

ctz-euclide

Typst Port

Euclidean Geometry for Typst

A comprehensive geometry package built on CeTZ
Version 0.1.0
Nathan Scheinmann

Contents

1. Introduction	5
1.1. Features	5
1.2. Installation	5
1.3. Basic Usage	5
2. Core Concepts	6
2.1. The Point Registry	6
2.2. Figure Scaling	6
2.3. Coordinate Systems	6
3. Point Definitions	6
3.1. Basic Points — <code>ctz-def-points</code>	6
3.2. Midpoint — <code>ctz-def-midpoint</code>	6
3.3. Regular Polygons — <code>ctz-def-regular-polygon</code>	7
3.4. Named Polygons — <code>ctz-def-polygon</code> / <code>ctz-label-polygon</code>	7
3.5. Linear Combination — <code>ctz-def-linear</code>	7
4. Line Constructions	8
4.1. Perpendicular — <code>ctz-def-perp</code>	8
4.2. Parallel — <code>ctz-def-para</code>	8
4.3. Angle Bisector — <code>ctz-def-bisect</code>	8
4.4. Perpendicular Bisector — <code>ctz-def-mediator</code>	9
5. Intersections	10
5.1. Line–Line — <code>ctz-def-ll</code>	10
5.2. Line–Circle — <code>ctz-def-lc</code>	10
5.3. Circle–Circle — <code>ctz-def-cc</code>	10
6. Triangle Centers	12
6.1. Basic Centers	12
6.1.1. Centroid — <code>ctz-def-centroid</code>	12
6.1.2. Circumcenter — <code>ctz-def-circumcenter</code>	12
6.1.3. Incenter — <code>ctz-def-incenter</code>	13
6.1.4. Orthocenter — <code>ctz-def-orthocenter</code>	13
6.2. The Euler Line	13
6.3. Right Triangles via Thales’ Theorem	14
6.4. Advanced Centers	15
7. Transformations	16
7.1. Rotation — <code>rotate</code>	16
7.2. Reflection — <code>ctz-def-reflect</code>	16
7.3. Homothety (Scaling) — <code>scale</code>	16
7.4. Projection — <code>ctz-def-project</code>	17
7.5. Inversion — <code>ctz-def-inversion</code>	17
7.6. Object Duplication — <code>ctz-duplicate</code>	18
7.7. Polymorphic Rotation	18
8. Drawing & Styling	20
8.1. Points — <code>ctz-draw-points</code>	20
8.2. Unified Drawing — <code>ctz-draw</code>	20
8.2.1. Drawing Named Objects	20
8.2.2. Drawing Unnamed Constructs	20
8.2.2.1. Points with Labels	20
8.2.2.2. Paths and Polylines	20
8.2.2.3. Circles	21
8.2.2.4. Conics (Ellipses, Parabolas, Projectiles)	21
8.2.2.5. Arcs and Semicircles	21
8.2.2.6. Line Segments	21

8.2.3. Complete Example	22
8.3. Labels — <code>ctz-draw-labels</code>	22
8.4. Labels & Points — Unified API (Recommended)	22
8.5. Segments — <code>ctz-draw-segment</code>	23
8.6. Segment Measurements — <code>ctz-draw-measure-segment</code>	24
8.7. Paths — <code>ctz-draw-path</code>	25
8.8. Global Styling — <code>ctz-style</code>	26
8.9. Angle Marking — <code>ctz-draw-angle</code>	27
8.10. Right Angle Mark — <code>ctz-draw-mark-right-angle</code>	27
9. Circles	29
9.1. Named Circle — <code>ctz-def-circle</code> / <code>ctz-draw-circle</code>	29
9.2. Circumcircle — <code>ctz-draw-circumcircle</code>	29
9.3. Incircle — <code>ctz-draw-incircle</code>	29
10. Conics	30
10.1. Ellipse — <code>ctz-def-ellipse</code> / <code>ctz-draw-ellipse</code>	30
10.2. Parabola (Focus + Directrix) — <code>ctz-def-parabola</code> / <code>ctz-draw-parabola</code>	30
10.3. Parabola (Focus + p + Angle) — <code>ctz-draw-parabola-focus</code>	30
10.4. Projectile Trajectory — <code>ctz-def-projectile</code> / <code>ctz-draw-projectile</code>	30
10.5. Conic Tangents	30
10.6. Ellipse Helpers	30
10.7. Conics — Visual Example	31
10.8. Circle Through Point — <code>ctz-draw-circle-through</code>	32
10.9. Semicircle — <code>ctz-draw-semicircle</code>	32
11. Clipping	33
11.1. Set Clip Region — <code>ctz-set-clip</code>	33
11.2. Draw Clipped Line — <code>ctz-draw-line-global-clip</code>	33
11.3. Draw Clipped Segment — <code>ctz-draw-seg-global-clip</code>	33
12. Grid & Annotation Helpers	33
12.1. Grid Positioning	33
12.1.1. Triangular Grid	33
12.1.2. Pascal's Triangle	34
12.1.3. Row and Diagonal Labels	34
12.1.4. Rectangular Grid	35
12.1.5. Hexagonal Grid	35
12.2. Annotation Helpers	36
12.2.1. Cell Highlighting	36
12.2.2. Curved Arrows	36
12.2.3. Smooth Arrows	36
12.2.4. Addition Indicator	37
12.2.5. Braces	37
13. Raw Algorithms	38
14. Function Reference	40
14.1. Point Definition	40
14.2. Line Constructions	40
14.3. Intersections	40
14.4. Triangle Centers	40
14.5. Special Triangles	40
14.6. Transformations	40
14.7. Drawing	40
14.8. Clipping	41
14.9. Grid Positioning	41
14.10. Grid Drawing	41

