ctz-euclide

Typst Port

Euclidean Geometry for Typst

A comprehensive geometry package built on CeTZ
Version 0.1.0

Contents

1. Introduction 4
   1.1. Features 4
   1.2. Installation 4
   1.3. Basic Usage 4
2. Core Concepts 5
   2.1. The Point Registry 5
   2.2. Figure Scaling 5
   2.3. Coordinate Systems 5
3. Point Definitions 6
   3.1. Basic Points — `pts` 6
   3.2. Midpoint — `midpoint` 6
   3.3. Regular Polygons — `regular-polygon` 6
   3.4. Linear Combination — `linear` 6
4. Line Constructions 7
   4.1. Perpendicular — `perp` 7
   4.2. Parallel — `para` 7
   4.3. Angle Bisector — `bisect` 7
   4.4. Perpendicular Bisector — `mediator` 8
5. Intersections 9
   5.1. Line–Line — `ll` 9
   5.2. Line–Circle — `lc` 9
   5.3. Circle–Circle — `cc` 9
6. Triangle Centers 11
   6.1. Basic Centers 11
      6.1.1. Centroid — `centroid` 11
      6.1.2. Circumcenter — `circumcenter` 11
      6.1.3. Incenter — `incenter` 11
      6.1.4. Orthocenter — `orthocenter` 12
   6.2. The Euler Line 12
   6.3. Advanced Centers 14
7. Transformations 15
   7.1. Rotation — `rotate` 15
   7.2. Reflection — `reflect` 15
   7.3. Homothety (Scaling) — `scale` 15
   7.4. Projection — `project` 16
8. Drawing & Styling 17
   8.1. Points — `points` 17
   8.2. Labels — `labels` 17
   8.3. Global Styling — `style` 17
   8.4. Angle Marking — `angle` 17
      8.4.1. Angle Label Positioning 18
   8.5. Segment Marks — `mark-segment` 18
   8.6. Right Angle Marks — `mark-right-angle` 19
9. Grid & Axes 20
   9.1. Basic Grid — `grid` 20
   9.2. Axes — `axes` 20
   9.3. Grid with Subdivisions 20
10. Clipping 22
   10.1. Global Clipping (Recommended) 22
   10.2. Manual Clipping 22
11. API Reference 23

11.1. Initialization 23
11.2. Point Definitions 23
11.3. Line Constructions 23
11.4. Intersections 23
11.5. Triangle Centers 23
11.6. Special Triangles 24
11.7. Transformations 24
11.8. Drawing 24
11.9. Marking & Annotation 24
11.10. Styling 24
11.11. Grid & Axes 24
11.12. Clipping 25
12. Figures Gallery 26

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 "@preview/cetz:0.4.2" as cetz
#import "@preview/ctz-euclide:0.1.0" as ctz-lib

#let ctz = ctz-lib.create-api(cetz)
```

All figures must begin with:

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

  // Your geometry code here
})
```

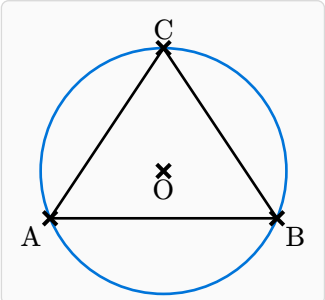
The `(ctz.init)()` call initializes the point registry and coordinate resolver.

1.3. Basic Usage

Code

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

  // Define points
  (ctz.pts)(A: (0, 0), B: (4, 0), C: (2, 3))

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

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

  // Draw and label points
  (ctz.points)("A", "B", "C", "O")
  (ctz.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.pts)(A: (0, 0), B: (3, 4))  // Register points A and B
line("A", "B")                    // Use them directly in CeTZ
```

Under the hood, `(ctz.init)()` installs a coordinate resolver that translates `"A"` to the stored coordinates. Both `"A"` and `"tkz:A"` resolve to the same point.

2.2. Figure Scaling

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

```
#cetz.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.pts)(A: (2, 3))

// Using existing CeTZ coordinates
(ctz.pts)(B: (rel: (1, 1), to: "A"))

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

3. Point Definitions

3.1. Basic Points — `pts`

Define one or more points at specific coordinates:

```
(ctz.pts)(A: (0, 0), B: (4, 0), C: (2, 3))
```

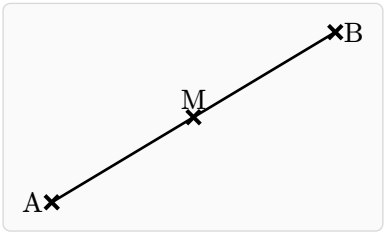
3.2. Midpoint — `midpoint`

Find the midpoint of a segment:

Code

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

  (ctz.pts)(A: (0, 0), B: (5, 3))
  (ctz.midpoint)("M", "A", "B")

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

Figure



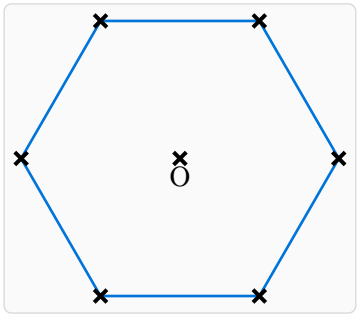
3.3. Regular Polygons — `regular-polygon`

Generate vertices of a regular n-gon:

Code

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

  (ctz.pts)(O: (0, 0), A: (3, 0))
  (ctz.regular-polygon)(("A", "B", "C", "D", "E",
"F"), "O", "A")

