CeTZ

ein Typst Zeichenpacket

Johannes Wolf
fenjalien

Version 0.2.0

1 Introduction

This package provides a way to draw stuff using a similar API to Processing but with relative coordinates and anchors from TikZ. You also won't have to worry about accidentally drawing over other content as the canvas will automatically resize. And remember: up is positive!

The name CeTZ is a recursive acronym for "CeTZ, ein Typst Zeichenpacket" (german for "CeTZ, a Typst drawing package") and is pronounced like the word "Cats".

2 Usage

This is the minimal starting point:

```
#import "@preview/cetz:0.2.0"
#cetz.canvas({
  import cetz.draw: *
  ...
})
```

Note that draw functions are imported inside the scope of the `canvas` block. This is recommended as draw functions override Typst's functions such as `line`.

2.1 CeTZ Unique Argument Types

Many CeTZ functions expect data in certain formats which we will call types. Note that these are actually made up of Typst primitives.

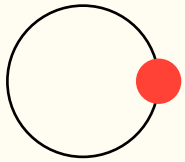
`coordinate` Any coordinate system. See coordinate-systems.

`number` Any of `float`, `integer` or `length`

`style` Named arguments (or a dictionary if used for a single argument) of style key-values

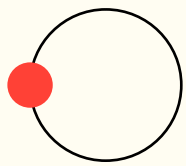
2.2 Anchors

Anchors are named positions relative to named elements. To use an anchor of an element, you must give the element a name using the `name` argument. All elements with the `name` argument allow anchors.



```
// Name the circle
circle((0,0), name: "circle")

// Draw a smaller red circle at "circle"'s east anchor
fill(red)
stroke(none)
circle("circle.east", radius: 0.3)
```

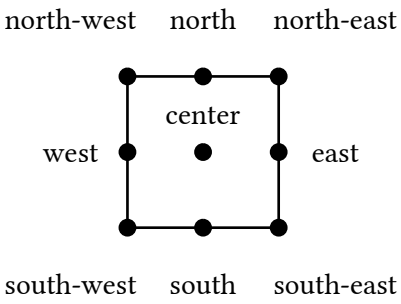
Elements can be placed relative to their own anchors if they have an argument called `anchor`:



```
// An element does not have to be named
// in order to use its own anchors.
circle((0,0), anchor: "west")

// Draw a smaller red circle at the origin
fill(red)
stroke(none)
circle((0,0), radius: 0.3)
```

2.2.1 Compass Anchors

Some elements support compass anchors. TODO



3 Draw Function Reference

3.1 Canvas

```
canvas(background: none, length: 1cm, debug: false, body)
```

background `color` (default: none)

A color to be used for the background of the canvas.

length `length` (default: 1cm)

Used to specify what 1 coordinate unit is.

debug `bool` (default: false)

Shows the bounding boxes of each element when `true`.

body

A code block in which functions from `draw.typ` have been called.

3.2 Styling

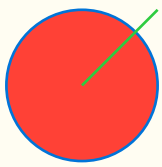
You can style draw elements by passing the relevant named arguments to their draw functions. All elements that draw something have stroke and fill styling unless said otherwise.

fill `color` or `none` Default: `none`

How to fill the drawn element.

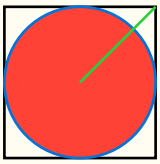
stroke `none` or `auto` or `length` or `color` or `dictionary` or `stroke` Default: `1pt + luma(0%)`

How to stroke the border or the path of the draw element. See Typst's line documentation for more details: https://typst.app/docs/reference/visualize/line/#parameters-stroke



```
// Draws a red circle with a blue border
circle((0, 0), fill: red, stroke: blue)
// Draws a green line
line((0, 0), (1, 1), stroke: green)
```

Instead of having to specify the same styling for each time you want to draw an element, you can use the `set-style` function to change the style for all elements after it. You can still pass styling to a draw function to override what has been set with `set-style`. You can also use the `fill()` and `stroke()` functions as a shorthand to set the fill and stroke respectively.

