beam

draw optics experiment setups with CeTZ

v0.1.0 2026-01-19

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

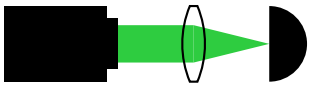
1 Getting Started

beam is heavily inspired by zap⚡. The usage should feel very familiar to those accustomed to it.

```typ
#import "@preview/beam:0.1.0"

#beam.setup({
    import beam: *

    // draw your setup, for example...
    laser("laser", (0, 0))
    lens("l1", (1, 0))
    detector("cam", (2, 0))
    beam("", "laser", "l1")
    focus("", "l1", "cam")
})
```



All components accept 1-3 coordinates[1]. 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 alignes the component between the given points. The position can be adjusted by passing `position: <ratio>` to the component[2].

3 coordinates alignes 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.

[1] except `beam()`, it takes ≥ 2 coordinates

[2] 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.

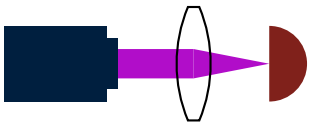
Styling can be applied globally or locally on any given component.

```typ
#import "@preview/beam:0.1.0"

#beam.setup({
    import beam: *

    set-beam-style(
        beam: (stroke: 11pt + purple),
        detector: (fill:
        red.darken(50%)),
    )

    laser("laser", (0, 0), fill: navy)
    lens("l1", (1, 0), scale: 1.5)
    detector("cam", (2, 0))
    beam("", "laser", "l1")
    focus("", "l1", "cam")
})
```



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()`.

3.1 beam

laser beam

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

3.1.1 Parameters

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

3.2 beam-splitter

beam splitter cube

```typ
#beam.setup({
    import beam: *
    beam-splitter("", (0, 0))
})
```

3.2.1 Parameters

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

flip `bool`

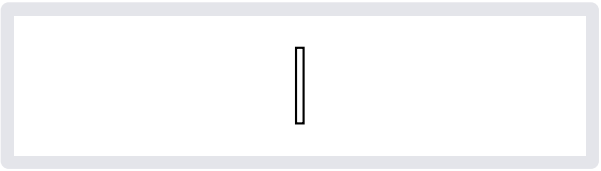
flip along local y-axis

Default: `false`

3.3 beam-splitter-plate

beam splitter plate

```typ
#beam.setup({
    import beam: *
    beam-splitter-plate("", (0, 0))
})
```



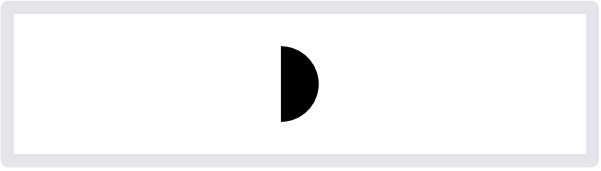
3.3.1 Parameters

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

3.4 detector

detector / camera

```typ
#beam.setup({
    import beam: *
    detector("", (0, 0))
})
```



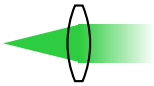
3.4.1 Parameters

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

3.5 fade

fading laser beam

```typ
#beam.setup({
    import beam: *
    lens("l1", (0, 0), (2, 0))
    focus("", "l1", "l1.in")
    fade("", "l1", "l1.out")
})
```



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

```typ
#beam.setup({
    import beam: *
    filter("", (0, 0))
    flip-filter("", (2, 0))
})
```

3.6.1 Parameters

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

3.7 filter-rot

rotational filter / filter wheel

```typ
#beam.setup({
    import beam: *
    filter-rot("", (0, 0))
})
```

3.7.1 Parameters

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

3.8 focus

focusing laser beam

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

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

```typ
#beam.setup({
    import beam: *
    grating("", (0, 0))
})
```

3.9.1 Parameters

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

3.10 laser

laser source

```
#beam.setup({
    import beam: *
    laser("", (0, 0))
})
```

3.10.1 Parameters

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

3.11 lens

lens

```
#beam.setup({
    import beam: *
    lens("", (0, 0))
    lens("", (1.5, 0), kind: ")(")
    lens("", (3, 0), kind: "(|")
})
```

3.11.1 Parameters

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

kind `str`

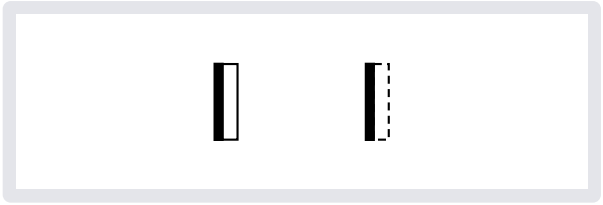
what kind of lens to draw. Supported lenses are `"()"`, `"(("`, `"))"`, `")("`, `"(|"`, `")|"`, `"|)"`, `"|("` and `"||"`