14.11. Annotations	41
15. Gallery Examples	42
16. Advanced Examples	54

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, inversion
- **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 "@local/ctz-euclide:0.1.0": *
```

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-*`.

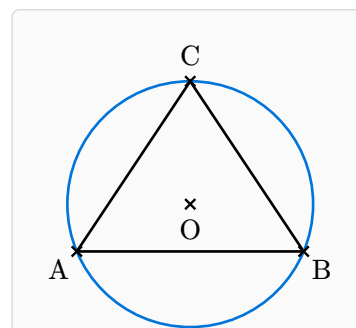
The `ctz-init()` call initializes the point registry and coordinate resolver.

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"), labels: (  
    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(segment: ("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:

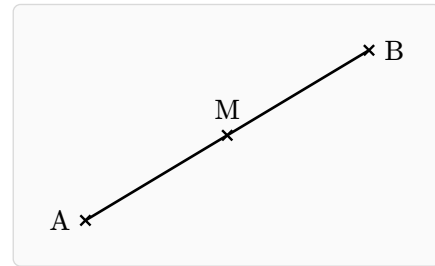
Code

```
#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(segment: ("A", "B"), stroke: black)
  ctz-draw(points: ("A", "B", "M"), labels: (
    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.

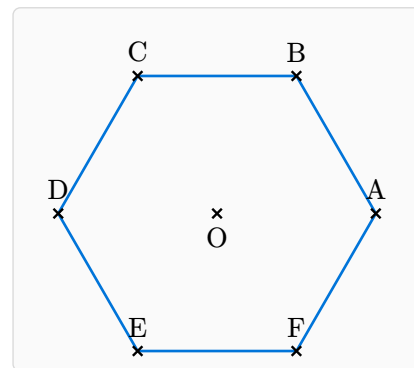
Code

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

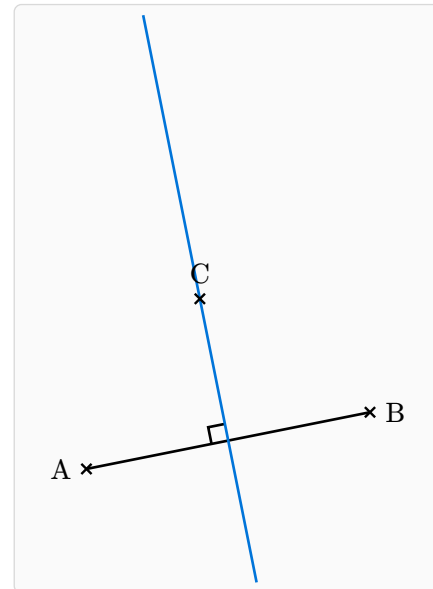
4.1. Perpendicular — `ctz-def-perp`

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(segment: ("A", "B"), stroke: black)  
  ctz-draw(segment: ("P1", "P2"), stroke: blue)  
  ctz-draw-mark-right-angle("A", "H", "C", size: 0.3)  
  
  ctz-draw(points: ("A", "B", "C"), labels: (  
    A: "left", B: "right", C: "above")  
  })  
})
```

Figure



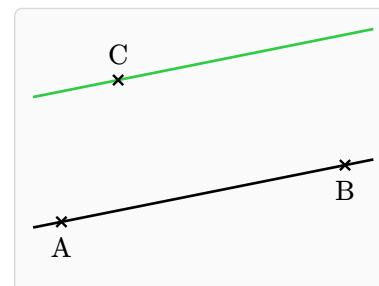
4.2. Parallel — `ctz-def-para`

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

Figure



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

Construct the bisector of an angle:

Code

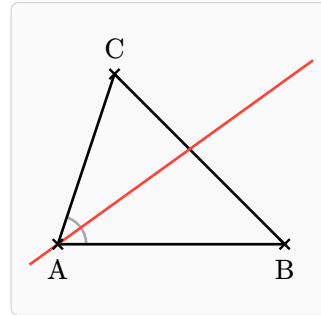
```
#ctz-canvas(length: 0.8cm, {
  import cetz.draw: *
  ctz-init()

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

Figure



4.4. Perpendicular Bisector — ctz-def-mediator

Construct the perpendicular bisector of a segment:

Code

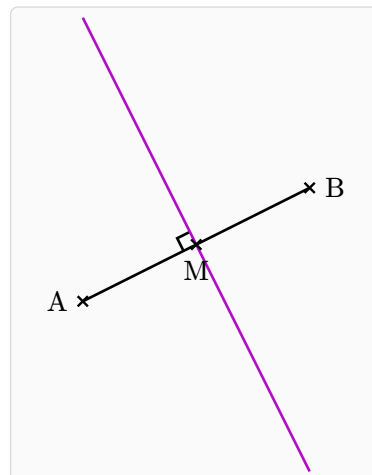
```
#ctz-canvas(length: 0.75cm, {
  import cetz.draw: *
  ctz-init()

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

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

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

Figure



5. Intersections

5.1. Line–Line — `ctz-def-ll`

Find the intersection of two lines:

Code

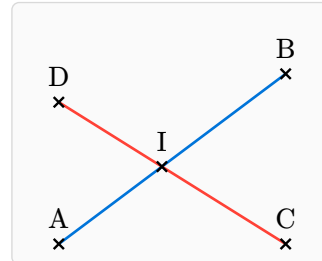
```
#ctz-canvas(length: 0.8cm, {
  import cetz.draw: *
  ctz-init()

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

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

Figure



5.2. Line–Circle — `ctz-def-lc`

Find intersections of a line with a circle:

Code

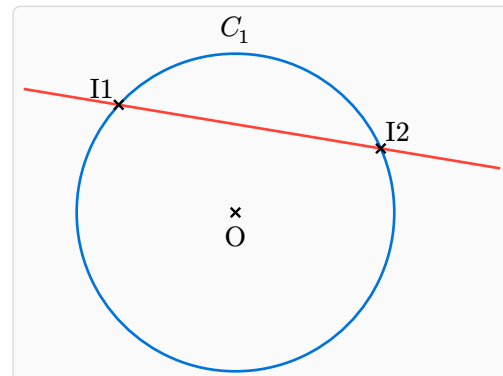
```
#ctz-canvas(length: 0.7cm, {
  import cetz.draw: *
  ctz-init()

  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("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"), labels: (
    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

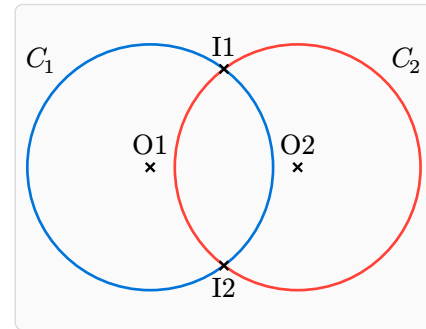
```
#ctz-canvas(length: 0.65cm, {
  import cetz.draw: *
  ctz-init()

  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("C1", stroke: blue)
  ctz-draw("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"), labels:
  (
    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()

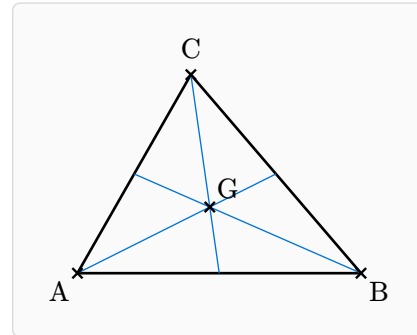
  ctz-def-points(A: (0, 0), B: (5, 0), C: (2, 3.5))
  ctz-def-centroid("G", "A", "B", "C")

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

  ctz-draw(points: ("A", "B", "C", "G"), labels: (
    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()

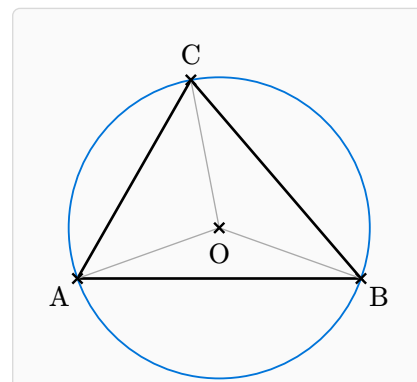
  ctz-def-points(A: (0, 0), B: (5, 0), C: (2, 3.5))
  ctz-def-circumcenter("O", "A", "B", "C")

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

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

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

Figure



6.1.3. Incenter — ctz-def-incenter

Center of the inscribed circle:

Code

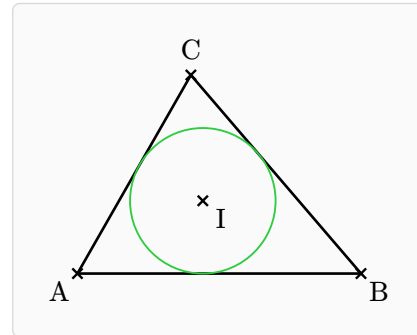
```
#ctz-canvas(length: 0.75cm, {
  import cetz.draw: *
  ctz-init()

  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"), labels: (
    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()

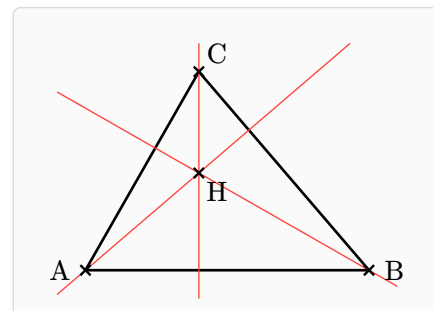
  ctz-def-points(A: (0, 0), B: (5, 0), C: (2, 3.5))
  ctz-def-orthocenter("H", "A", "B", "C")

  // 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"), labels: (
    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()

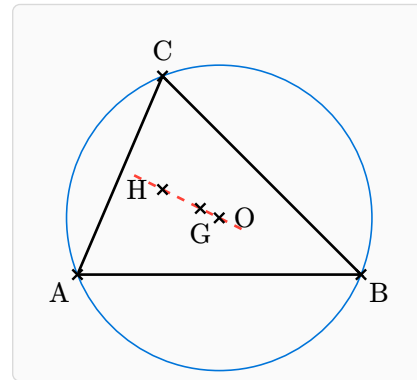
  ctz-def-points(A: (0, 0), B: (5, 0), C: (1.5, 3.5))

  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"),
    labels: {
      A: "below left",
      B: "below right",
      C: "above",
      H: "left",
      G: "below",
      O: "right"
    })
})
```

Figure



6.3. Right Triangles via Thales' Theorem

Thales' theorem states that any triangle inscribed in a semicircle with the diameter as its base has a right angle at the opposite vertex. The `ctz-def-thales-triangle()` function creates such triangles.

Code

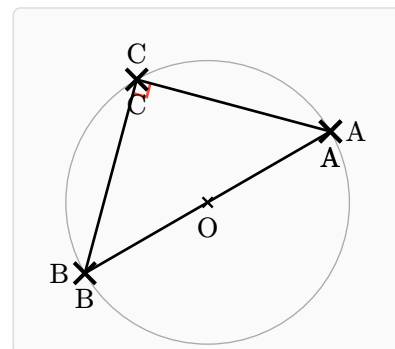
```
#ctz-canvas(length: 0.75cm, {
  import cetz.draw: *
  ctz-init()

  ctz-def-points(O: (0, 0))
  ctz-def-thales-triangle("A", "B", "C", "O", 2.5,
    base-angle: 30, orientation: "left")

  ctz-draw(circle-r: (_pt("O"), 2.5), stroke: gray +
    0.5pt)
  ctz-draw-path("A--B--C--A", stroke: black + 1pt)
  ctz-draw-mark-right-angle("A", "C", "B", color:
    red)

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

Figure



Parameters:

- `name-a`, `name-b`: Diameter endpoints (base of triangle)
- `name-c`: Vertex with the right angle
- `center`: Circle center
- `radius`: Circle radius
- `base-angle`: Rotation angle for the diameter (default: 0)
- `orientation`: "left" or "right" - position of right angle vertex

6.4. 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 ctz.draw: *
  ctz-init()

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

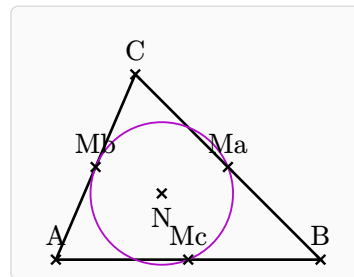
  ctz-draw(line: ("A", "B", "C", "A"), stroke: black)

  // 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"), labels: (
    N: "below"
  ))
})
```

Figure



7. Transformations

7.1. Rotation — rotate

Rotate a point around a center:

Code

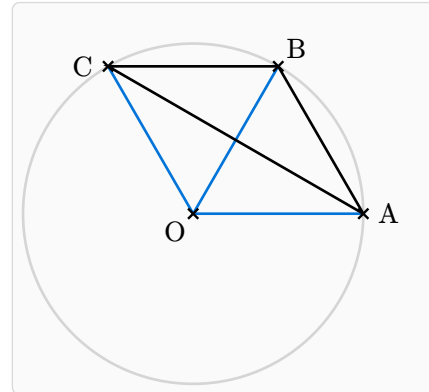
```
#ctz-canvas(length: 0.75cm, {
  import cetz.draw: *
  ctz-init()

  ctz-def-points(0: (2, 2), A: (5, 2))
  ctz-def-rotation("B", "A", "O", 60)
  ctz-def-rotation("C", "A", "O", 120)

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

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

Figure



7.2. Reflection — ctz-def-reflect

Reflect a point across a line:

Code

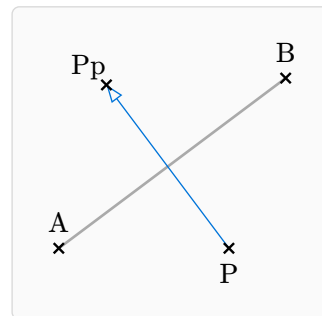
```
#ctz-canvas(length: 0.75cm, {
  import cetz.draw: *
  ctz-init()

