ctz-euclide

Typst Port

Euclidean Geometry for Typst

A comprehensive geometry package built on CeTZ
Version 0.1.0
Nathan Scheinmann

Contents

1. Introduction

`ctz-euclide` is a geometry package for Typst, a port of the LaTeX package `tkz-euclide`. Built on top of CeTZ (a powerful drawing library), it provides high-level constructions for Euclidean geometry.

1.1. Features

- **Point Registry**: Define points once, reference them by name throughout your figure
- **Geometric Constructions**: Perpendiculars, parallels, bisectors, mediators
- **Intersections**: Line–line, line–circle, circle–circle with multiple solution handling
- **Triangle Centers**: Centroid, circumcenter, incenter, orthocenter, and 10+ specialized centers
- **Special Triangles**: Medial, orthic, intouch triangles
- **Transformations**: Rotation, reflection, translation, homothety, projection
- **Drawing & Styling**: Points, labels, angles, segments with tick marks
- **Grid & Axes**: Coordinate systems with customizable appearance
- **Clipping**: Mathematical line clipping for clean bounded figures

1.2. Installation

Import the package in your Typst document:

```
#import "../src/lib.typ": *
```

All figures use the `ctz-canvas` function (re-exported from CeTZ):

```
#ctz-canvas({
  import cetz.draw: *
  ctz-init()

  // Your geometry code here
})
```

Naming notes:
- All public functions are prefixed with `ctz-` to avoid conflicts.
- Point creation and drawing use `ctz-def-points` and `ctz-draw-points`.
- Other constructors use `ctz-def-*`, and drawing utilities use `ctz-draw-*`.

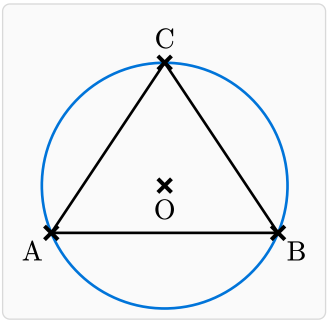
The `ctz-init()` call initializes the point registry and coordinate resolver.

1.3. Basic Usage

Code

```
#ctz-canvas(length: 0.8cm, {
  import cetz.draw: *
  ctz-init()

  // Define points
  ctz-def-points(A: (0, 0), B: (4, 0), C: (2, 3))

  // Draw triangle
  ctz-draw-line("A", "B", "C", "A", stroke: black)

  // Find circumcenter and draw circumcircle
  ctz-def-circumcenter("O", "A", "B", "C")
  ctz-draw-circle-through("O", "A", stroke: blue)

  // Draw and label points
  ctz-draw-points("A", "B", "C", "O")
  ctz-draw-labels("A", "B", "C", "O",
    A: "below left", B: "below right",
    C: "above", O: "below")
})
```

Figure



2. Core Concepts

2.1. The Point Registry

The point registry is the heart of `ctz-euclide`. Once you define a point with a name, that name can be used directly in CeTZ drawing commands.

```
ctz-def-points(A: (0, 0), B: (3, 4))  // Register points A and B
ctz-draw-line("A", "B")              // Use them directly in CeTZ
```

Under the hood, `ctz-init()` installs a coordinate resolver that translates `"A"` to the stored coordinates.

2.2. Figure Scaling

Control the size of your figures using CeTZ's `length` parameter:

```
#ctz-canvas(length: 0.8cm, { ... })
```

This scales everything proportionally, including stroke widths. Typical values:

- `0.6cm` – small inline figures
- `0.8cm` – standard examples
- `1.0cm` – large detailed figures

2.3. Coordinate Systems

Points can be defined in multiple ways:

```
// Explicit coordinates
ctz-def-points(A: (2, 3))

// Using existing CeTZ coordinates
ctz-def-points(B: (rel: (1, 1), to: "A"))

// Mixed: numbers and existing points
ctz-def-points(C: (4, 0), D: "A", E: (3, 2))
```

3. Point Definitions

3.1. Basic Points — `ctz-def-points`

Define one or more points at specific coordinates:

```
ctz-def-points(A: (0, 0), B: (4, 0), C: (2, 3))
```

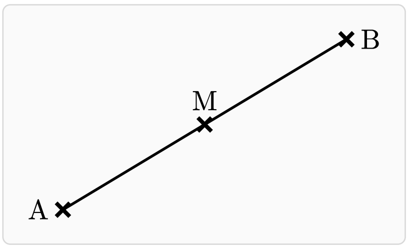
3.2. Midpoint — `ctz-def-midpoint`

Find the midpoint of a segment:

Code

```
#ctz-canvas(length: 0.8cm, {
  import cetz.draw: *
  ctz-init()

  ctz-def-points(A: (0, 0), B: (5, 3))
  ctz-def-midpoint("M", "A", "B")

  ctz-draw-line("A", "B", stroke: black)
  ctz-draw-points("A", "B", "M")
  ctz-draw-labels("A", "B", "M",
    A: "left", B: "right", M: "above")
})
```

Figure



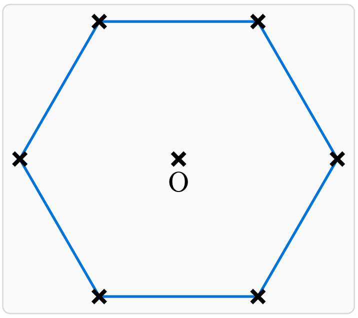
3.3. Regular Polygons — `ctz-def-regular-polygon`

Generate vertices of a regular n-gon. If you pass a polygon name first, it is registered and can be drawn/labeled by name: You can also mark all sides during drawing with `mark` and optional `mark-opts`.

Code

```
#ctz-canvas(length: 0.7cm, {
  import cetz.draw: *
  ctz-init()

  ctz-def-points(O: (0, 0), A: (3, 0))
  // O is the center; A is the starting vertex that
fixes the radius/angle.
  ctz-def-regular-polygon("Hex", ("A", "B", "C", "D",
"E", "F"), "O", "A")

  ctz-draw-polygon("Hex", stroke: blue)
  ctz-draw-points("A", "B", "C", "D", "E", "F", "O")
  ctz-draw-labels("O", O: "below")
})
```

Figure



3.4. Named Polygons — `ctz-def-polygon` / `ctz-label-polygon`

Define a polygon once and draw/label it by name:

```
ctz-def-points(A: (0, 0), B: (4, 0), C: (4, 2), D: (0, 2))
ctz-def-polygon("P1", "A", "B", "C", "D")
ctz-draw-polygon("P1", stroke: black)
ctz-label-polygon("P1", $P_1$, pos: "center")
```

3.5. Linear Combination — `ctz-def-linear`

Define a point along a line: $P = A + k(B - A)$

```
ctz-def-linear("P", "A", "B", 0.3)  // P is 30% from A to B
ctz-def-linear("Q", "A", "B", 1.5)  // Q extends beyond B
```

4. Line Constructions

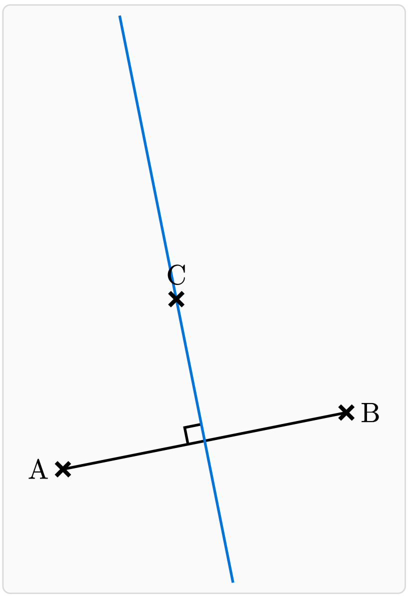
4.1. Perpendicular — `ctz-def-perp`

Construct a perpendicular line through a point:

Code

```
#ctz-canvas(length: 0.75cm, {
  import cetz.draw: *
  ctz-init()

  ctz-def-points(A: (0, 0), B: (5, 1), C: (2, 3))
  ctz-def-perp("P1", "P2", ("A", "B"), "C")
  ctz-def-project("H", "C", "A", "B")

  ctz-draw-line("A", "B", stroke: black)
  ctz-draw-line("P1", "P2", stroke: blue)
  ctz-draw-mark-right-angle("A", "H", "C", size: 0.3)

  ctz-draw-points("A", "B", "C")
  ctz-draw-labels("A", "B", "C",
    A: "left", B: "right", C: "above")
})
```

Figure



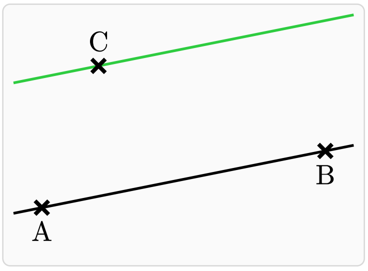
4.2. Parallel — `ctz-def-para`

Construct a parallel line through a point:

Code

```
#ctz-canvas(length: 0.75cm, {
  import cetz.draw: *
  ctz-init()

  ctz-def-points(A: (0, 0), B: (5, 1), C: (1, 2.5))
  ctz-def-para("P1", "P2", ("A", "B"), "C")

  ctz-set-clip(-0.5, -0.5, 5.5, 3.5)
  ctz-draw-line-global-clip("A", "B", add: (2, 2),
stroke: black)
  ctz-draw-line-global-clip("P1", "P2", add: (2, 2),
stroke: green)

  ctz-draw-points("A", "B", "C")
  ctz-draw-labels("A", "B", "C",
    A: "below", B: "below", C: "above")
})
```