  line("A", "B", "C", "D", "E", "F", "A", stroke:
blue)
  (ctz.points)("A", "B", "C", "D", "E", "F", "O")
  (ctz.labels)("O", O: "below")
})
```

Figure



3.4. Linear Combination — `linear`

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

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

4. Line Constructions

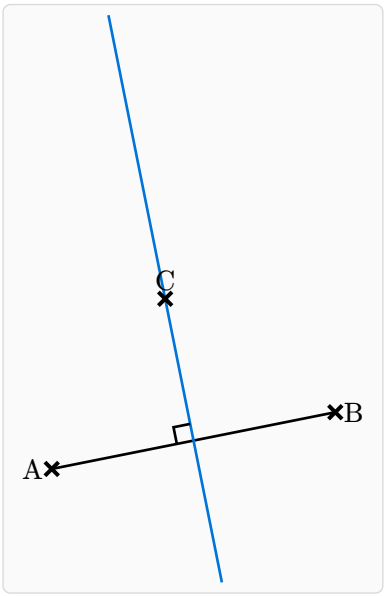
4.1. Perpendicular — `perp`

Construct a perpendicular line through a point:

Code

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

  (ctz.pts)(A: (0, 0), B: (5, 1), C: (2, 3))
  (ctz.perp)("P1", "P2", ("A", "B"), "C")
  (ctz.project)("H", "C", "A", "B")

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

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

Figure



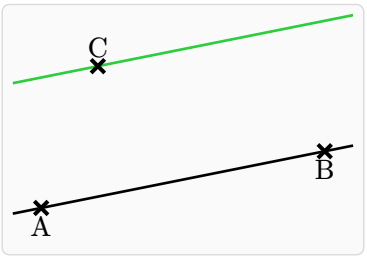
4.2. Parallel — `para`

Construct a parallel line through a point:

Code

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

  (ctz.pts)(A: (0, 0), B: (5, 1), C: (1, 2.5))
  (ctz.para)("P1", "P2", ("A", "B"), "C")

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

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

Figure



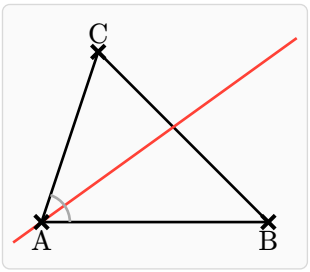
4.3. Angle Bisector — `bisect`

Construct the bisector of an angle:

Code

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

  (ctz.pts)(A: (0, 0), B: (4, 0), C: (1, 3))
  (ctz.bisect)("D1", "D2", "C", "A", "B")

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

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

Figure



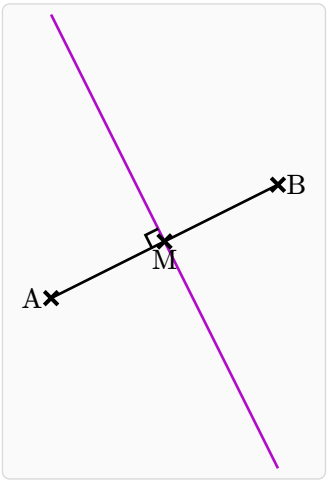
4.4. Perpendicular Bisector — `mediator`

Construct the perpendicular bisector of a segment:

Code

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

  (ctz.pts)(A: (1, 1), B: (5, 3))
  (ctz.mediator)("M1", "M2", "A", "B")
  (ctz.midpoint)("M", "A", "B")

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

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

Figure



5. Intersections

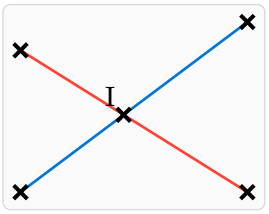
5.1. Line–Line — `ll`

Find the intersection of two lines:

Code

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

  (ctz.pts)(A: (0, 0), B: (4, 3),
            C: (4, 0), D: (0, 2.5))
  (ctz.ll)("I", ("A", "B"), ("C", "D"))

  line("A", "B", stroke: blue)
  line("C", "D", stroke: red)

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

Figure



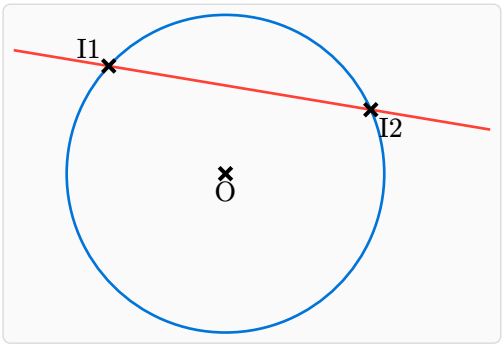
5.2. Line–Circle — `lc`

Find intersections of a line with a circle:

Code

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

  (ctz.pts)(O: (0, 0), R: (3, 0),
            A: (-2, 2), B: (4, 1))
  (ctz.lc)(("I1", "I2"), ("A", "B"),
    (center: "O", through: "R"))

  circle("O", "R", stroke: blue)
  (ctz.set-clip)(-4, -4, 5, 4)
  (ctz.line)("A", "B", add: (2, 2), stroke: red)

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

Figure



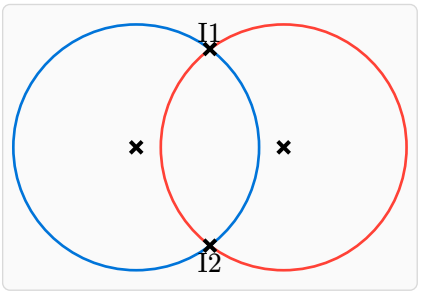
5.3. Circle–Circle — `cc`

Find intersections of two circles:

Code

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

  (ctz.pts)(O1: (0, 0), O2: (3, 0),
            R1: (2.5, 0), R2: (5.5, 0))
  (ctz.cc)(("I1", "I2"),
    (center: "O1", through: "R1"),
    (center: "O2", through: "R2"))

  circle("O1", "R1", stroke: blue)
  circle("O2", "R2", stroke: red)

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

Figure



6. Triangle Centers

6.1. Basic Centers

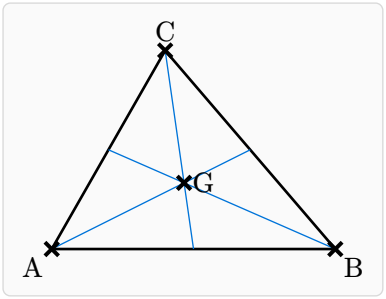
6.1.1. Centroid — `centroid`

The intersection of medians (center of mass):

Code

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

  (ctz.pts)(A: (0, 0), B: (5, 0), C: (2, 3.5))
  (ctz.centroid)("G", "A", "B", "C")

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

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

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

Figure



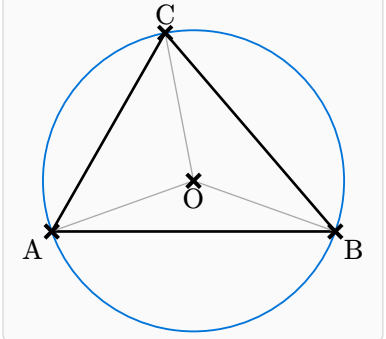
6.1.2. Circumcenter — `circumcenter`

Center of the circumscribed circle:

Code

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

  (ctz.pts)(A: (0, 0), B: (5, 0), C: (2, 3.5))
  (ctz.circumcenter)("O", "A", "B", "C")

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

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

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

Figure



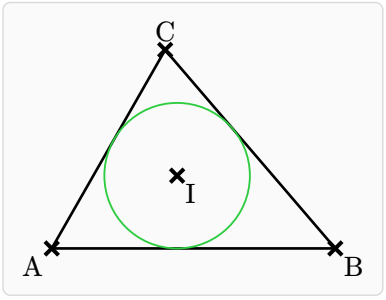
6.1.3. Incenter — `incenter`

Center of the inscribed circle:

Code

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

  (ctz.pts)(A: (0, 0), B: (5, 0), C: (2, 3.5))
  (ctz.incenter)("I", "A", "B", "C")

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

  // Angle bisectors
  (ctz.bisect)("Ba1", "Ba2", "I", "B", "C")
  (ctz.bisect)("Bb1", "Bb2", "I", "A", "C")
  (ctz.bisect)("Bc1", "Bc2", "I", "A", "B")

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

Figure



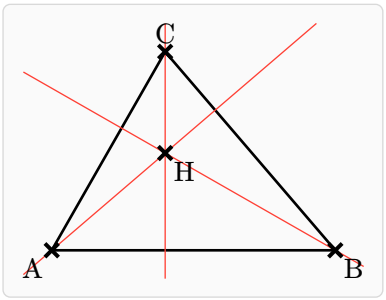
6.1.4. Orthocenter — `orthocenter`

Intersection of altitudes:

Code

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

  (ctz.pts)(A: (0, 0), B: (5, 0), C: (2, 3.5))
  (ctz.orthocenter)("H", "A", "B", "C")

  // Altitudes (extended as lines)
  (ctz.perp)("Ha1", "Ha2", ("B", "C"), "A")
  (ctz.perp)("Hb1", "Hb2", ("A", "C"), "B")
  (ctz.perp)("Hc1", "Hc2", ("A", "B"), "C")

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

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

Figure



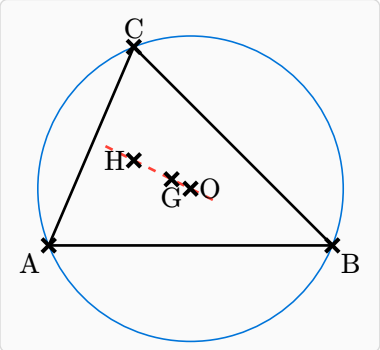
6.2. The Euler Line

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

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

  (ctz.pts)(A: (0, 0), B: (5, 0), C: (1.5, 3.5))

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

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

  (ctz.points)("A", "B", "C", "H", "G", "O")
  (ctz.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:

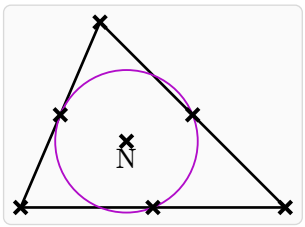
- `lemoine` — Symmedian point (Lemoine point)
- `nagel` — Nagel point
- `gergonne` — Gergonne point
- `spieker` — Spieker center (incenter of medial triangle)
- `euler` — Nine-point circle center
- `feuerbach` — Feuerbach point
- `mittenpunkt` — Mittenpunkt
- `excenter` — Excenter (specify vertex: `"a"`,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

Example with Euler (nine-point) circle:

Code

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

  (ctz.pts)(A: (0, 0), B: (5, 0), C: (1.5, 3.5))
  (ctz.euler)("N", "A", "B", "C")

  line("A", "B", "C", "A", stroke: black)

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

  circle("N", "Ma", stroke: purple + 0.7pt)

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

Figure



7. Transformations

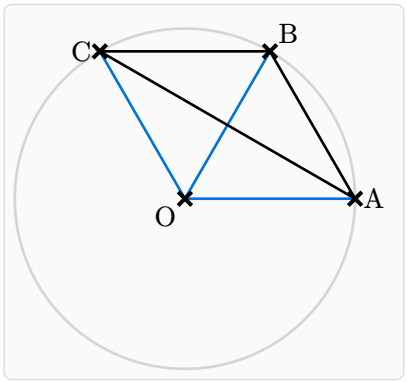
7.1. Rotation — `rotate`

Rotate a point around a center:

Code

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

  (ctz.pts)(O: (2, 2), A: (5, 2))
  (ctz.rotate)("B", "A", "O", 60)
  (ctz.rotate)("C", "A", "O", 120)

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

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

Figure



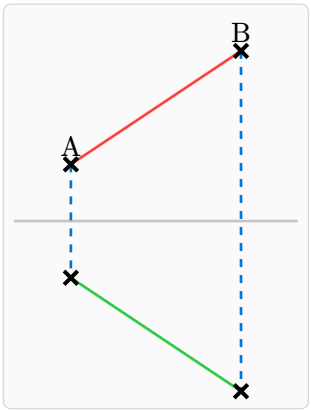
7.2. Reflection — `reflect`

Reflect a point across a line:

Code

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

  (ctz.pts)(A: (1, 1), B: (4, 3),
            L1: (0, 0), L2: (5, 0))
  (ctz.reflect)("Ap", "A", "L1", "L2")
  (ctz.reflect)("Bp", "B", "L1", "L2")

  line("L1", "L2", stroke: gray.lighten(30%))
  line("A", "Ap", stroke: (paint: blue, dash:
"dashed"))
  line("B", "Bp", stroke: (paint: blue, dash:
"dashed"))
  line("A", "B", stroke: red)
  line("Ap", "Bp", stroke: green)

  (ctz.points)("A", "B", "Ap", "Bp")
  (ctz.labels)("A", "B",
    A: "above", B: "above")
})
```