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

  ctz-draw(segment: ("A", "B"), stroke: gray)
  ctz-draw(path: "P--Pp", stroke: blue + 0.5pt, mark:
(end: ">"), points: false, labels: false)

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

Figure



7.3. Homothety (Scaling) — scale

Scale a point from a center:

Code

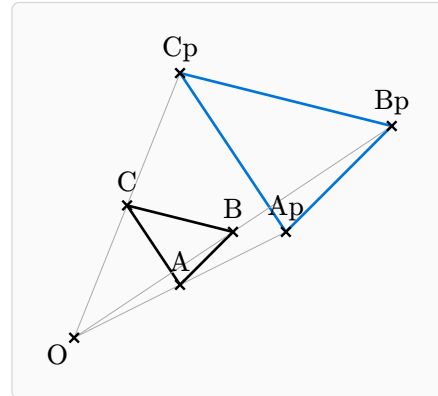
```
#ctz-canvas(length: 0.7cm, {
  import cetz.draw: *
  ctz-init()

  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(segment: ("O", "Ap"), stroke: gray +
0.3pt)
  ctz-draw(segment: ("O", "Bp"), stroke: gray +
0.3pt)
  ctz-draw(segment: ("O", "Cp"), stroke: gray +
0.3pt)

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

Figure



7.4. Projection — ctz-def-project

Project a point onto a line:

Code

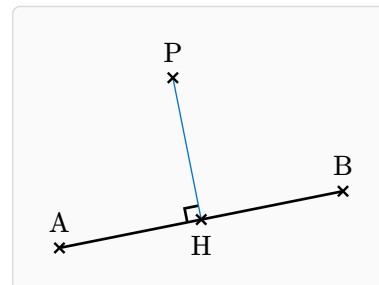
```
#ctz-canvas(length: 0.75cm, {
  import cetz.draw: *
  ctz-init()

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

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

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

Figure



7.5. Inversion — ctz-def-inversion

Invert points, lines, or circles through a circle:

Code

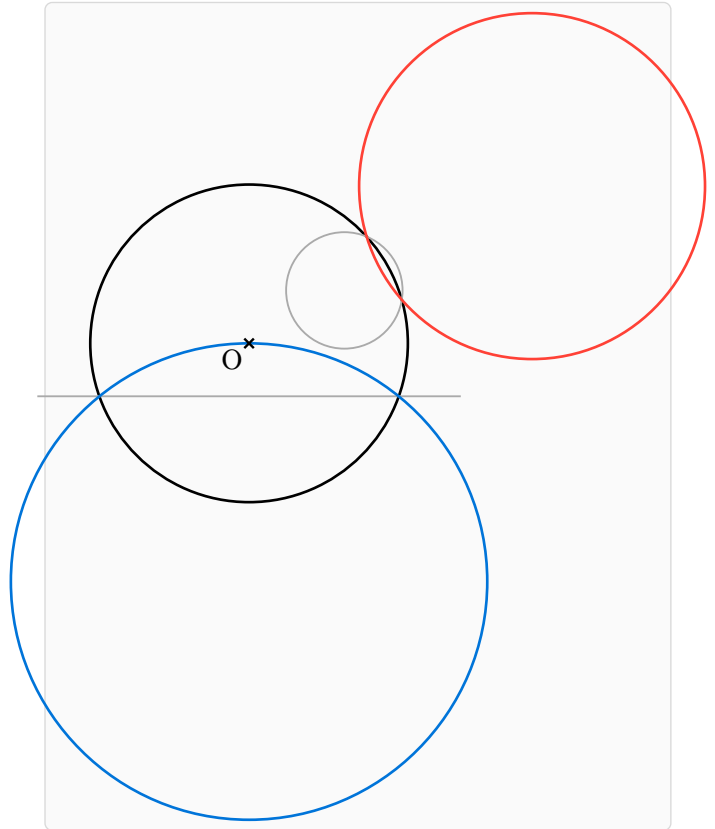
```
#ctz-canvas(length: 0.7cm, {
  import cetz.draw: *
  ctz-init()
  ctz-set-clip(-5, -4.5, 5, 4)

  ctz-def-points(0: (0, 0), A: (-4, -1), B: (4, -1),
  C: (1.8, 1))
  ctz-def-line("L", "A", "B")
  ctz-def-circle("C1", "C", radius: 1.1)

  ctz-def-inversion("Li", "L", "0", 3)
  ctz-def-inversion("C1i", "C1", "0", 3)

  ctz-draw(circle-r: (_pt("0"), 3), stroke: black +
  1pt)
  ctz-draw("L", stroke: gray + 0.7pt)
  ctz-draw("C1", stroke: gray + 0.7pt)
  ctz-draw("Li", stroke: blue + 1pt)
  ctz-draw("C1i", stroke: red + 1pt)
  ctz-draw(points: ("0"), labels: (0: "below left"))
})
```

Figure



7.6. Object Duplication — ctz-duplicate

Duplicate any geometric object. For polygons, explicit point names must be provided.