Figure



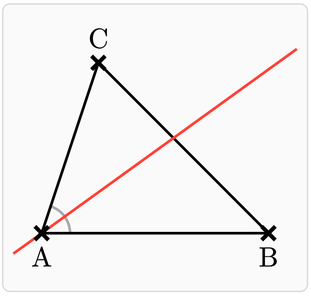
4.3. Angle Bisector — `ctz-def-bisect`

Construct the bisector of an angle:

Code

```
#ctz-canvas(length: 0.8cm, {
  import cetz.draw: *
  ctz-init()

  ctz-def-points(A: (0, 0), B: (4, 0), C: (1, 3))
  ctz-def-bisect("D1", "D2", "C", "A", "B")

  ctz-set-clip(-0.5, -0.5, 4.5, 3.5)
  ctz-draw-line("A", "B", "C", "A", stroke: black)
  ctz-draw-seg-global-clip("D1", "D2", stroke: red)

  ctz-draw-angle("A", "C", "B", radius: 0.5, stroke:
gray)
  ctz-draw-points("A", "B", "C")
  ctz-draw-labels("A", "B", "C",
    A: "below", B: "below", C: "above")
})
```

Figure



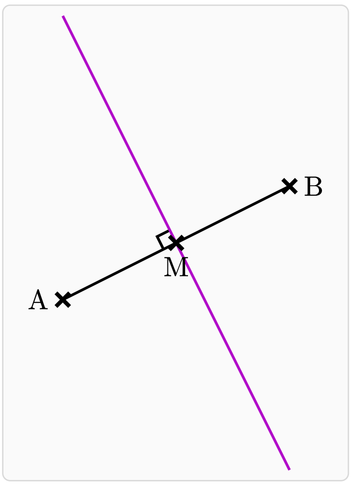
4.4. Perpendicular Bisector — `ctz-def-mediator`

Construct the perpendicular bisector of a segment:

Code

```
#ctz-canvas(length: 0.75cm, {
  import cetz.draw: *
  ctz-init()

  ctz-def-points(A: (1, 1), B: (5, 3))
  ctz-def-mediator("M1", "M2", "A", "B")
  ctz-def-midpoint("M", "A", "B")

  ctz-draw-line("A", "B", stroke: black)
  ctz-draw-line("M1", "M2", stroke: purple)
  ctz-draw-mark-right-angle("M1", "M", "A", size:
0.25)

  ctz-draw-points("A", "B", "M")
  ctz-draw-labels("A", "B", "M",
    A: "left", B: "right", M: "below")
})
```

Figure



5. Intersections

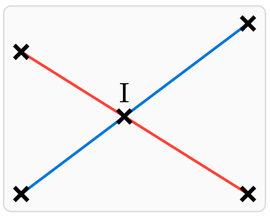
5.1. Line–Line — `ctz-def-ll`

Find the intersection of two lines:

Code

```
#ctz-canvas(length: 0.8cm, {
  import cetz.draw: *
  ctz-init()

  ctz-def-points(A: (0, 0), B: (4, 3),
           C: (4, 0), D: (0, 2.5))
  ctz-def-line("L1", "A", "B")
  ctz-def-line("L2", "C", "D")
  ctz-def-ll("I", "L1", "L2")

  ctz-draw-line("L1", stroke: blue)
  ctz-draw-line("L2", stroke: red)

  ctz-draw-points("A", "B", "C", "D", "I")
  ctz-draw-labels("I", I: "above")
})
```

Figure



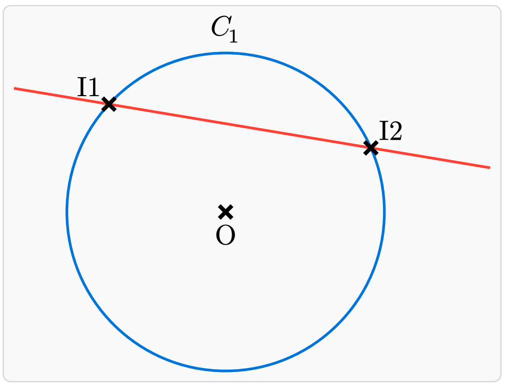
5.2. Line–Circle — `ctz-def-lc`

Find intersections of a line with a circle:

Code

```
#ctz-canvas(length: 0.7cm, {
  import cetz.draw: *
  ctz-init()

  ctz-def-points(O: (0, 0), R: (3, 0),
           A: (-2, 2), B: (4, 1))
  ctz-def-line("L1", "A", "B")
  ctz-def-circle("C1", "O", through: "R")
  ctz-def-lc(("I1", "I2"), "L1", "C1")

  ctz-draw-circle("C1", stroke: blue)
  ctz-label-circle("C1", $C_1$, pos: "above", dist:
0.2)
  ctz-set-clip(-4, -4, 5, 4)
  ctz-draw-line-global-clip("A", "B", add: (2, 2),
stroke: red)

  ctz-draw-points("O", "I1", "I2")
  ctz-draw-labels("O", "I1", "I2",
    O: "below", I1: "above left", I2: "above right")
})
```

Figure



Named line/circle form:

```
ctz-def-line("L1", "A", "B")
ctz-def-circle("C1", "O", radius: 3)
ctz-def-lc(("I1", "I2"), "L1", "C1")
```

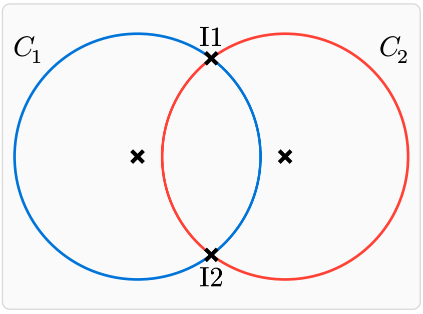
5.3. Circle–Circle — `ctz-def-cc`

Find intersections of two circles:

Code

```
#ctz-canvas(length: 0.65cm, {
  import cetz.draw: *
  ctz-init()

  ctz-def-points(O1: (0, 0), O2: (3, 0),
            R1: (2.5, 0), R2: (5.5, 0))
  ctz-def-circle("C1", "O1", through: "R1")
  ctz-def-circle("C2", "O2", through: "R2")
  ctz-def-cc(("I1", "I2"), "C1", "C2")

  ctz-draw-circle("C1", stroke: blue)
  ctz-draw-circle("C2", stroke: red)
  ctz-label-circle("C1", $C_1$, pos: "above left",
dist: 0.2)
  ctz-label-circle("C2", $C_2$, pos: "above right",
dist: 0.2)

  ctz-draw-points("O1", "O2", "I1", "I2")
  ctz-draw-labels("I1", "I2",
    I1: "above", I2: "below")
})
```

Figure



Named circle form:

```
ctz-def-circle("C1", "O1", through: "R1")
ctz-def-circle("C2", "O2", through: "R2")
ctz-def-cc(("I1", "I2"), "C1", "C2")
```

6. Triangle Centers

6.1. Basic Centers

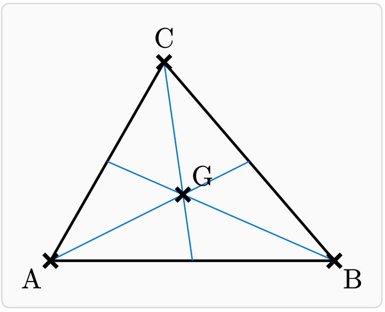
6.1.1. Centroid — `ctz-def-centroid`

The intersection of medians (center of mass):

Code

```
#ctz-canvas(length: 0.8cm, {
  import cetz.draw: *
  ctz-init()

  ctz-def-points(A: (0, 0), B: (5, 0), C: (2, 3.5))
  ctz-def-centroid("G", "A", "B", "C")

  // Draw medians
  ctz-def-midpoint("Ma", "B", "C")
  ctz-def-midpoint("Mb", "A", "C")
  ctz-def-midpoint("Mc", "A", "B")

  ctz-draw-line("A", "B", "C", "A", stroke: black)
  ctz-draw-line("A", "Ma", stroke: blue + 0.5pt)
  ctz-draw-line("B", "Mb", stroke: blue + 0.5pt)
  ctz-draw-line("C", "Mc", stroke: blue + 0.5pt)

  ctz-draw-points("A", "B", "C", "G")
  ctz-draw-labels("A", "B", "C", "G",
    A: "below left", B: "below right",
    C: "above", G: "above right")
})
```

Figure



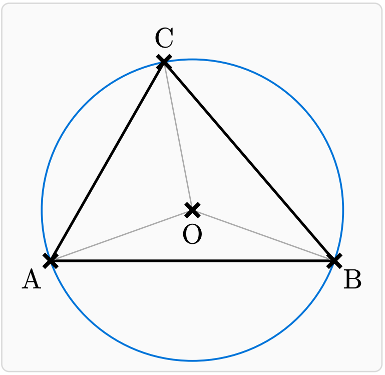
6.1.2. Circumcenter — `ctz-def-circumcenter`

Center of the circumscribed circle:

Code

```
#ctz-canvas(length: 0.75cm, {
  import cetz.draw: *
  ctz-init()

  ctz-def-points(A: (0, 0), B: (5, 0), C: (2, 3.5))
  ctz-def-circumcenter("O", "A", "B", "C")

  ctz-draw-line("A", "B", "C", "A", stroke: black)
  ctz-draw-circle-through("O", "A", stroke: blue +
0.7pt)

  ctz-draw-line("O", "A", stroke: gray + 0.5pt)
  ctz-draw-line("O", "B", stroke: gray + 0.5pt)
  ctz-draw-line("O", "C", stroke: gray + 0.5pt)

  ctz-draw-points("A", "B", "C", "O")
  ctz-draw-labels("A", "B", "C", "O",
    A: "below left", B: "below right",
    C: "above", O: "below")
})
```

