

# beam

draw optics experiment setups with CeTZ

v0.1.0

2026-01-20

<https://github.com/bendix4620/beam>

## ABSTRACT

In a landscape dominated by copy pasting inkscape templates beam aims to simplify the creation of schematics for experiment setups in the field of optics.

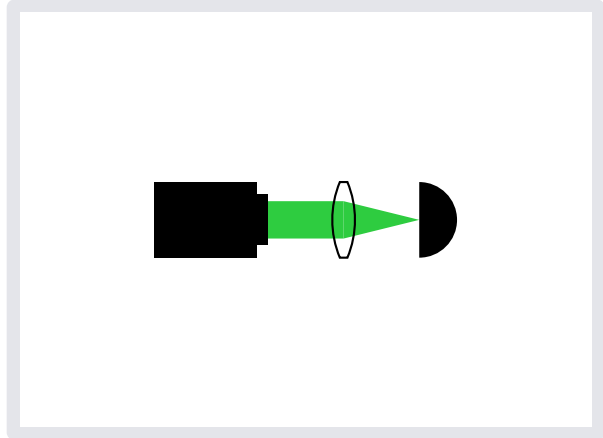
## CONTENTS

1 Getting Started .....	2
2 Styling .....	3
3 Components .....	4
4 Custom Components .....	11
5 Internals .....	12

# 1 GETTING STARTED

beam is heavily inspired by [zap](#) ⚡. The usage should feel very familiar to those accustomed to it.

```
1  #import "@preview/beam:0.1.0" typ
2
3  #beam.setup({
4      import beam: *
5
6      // draw your setup, for example...
7      laser("laser", (0, 0))
8      lens("l1", (1, 0))
9      detector("cam", (2, 0))
10     beam("", "laser", "l1")
11     focus("", "l1", "cam")
12 })
```



All components accept 1-3 coordinates<sup>1</sup>. The coordinates can be anything that [CeTZ](#) can parse and are used to position the components automatically

**1 coordinate** places the component at the given point. It can be rotated by passing `rotate: <angle>` to the component's function.

**2 coordinates** aligns the component between the given points. The position can be adjusted by passing `position: <ratio>` to the component<sup>2</sup>.

**3 coordinates** aligns the component to mimic reflection/refraction. It is placed at the middle point and is rotated to face the bisector of the angle spanned by the 3 points.

Please note that there is no standard for depicting optical components, so I gathered all the inspiration from colleagues, friends and certain [Inkscape template](#) and simply drew some symbols.

---

<sup>1</sup>except `beam()`, it takes  $\geq 2$  coordinates

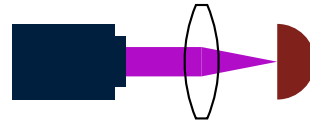
<sup>2</sup>For some components this value is always fixed

## 2 STYLING

Styling works just like in [CeTZ](#). However, beam uses a dedicated function for styling to not interfere with other [CeTZ](#)-based libraries.

Styling can be applied globally or locally on any given component.

```
1  #import "@preview/beam:0.1.0" typ
2
3  #beam.setup({
4    import beam: *
5
6    set-beam-style(
7      beam: (stroke: 11pt + purple),
8      detector: (fill:
9        red.darken(50%)),
10    )
11  laser("laser", (0, 0), fill: navy)
12  lens("l1", (1, 0), scale: 1.5)
13  detector("cam", (2, 0))
14  beam("", "laser", "l1")
15  focus("", "l1", "cam")
16 }
```



## 3 COMPONENTS

All components follow the same interface. Below is a list of the available optical components. Parameters without dedicated description are passed to `component()`.

- [beam\(\)](#)
- [beam-splitter\(\)](#)
- [beam-splitter-plate\(\)](#)
- [detector\(\)](#)
- [fade\(\)](#)
- [filter\(\)](#)
- [filter-rot\(\)](#)
- [focus\(\)](#)
- [grating\(\)](#)
- [laser\(\)](#)
- [lens\(\)](#)
- [mirror\(\)](#)
- [objective\(\)](#)
- [pinhole\(\)](#)
- [prism\(\)](#)
- [sample\(\)](#)