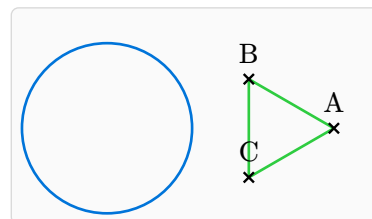
Code

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

  // Duplicate a circle
  ctz-def-circle("c1", (0, 0), radius: 1.5)
  ctz-duplicate("c2", "c1")
  ctz-draw("c1", stroke: blue)

  // Duplicate a polygon
  ctz-def-regular-polygon("tri", ("A", "B", "C"),
  (3, 0), (4, 0), n: 3)
  ctz-duplicate("tri2", "tri",
  points: ("A2", "B2", "C2"))
  ctz-draw("tri", stroke: green)
  ctz-draw(points: ("A", "B", "C"), labels: true)
})
```

Figure



For points, lines, and circles, duplication is straightforward. For polygons, you must provide explicit point names so the vertices can be referenced independently.

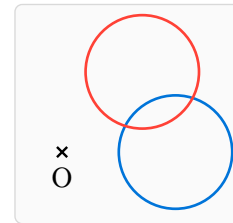
7.7. Polymorphic Rotation

The `ctz-def-rotation()` function works on all object types: points, lines, circles, and polygons.

Code

```
#ctz-canvas(length: 0.75cm, {  
  import cetz.draw: *  
  ctz-init()  
  
  // Rotate a circle  
  ctz-def-points(0: (0, 0))  
  ctz-def-circle("c1", (2, 0), radius: 1)  
  ctz-def-rotation("c2", "c1", "0", 45)  
  
  ctz-draw("c1", stroke: blue)  
  ctz-draw("c2", stroke: red)  
  ctz-draw(points: ("0"), labels: (  
    0: "below"  
  ))  
})
```

Figure



For lines, both endpoints are rotated. For circles, the center is rotated while radius remains constant. For polygons, all constituent points are rotated in place.

8. Drawing & Styling

8.1. Points — `ctz-draw-points`

Draw points at named locations:

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

8.2. Unified Drawing — `ctz-draw`

The `ctz-draw()` function provides a unified interface for drawing both **named objects** and **unnamed constructs**.

8.2.1. Drawing Named Objects

Use `ctz-draw()` to draw any object type without remembering type-specific commands. It automatically detects whether the object is a point, line, circle, polygon, or conic (ellipse/parabola/projectile).

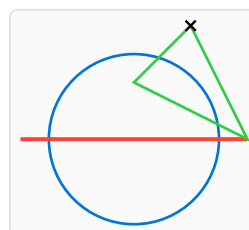
Code

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

  ctz-def-circle("c1", (0, 0), radius: 1.5)
  ctz-def-line("l1", (-2, 0), (2, 0))
  ctz-def-polygon("tri", "A", "B", "C")
  ctz-def-points(A: (1, 2), B: (2, 0), C: (0, 1))

  ctz-draw("c1", stroke: blue, fill: none)
  ctz-draw("l1", stroke: red + 1.5pt)
  ctz-draw("tri", stroke: green)
  ctz-draw("A")
})
```

Figure



8.2.2. Drawing Unnamed Constructs

You can also use `ctz-draw()` to draw geometric objects directly without defining them first using named parameters.

8.2.2.1. Points with Labels

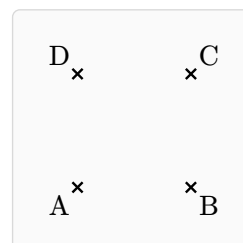
Draw multiple points at once, optionally with labels:

Code

```
#ctz-canvas(length: 0.75cm, {
  import ctz.draw: *
  ctz-init()
  ctz-def-points(A: (0, 0), B: (2, 0), C: (2, 2), D:
    (0, 2))

  // Draw points with custom label positions
  ctz-draw(points: ("A", "B", "C", "D"), labels: {
    A: "below left",
    B: "below right",
    C: "above right",
    D: "above left"
  })
})
```

Figure



Use `labels: true` for default label positioning, or omit the `labels` parameter to draw points without labels.

8.2.2.2. Paths and Polylines

Draw paths using the `path:` parameter for CeTZ-style path syntax, or `line:` for polylines through points:

```
ctz-draw(path: "A--B--C--A", stroke: black) // Close the path by repeating first point
ctz-draw(line: ("A", "B", "C", "D"), stroke: red) // Open polyline
```

8.2.2.3. Circles

Draw circles without naming them:

```
// Circle through two points (center and point on circumference)
ctz-draw(circle-through: ("O", "A"), stroke: blue)

// Circle by center and radius
ctz-draw(circle-r: ((0, 0), 1.5), stroke: green)

// Circle by diameter endpoints
ctz-draw(circle-diameter: ("A", "B"), stroke: purple)

// Circumcircle of triangle
ctz-draw(circumcircle: ("A", "B", "C"), stroke: red)

// Incircle of triangle
ctz-draw(incircle: ("A", "B", "C"), stroke: teal)
```