Figure



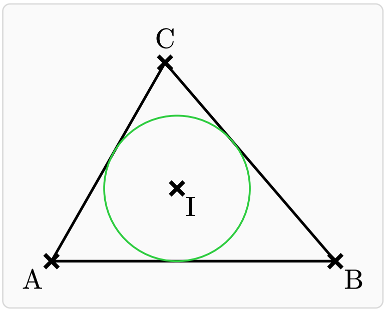
6.1.3. Incenter — `ctz-def-incenter`

Center of the inscribed circle:

Code

```
#ctz-canvas(length: 0.75cm, {
  import cetz.draw: *
  ctz-init()

  ctz-def-points(A: (0, 0), B: (5, 0), C: (2, 3.5))
  ctz-def-incenter("I", "A", "B", "C")

  ctz-draw-line("A", "B", "C", "A", stroke: black)
  ctz-draw-incircle("A", "B", "C", stroke: green +
0.7pt)

  ctz-draw-points("A", "B", "C", "I")
  ctz-draw-labels("A", "B", "C", "I",
    A: "below left", B: "below right",
    C: "above", I: "below right")
})
```

Figure



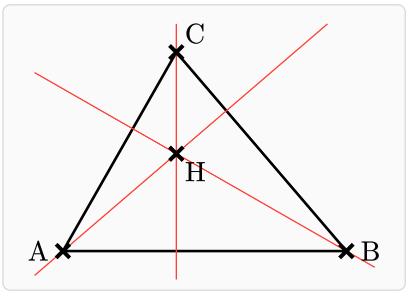
6.1.4. Orthocenter — `ctz-def-orthocenter`

Intersection of altitudes:

Code

```
#ctz-canvas(length: 0.75cm, {
  import cetz.draw: *
  ctz-init()

  ctz-def-points(A: (0, 0), B: (5, 0), C: (2, 3.5))
  ctz-def-orthocenter("H", "A", "B", "C")

  // Altitudes
  ctz-def-perp("Ha1", "Ha2", ("B", "C"), "A")
  ctz-def-perp("Hb1", "Hb2", ("A", "C"), "B")
  ctz-def-perp("Hc1", "Hc2", ("A", "B"), "C")

  ctz-set-clip(-0.5, -0.5, 5.5, 4)
  ctz-draw-line("A", "B", "C", "A", stroke: black)
  ctz-draw-line-global-clip("A", "Ha1", add: (2, 2),
stroke: red + 0.5pt)
  ctz-draw-line-global-clip("B", "Hb1", add: (2, 2),
stroke: red + 0.5pt)
  ctz-draw-line-global-clip("C", "Hc1", add: (2, 2),
stroke: red + 0.5pt)

  ctz-draw-points("A", "B", "C", "H")
  ctz-draw-labels("A", "B", "C", "H",
    A: "left", B: "right",
    C: "above right", H: "below right")
})
```

Figure



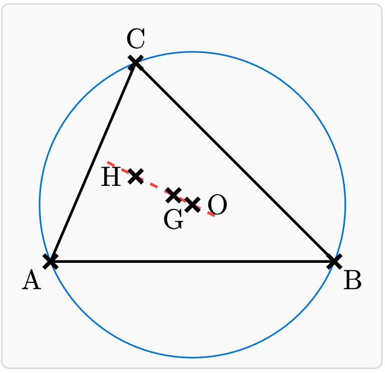
6.2. The Euler Line

In any non-equilateral triangle, the orthocenter H, centroid G, and circumcenter O are collinear. This line is called the **Euler line**, and remarkably, G divides HO in the ratio $2 : 1$.

Code

```
#ctz-canvas(length: 0.75cm, {
  import cetz.draw: *
  ctz-init()

  ctz-def-points(A: (0, 0), B: (5, 0), C: (1.5, 3.5))

  ctz-def-orthocenter("H", "A", "B", "C")
  ctz-def-centroid("G", "A", "B", "C")
  ctz-def-circumcenter("O", "A", "B", "C")

  ctz-set-clip(-0.5, -0.5, 5.5, 4)
  ctz-draw-line("A", "B", "C", "A", stroke: black)
  ctz-draw-line-add("H", "O", add: 0.5, stroke:
(paint: red, dash: "dashed"))
  ctz-draw-circle-through("O", "A", stroke: blue +
0.6pt)

  ctz-draw-points("A", "B", "C", "H", "G", "O")
  ctz-draw-labels("A", "B", "C", "H", "G", "O",
    A: "below left", B: "below right", C: "above",
    H: "left", G: "below", O: "right")
})
```

Figure



6.3. Advanced Centers

`ctz-euclide` supports 10+ specialized triangle centers:

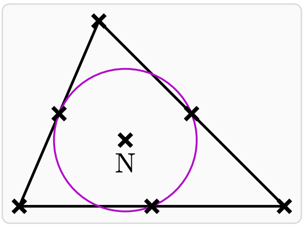
- `ctz-def-lemoine` — Symmedian point (Lemoine point)
- `ctz-def-nagel` — Nagel point
- `ctz-def-gergonne` — Gergonne point
- `ctz-def-spieker` — Spieker center (incenter of medial triangle)
- `ctz-def-euler` — Nine-point circle center
- `ctz-def-feuerbach` — Feuerbach point
- `ctz-def-mittenpunkt` — Mittenpunkt
- `ctz-def-excenter` — Excenter (specify vertex: `"a"`, `"b"`, or `"c"`)

Example with Euler (nine-point) circle:

Code

```
#ctz-canvas(length: 0.7cm, {
  import cetz.draw: *
  ctz-init()

  ctz-def-points(A: (0, 0), B: (5, 0), C: (1.5, 3.5))
  ctz-def-euler("N", "A", "B", "C")

  ctz-draw-line("A", "B", "C", "A", stroke: black)

  // Nine-point circle passes through midpoints
  ctz-def-midpoint("Ma", "B", "C")
  ctz-def-midpoint("Mb", "A", "C")
  ctz-def-midpoint("Mc", "A", "B")

  ctz-draw-circle-through("N", "Ma", stroke: purple +
0.7pt)

  ctz-draw-points("A", "B", "C", "N", "Ma", "Mb",
"Mc")
  ctz-draw-labels("N", N: "below")
})
```

Figure



7. Transformations

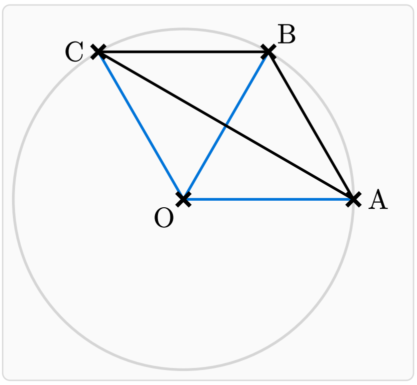
7.1. Rotation — `rotate`

Rotate a point around a center:

Code

```
#ctz-canvas(length: 0.75cm, {
  import cetz.draw: *
  ctz-init()

  ctz-def-points(O: (2, 2), A: (5, 2))
  ctz-def-rotation("B", "A", "O", 60)
  ctz-def-rotation("C", "A", "O", 120)

  ctz-draw-circle-r("O", 3, stroke:
gray.lighten(50%))
  ctz-draw-line("O", "A", stroke: blue)
  ctz-draw-line("O", "B", stroke: blue)
  ctz-draw-line("O", "C", stroke: blue)
  ctz-draw-line("A", "B", "C", "A", stroke: black)

  ctz-draw-points("O", "A", "B", "C")
  ctz-draw-labels("O", "A", "B", "C",
    O: "below left", A: "right",
    B: "above right", C: "left")
})
```

Figure



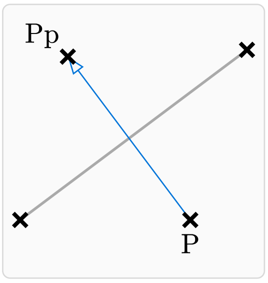
7.2. Reflection — `ctz-def-reflect`

Reflect a point across a line:

Code

```
#ctz-canvas(length: 0.75cm, {
  import cetz.draw: *
  ctz-init()

  ctz-def-points(A: (0, 0), B: (4, 3), P: (3, 0))
  ctz-def-reflect("Pp", "P", "A", "B")

  ctz-draw-line("A", "B", stroke: gray)
  ctz-draw-line("P", "Pp", stroke: blue + 0.5pt,
mark: (end: ">"))

  ctz-draw-points("A", "B", "P", "Pp")
  ctz-draw-labels("P", "Pp", P: "below", Pp: "above
left")
})
```

Figure



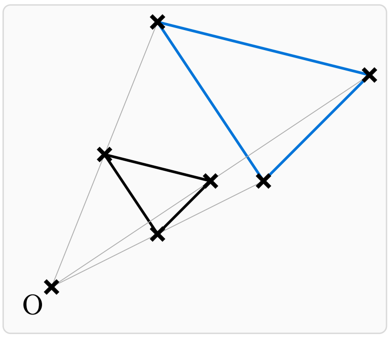
7.3. Homothety (Scaling) — `scale`

Scale a point from a center:

Code

```
#ctz-canvas(length: 0.7cm, {
  import cetz.draw: *
  ctz-init()

  ctz-def-points(O: (0, 0), A: (2, 1), B: (3, 2), C:
(1, 2.5))
  ctz-def-homothety("Ap", "A", "O", 2)
  ctz-def-homothety("Bp", "B", "O", 2)
  ctz-def-homothety("Cp", "C", "O", 2)

  ctz-draw-line("A", "B", "C", "A", stroke: black)
  ctz-draw-line("Ap", "Bp", "Cp", "Ap", stroke: blue)
  ctz-draw-line("O", "Ap", stroke: gray + 0.3pt)
  ctz-draw-line("O", "Bp", stroke: gray + 0.3pt)
  ctz-draw-line("O", "Cp", stroke: gray + 0.3pt)

  ctz-draw-points("O", "A", "B", "C", "Ap", "Bp",
"Cp")
  ctz-draw-labels("O", O: "below left")
})
```