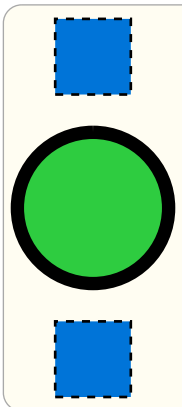


```
// Draws an empty square with a black border
rect((-1, -1), (1, 1))

// Sets the global style to have a fill of red and a stroke of blue
set-style(stroke: blue, fill: red)
circle((0,0))

// Draws a green line despite the global stroke is blue
line((), (1,1), stroke: green)
```

When using a dictionary for a style, it is important to note that they update each other instead of overriding the entire option like a non-dictionary value would do. For example, if the stroke is set to `(paint: red, thickness: 5pt)` and you pass `(paint: blue)`, the stroke would become `(paint: blue, thickness: 5pt)`.

```
// Sets the stroke to red with a thickness of 5pt
set-style(stroke: (paint: red, thickness: 5pt))
// Draws a line with the global stroke
line((0,0), (1,0))
// Draws a blue line with a thickness of 5pt because dictionaries update the style
line((0,0), (1,1), stroke: (paint: blue))
// Draws a yellow line with a thickness of 1pt because other values override the style
line((0,0), (0,1), stroke: yellow)
```

You can also specify styling for each type of element. Note that dictionary values will still update with its global value, the full hierarchy is `function > element type > global`. When the value of a style is `auto`, it will become exactly its parent style.



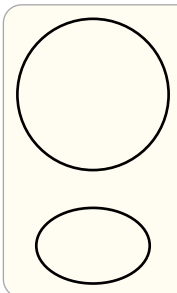
```
set-style(
  // Global fill and stroke
  fill: green,
  stroke: (thickness: 5pt),
  // Stroke and fill for only rectangles
  rect: (stroke: (dash: "dashed"), fill: blue),
)
rect((0,0), (1,1))
circle((0.5, -1.5))
rect((0,-3), (1, -4), stroke: (thickness: 1pt))
```

```
// Its a nice drawing okay
set-style(
  rect: (
    fill: red,
    stroke: none
  ),
  line: (
    fill: blue,
    stroke: (dash: "dashed")
  ),
)
rect((0,0), (1,1))

line((0, -1.5), (0.5, -0.5), (1, -1.5), close: true)

circle((0.5, -2.5), radius: 0.5, fill: green)
```

3.3 Elements

3.3.1 circle

Draws a circle or ellipse.



```
circle((0,0))
// Draws an ellipse
circle((0,-2), radius: (0.75, 0.5))
```

Parameters

```
circle(
  position: coordinate,
  name: none string,
  anchor: none string,
  ..style: style
)
```

position `coordinate`

The position to place the circle on.

Style Root `circle`

Style Keys

radius `number` or `array` Default: `1`

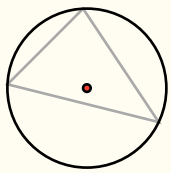
A number that defines the size of the circle's radius. Can also be set to a tuple of two numbers to define the radii of an ellipse, the first number is the `x` radius and the second is the `y` radius.

Anchors

Supports compass anchors. The "center" anchor is the default.

3.3.2 circle-through

Draws a circle through three coordinates



```
let (a, b, c) = ((0,0), (2,-.5), (1,1))
line(a, b, c, close: true, stroke: gray)
circle-through(a, b, c, name: "c")
circle("c.center", radius: .05, fill: red)
```

Parameters

```
circle-through(
  a: coordinate,
  b: coordinate,
  c: coordinate,
  name: none string,
  anchor: none string,
  ..style: style
)
```

a `coordinate`
Coordinate a

b `coordinate`
Coordinate b

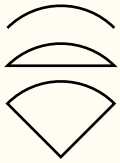
c `coordinate`
Coordinate c

Style Root `circle`

Anchors
Supports the same anchors as `circle` as well as:
a Coordinate a
b Coordinate b
c Coordinate c

3.3.3 arc

Draws a circular segment.



```
arc((0,0), start: 45deg, stop: 135deg)
arc((0,-0.5), start: 45deg, delta: 90deg, mode: "CLOSE")
arc((0,-1), stop: 135deg, delta: 90deg, mode: "PIE")
```

Note that two of the three angle arugments (`start`, `stop` and `delta`) must be set.