Figure



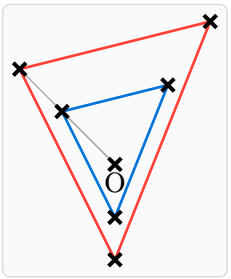
7.3. Homothety (Scaling) — `scale`

Scale a point from a center:

Code

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

  (ctz.pts)(O: (2, 2),
            A: (1, 3), B: (3, 3.5), C: (2, 1))
  (ctz.scale)("Ap", "A", "O", 1.8)
  (ctz.scale)("Bp", "B", "O", 1.8)
  (ctz.scale)("Cp", "C", "O", 1.8)

  line("A", "B", "C", "A", stroke: blue)
  line("Ap", "Bp", "Cp", "Ap", stroke: red)

  line("O", "A", stroke: gray + 0.5pt)
  line("O", "Ap", stroke: gray + 0.5pt)

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

Figure



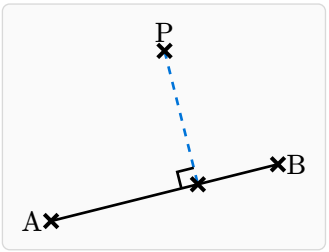
7.4. Projection — `project`

Project a point onto a line:

Code

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

  (ctz.pts)(A: (1, 1), B: (5, 2),
            P: (3, 4))
  (ctz.project)("Pp", "P", "A", "B")

  line("A", "B", stroke: black)
  line("P", "Pp", stroke: (paint: blue, dash:
"dashed"))
  (ctz.mark-right-angle)("P", "Pp", "A", size: 0.3)

  (ctz.points)("A", "B", "P", "Pp")
  (ctz.labels)("A", "B", "P",
    A: "left", B: "right", P: "above")
})
```

Figure



8. Drawing & Styling

8.1. Points — `points`

Draw point markers:

```
(ctz.points)("A", "B", "C")
```

8.2. Labels — `labels`

Add labels with automatic positioning:

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

Position keywords: `"above"`, `"below"`, `"left"`, `"right"`, and combinations like `"above left"`.

For fine control, use offset tuples:

```
(ctz.labels)("A",
  A: (pos: "below", offset: (0.1, -0.2)))
```

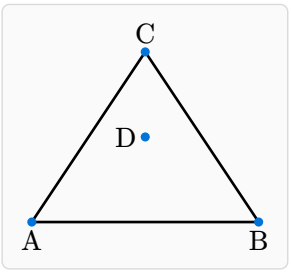
8.3. Global Styling — `style`

Set default appearance for all elements:

Code

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

  // Set global point style
  (ctz.style)(point: (
    shape: "circle",
    size: 0.08,
    fill: blue,
    stroke: none
  ))

  (ctz.pts)(A: (0, 0), B: (4, 0),
            C: (2, 3), D: (2, 1.5))
  (ctz.polygon)("A", "B", "C", stroke: black)

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

Figure



Point shapes: `"cross"`, `"dot"`, `"circle"`, `"plus"`, `"square"`, `"diamond"`, `"triangle"`.

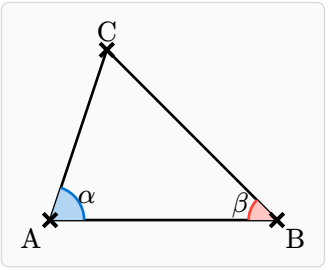
8.4. Angle Marking — `angle`

Mark and label angles:

Code

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

  (ctz.pts)(A: (0, 0), B: (4, 0), C: (1, 3))
  line("A", "B", "C", "A", stroke: black)

  (ctz.angle)("A", "B", "C",
    label: $alpha$,
    radius: 0.6,
    fill: blue.lighten(70%),
    stroke: blue)

  (ctz.angle)("B", "C", "A",
    label: $beta$,
    radius: 0.5,
    fill: red.lighten(70%),
    stroke: red)

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

Figure



8.4.1. Angle Label Positioning

The angle label is automatically placed along the **angle bisector** (the line that divides the angle in half). You can fine-tune the label position using:

- `radius`: Distance from vertex to the arc
- `label-radius`: Distance from vertex to the label (default: `radius + 0.2`)
- `label-offset`: A tuple `(along, perp)` for fine positioning:
  - `along`: Moves label along the bisector (positive = farther from vertex, negative = closer)
  - `perp`: Moves label perpendicular to bisector (positive = counterclockwise, negative = clockwise)

The offset is relative to the bisector's coordinate system, not the canvas axes.

Example with offset:

```
(ctz.angle)("A", "B", "C",
  label: $alpha$,
  radius: 0.6,
  label-radius: 0.9,      // Label at distance 0.9 from vertex
  label-offset: (0.1, 0.15),  // Shift slightly outward and counterclockwise
  fill: blue.lighten(70%))
```

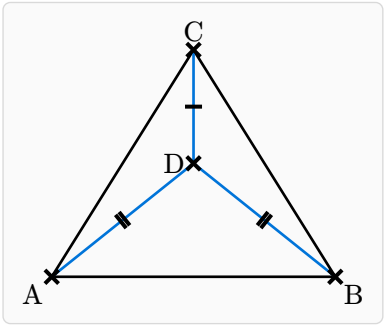
8.5. Segment Marks — `mark-segment`

Mark equal segments with tick marks:

Code

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

  (ctz.pts)(A: (0, 0), B: (5, 0),
            C: (2.5, 4), D: (2.5, 2))
  line("A", "B", "C", "A", stroke: black)
  line("C", "D", stroke: blue)
  line("D", "A", stroke: blue)
  line("D", "B", stroke: blue)

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

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

Figure



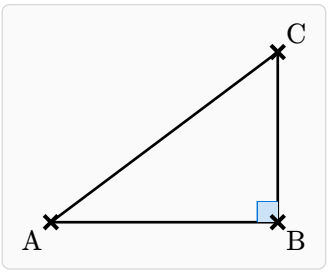
8.6. Right Angle Marks — `mark-right-angle`

Mark 90° angles:

Code

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

  (ctz.pts)(A: (0, 0), B: (4, 0), C: (4, 3))
  line("A", "B", "C", "A", stroke: black)

  (ctz.mark-right-angle)("A", "B", "C",
    size: 0.35,
    color: blue,
    fill: blue.lighten(80%))

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

Figure



9. Grid & Axes

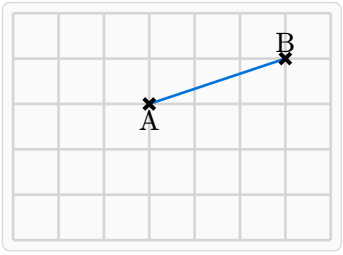
9.1. Basic Grid — `grid`

Draw a coordinate grid:

Code

```
#cetz.canvas(length: 0.6cm, {
  import cetz.draw: *
  (ctz.init)()

  (ctz.grid)(xmin: -2, xmax: 5,
             ymin: -1, ymax: 4,
             stroke: gray.lighten(50%))

  (ctz.pts)(A: (1, 2), B: (4, 3))
  line("A", "B", stroke: blue + 1pt)

  (ctz.points)("A", "B")
  (ctz.labels)("A", "B",
    A: "below", B: "above")
})
```

Figure



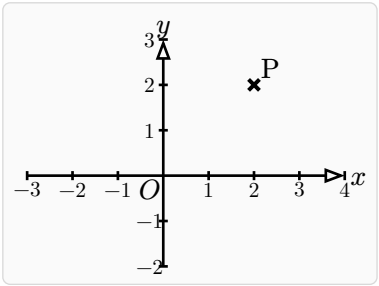
9.2. Axes — `axes`

Draw X and Y axes with labels:

Code

```
#cetz.canvas(length: 0.6cm, {
  import cetz.draw: *
  (ctz.init)()

  (ctz.axes)(xmin: -3, xmax: 4,
             ymin: -2, ymax: 3,
             x-label: $x$,
             y-label: $y$,
             origin-label: $0$)

  (ctz.pts)(P: (2, 2))
  (ctz.points)("P")
  (ctz.labels)("P", P: "above right")
})
```

Figure



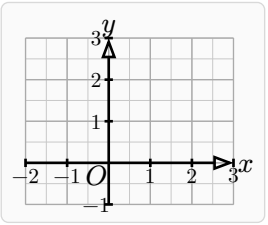
9.3. Grid with Subdivisions

Add finer sub-grid lines:

Code

```
#cetz.canvas(length: 0.55cm, {
  import cetz.draw: *
  (ctz.init)()

  (ctz.grid)(xmin: -2, xmax: 3,
             ymin: -1, ymax: 3,
             sub: true,
             sub-xstep: 0.5,
             sub-ystep: 0.5,
             sub-stroke: luma(200) + 0.25pt,
             stroke: gray + 0.5pt)