Figure



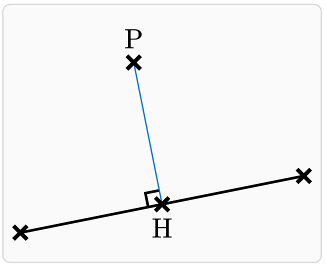
7.4. Projection — `ctz-def-project`

Project a point onto a line:

Code

```
#ctz-canvas(length: 0.75cm, {
  import cetz.draw: *
  ctz-init()

  ctz-def-points(A: (0, 0), B: (5, 1), P: (2, 3))
  ctz-def-project("H", "P", "A", "B")

  ctz-draw-line("A", "B", stroke: black)
  ctz-draw-line("P", "H", stroke: blue + 0.5pt)
  ctz-draw-mark-right-angle("A", "H", "P", size:
0.25)

  ctz-draw-points("A", "B", "P", "H")
  ctz-draw-labels("P", "H", P: "above", H: "below")
})
```

Figure



8. Drawing & Styling

8.1. Points — `ctz-draw-points`

Draw points at named locations:

```
ctz-draw-points("A", "B", "C")
```

8.2. Labels — `ctz-draw-labels`

Add labels to points with positioning:

```
ctz-draw-labels("A", "B", "C",
  A: "below left",
  B: "below right",
  C: "above")
```

Positions: `"above"`, `"below"`, `"left"`, `"right"`, `"above left"`, etc.

Custom offset:

```
ctz-draw-labels("O", O: (pos: "below", offset: (0, -0.15)))
```

More placement controls (position, offset, distance):

```
ctz-draw-labels("A", "B", "C",
  A: (pos: "above", dist: 0.25),
  B: (pos: "right", offset: (0.1, 0)),
  C: (pos: "below left", offset: (-0.05, -0.05)))
```

8.3. Segments — `ctz-draw-segment`

Draw a segment with optional arrow or bar tips and a dimension label:

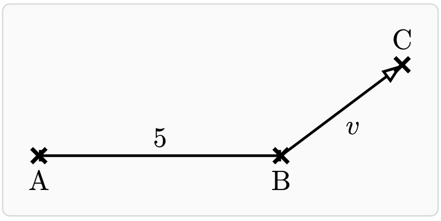
```
ctz-draw-segment("A", "B", arrows: "|-|", dim: $5$, dim-pos: "above")
```

Supported `arrows`: `--` (none), `->`, `<-`, `<->`, `|-|`, `|->`, `<-|`.

Code

```
#ctz-canvas(length: 0.8cm, {
  import cetz.draw: *
  ctz-init()

  ctz-def-points(A: (0, 0), B: (4, 0), C: (6, 1.5))

  ctz-draw-segment("A", "B", arrows: "|-|", dim: $5$,
dim-pos: "above")
  ctz-draw-segment("B", "C", arrows: "->", dim: $v$,
dim-pos: "below")
  ctz-draw-points("A", "B", "C")
  ctz-draw-labels("A", "B", "C", A: "below", B:
"below", C: "above")
})
```

Figure



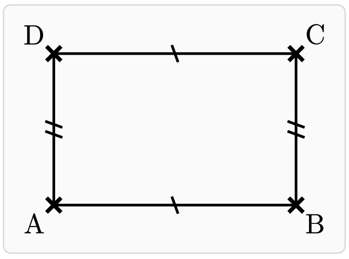
Mark equal-length segments with ticks:

```
ctz-draw-mark-segment("A", "B", mark: 1)
ctz-draw-mark-segment("C", "D", mark: 2)
```

Code

```
#ctz-canvas(length: 0.8cm, {
  import cetz.draw: *
  ctz-init()

  ctz-def-points(A: (0, 0), B: (4, 0), C: (4, 2.5),
D: (0, 2.5))
  ctz-draw-polygon("A", "B", "C", "D", stroke: black)

  // Opposite sides equal
  ctz-draw-mark-segment("A", "B", mark: 1)
  ctz-draw-mark-segment("C", "D", mark: 1)
  ctz-draw-mark-segment("B", "C", mark: 2)
  ctz-draw-mark-segment("D", "A", mark: 2)

  ctz-draw-points("A", "B", "C", "D")
  ctz-draw-labels("A", "B", "C", "D",
    A: "below left", B: "below right",
    C: "above right", D: "above left")
})
```

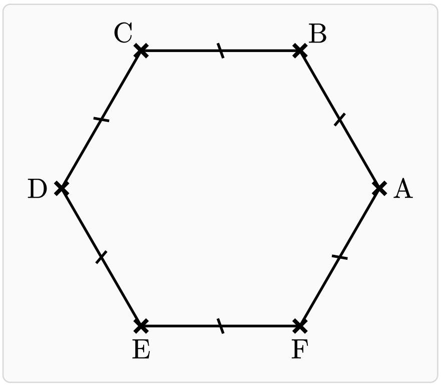
Figure



Code

```
#ctz-canvas(length: 0.7cm, {
  import cetz.draw: *
  ctz-init()

  ctz-def-points(O: (0, 0), A: (3, 0))
  ctz-def-regular-polygon("Hex", ("A", "B", "C", "D",
"E", "F"), "O", "A")

  ctz-draw-regular-polygon(("A", "B", "C", "D", "E",
"F"),
    stroke: black, mark: 1)

  // Mark all sides with the same tick
  // (use mark-opts to customize size/position)

  ctz-draw-points("A", "B", "C", "D", "E", "F")
  ctz-draw-labels("A", "B", "C", "D", "E", "F",
    A: "right", B: "above right", C: "above left",
    D: "left", E: "below", F: "below")
})
```

Figure



8.4. Segment Measurements — `ctz-draw-measure-segment`

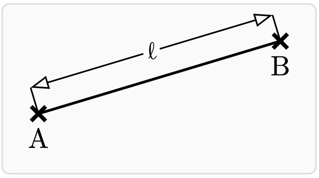
Draw an offset measurement line with dotted fences and a centered label. The line breaks around the label and uses open arrowheads by default.

```
ctz-draw-measure-segment("A", "B", label: $5$, offset: 0.3, side: "left")
```

Code

```
#ctz-canvas(length: 0.8cm, {
  import cetz.draw: *
  ctz-init()

  ctz-def-points(A: (0, 0), B: (4, 1.2))
  ctz-draw-segment("A", "B", stroke: black + 1pt)

  // Minimal measurement example
  ctz-draw-measure-segment("A", "B", label: $ell$,
offset: 0.45, side: "left",
    fence-dash: "dotted")

  ctz-draw-points("A", "B")
  ctz-draw-labels("A", "B", A: "below", B: "below")
})
```

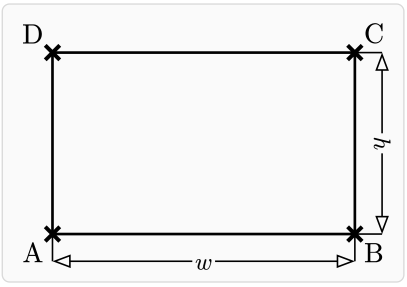
Figure



Code

```
#ctz-canvas(length: 0.8cm, {
  import cetz.draw: *
  ctz-init()

  ctz-def-points(A: (0, 0), B: (5, 0), C: (5, 3), D:
(0, 3))
  ctz-draw-polygon("A", "B", "C", "D", stroke: black
+ 1pt)

  // Rectangle measurements (width and height)
  ctz-draw-measure-segment("A", "B", label: $w$,
offset: 0.45, side: "below")
  ctz-draw-measure-segment("C", "B", label: $h$,
offset: -0.45, side: "right")

  ctz-draw-points("A", "B", "C", "D")
  ctz-draw-labels("A", "B", "C", "D",
    A: "below left", B: "below right", C: "above
right", D: "above left")
})
```

Figure



8.5. Paths — `ctz-draw-path`

Draw polylines with per-segment tips using a TikZ-like string:

```
ctz-draw-path("A--B->C|-|D", stroke: black)
```

Supported connectors: `--`, `->`, `<-`, `<->`, `|-|`, `|->`, `<-|`.

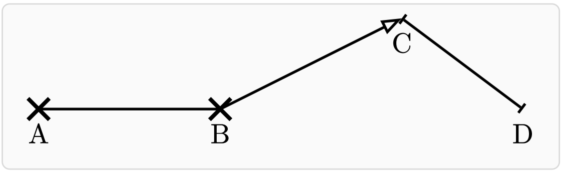
By default, `ctz-draw-path` draws points as crosses for normal segments and hides points that touch a bar connector (`|-|`, `|->`, `<-|`). Labels default to `below`. You can override per-point placements or point styles in the path with `{...}`, or via `label-overrides`.