### 3.1 beam

laser beam

```
1 #beam.setup({  
2     import beam: *  
3     mirror("m1", (0, 1), (1, 0), (2,  
4         1))  
5     beam("", "m1.in", "m1", "m1.out")  
6 })
```

typ



#### 3.1.1 Parameters

```
beam(  
    name: str,  
    ..points-style-decoration: coordinate style decoration  
)
```

### 3.2 beam-splitter

beam splitter cube

```
1 #beam.setup({  
2     import beam: *  
3     beam-splitter("", (0, 0))  
4 })
```

typ



### 3.2.1 Parameters

```
beam-splitter(  
    name: name,  
    ..points-style-decoration: points style decoration,  
    flip: bool  
)
```

**flip** bool

flip along local y-axis

Default: false

## 3.3 beam-splitter-plate

beam splitter plate

```
1 #beam.setup({  
2     import beam: *  
3     beam-splitter-plate("", (0, 0))  
4 })
```

typ



### 3.3.1 Parameters

```
beam-splitter-plate(  
    name: str,  
    ..points-style-decoration: coordinate style decoration  
)
```

## 3.4 detector

detector / camera

```
1 #beam.setup({  
2     import beam: *  
3     detector("", (0, 0))  
4 })
```

typ



### 3.4.1 Parameters

```
detector(  
    name: str,  
    ..points-style-decoration: coordinate style decoration  
)
```

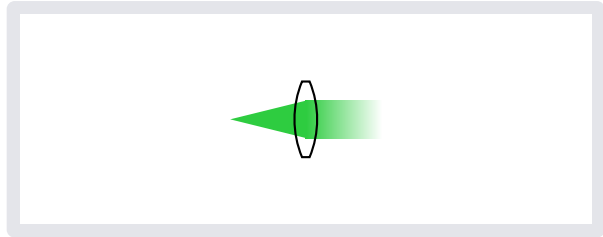
## 3.5 fade

fading laser beam

```

1 #beam.setup({
2     import beam: *
3     lens("l1", (0, 0), (2, 0))
4     focus("", "l1", "l1.in")
5     fade("", "l1", "l1.out")
6 })

```



### 3.5.1 Parameters

```

fade(
    name: str,
    ..points-style-decoration: coordinate style decoration,
    flip: bool
)

```

**flip** bool

flip the fade direction

Default: false

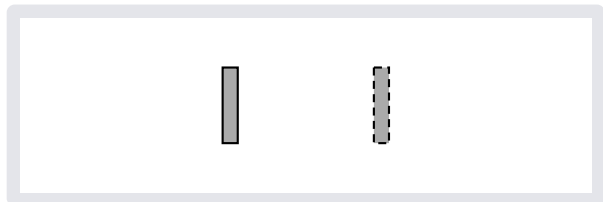
## 3.6 filter

filter

```

1 #beam.setup({
2     import beam: *
3     filter("", (0, 0))
4     flip-filter("", (2, 0))
5 })

```



### 3.6.1 Parameters

```

filter(
    name: str,
    ..points-style-decoration: coordinate style decoration
)

```

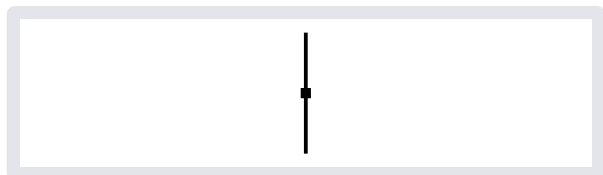
## 3.7 filter-rot

rotational filter / filter wheel

```

1 #beam.setup({
2     import beam: *
3     filter-rot("", (0, 0))
4 })

```



### 3.7.1 Parameters

```
filter-rot(  
    name: str,  
    ..points-style-decoration: coordinate style decoration,  
    flip: true  
)
```

**flip** true

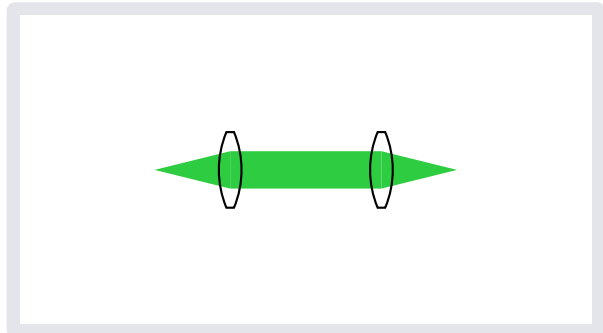
flip the filter

Default: false

## 3.8 focus

focusing laser beam

```
1 #beam.setup({  
2     import beam: *  
3     lens("l1", (1, 0))  
4     lens("l2", (3, 0))  
5     beam("", "l1", "l2")  
6     focus("", (0, 0), "l1", flip:  
7         true)  
8 })
```