Parameters

```
arc(
  position: coordinate,
  start: auto angle,
  stop: auto angle,
  delta: auto angle,
  name: none string,
  anchor: none string,
  ..style: style
)
```

position `coordinate`
Position to place the arc at.

start `auto` or `angle` Default: `"auto"`
The angle at which the arc should start. Remember that `0deg` points directly towards the right and `90deg` points up.

stop `auto` or `angle` Default: `"auto"`
The angle at which the arc should stop.

delta `auto` or `angle` Default: `"auto"`
The change in angle away start or stop.

Style Root `arc`
Style Keys

radius `number` or `array` Default: `1`

The radius of the arc. An eliptical arc can be created by passing a tuple of numbers where the first element is the x radius and the second element is the y radius.

mode `string` Default: `"OPEN"`

The options are: “OPEN” no additional lines are drawn so just the arc is shown; “CLOSE” a line is drawn from the start to the end of the arc creating a circular segment; “PIE” lines are drawn from the start and end of the arc to the origin creating a circular sector.

Anchors

Supports compass anchors when `mode` is “PIE”

center The center of the arc, this is the default anchor.

arc-center The midpoint of the arc’s curve.

chord-center Center of chord of the arc drawn between the start and end point.

origin The origin of the arc’s circle.

arc-start The position at which the arc’s curve starts.

arc-end The position of the arc’s curve end.

3.3.4 mark

Draws a single mark pointing at a target coordinate

```
mark((0,0), (1,0), symbol: ">", fill: black)
mark((0,0), (1,1), symbol: ">", scale: 3, fill: black)
```

Or as part of a path based element that supports the `mark` style key:

```
rotate(-90deg)
set-style(mark: (fill: black))
line((1, -1), (1, 9), stroke: (paint: gray, dash: "dotted"))
line((0, 8), (rel: (1, 0)), mark: (end: "left-harpoon"))
line((0, 7), (rel: (1, 0)), mark: (end: "right-harpoon"))
line((0, 6), (rel: (1, 0)), mark: (end: "<>"))
line((0, 5), (rel: (1, 0)), mark: (end: "o"))
line((0, 4), (rel: (1, 0)), mark: (end: "|"))
line((0, 3), (rel: (1, 0)), mark: (end: "<"))
line((0, 2), (rel: (1, 0)), mark: (end: ">"))
set-style(mark: (fill: none))
line((0, 1), (rel: (1, 0)), mark: (end: "<"))
line((0, 0), (rel: (1, 0)), mark: (end: ">"))
```

Parameters

```
mark(
  from: coordinate,
  to: coordinate,
  ..style: style
)
```

from `coordinate`
The position to place the mark.

to `coordinate`
The position the mark should point towards.

Style Root `mark`
Style Keys

symbol `string` Default: `">"`
The type of mark to draw when using the `mark` function.

start `string` or `none` or `array` Default: `none`
The type of mark to draw at the start of a path.

end `string` or `none` or `array` Default: `none`
The type of mark to draw at the end of a path.

length `number` Default: `0.2`
The length of the mark along its direction it is pointing.

width `number` Default: `0.15`
The width of the mark along the normal of its direction.

inset `number` Default: `0.05`
The distance by which something inside the arrow tip is set inwards.

scale `float` Default: `1`
A factor that is applied to the mark's length, width and inset.

sep `number` Default: `1`
The distance between multiple marks along their path.

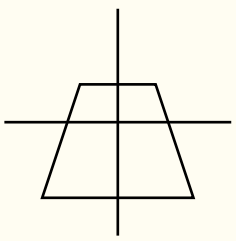
flex `boolean` Default: `true`
Only applicable when marks are used on curves such as bezier and hobby. If true, the mark will point along the secant of the curve. If false, the tangent at the marks tip is used.

position-samples `integer` Default: `30`
Only applicable when marks are used on curves such as bezier and hobby. The maximum number of samples to use for calculating curve positions. A higher number gives better results but may slow down compilation.

Note: The size of the mark depends on its style values, not the distance between `from` and `to`, which only determine its orientation.