Default behavior, label are placed below

Code

```
#ctz-canvas(length: 0.8cm, {
  import cetz.draw: *
  ctz-init()

  ctz-def-points(
    A: (0, 0), B: (3, 0), C: (6, 1.5), D: (8, 0),
  )

  // Default labels below, default point styles
  ctz-draw-path("A--B->C|-|D", stroke: black)
})
```

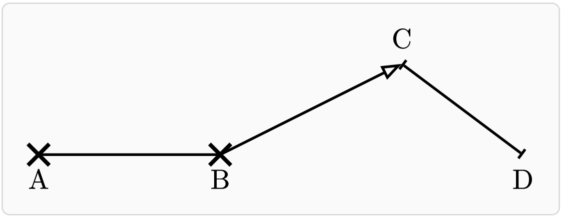
Figure



Code

```
#ctz-canvas(length: 0.8cm, {
  import cetz.draw: *
  ctz-init()

  ctz-def-points(
    A: (0, 0), B: (3, 0), C: (6, 1.5), D: (8, 0),
  )

  ctz-draw-path("A{below}--B{below}->C{above}|-|
D{below}", stroke: black)
})
```

Figure



Override placements using `label-overrides`:

```
ctz-draw-path("A--B->C|-|D",
  label-overrides: (A: "left", C: "above right"))
```

Customize point appearance or disable points/labels:

```
ctz-draw-path("A--B->C|-|D",
  point-style: "circle",
  point-color: red,
  label-pos: "above")
```

```
ctz-draw-path("A--B->C|-|D",
  points: false,
  labels: false)
```

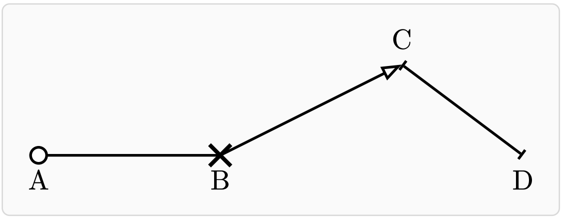
Per-point overrides inside the path:

```
ctz-draw-path("A{below, style: circle}--B{below}->C{above, style: none}|-|D{below}",
  stroke: black)
```

Code

```
#ctz-canvas(length: 0.8cm, {
  import cetz.draw: *
  ctz-init()

  ctz-def-points(
    A: (0, 0), B: (3, 0), C: (6, 1.5), D: (8, 0),
  )

  ctz-draw-path("A{below, style: circle}--B{below}-
>C{above}|-|D{below}",
    stroke: black)
})
```

Figure



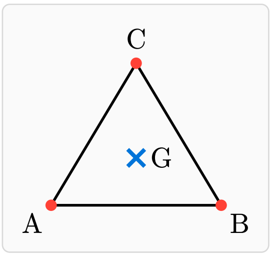
8.6. Global Styling — `ctz-style`

Set default styles for points and labels:

Code

```
#ctz-canvas(length: 0.8cm, {
  import cetz.draw: *
  ctz-init()
  ctz-style(point: (shape: "dot", size: 0.1, fill:
red))

  ctz-def-points(A: (0, 0), B: (3, 0), C: (1.5, 2.5))
  ctz-draw-line("A", "B", "C", "A", stroke: black)
  ctz-draw-points("A", "B", "C")

  ctz-style(point: (shape: "cross", size: 0.15,
stroke: blue + 1.5pt))
  ctz-def-centroid("G", "A", "B", "C")
  ctz-draw-points("G")

  ctz-draw-labels("A", "B", "C", "G",
    A: "below left", B: "below right",
    C: "above", G: "right")
})
```

Figure



Point shapes: `"dot"`, `"cross"`, `"circle"`, `"square"`

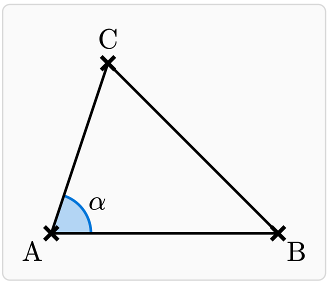
8.7. Angle Marking — `ctz-draw-angle`

Mark and label angles:

Code

```
#ctz-canvas(length: 0.8cm, {
  import cetz.draw: *
  ctz-init()

  ctz-def-points(A: (0, 0), B: (4, 0), C: (1, 3))
  ctz-draw-line("A", "B", "C", "A", stroke: black)

  ctz-draw-angle("A", "B", "C",
    label: $alpha$,
    radius: 0.7,
    fill: blue.lighten(70%),
    stroke: blue)

  ctz-draw-points("A", "B", "C")
  ctz-draw-labels("A", "B", "C",
    A: "below left", B: "below right", C: "above")
})
```

Figure



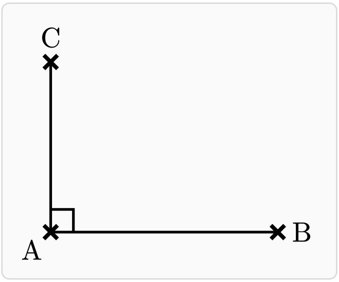
8.8. Right Angle Mark — `ctz-draw-mark-right-angle`

Mark a right angle with a small square:

Code

```
#ctz-canvas(length: 0.8cm, {
  import cetz.draw: *
  ctz-init()

  ctz-def-points(A: (0, 0), B: (4, 0), C: (0, 3))
  ctz-draw-line("A", "B", stroke: black)
  ctz-draw-line("A", "C", stroke: black)

  ctz-draw-mark-right-angle("B", "A", "C", size: 0.4)

  ctz-draw-points("A", "B", "C")
  ctz-draw-labels("A", "B", "C",
    A: "below left", B: "right", C: "above")
})
```

Figure



9. Circles

9.1. Named Circle — `ctz-def-circle` / `ctz-draw-circle`

Define a circle once and draw/label it by name. This uses the “define → draw → label” pattern:

- `ctz-def-circle` stores geometry under a name.
- `ctz-draw-circle` renders it later (you can style it each time).
- `ctz-label-circle` places text relative to the circle without recomputing center/radius.

```
ctz-def-points(O: (0, 0), A: (3, 0))
ctz-def-circle("C1", "O", through: "A")
ctz-draw-circle("C1", stroke: gray)
ctz-label-circle("C1", $C_1$, pos: "above right", dist: 0.2)
```

More label placement controls:

```
ctz-label-circle("C1", $C_1$,
  pos: "above",
  dist: 0.25,
  offset: (0.1, 0))
```

9.2. Circumcircle — `ctz-draw-circumcircle`

Draw the circumscribed circle of a triangle:

```
ctz-draw-circumcircle("A", "B", "C", stroke: blue + 1pt)
```

9.3. Incircle — `ctz-draw-incircle`

Draw the inscribed circle of a triangle:

```
ctz-draw-incircle("A", "B", "C", stroke: green + 1pt)
```

9.4. Circle Through Point — `ctz-draw-circle-through`

Draw a circle with given center passing through a point:

```
ctz-draw-circle-through("O", "A", stroke: blue)
```

9.5. Semicircle — `ctz-draw-semicircle`

Draw a semicircle on a diameter:

```
ctz-draw-semicircle("A", "B", stroke: blue)
```

10. Clipping

Lines that extend infinitely need to be clipped to the visible region.

10.1. Set Clip Region — `ctz-set-clip`

Define a rectangular clip boundary:

```
ctz-set-clip(xmin, ymin, xmax, ymax)
```

10.2. Draw Clipped Line — `ctz-draw-line-global-clip`

Draw a line that is automatically clipped:

```
ctz-draw-line-global-clip("A", "B", add: (2, 2), stroke: blue)
```

The `add` parameter extends the line beyond the two points before clipping.

10.3. Draw Clipped Segment — `ctz-draw-seg-global-clip`

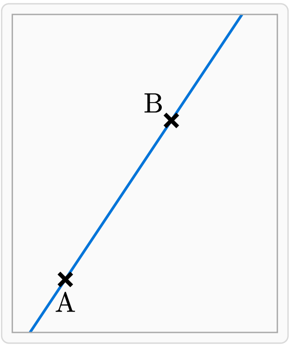
Draw a segment with clipping:

```
ctz-draw-seg-global-clip("A", "B", stroke: red)
```

Code

```
#ctz-canvas(length: 0.7cm, {
  import cetz.draw: *
  ctz-init()

  ctz-def-points(A: (0, 0), B: (2, 3))

  // Set clip boundary
  ctz-set-clip(-1, -1, 4, 5)

  // Draw extended line (clipped)
  ctz-draw-line-global-clip("A", "B", add: (5, 5),
stroke: blue)

  // Show clip boundary
  ctz-show-clip(stroke: gray + 0.5pt)

  ctz-draw-points("A", "B")
  ctz-draw-labels("A", "B", A: "below", B: "above
left")
})
```

Figure



11. Raw Algorithms

For direct computation without the point registry, use the `raw` dictionary:

```
// Direct centroid calculation
let center = raw.ctz-def-centroid((0,0,0), (3,0,0), (1.5,2.5,0))

// Distance between points
let d = raw.dist((0,0,0), (3,4,0))  // Returns 5

// Line-line intersection
let pt = raw.line-line((0,0,0), (1,1,0), (0,1,0), (1,0,0), ray: true)
```

Available raw functions:

- **Intersections**: `line-line`, `line-circle`, `circle-circle`
- **Triangle centers**: `ctz-def-centroid`, `ctz-def-circumcenter`, `ctz-def-incenter`, `ctz-def-orthocenter`, `euler-center`, `ctz-def-lemoine`, etc.
- **Transformations**: `ctz-def-rotation`, `reflection`, `translation`, `ctz-def-homothety`, `projection`
- **Utilities**: `ctz-def-midpoint`, `dist`, `angle-at-vertex`, `triangle-area`, etc.