Default: `"()"`

3.12 mirror

mirror

```typ
#beam.setup({
    import beam: *
    mirror("", (0, 0))
    flip-mirror("", (2, 0))
})
```



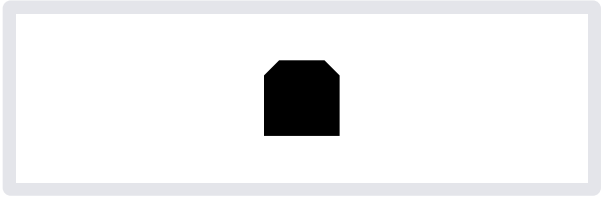
3.12.1 Parameters

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

3.13 objective

microscopy objective

```typ
#beam.setup({
    import beam: *
    objective("", (0, 0), rotate: 90deg)
})
```



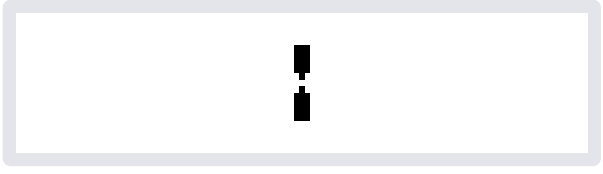
3.13.1 Parameters

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

3.14 pinhole

pinhole / aperture

```typ
#beam.setup({
    import beam: *
    pinhole("", (0, 0))
})
```



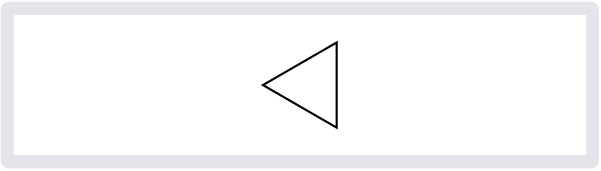
3.14.1 Parameters

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

3.15 prism

dispersive prism

```
#beam.setup({
    import beam: *
    prism("", (0, 0))
})
```



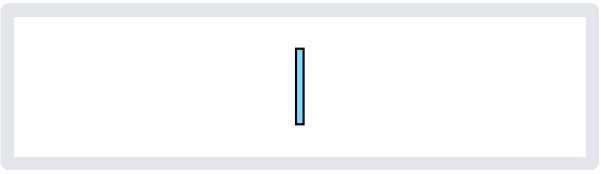
3.15.1 Parameters

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

3.16 sample

sample

```
#beam.setup({
    import beam: *
    sample("", (0, 0))
})
```



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()`.

```typ
#import "@preview/beam:0.1.0"

#import beam: cetz, component, interface

// draw a simple rectangle
#let custom(name, ..params) = {
    let w = 2
    let h = 1

    let sketch(ctx, points, style) = {
        interface(
            (-w / 2, -h / 2),
            (w / 2, h / 2),
            io: points.len() < 2,
        )

        cetz.draw.rect("bounds.north-east", "bounds.south-west", ..style)
    }
    component("my-custom-component", sketch: sketch, name, ..params)
}

#beam.setup({
    import beam: *
    custom("c", (0, 0), (3, 0))
    beam("", "c.in", "c.out")
})
```

5 Internals

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 identifyer. 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 identifyer. 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 vlaid 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

```typst
#cetz.canvas({
  import cetz.draw: *
  init-beam()
  // draw setup here
})
```

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 bounidng 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

```typst
#beam.setup({
    import beam: *
    // draw setup here
})
```

5.6.1 Parameters

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

5.7 sketch-axis

draw the optical axis

```typ
#beam.setup({
  import beam: *
  mirror("m1", (), axis: true)
})
```

5.7.1 Parameters

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

enabled `bool`

Wether to draw the axis

Default: `false`

length `int` or `float`

axis length

Default: `1`

stroke `stroke` or `dictionary`

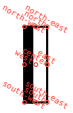
axis stroke

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

5.8 sketch-debug

draw debug information

```typ
#beam.setup({
  import beam: *
  mirror("m1", (), debug: true)
})
```



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`

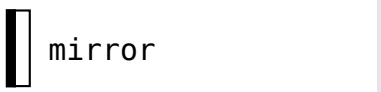
anchor name text color

Default: `red`

5.9 sketch-label

draw the component label

```typ
#beam.setup({
  import beam: *
  mirror("m1", (), label: [mirror])
})
```



5.9.1 Parameters

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

pos `str` or `angle`

where to position the label, given as anchor of bounds or angle

Default: `90deg`

scope `str`

relative to wich scope the label should be positioned

- `"local"` relative to local component coordinate system
- `"parent"` relative to coordinate system the component is placed in
- `"global"` relative to the canvas's coordinate system

Default: `"global"`

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`

rotate `auto` or `angle`

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

Default: `0deg`

padding `length` or `dictionary`

label content padding

Default: `7pt`

..style `style`

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