### 3.8.1 Parameters

```
focus(  
    name: str,  
    ..points-style-decoration: coordinate style decoration,  
    flip: bool  
)
```

**flip** bool

flip the focus direction

Default: false

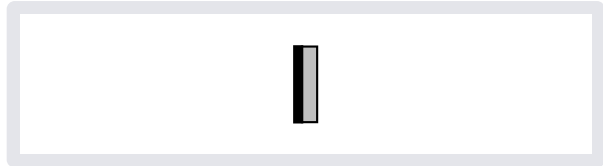
## 3.9 grating

refraction grating

```

1 #beam.setup({
2     import beam: *
3     grating("", (0, 0))
4 })

```



### 3.9.1 Parameters

```

grating(
    name: str,
    ..points-style-decoration: points style decoration
)

```

## 3.10 laser

laser source

```

1 #beam.setup({
2     import beam: *
3     laser("", (0, 0))
4 })

```



### 3.10.1 Parameters

```

laser(
    name: str,
    ..points-style-decoration: coordinate style decoration
)

```

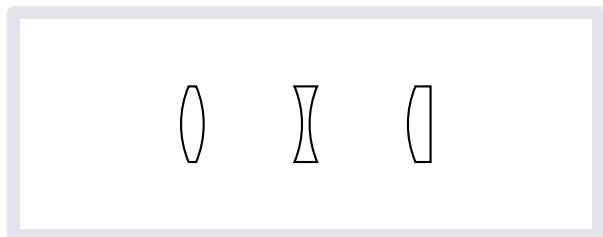
## 3.11 lens

lens

```

1 #beam.setup({
2     import beam: *
3     lens("", (0, 0))
4     lens("", (1.5, 0), kind: "|")
5     lens("", (3, 0), kind: "||")
6 })

```



### 3.11.1 Parameters

```

lens(
    name: str,
    kind: str,
    ..points-style-decoration: coordinate style decoration
)

```



**kind**    `str`

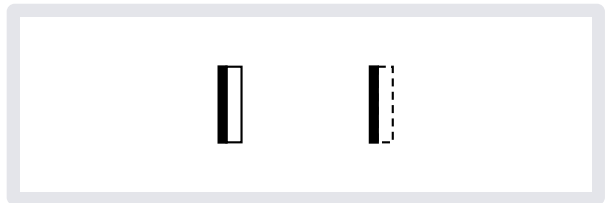
what kind of lens to draw. Supported lenses are "`()`", "`((, ))`", "`()(`", "`(|`", "`)|`", "`|)`", "`|(|`" and "`||`"

Default: "`()`"

### 3.12 mirror

mirror

```
1 #beam.setup({  
2     import beam: *  
3     mirror("", (0, 0))  
4     flip-mirror("", (2, 0))  
5 })
```



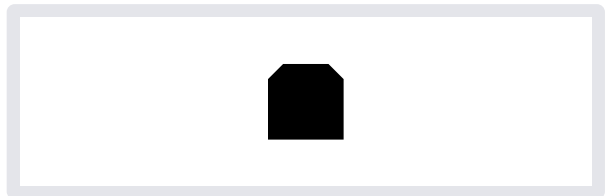
#### 3.12.1 Parameters

```
mirror(  
    name: str,  
    ..points-style-decoration: points style decoration  
)
```

### 3.13 objective

microscopy objective

```
1 #beam.setup({  
2     import beam: *  
3     objective("", (0, 0), rotate:  
4         90deg)  
4 })
```



#### 3.13.1 Parameters

```
objective(  
    name: str,  
    ..points-style-decoration: coordinate style decoration  
)
```

### 3.14 pinhole

pinhole / aperture

```

1 #beam.setup({
2     import beam: *
3     pinhole("", (0, 0))
4 })