12. Function Reference

12.1. Point Definition

- `ctz-def-points(A: (x, y), ...)` — Define named points
- `ctz-def-line(name, a, b)` — Named line from two points
- `ctz-def-circle(name, center, radius|through)` — Named circle
- `ctz-def-polygon(name, a, b, c, ...)` — Named polygon
- `ctz-def-midpoint(name, a, b)` — Midpoint of segment
- `ctz-def-linear(name, a, b, k)` — Point at ratio k along line
- `ctz-def-regular-polygon([name,] names, center, first)` — Regular n-gon vertices (optionally register polygon name)
- `ctz-def-point-on-circle(name, center, radius, angle)` — Point on circle at angle
- `ctz-def-equilateral(name, a, b)` — Third vertex of equilateral triangle
- `ctz-def-golden(name, a, b)` — Golden ratio point

12.2. Line Constructions

- `ctz-def-perp(n1, n2, line, through)` — Perpendicular through point
- `ctz-def-para(n1, n2, line, through)` — Parallel through point
- `ctz-def-bisect(n1, n2, a, vertex, c)` — Angle bisector
- `ctz-def-mediator(n1, n2, a, b)` — Perpendicular bisector

12.3. Intersections

- `ctz-def-ll(name, line1, line2)` — Line-line intersection
- `ctz-def-lc(names, line, circle)` — Line-circle intersections
- `ctz-def-cc(names, circle1, circle2)` — Circle-circle intersections

12.4. Triangle Centers

- `ctz-def-centroid`, `ctz-def-circumcenter`, `ctz-def-incenter`, `ctz-def-orthocenter`
- `ctz-def-euler`, `ctz-def-lemoine`, `ctz-def-nagel`, `ctz-def-gergonne`, `ctz-def-spieker`
- `ctz-def-feuerbach`, `ctz-def-mittenpunkt`, `ctz-def-excenter`

12.5. Special Triangles

- `ctz-def-medial-triangle(na, nb, nc, a, b, c)` — Medial triangle
- `ctz-def-orthic-triangle(na, nb, nc, a, b, c)` — Orthic triangle
- `ctz-def-intouch-triangle(na, nb, nc, a, b, c)` — Intouch triangle

12.6. Transformations

- `ctz-def-rotation(name, source, center, angle)` — Rotation
- `ctz-def-reflect(name, source, line-a, line-b)` — Reflection
- `ctz-def-translate(name, source, vector)` — Translation
- `ctz-def-homothety(name, source, center, factor)` — Homothety
- `ctz-def-project(name, source, line-a, line-b)` — Projection

12.7. Drawing

- `ctz-draw-points(names...)` — Draw points
- `ctz-draw-labels(names, placements)` — Label points
- `ctz-style(point: (...))` — Set styling
- `ctz-draw-angle(vertex, a, b, ...)` — Mark angle
- `ctz-draw-mark-right-angle(a, vertex, c, ...)` — Right angle mark
- `ctz-draw-segment(a, b, ...)` — Draw segment
- `ctz-draw-measure-segment(a, b, ...)` — Offset measurement with fences and label
- `ctz-draw-path(spec, ...)` — Draw path with per-segment tips
- `ctz-draw-polygon(points...)` — Draw polygon (triangle, quadrilateral, etc.)

- `ctz-draw-fill-polygon(points...)` — Fill polygon
- `ctz-draw-regular-polygon(names, ...)` — Draw regular polygon by vertex names
- `ctz-draw-fill-regular-polygon(names, ...)` — Fill regular polygon by vertex names
- `ctz-draw-circle(name, ...)` — Draw named circle
- `ctz-label-circle(name, label, ...)` — Label named circle
- `ctz-label-polygon(name, label, ...)` — Label named polygon
- `ctz-draw-circumcircle(a, b, c, ...)` — Circumscribed circle
- `ctz-draw-incircle(a, b, c, ...)` — Inscribed circle

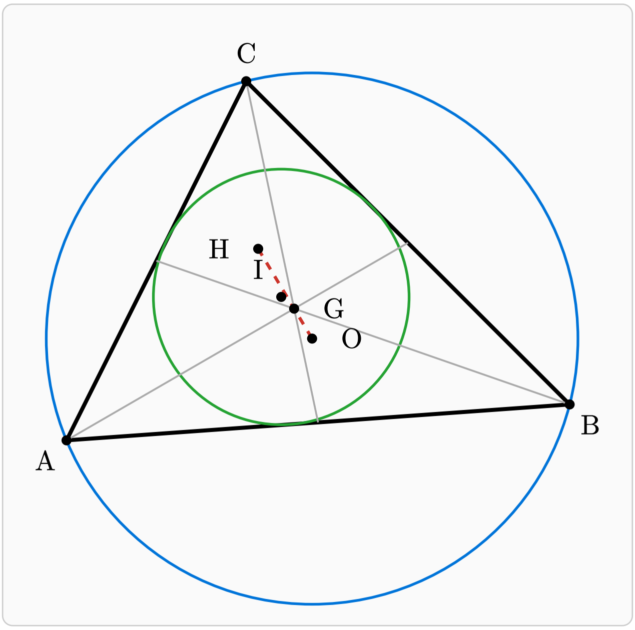
12.8. Clipping

- `ctz-set-clip(xmin, ymin, xmax, ymax)` — Set clip region
- `ctz-clear-clip()` — Clear clip region
- `ctz-draw-line-global-clip(a, b, ...)` — Draw clipped line
- `ctz-draw-seg-global-clip(a, b, ...)` — Draw clipped segment

13. Gallery Examples

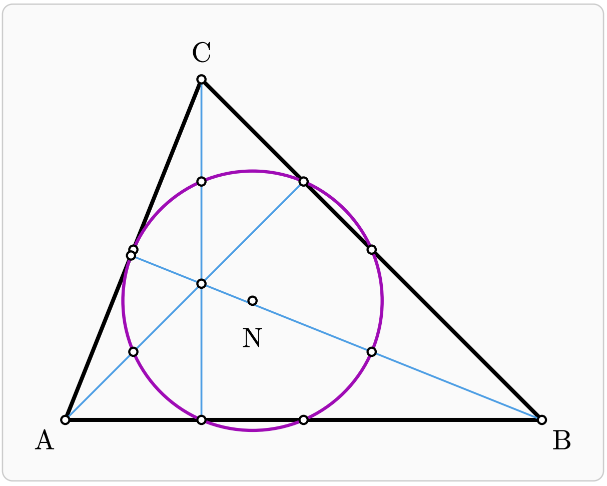
The following pages showcase advanced geometric constructions using ctz-euclide. Each example demonstrates different features and techniques.

Triangle Centers



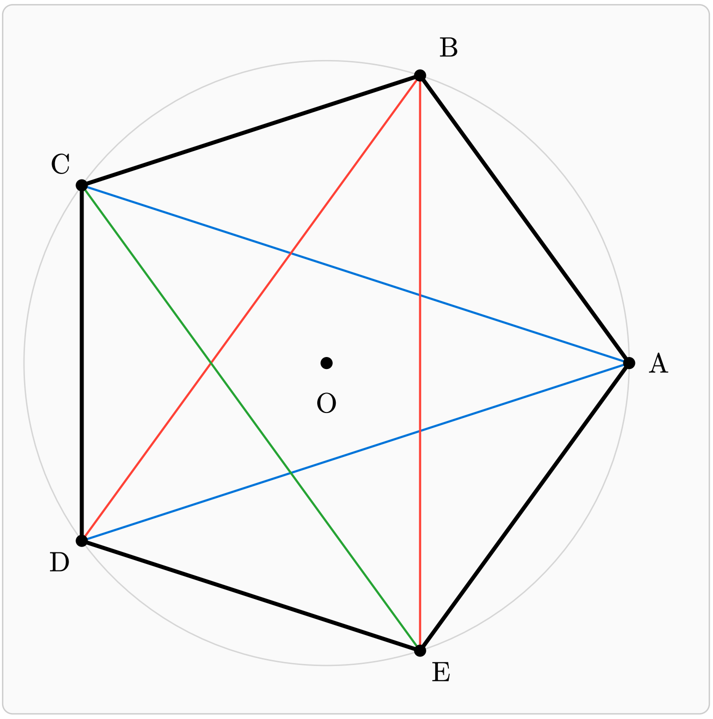
```
#ctz-canvas(length: 0.95cm, {
  import cetz.draw: *
  ctz-init()
  ctz-style(point: (shape: "dot", size: 0.07, fill: black))

  ctz-def-points(A: (0, 0), B: (7, 0.5), C: (2.5, 5))
  ctz-draw-line("A", "B", "C", "A", stroke: black + 1.5pt)

  ctz-def-centroid("G", "A", "B", "C")
  ctz-def-circumcenter("O", "A", "B", "C")
  ctz-def-incenter("I", "A", "B", "C")
  ctz-def-orthocenter("H", "A", "B", "C")

  // Euler line (H, G, O are collinear)
  ctz-draw-line("H", "O", stroke: (
    paint: red.darken(20%),
    dash: "dashed",
    thickness: 1.2pt,
  ))

  // Circumcircle and incircle
  ctz-draw-circle-through("O", "A", stroke: blue + 1pt)
  ctz-draw-incircle("A", "B", "C", stroke: green.darken(20%) + 1pt)

  // Draw medians to centroid
  ctz-def-midpoint("Ma", "B", "C")
  ctz-def-midpoint("Mb", "A", "C")
  ctz-def-midpoint("Mc", "A", "B")
  ctz-draw-line("A", "Ma", stroke: gray + 0.7pt)
  ctz-draw-line("B", "Mb", stroke: gray + 0.7pt)
  ctz-draw-line("C", "Mc", stroke: gray + 0.7pt)

  ctz-draw-points("A", "B", "C", "G", "O", "I", "H")
  ctz-draw-labels(
    "A", "B", "C", "G", "O", "I", "H",
    A: "below left",
    B: "below right",
    C: "above",
    G: "right",
    O: "below",
    I: "right",
    H: "left",
  )
})
```