  (ctz.axes)(xmin: -2, xmax: 3,
             ymin: -1, ymax: 3,
             labels: true,
             ticks: true)
})
```

Figure



10. Clipping

When drawing extended lines (like altitudes or angle bisectors that go beyond triangle sides), you may want to clip them to a viewing region. `ctz-euclide` provides mathematical line clipping using the Cohen-Sutherland algorithm.

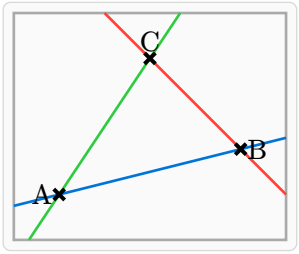
10.1. Global Clipping (Recommended)

Set a clip region once, then all `line` commands automatically clip:

Code

```
#cetz.canvas(length: 0.6cm, {
  import cetz.draw: *
  (ctz.init)()

  (ctz.pts)(A: (0, 1), B: (4, 2), C: (2, 4))

  // Set global clip region
  (ctz.set-clip)(-1, 0, 5, 5)
  (ctz.show-clip)(stroke: gray)

  // These lines auto-clip to bounds
  (ctz.line)("A", "B", add: (5, 5), stroke: blue)
  (ctz.line)("B", "C", add: (5, 5), stroke: red)
  (ctz.line)("C", "A", add: (5, 5), stroke: green)

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

Figure



API:

- `(ctz.set-clip)(xmin, ymin, xmax, ymax)` — Set global clip region
- `(ctz.show-clip)(stroke: ...)` — Draw the clip boundary
- `(ctz.line)(a, b, add: (n, m), stroke: ...)` — Extended line (auto-clips)
- `(ctz.seg)(a, b, stroke: ...)` — Simple segment (auto-clips)
- `(ctz.clear-clip)()` — Remove clipping

10.2. Manual Clipping

For per-line control, specify bounds directly:

```
(ctz.clipped-line-add)("A", "B", xmin, ymin, xmax, ymax,
  add: (10, 10), stroke: blue)
```

11. API Reference

11.1. Initialization

`(ctz.init)()` Initialize the point registry and coordinate resolver. **Required** at the start of every figure.

11.2. Point Definitions

`(ctz.pts)(...)` Define points with named arguments: `(ctz.pts)(A: (0, 0), B: (3, 4))`

`(ctz.midpoint)(name, a, b)` Midpoint of segment AB

`(ctz.linear)(name, a, b, k)` Point P = A + k(B - A)

`(ctz.barycentric)(name, a, b, c, wa, wb, wc)` Barycentric combination

`(ctz.regular-polygon)(names, center, first)` Regular polygon vertices

`(ctz.point-on-circle)(name, center, radius, angle)` Point on circle at angle (degrees)

`(ctz.equilateral)(name, a, b)` Third vertex of equilateral triangle

`(ctz.square)(c, d, a, b)` Complete a square given two vertices

`(ctz.golden)(name, a, b)` Golden ratio point on AB

11.3. Line Constructions

`(ctz.perp)(p1, p2, (a, b), through)` Perpendicular to line AB through point

`(ctz.para)(p1, p2, (a, b), through)` Parallel to line AB through point

`(ctz.bisect)(p1, p2, a, vertex, c)` Angle bisector at vertex

`(ctz.mediator)(p1, p2, a, b)` Perpendicular bisector of AB

11.4. Intersections

`(ctz.ll)(name, (a, b), (c, d))` Line-line intersection

`(ctz.lc)(names, (a, b), circle)` Line-circle intersections. Circle: `(center: "O", through: "A")` or `(center: "O", radius: r)`

`(ctz.cc)(names, circle1, circle2)` Circle-circle intersections

11.5. Triangle Centers

`(ctz.centroid)(name, a, b, c)` Center of mass (medians intersection)

`(ctz.circumcenter)(name, a, b, c)` Circumscribed circle center

`(ctz.incenter)(name, a, b, c)` Inscribed circle center

`(ctz.orthocenter)(name, a, b, c)` Altitudes intersection

`(ctz.euler)(name, a, b, c)` Nine-point circle center

`(ctz.lemoine)(name, a, b, c)` Lemoine point (symmedian point)

`(ctz.nagel)(name, a, b, c)` Nagel point

`(ctz.gergonne)(name, a, b, c)` Gergonne point

`(ctz.spieker)(name, a, b, c)` Spieker center

`(ctz.feuerbach)(name, a, b, c)` Feuerbach point

`(ctz.mittenpunkt)(name, a, b, c)` Mittenpunkt

`(ctz.excenter)(name, a, b, c, vertex: "a")` Excenter opposite to vertex

11.6. Special Triangles

`(ctz.medial)(ma, mb, mc, a, b, c)` Medial triangle (midpoints)

`(ctz.orthic)(ha, hb, hc, a, b, c)` Orthic triangle (feet of altitudes)

`(ctz.intouch)(ta, tb, tc, a, b, c)` Intouch triangle (incircle tangency points)

11.7. Transformations

`(ctz.rotate)(name, source, center, angle)` Rotate point around center (degrees)

`(ctz.reflect)(name, source, a, b)` Reflect point across line AB

`(ctz.translate)(name, source, vector)` Translate by vector `(dx, dy)` or `(a, b)`

`(ctz.scale)(name, source, center, factor)` Homothety (scale from center)

`(ctz.project)(name, source, a, b)` Orthogonal projection onto line AB

`(ctz.symmetry)(name, source, center)` Central symmetry

`(ctz.inversion)(name, source, center, radius)` Circle inversion

11.8. Drawing

`(ctz.points)(...)` Draw point markers: `(ctz.points)("A", "B", "C")`

`(ctz.labels)(...)` Label points with positioning: `(ctz.labels)("A", "B", A: "left", B: "right")`

`(ctz.polygon)(...)` Draw polygon: `(ctz.polygon)("A", "B", "C", close: true, stroke: black)`

`(ctz.segment)(a, b, ...)` Draw segment with optional arrows: `arrows: "<->"`, `dim: $5$`

`(ctz.line-add)(a, b, add: (n, m), ...)` Extended line beyond AB

`(ctz.circle-r)(center, radius, ...)` Circle with explicit radius

`(ctz.circle-through)(center, through, ...)` Circle through point

`(ctz.circle-diameter)(a, b, ...)` Circle with AB as diameter

`(ctz.circumcircle)(a, b, c, ...)` Circumscribed circle of triangle

`(ctz.incircle)(a, b, c, ...)` Inscribed circle of triangle

`(ctz.semicircle)(a, b, above: true, ...)` Semicircle with AB as diameter

`(ctz.sector)(center, start, end, ...)` Circular sector

`(ctz.arc)(center, start, end, ...)` Arc through two points

`(ctz.arc-r)(center, radius, start-angle, end-angle, ...)` Arc with explicit angles