8.2.2.4. Conics (Ellipses, Parabolas, Projectiles)

Draw conics without naming them:

```
// Ellipse by center, radii, and rotation angle
ctz-draw(ellipse: ((0, 0), 3, 2, 20deg), stroke: black)

// Parabola by focus + directrix (line or two points)
ctz-draw(parabola: ((0, 0), ((-2, -3), (-2, 3)), 4), stroke: black)

// Projectile by origin + velocity (optional gravity, vectors, etc.)
ctz-draw(projectile: (origin: (0, 0), velocity: (4, 6), y-floor: 0, vectors: true), stroke: blue)
```

8.2.2.5. Arcs and Semicircles

```
// Arc by center and two points
ctz-draw(arc: (center: "O", start: "A", end: "B"), stroke: orange)

// Arc by center, radius, and angles (in degrees)
ctz-draw(arc-r: ((0, 0), 2, 0, 90), stroke: black)

// Semicircle by diameter endpoints
ctz-draw(semicircle: ("A", "B"), stroke: blue, fill: blue.lighten(80%))
```

8.2.2.6. Line Segments

```
ctz-draw(segment: ("A", "B"), stroke: maroon + 1.5pt)
```

8.2.3. Complete Example

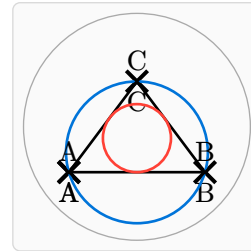
Code

```
#ctz-canvas(length: 0.6cm, {
  import cetz.draw: *
  ctz-init()
  ctz-def-points(A: (0, 0), B: (3, 0), C: (1.5, 2))

  // Named circle
  ctz-def-circle("myCircle", (1.5, 1), radius: 2.5)
  ctz-draw("myCircle", stroke: gray + 0.5pt)

  // Unnamed constructs
  ctz-draw(path: "A--B--C--A", stroke: black)
  ctz-draw(circumcircle: ("A", "B", "C"), stroke:
blue)
  ctz-draw(incircle: ("A", "B", "C"), stroke: red)
  ctz-draw(points: ("A", "B", "C"), labels: true)
})
```

Figure



8.3. 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("0", 0: (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)))
```

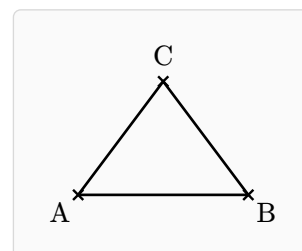
8.4. Labels & Points — Unified API (Recommended)

The modern approach combines point drawing and labeling in a single call:

Code

```
#ctz-canvas(length: 0.75cm, {
  import cetz.draw: *
  ctz-init()
  ctz-def-points(A: (0, 0), B: (3, 0), C: (1.5, 2))
  ctz-draw(line: ("A", "B", "C", "A"), stroke: black)
  ctz-draw(points: ("A", "B", "C"), labels: (
    A: "below left",
    B: "below right",
    C: "above"
  ))
})
```

Figure



This unified API replaces separate `ctz-draw-points()` and `ctz-draw-labels()` calls. The old API remains supported for backward compatibility.

You can also label points that were drawn earlier:

```
ctz-draw(points: ("A", "B")) // Draw points without labels
// ... other drawing commands ...
ctz-draw(labels: (A: "below", B: "above")) // Add labels later
```

8.5. 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), ->, <-, <->, |-|, |->, <-|.

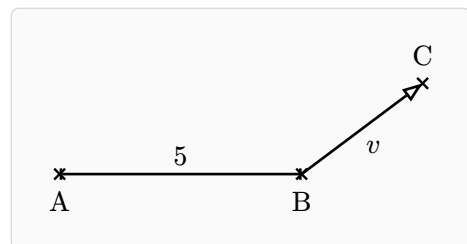
Code

```
#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"), labels: (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

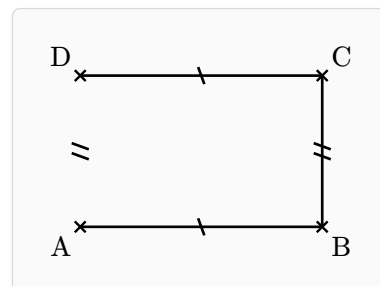
```
#ctz-canvas(length: 0.8cm, {
  import cetz.draw: *
  ctz-init()

  ctz-def-points(A: (0, 0), B: (4, 0), C: (4, 2.5),
    D: (0, 2.5))
  ctz-draw(line: ("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"), labels: (
    A: "below left",
    B: "below right",
    C: "above right",
    D: "above left"
  ))
})
```