3.3.5 line

Draws a line, more than two points can be given to create a line-strip.



```
line((-1.5, 0), (1.5, 0))
line((0, -1.5), (0, 1.5))
line((-1, -1), (-0.5, 0.5), (0.5, 0.5), (1, -1), close: true)
```

Parameters

```
line(
  ..pts-style: coordinates style,
  close: bool,
  name: none string
)
```

..pts-style `coordinates` or `style`

Positional two or more coordinates to draw lines between. Accepts style key-value pairs.

close `bool` Default: `"false"`

If true, the line-strip gets closed to form a polygon

Style Root `line`

Style Keys

Supports marks

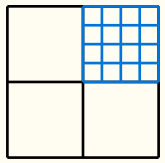
Anchors

start The line's start position

end The line's end position

3.3.6 grid

Draw a grid between two coordinates



```
// Draw a grid
grid((0,0), (2,2))

// Draw a smaller blue grid
grid((1,1), (2,2), stroke: blue, step: .25)
```

Style Root `grid`

Anchors

Supports compass anchors.

Parameters

```
grid(
  from: coordinate,
  to: coordinate,
  step: number,
  name: none string,
  help-lines,
  ..style: style
)
```

from `coordinate`

The top left of the grid

to `coordinate`

The bottom right of the grid

step `number` Default: `"1"`

Grid spacing.

help-lines Default: `"false"`

3.3.7 content

Positions Typst content in the canvas. Note that the content itself is not transformed only its position is.

Hello World!
```
content((0,0), [Hello World!])
```

To put text on a line you can let the function calculate the angle between its position and a second coordinate by passing it to `angle`:


```
line((0, 0), (3, 1), name: "line")
content(
  ("line.start", 0.5, "line.end"),
  angle: "line.end",
  padding: .1,
  anchor: "south",
  [Text on a line]
)
```

Parameters

content(
..args-style: `coordinate` `content` `style`,
angle: `angle` `coordinate`,
anchor: `none` `string`,
name: `none` `string`
)

..args-style `coordinate` or `content` or `style`

When one coordinate is given as a positional argument, the content will be placed at that position. When two coordinates are given as positional arguments, the content will be placed inside a rectangle between the two positions. All named arguments are styling and any additional positional arguments will panic.

angle `angle` or `coordinate` Default: `"0deg"`

Rotates the content by the given angle. A coordinate can be given to rotate the content by the angle between it and the first coordinate given in `args`. This effectively points the right hand side of the content towards the coordinate. This currently exists because Typst's rotate function does not change the width and height of content.

Style Root `content`

Style Keys

padding `number` or `dictionary` Default: `0`

Sets the spacing around content. Can be a single number to set padding on all sides or a dictionary to specify each side specifically. The dictionary follows Typst's `pad` function: https://typst.app/docs/reference/layout/pad/

frame `string` or `none` Default: `none`

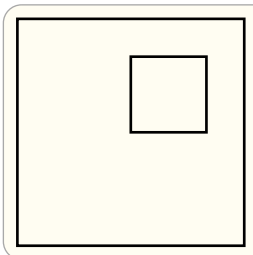
Sets the frame style. Can be `none`, "rect" or "circle" and inherits the `stroke` and `fill` style.

Anchors

Supports compass anchors.

3.3.8 rect

Draws a rectangle between two coordinates.



```
rect((0,0), (1,1))
rect((-1.5, 1.5), (1.5, -1.5))
```

Style Root `rect`

Anchors

Supports compass anchors.

Parameters

rect(
  a: `coordinate`,
  b: `coordinate`,
  name: `none` `string`,
  anchor: `none` `string`,
  ..style: `style`
)

a `coordinate`

Coordinate of the top left corner of the rectangle.

b `coordinate`

Coordinate of the bottom right corner of the rectanlge. You can draw a rectangle with a specified width and height by using relative coordinates for this parameter `(rel: (width, height))`.

3.3.9 bezier

Draws a quadratic or cubic bezier curve

```
let (a, b, c) = ((0, 0), (2, 0), (1, 1))
line(a, c,  b, stroke: gray)
bezier(a, b, c)

let (a, b, c, d) = ((0, -1), (2, -1), (.5, -2), (1.5, 0))
line(a, c, d, b, stroke: gray)
bezier(a, b, c, d)
```

Parameters

bezier(
  start: `coordinate`,
  end: `coordinate`,
  ..ctrl-style: `coordinate` `style`,
  name: `none` `string`
)

start `coordinate`

Start position

end `coordinate`
End position (last coordinate)

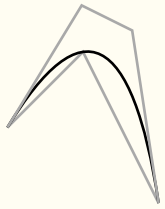
..ctrl-style `coordinate` or `style`
The first two positional arguments are taken as cubic bezier control points, where the first is the start control point and the second is the end control point. One control point can be given for a quadratic bezier curve instead. Named arguments are for styling.

