```
\pgfusepath{stroke}%  
}  
\state{final}{%  
  \pgfsetlinewidth{\pgflinewidth}%  
  \pgfpathmoveto{\pgfpointorigin}%  
  \color{\endcolor!\x!\startcolor}%  
  \pgfusepath{stroke}%  
}  
}
```