Nine-Point Circle



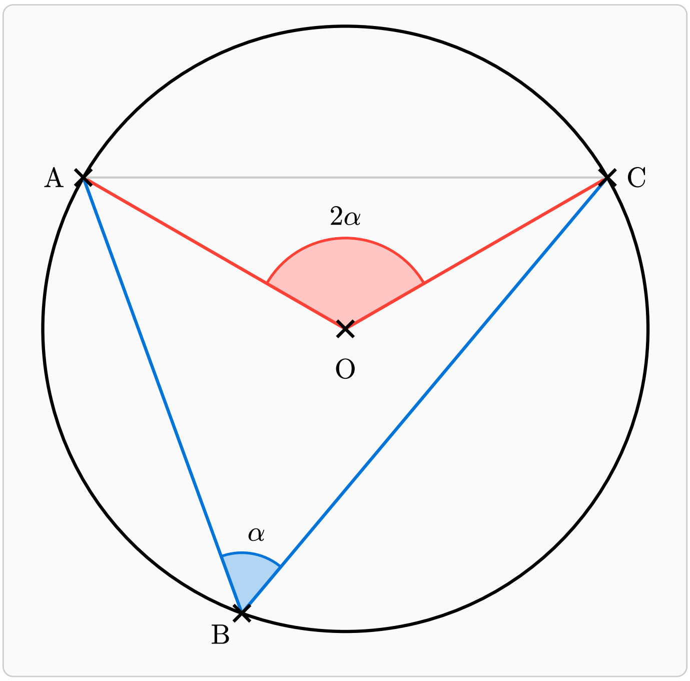
```
#ctz-canvas(length: 0.9cm, {
  import cetz.draw: *
  ctz-init()
  ctz-style(point: (shape: "circle", size: 0.06, stroke: black + 0.8pt, fill: white))

  ctz-def-points(A: (0, 0), B: (7, 0), C: (2, 5))
  ctz-draw-line("A", "B", "C", "A", stroke: black + 1.5pt)

  // Midpoints of sides
  ctz-def-midpoint("Ma", "B", "C")
  ctz-def-midpoint("Mb", "A", "C")
  ctz-def-midpoint("Mc", "A", "B")

  // Feet of altitudes
  ctz-def-project("Ha", "A", "B", "C")
  ctz-def-project("Hb", "B", "A", "C")
  ctz-def-project("Hc", "C", "A", "B")

  // Orthocenter and midpoints to vertices
  ctz-def-orthocenter("H", "A", "B", "C")
  ctz-def-midpoint("MHa", "H", "A")
  ctz-def-midpoint("MHb", "H", "B")
  ctz-def-midpoint("MHc", "H", "C")

  // Nine-point circle passes through all 9 points
  ctz-def-euler("N", "A", "B", "C")
  ctz-draw-circle-through("N", "Ma", stroke: purple.darken(10%) + 1.2pt)

  // Draw altitudes
  ctz-draw-line("A", "Ha", stroke: blue.lighten(30%) + 0.7pt)
  ctz-draw-line("B", "Hb", stroke: blue.lighten(30%) + 0.7pt)
  ctz-draw-line("C", "Hc", stroke: blue.lighten(30%) + 0.7pt)

  ctz-draw-points("A", "B", "C", "N", "Ma", "Mb", "Mc", "Ha", "Hb", "Hc", "MHa", "MHb", "MHc", "H")
  ctz-draw-labels(
    "A", "B", "C", "N",
    A: "left",
    B: "right",
    C: "above",
    N: "below",
  )
})
```

Regular Pentagon



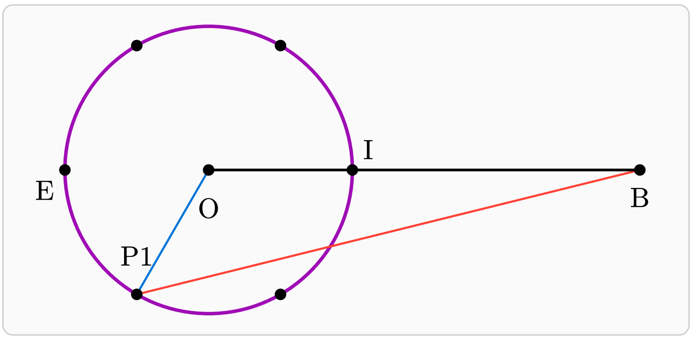
```
#ctz-canvas(length: 1cm, {
  import cetz.draw: *
  ctz-init()
  ctz-style(point: (shape: "dot", size: 0.08, fill: black))

  ctz-def-points(O: (0, 0), V1: (4, 0))
  ctz-def-regular-polygon("Pent", ("A", "B", "C", "D", "E"), "O", "V1")

  // Pentagon
  ctz-draw-polygon("Pent", stroke: black + 1.5pt)

  // All diagonals
  ctz-draw-line("A", "C", stroke: blue + 0.8pt)
  ctz-draw-line("A", "D", stroke: blue + 0.8pt)
  ctz-draw-line("B", "D", stroke: red + 0.8pt)
  ctz-draw-line("B", "E", stroke: red + 0.8pt)
  ctz-draw-line("C", "E", stroke: green.darken(20%) + 0.8pt)

  // Center
  ctz-draw-circle-r("O", 4, stroke: gray.lighten(50%) + 0.5pt)

  ctz-draw-points("A", "B", "C", "D", "E", "O")
  ctz-draw-labels(
    "A", "B", "C", "D", "E", "O",
    A: "right",
    B: (pos: "above right", offset: (0.1, 0.1)),
    C: "above left",
    D: "below left",
    E: "below right",
    O: "below",
  )
})
```

Inscribed Angle Theorem



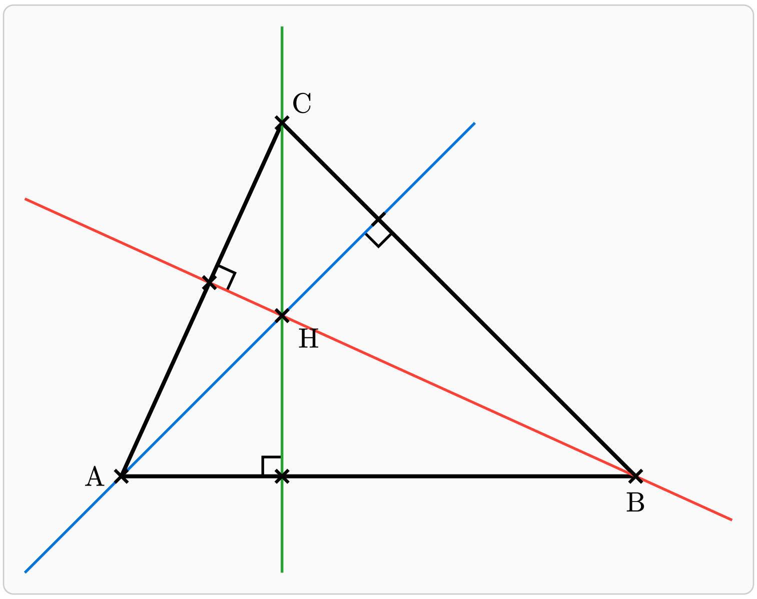
```
#ctz-canvas(length: 1cm, {
  import cetz.draw: *
  ctz-init()
  ctz-style(point: (shape: "cross", size: 0.11, stroke: black + 1.2pt))

  ctz-def-points(O: (0, 0), R: (4, 0))
  ctz-draw-circle-r("O", 4, stroke: black + 1.2pt)

  // Place points on circle
  ctz-def-rotation("A", "R", "O", 150)
  ctz-def-rotation("C", "R", "O", 30)
  ctz-def-rotation("B", "R", "O", 250)

  // Inscribed angle at B
  ctz-draw-line("A", "B", stroke: blue + 1.2pt)
  ctz-draw-line("B", "C", stroke: blue + 1.2pt)
  ctz-draw-angle("B", "A", "C", label: $alpha$, radius: 0.8,
    fill: blue.lighten(70%), stroke: blue)

  // Central angle at O (twice the inscribed angle)
  ctz-draw-line("O", "A", stroke: red + 1.2pt)
  ctz-draw-line("O", "C", stroke: red + 1.2pt)
  ctz-draw-angle("O", "A", "C", label: $2alpha$, radius: 1.2,
    fill: red.lighten(70%), stroke: red)

  ctz-draw-line("A", "C", stroke: gray.lighten(40%) + 0.8pt)
  ctz-draw-points("O", "A", "B", "C")
  ctz-draw-labels(
    "O", "A", "B", "C",
    O: "below",
    A: "left",
    B: "right",
    C: "above",
  )
})
```

Apollonius Circle



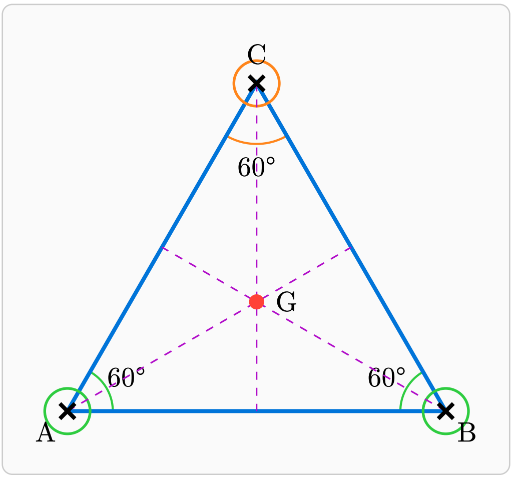
```
#ctz-canvas(length: 0.95cm, {
  import cetz.draw: *
  ctz-init()
  ctz-style(point: (shape: "dot", size: 0.08, fill: black))

  ctz-def-points(A: (-3, 0), B: (3, 0))

  // Apollonius circle: locus of points P where PA/PB = k
  let k = 2

  // External and internal division points
  ctz-def-points(E: (-5, 0), I: (-1, 0))

  // Center is midpoint of E and I
  ctz-def-midpoint("O", "E", "I")
  ctz-draw-circle-through("O", "E", stroke: purple.darken(10%) + 1.3pt)

  // Show some points on the circle
  ctz-def-rotation("P1", "E", "O", 60)
  ctz-def-rotation("P2", "E", "O", 120)
  ctz-def-rotation("P3", "E", "O", -60)
  ctz-def-rotation("P4", "E", "O", -120)

  // For P1: PA/PB = k = 2
  ctz-draw-line("P1", "A", stroke: blue + 0.8pt)
  ctz-draw-line("P1", "B", stroke: red + 0.8pt)
  ctz-draw-line("A", "B", stroke: black + 1pt)

  ctz-draw-points( "B", "O", "E", "I", "P1", "P2", "P3", "P4")
  ctz-draw-labels(
     "B", "E", "I", "P1", "O",
    B: "below right",
    E: "left",
    I: "above right",
    P1: "above",
    O: "below",
  )
})
```