Style Root `bezier`
Style Keys
Supports marks.

Anchors
ctrl-n nth control point where n is an integer starting at 0
start The start position of the curve.
end The end position of the curve.

3.3.10 bezier-through

Draw a cubic bezier curve through a set of three points. See `bezier` for style and anchor details.



```
let (a, b, c) = ((0, 0), (1, 1), (2, -1))
line(a, b, c, stroke: gray)
bezier-through(a, b, c, name: "b")

// Show calculated control points
line(a, "b.ctrl-0", "b.ctrl-1", c, stroke: gray)
```

Parameters

bezier-through(
start: `coordinate`,
pass-through: `coordinate`,
end: `coordinate`,
name: `none` `string`,
..style: `style`
)

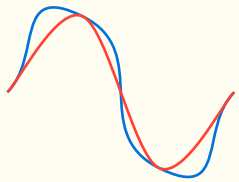
start `coordinate`
Start position

pass-through `coordinate`
Curve mid-point

end `coordinate`
End coordinate

3.3.11 catmull

Draw a Catmull-Rom curve through a set of points.



```
catmull((0,0), (1,1), (2,-1), (3,0), tension: .4, stroke: blue)
catmull((0,0), (1,1), (2,-1), (3,0), tension: .5, stroke: red)
```

Parameters

catmull(
..pts-style: `coordinate` `style`,
close: `bool`,
name: `none` `string`
)

..pts-style `coordinate` or `style`

Positional arguments should be coordinates that the curve should pass through. Named arguments are for styling.

close `bool` Default: `"false"`

Closes the curve with a straight line between the start and end of the curve.

Style Root `catmull`

Style Keys

tension `float` Default: `0.5`

I need a description

Supports marks.

Anchors

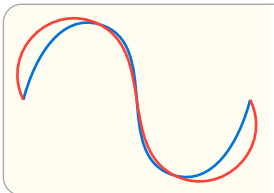
start The position of the start of the curve.

end The position of the end of the curve.

pt-n The nth given position (0 indexed so “pt-0” is equal to “start”)

3.3.12 hobby

Draws a Hobby curve through a set of points.



```
hobby((0, 0), (1, 1), (2, -1), (3, 0), omega: 0, stroke: blue)
hobby((0, 0), (1, 1), (2, -1), (3, 0), omega: 1, stroke: red)
```

Parameters

hobby(
..pts-style: `coordinate` `style`,
ta: `auto` `array`,
tb: `auto` `array`,
close: `bool`,
name: `none` `string`
)

..pts-style `coordinate` or `style`

Positional arguments are the coordinates to use to draw the curve with, a minimum of two is required. Named arguments are for styling.

ta `auto` or `array` Default: `"auto"`

Outgoing tension at `pts.at(n)` from `pts.at(n)` to `pts.at(n+1)`. The number given must be one less than the number of points.

tb `auto` or `array` Default: `"auto"`

Incoming tension at `pts.at(n+1)` from `pts.at(n)` to `pts.at(n+1)`. The number given must be one less than the number of points.

close `bool` Default: `"false"`

Closes the curve with a straight line between the start and end of the curve.

Style Root `hobby`

Style Keys

Supports marks.

omega `idk` Default: `none`

The curve's curlyness

rho `idk` Default: `none`

Anchors

start The position of the start of the curve.

end The position of the end of the curve.

pt-n The nth given position (0 indexed, so "pt-0" is equal to "start")

3.3.13 merge-path

Merges two or more paths by concattenating their elements. Anchors and visual styling, such as `stroke` and `fill`, are not preserved. When an element's path does not start at the same position the previous element's path ended, a straight line is drawn between them so that the final path is continuous. You must then pay attention to the direction in which element paths are drawn.

```
merge-path(fill: white, {
  line((0, 0), (1, 0))
  bezier((), (0, 0), (1,1), (0,1))
})
```

Parameters

merge-path(
 body: `elements`,
 close: `bool`,
 name: `none` `string`,
 ..style: `style`
)

body `elements`

Elements with paths to be merged together.

close `bool` Default: `"false"`

Close the path with a straight line from the start of the path to its end.

Anchors

start The start of the merged path.

end The end of the merged path.