```



### 3.14.1 Parameters

```

pinhole(
    name: str,
    ..points-style-decoration: coordinate style decoration
)

```

## 3.15 prism

dispersive prism

```

1 #beam.setup({
2     import beam: *
3     prism("", (0, 0))
4 })

```



### 3.15.1 Parameters

```

prism(
    name: str,
    ..points-style-decoration: coordinate style decoration
)

```

## 3.16 sample

sample

```

1 #beam.setup({
2     import beam: *
3     sample("", (0, 0))
4 })

```



### 3.16.1 Parameters

```

sample(
    name: str,
    ..points-style-decoration: points style decoration
)

```

## 4 CUSTOM COMPONENTS

Custom components can be easily created with the help of [component\(\)](#) and [interface\(\)](#).

```
1  #import "@preview/beam:0.1.0" typ
2
3  #import beam: cetz, component,
   interface
4
5  // draw a simple rectangle
6  #let custom(name, ..params) = {
7      let w = 2
8      let h = 1
9
10     let sketch(ctx, points, style) = {
11         interface(
12             (-w / 2, -h / 2),
13             (w / 2, h / 2),
14             io: points.len() < 2,
15         )
16
17         cetz.draw.rect("bounds.north-
18             east", "bounds.south-
19             west", ..style)
20     }
21     component("my-custom-component",
22         sketch: sketch, name, ..params)
23 }
24
25 #beam.setup({
26     import beam: *
27     custom("c", (0, 0), (3, 0))
28     beam("", "c.in", "c.out")
29 })
```



## 5 INTERNALS

- [component\(\)](#)
- [get-beam-style\(\)](#)
- [init-beam\(\)](#)
- [interface\(\)](#)
- [set-beam-style\(\)](#)
- [setup\(\)](#)
- [sketch-axis\(\)](#)
- [sketch-debug\(\)](#)
- [sketch-label\(\)](#)

### 5.1 component

Handle component creation

#### 5.1.1 Parameters

```
component(  
    root: str,  
    sketch: function,  
    num-points: array,  
    name: str,  
    ..points-style: coordinate style,  
    position: ratio,  
    rotate: angle,  
    axis: auto bool style,  
    debug: auto bool style,  
    label: auto none content dictionary  
)
```

**root**    `str`

Component type identifier. Used to find the correct style

**sketch**    `function`

Function that draws the component. Takes `context`, `array of vector` and `style`

Default: `(ctx, points, style) => {}`

**num-points**    `array`

Number of points supported by the component

Default: `(1, 2)`

**name**    `str`

Component identifier. Used by `cetz` to reference the component

**..points-style** coordinate or style

Points (positional) and style (named) just like when using cetz's shapes

**position** ratio

Position between start and end point. Only works when 2 points are given

Default: 50%

**rotate** angle

Rotate the component. Only works when 1 point is given.

Default: 0deg

**axis** auto or bool or style

optical axis decoration

- auto uses global style (equiv. to (:))
- bool will turn axis on or off (equiv. to (enabled: axis))
- dictionary will be merged with global style.

See named parameters of [sketch-axis\(\)](#) for valid definitions

Default: auto

**debug** auto or bool or style

debug info decoration

- auto uses global style (equiv. to (:))
- bool will turn axis on or off (equiv. to (enabled: debug))
- dictionary will be merged with global style.

See named parameters of [sketch-debug\(\)](#) for valid definitions

Default: auto

**label** auto or none or content or dictionary

label decoration

- auto uses global style (equiv. to (:))
- none and content will overwrite the displayed content (equiv. to (content: label))
- dictionary will be merged with global style.

See named parameters of [sketch-label\(\)](#) for valid definitions

Default: auto

## 5.2 get-beam-style

get currently active style

### 5.2.1 Parameters

`get-beam-style`(ctx: `context`)

## 5.3 init-beam

initialize beam

Useful when working with other cetz extensions (for example [zap](#)) that bring their own canvas

```
1 #cetz.canvas({  
2   import cetz.draw: *  
3   init-beam()  
4   // draw setup here  
5 })
```

typst

### 5.3.1 Parameters

`init-beam`()

## 5.4 interface

Create a bounding box for a component

cetz's `rect-around()` does not work properly on groups with rotation, so a manual bounding box is necessary