11.9. Marking & Annotation

`(ctz.angle)(vertex, a, b, ...)` Mark angle at vertex. Options: `label`, `radius`, `fill`, `stroke`

`(ctz.mark-segment)(a, b, mark: n, ...)` Tick marks for equal segments (n = 1, 2, 3, …)

`(ctz.mark-right-angle)(a, vertex, c, ...)` Mark 90° angle

`(ctz.fill-angle)(vertex, a, c, ...)` Fill angle without border

`(ctz.label-segment)(a, b, label: $5$, ...)` Label a segment

11.10. Styling

`(ctz.style)(...)` Set global defaults. Example: `(ctz.style)(point: (shape: "dot", size: 0.08))`

11.11. Grid & Axes

`(ctz.grid)(...)` Draw grid. Options: `xmin`, `xmax`, `ymin`, `ymax`, `xstep`, `ystep`, `sub`, `stroke`

`(ctz.axes)(...)` Draw axes with labels and ticks

`(ctz.hline)(y, xmin: ..., xmax: ..., stroke: ...)` Horizontal line

`(ctz.vline)(x, ymin: ..., ymax: ..., stroke: ...)` Vertical line

`(ctz.text)(x, y, content, ...)` Place text at coordinates

11.12. Clipping

`(ctz.set-clip)(xmin, ymin, xmax, ymax)` Set global clip region

`(ctz.show-clip)(stroke: ...)` Draw clip boundary

`(ctz.line)(a, b, add: (n, m), stroke: ...)` Auto-clipped extended line

`(ctz.seg)(a, b, stroke: ...)` Auto-clipped segment

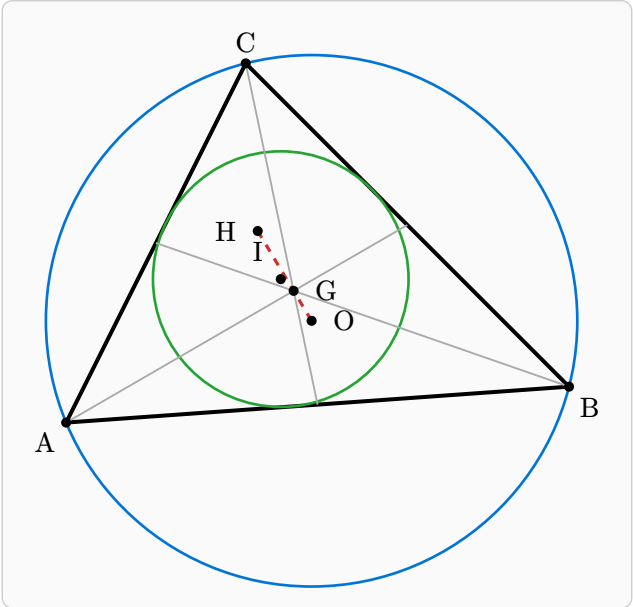
`(ctz.clear-clip)()` Remove global clipping

`(ctz.clipped-line-add)(a, b, xmin, ymin, xmax, ymax, ...)` Manual per-line clipping

12. Figures Gallery

The following pages showcase various geometric constructions and their capabilities.

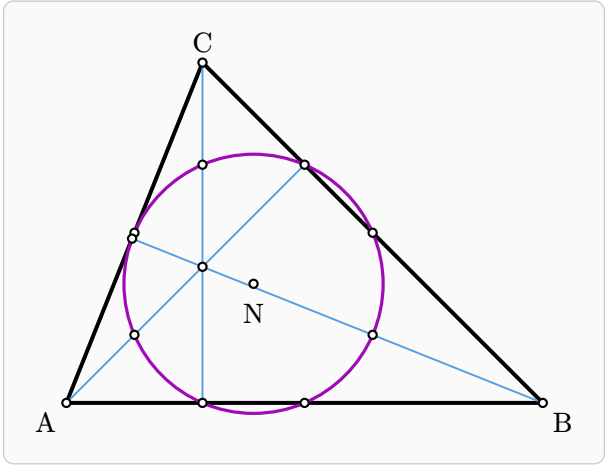
Complete Triangle Centers



Euler Line (red dashed): $O - G - H$
Circumcircle (blue), Incircle (green), Medians (gray)

```
(ctz.pts)(A: (0, 0), B: (7, 0.5), C: (2.5, 5))
(ctz.centroid)("G", "A", "B", "C")
(ctz.circumcenter)("O", "A", "B", "C")
(ctz.incenter)("I", "A", "B", "C")
(ctz.orthocenter)("H", "A", "B", "C")

line("A", "B", "C", "A", stroke: black + 1.5pt)
line("H", "O", stroke: (paint: red, dash: "dashed"))
circle("O", "A", stroke: blue + 1pt)
(ctz.incircle)("A", "B", "C", stroke: green + 1pt)
```

Nine-Point (Euler) Circle

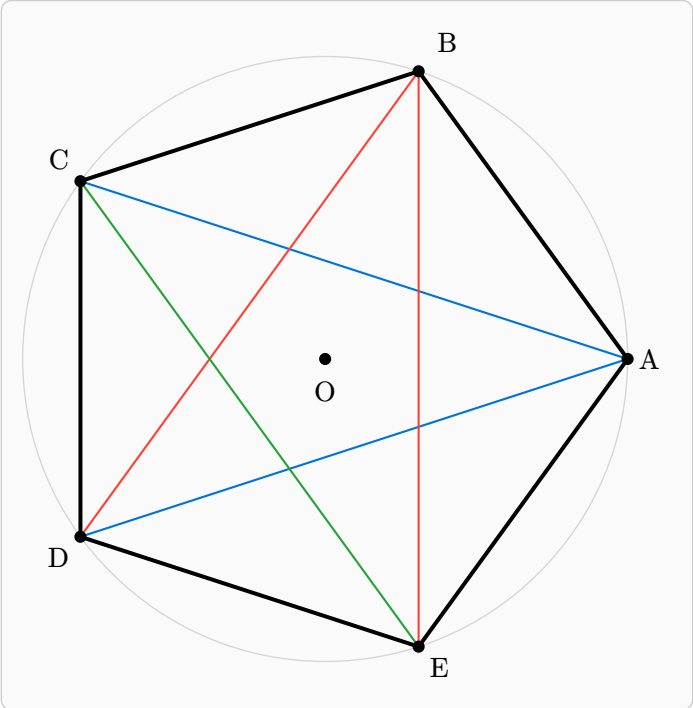


The nine-point circle passes through:

- Midpoints of sides (M_a, M_b, M_c)
- Feet of altitudes (H_a, H_b, H_c)
- Midpoints from H to vertices

```
(ctz.pts)(A: (0, 0), B: (7, 0), C: (2, 5))
(ctz.midpoint)("Ma", "B", "C")
(ctz.midpoint)("Mb", "A", "C")
(ctz.midpoint)("Mc", "A", "B")
(ctz.project)("Ha", "A", "B", "C")
(ctz.project)("Hb", "B", "A", "C")
(ctz.project)("Hc", "C", "A", "B")
(ctz.orthocenter)("H", "A", "B", "C")
(ctz.midpoint)("MHa", "H", "A")
(ctz.midpoint)("MHb", "H", "B")
(ctz.midpoint)("MHc", "H", "C")
(ctz.euler)("N", "A", "B", "C")
circle("N", "Ma", stroke: purple.darken(10%) + 1.2pt)
```

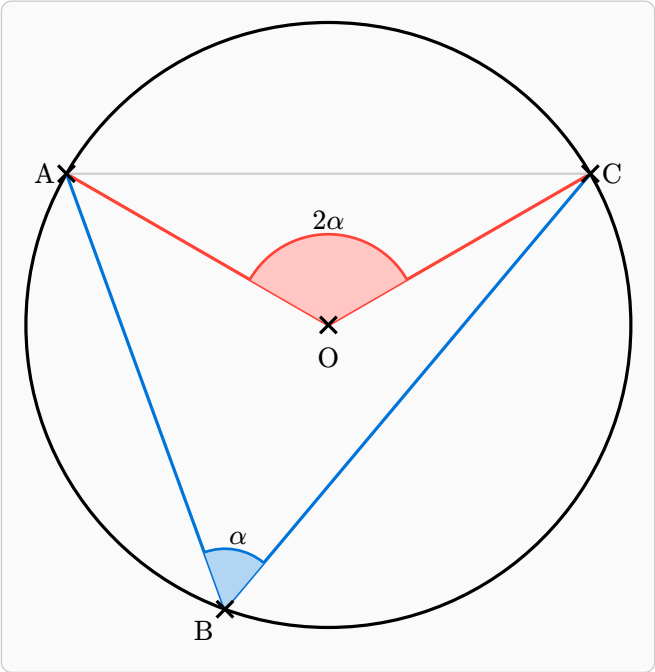
Regular Pentagon with Diagonals



Regular pentagon with all diagonals forming a pentagram

```
(ctz.pts)(O: (0, 0), V1: (4, 0))
(ctz.regular-polygon)(("A", "B", "C", "D", "E"), "O", "V1")
line("A", "B", "C", "D", "E", "A", stroke: black + 1.5pt)
line("A", "C", stroke: blue + 0.8pt)
line("A", "D", stroke: blue + 0.8pt)
line("B", "D", stroke: red + 0.8pt)
line("B", "E", stroke: red + 0.8pt)
line("C", "E", stroke: green.darken(20%) + 0.8pt)
circle("O", radius: 4, stroke: gray.lighten(50%) + 0.5pt)
```

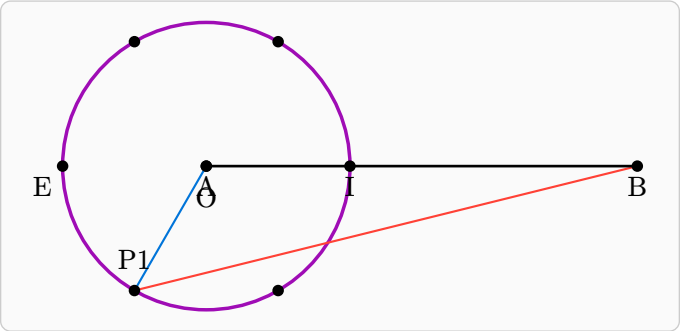
Inscribed Angle Theorem



Inscribed angle theorem: $\angle ABC = \frac{1}{2}\angle AOC$

```
(ctz.pts)(O: (0, 0), R: (4, 0))
circle("O", radius: 4, stroke: black + 1.2pt)
(ctz.rotate)("A", "R", "O", 150)
(ctz.rotate)("C", "R", "O", 30)
(ctz.rotate)("B", "R", "O", 250)
line("A", "B", stroke: blue + 1.2pt)
line("B", "C", stroke: blue + 1.2pt)
(ctz.angle)("B", "A", "C", label: $alpha$, radius: 0.8,
  fill: blue.lighten(70%), stroke: blue)
line("O", "A", stroke: red + 1.2pt)
line("O", "C", stroke: red + 1.2pt)
(ctz.angle)("O", "A", "C", label: $2alpha$, radius: 1.2,
  fill: red.lighten(70%), stroke: red)
```

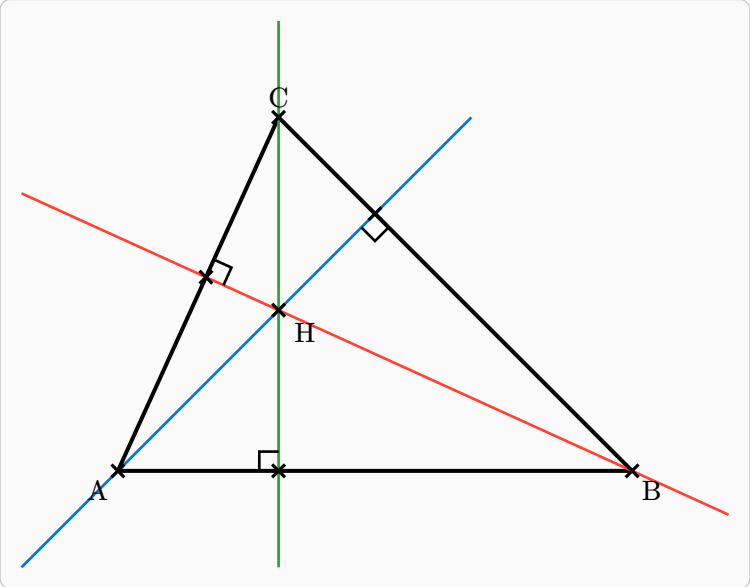
Geometric Locus: Apollonius Circle



Apollonius circle: locus of points P where $P\frac{A}{P}B = 2$
Points E (external) and I (internal) divide AB in ratio $2:1$

```
(ctz.pts)(A: (-3, 0), B: (3, 0))
let k = 2
(ctz.pts)(E: (-5, 0))
(ctz.pts)(I: (-1, 0))
(ctz.midpoint)("O", "E", "I")
circle("O", "E", stroke: purple.darken(10%) + 1.3pt)
(ctz.rotate)("P1", "E", "O", 60)
line("P1", "A", stroke: blue + 0.8pt)
line("P1", "B", stroke: red + 0.8pt)
line("A", "B", stroke: black + 1pt)
```