Orthocenter and Altitudes



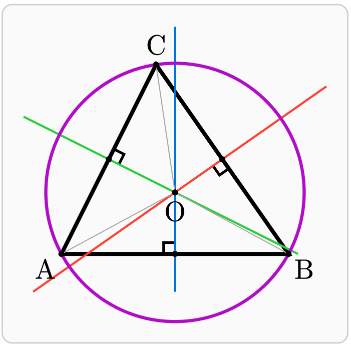
```
#ctz-canvas(length: 0.85cm, {
  import cetz.draw: *
  ctz-init()
  ctz-style(point: (shape: "cross", size: 0.1, stroke: black + 1.2pt))

  ctz-def-points(A: (0, 0), B: (8, 0), C: (2.5, 5.5))
  ctz-set-clip(-1.5, -1.5, 9.5, 7)

  // Triangle
  ctz-draw-line("A", "B", "C", "A", stroke: black + 1.5pt)

  // Extended altitudes (automatically clipped)
  ctz-def-perp("Ha1", "Ha2", ("B", "C"), "A")
  ctz-def-perp("Hb1", "Hb2", ("A", "C"), "B")
  ctz-def-perp("Hc1", "Hc2", ("A", "B"), "C")

  ctz-draw-line-global-clip("A", "Ha1", add: (1, 1.5), stroke: blue + 1pt)
  ctz-draw-line-global-clip("B", "Hb1", add: (1, 1.5), stroke: red + 1pt)
  ctz-draw-line-global-clip("C", "Hc1", add: (1, 2.5), stroke: green.darken(20%) + 1pt)

  // Orthocenter (intersection of altitudes)
  ctz-def-orthocenter("H", "A", "B", "C")

  // Feet of altitudes
  ctz-def-project("Ha", "A", "B", "C")
  ctz-def-project("Hb", "B", "A", "C")
  ctz-def-project("Hc", "C", "A", "B")

  ctz-draw-mark-right-angle("A", "Ha", "B", size: 0.3)
  ctz-draw-mark-right-angle("B", "Hb", "C", size: 0.3)
  ctz-draw-mark-right-angle("C", "Hc", "A", size: 0.3)

  ctz-draw-points("A", "B", "C", "H", "Ha", "Hb", "Hc")
  ctz-draw-labels(
    "A", "B", "C", "H",
    A: "left",
    B: "below",
    C: "above right",
    H: "right",
  )
})
```

Equilateral Triangle Construction



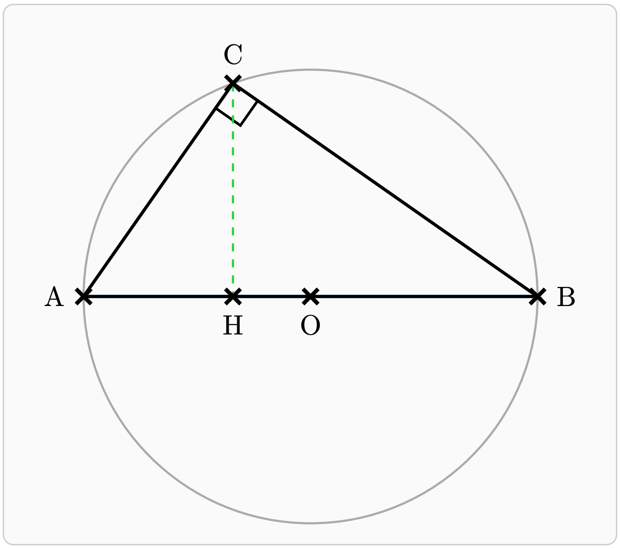
```
#ctz-canvas({
  import cetz.draw: *
  ctz-init()
  ctz-style(point: (shape: "cross", size: 0.1, stroke: black + 1.5pt))

  // Base of equilateral triangle
  ctz-def-points("A", (0, 0), "B", (5, 0))
  ctz-def-equilateral("C", "A", "B")

  // Draw triangle
  ctz-draw-line("A", "B", "C", "A", stroke: blue + 1.5pt)

  // Mark 60° angles
  ctz-draw-angle("A", "B", "C", label: $60°$, radius: 0.6, stroke: green + 0.8pt)
  ctz-draw-angle("B", "C", "A", label: $60°$, radius: 0.6, stroke: green + 0.8pt)
  ctz-draw-angle("C", "A", "B", label: $60°$, radius: 0.8, stroke: orange + 0.8pt)

  // Draw circles at vertices
  ctz-draw-circle-r("A", 0.3, stroke: green + 1pt)
  ctz-draw-circle-r("B", 0.3, stroke: green + 1pt)
  ctz-draw-circle-r("C", 0.3, stroke: orange + 1pt)

  // In equilateral triangle, all centers coincide
  ctz-def-centroid("G", "A", "B", "C")

  // Draw medians/altitudes
  ctz-def-midpoint("Ma", "B", "C")
  ctz-def-midpoint("Mb", "A", "C")
  ctz-def-midpoint("Mc", "A", "B")
  ctz-draw-line("A", "Ma", stroke: (paint: purple, thickness: 0.6pt, dash: "dashed"))
  ctz-draw-line("B", "Mb", stroke: (paint: purple, thickness: 0.6pt, dash: "dashed"))
  ctz-draw-line("C", "Mc", stroke: (paint: purple, thickness: 0.6pt, dash: "dashed"))

  ctz-draw-points("A", "B", "C", "G")
  ctz-draw-labels(
    "A", "B", "C", "G",
    A: "left",
    B: "right",
    C: "above",
    G: "below",
  )
})
```

Perpendicular Bisectors and Circumcircle



```
#ctz-canvas({
  import cetz.draw: *
  ctz-init()
  ctz-style(point: (shape: "dot", size: 0.08, fill: black))

  // Define triangle
  ctz-def-points("A", (0, 0), "B", (6, 0), "C", (2.5, 5))
  ctz-set-clip(-1, -1, 7, 6)

  // Calculate midpoints
  ctz-def-midpoint("Mab", "A", "B")
  ctz-def-midpoint("Mbc", "B", "C")
  ctz-def-midpoint("Mca", "C", "A")

  // Circumcenter is intersection of perpendicular bisectors
  ctz-def-circumcenter("O", "A", "B", "C")

  // Create perpendicular bisector lines
  ctz-def-mediator("Pab1", "Pab2", "A", "B")
  ctz-def-mediator("Pbc1", "Pbc2", "B", "C")
  ctz-def-mediator("Pca1", "Pca2", "C", "A")

  // Draw triangle
  ctz-draw-line("A", "B", "C", "A", stroke: black + 1.5pt)

  // Draw perpendicular bisectors (clipped)
  ctz-draw-seg-global-clip("Pab1", "Pab2", stroke: blue + 0.8pt)
  ctz-draw-seg-global-clip("Pbc1", "Pbc2", stroke: red + 0.8pt)
  ctz-draw-seg-global-clip("Pca1", "Pca2", stroke: green + 0.8pt)

  // Draw circumcircle
  ctz-draw-circumcircle("A", "B", "C", stroke: purple + 1.2pt)

  // Radii (equal length)
  ctz-draw-line("O", "A", stroke: gray + 0.4pt)
  ctz-draw-line("O", "B", stroke: gray + 0.4pt)
  ctz-draw-line("O", "C", stroke: gray + 0.4pt)

  ctz-draw-mark-right-angle("A", "Mab", "O", size: 0.3)
  ctz-draw-mark-right-angle("B", "Mbc", "O", size: 0.3)
  ctz-draw-mark-right-angle("C", "Mca", "O", size: 0.3)

  ctz-draw-points("A", "B", "C", "O", "Mab", "Mbc", "Mca")
  ctz-draw-labels(
    "A", "B", "C", "O",
    A: "left",
    B: "right",
    C: "above",
    O: "below",
  )
})
```

Thales' Theorem



```
#ctz-canvas({
  import cetz.draw: *
  ctz-init()
  ctz-style(point: (shape: "cross", size: 0.1, stroke: black + 1.5pt))

  // Circle with diameter AB
  ctz-def-points("O", (0, 0), "A", (-3, 0), "B", (3, 0))

  // Point C on circle
  ctz-def-point-on-circle("C", "O", 3, 110)

  // Draw circle and diameter
  ctz-draw-circle-r("O", 3, stroke: gray + 0.8pt)
  ctz-draw-line("A", "B", stroke: blue + 1.2pt)

  // Draw triangle ACB
  // By Thales' theorem: angle ACB = 90° (inscribed in semicircle)
  ctz-draw-line("A", "C", "B", "A", stroke: black + 1.2pt)

  // Mark the right angle at C
  ctz-draw-mark-right-angle("A", "C", "B", size: 0.4)

  // Draw altitude from C to AB
  ctz-def-project("H", "C", "A", "B")
  ctz-draw-line("C", "H", stroke: (paint: green, thickness: 0.8pt, dash: "dashed"))

  ctz-draw-points("A", "B", "C", "O", "H")
  ctz-draw-labels(
    "A", "B", "C", "O", "H",
    A: "left",
    B: "right",
    C: "above",
    O: "below",
    H: "below left",
  )
})
```