### 5.4.1 Parameters

```
interface(  
  ll: coordinate,  
  ur: coordinate,  
  io: bool  
)
```

**ll** `coordinate`

lower left point of the bbox

**ur** `coordinate`

upper right point of the bbox

**io** `bool`

wether to automaically create input and output anchors

Default: `false`

## 5.5 set-beam-style

change component style for the entire scope

### 5.5.1 Parameters

```
set-beam-style(..style: style)
```

## 5.6 setup

beam's canvas wrapper. Takes care of proper initialization

```
1 #beam.setup({  
2     import beam: *  
3     // draw setup here  
4 })
```

typst

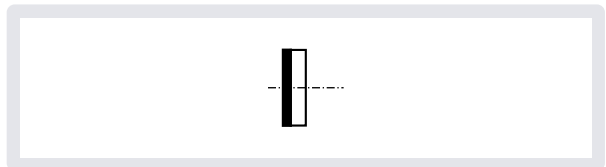
### 5.6.1 Parameters

```
setup(  
    body,  
    preamble,  
    ..params  
)
```

## 5.7 sketch-axis

draw the optical axis

```
1 #beam.setup({  
2     import beam: *  
3     mirror("m1", (), axis: true)  
4 })
```

typ

### 5.7.1 Parameters

```
sketch-axis(  
    enabled: bool,  
    length: int float,  
    stroke: stroke dictionary  
)
```

**enabled**    bool

Whether to draw the axis

Default: false

**length** `int` or `float`

axis length

Default: `1`

**stroke** `stroke` or `dictionary`

axis stroke

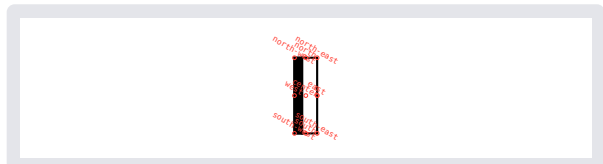
Default: (paint: black, thickness: `.5pt`, dash: `"densely-dash-dotted"`)

## 5.8 sketch-debug

draw debug information

```
1 #beam.setup({
2   import beam: *
3   mirror("ml", (), debug: true)
4 })
```

typ



### 5.8.1 Parameters

```
sketch-debug(
  name: str,
  enabled: bool,
  stroke: stroke dictionary,
  radius: length,
  angle: angle,
  shift: length,
  inset: length dictionary,
  fsize: length,
  fill: color
)
```

**enabled** `bool`

wether to draw debug info

Default: `false`

**stroke** `stroke` or `dictionary`

anchor marker stroke

Default: `.2pt + red`



**radius**   length

anchor marker radius

Default: **.7pt**

**angle**   angle

anchor name rotation

Default: **-30deg**

**shift**   length

anchor name shift

Default: **3pt**

**inset**   length or dictionary

anchor name inset

Default: **1pt**

**fsize**   length

anchor name font size

Default: **3pt**

**fill**   color

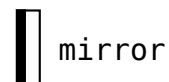
anchor name text color

Default: red

## 5.9 sketch-label

draw the component label

```
1 #beam.setup({  
2   import beam: *  
3   mirror("m1", (), label: [mirror])  
4 })
```



## 5.9.1 Parameters

```
sketch-label(  
    name: str,  
    rotation: angle,  
    scope: str,  
    pos: str angle,  
    content: none content,  
    anchor: auto str,  
    angle: auto angle,  
    padding: int float length dictionary,  
    ..style: style  
)
```

**scope** str

1	north/90deg	none
2		
3		
4	west/180deg     east/0deg	
5		
6		
7	south/270deg	

Imagine a component with anchors like above. The scope argument defines how the given angle is interpreted in terms of the components rotation. This does not account for external rotations on the canvas.

- "local" the position is resolved relative to the component bbox after rotation
- "parent" the position is resolved relative to the component bbox before rotation

Default: "local"

**pos** str or angle

where to position the label

Default: 90deg

**content** none or content

the label content

Default: none

**anchor** auto or str

label anchor. auto will try to pick anchor so that label and component do not overlap

Default: auto

**angle** `auto` or `angle`

rotate the label. `auto` will rotate the label with its position

Default: `0deg`

**padding** `int` or `float` or `length` or `dictionary`

label content padding

Default: `.1`

**..style** `style`

additional styling passed to `cetz's content()`