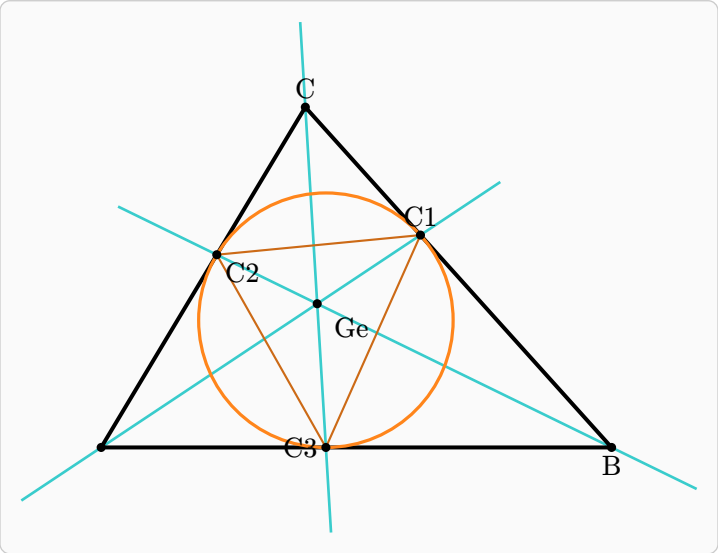
Orthocenter with Extended Altitudes



Altitudes extended beyond the triangle, clipped to viewing region

```
(ctz.pts)(A: (0, 0), B: (8, 0), C: (2.5, 5.5))
(ctz.set-clip)(-1.5, -1.5, 9.5, 7)
line("A", "B", "C", "A", stroke: black + 1.5pt)
(ctz.perp)("Ha1", "Ha2", ("B", "C"), "A")
(ctz.perp)("Hb1", "Hb2", ("A", "C"), "B")
(ctz.perp)("Hc1", "Hc2", ("A", "B"), "C")
(ctz.line)("A", "Ha1", add: (1, 1.5), stroke: blue + 1pt)
(ctz.line)("B", "Hb1", add: (1, 1.5), stroke: red + 1pt)
(ctz.line)("C", "Hc1", add: (1, 2.5), stroke: green.darken(20%) + 1pt)
(ctz.orthocenter)("H", "A", "B", "C")
(ctz.mark-right-angle)("A", "Ha", "B", size: 0.3)
```

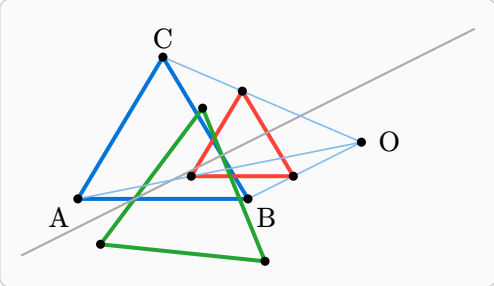
Gergonne Point and Contact Triangle



Gergonne point G_e: concurrence of lines from vertices to contact points of incircle with opposite sides

```
(ctz.pts)(A: (0, 0), B: (9, 0), C: (3.6, 6))
(ctz.set-clip)(-1.5, -1.5, 10.5, 8)
(ctz.gergonne)("Ge", "A", "B", "C")
(ctz.intouch)("C1", "C2", "C3", "A", "B", "C")
(ctz.line)("A", "C1", add: (0.25, 0.25), stroke: teal + 1pt)
(ctz.line)("B", "C2", add: (0.25, 0.25), stroke: teal + 1pt)
(ctz.line)("C", "C3", add: (0.25, 0.25), stroke: teal + 1pt)
line("A", "B", "C", "A", stroke: black + 1.5pt)
(ctz.incircle)("A", "B", "C", stroke: orange + 1.2pt)
line("C1", "C2", "C3", "C1", stroke: orange.darken(20%) + 0.8pt)
```

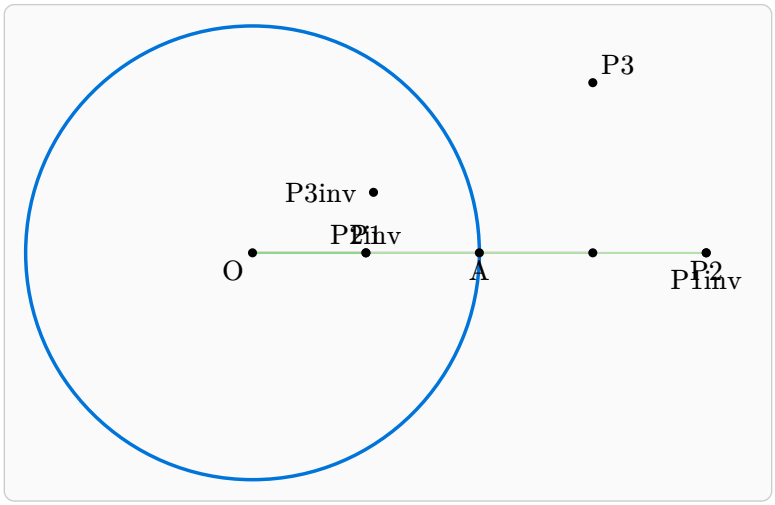
Homothety and Symmetry Transformations



Blue: original $\triangle ABC$ Red: homothety (center O, ratio 0.6)
Green: reflection across gray line

```
(ctz.pts)(A: (0, 0), B: (3, 0), C: (1.5, 2.5))
(ctz.pts)(O: (5, 1))
(ctz.scale)("Ah", "A", "O", 0.6)
(ctz.scale)("Bh", "B", "O", 0.6)
(ctz.scale)("Ch", "C", "O", 0.6)
(ctz.pts)(L1: (-1, -1), L2: (7, 3))
(ctz.reflect)("As", "A", "L1", "L2")
(ctz.reflect)("Bs", "B", "L1", "L2")
(ctz.reflect)("Cs", "C", "L1", "L2")
line("A", "B", "C", "A", stroke: blue + 1.5pt)
line("Ah", "Bh", "Ch", "Ah", stroke: red + 1.5pt)
line("As", "Bs", "Cs", "As", stroke: green.darken(20%) + 1.5pt)
```

Circle Inversion Transformation



Circle inversion: $OP \cdot OP' = r^2$ where r is the inversion radius
Points outside the circle map to inside, and vice versa

```
(ctz.pts)(O: (0, 0), A: (4, 0), P: (6, 0))
circle("O", "A", stroke: blue + 1.3pt)
(ctz.pts)(P1: (2, 0), P2: (8, 0), P3: (6, 3))
(ctz.inversion)("P1inv", "P1", "O", 4)
(ctz.inversion)("P2inv", "P2", "O", 4)
(ctz.inversion)("P3inv", "P3", "O", 4)
line("O", "P", stroke: gray + 0.8pt)
line("O", "P1", stroke: red.lighten(40%) + 0.7pt, stroke-dasharray: (3, 3))
line("O", "P2", stroke: green.lighten(40%) + 0.7pt, stroke-dasharray: (3, 3))
line("P1", "P1inv", stroke: red.lighten(60%) + 0.5pt)
line("P2", "P2inv", stroke: green.lighten(60%) + 0.5pt)
```