Figure



Code

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

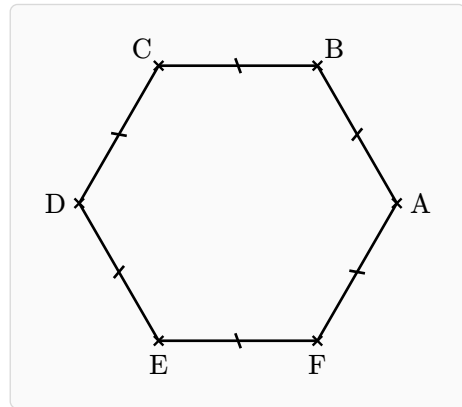
  ctz-def-points(0: (0, 0), A: (3, 0))
  ctz-def-regular-polygon("Hex", ("A", "B", "C", "D",
    "E", "F"), "0", "A")

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

Figure



8.6. 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

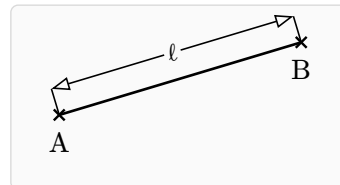
```
#ctz-canvas(length: 0.8cm, {
  import cetz.draw: *
  ctz-init()

  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"), labels: (
    A: "below",
    B: "below"
  ))
})
```

Figure



Code

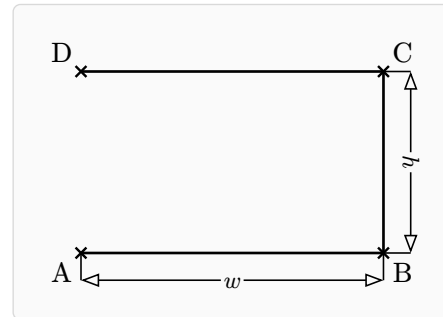
```
#ctz-canvas(length: 0.8cm, {
  import cetz.draw: *
  ctz-init()

  ctz-def-points(A: (0, 0), B: (5, 0), C: (5, 3), D:
(0, 3))
  ctz-draw(line: ("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"), labels: (
    A: "below left",
    B: "below right",
    C: "above right",
    D: "above left"
  ))
})
```

Figure



8.7. 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

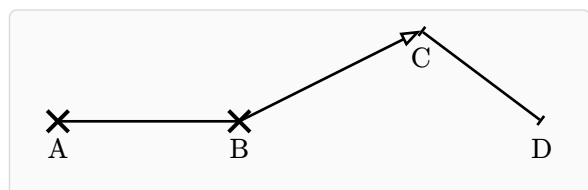
Code

```
#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



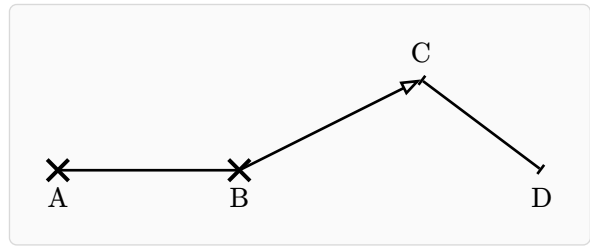
Code

```
#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)
```

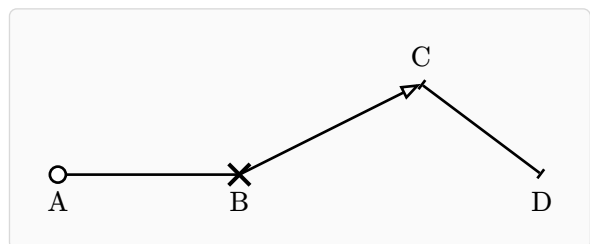
Code

```
#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



8.8. Global Styling — `ctz-style`

Set default styles for points and labels:

Code

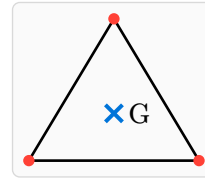
```
#ctz-canvas(length: 0.8cm, {
  import cetz.draw: *
  ctz-init()
  ctz-style(point: (shape: "dot", size: 0.1, fill:
red))

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

Figure



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

8.9. Angle Marking — ctz-draw-angle

Mark and label angles:

Code

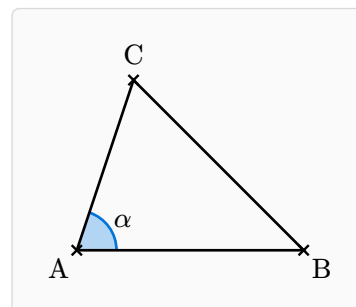
```
#ctz-canvas(length: 0.8cm, {
  import cetz.draw: *
  ctz-init()

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

Figure



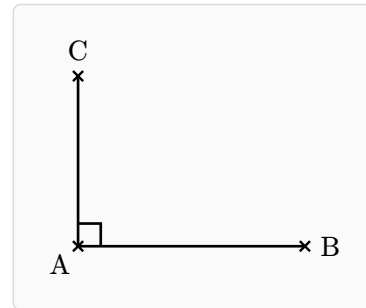
8.10. Right Angle Mark — ctz-draw-mark-right-angle

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(segment: ("A", "B"), stroke: black)  
  ctz-draw(segment: ("A", "C"), stroke: black)  
  
  ctz-draw-mark-right-angle("B", "A", "C", size: 0.4)  
  
  ctz-draw(points: ("A", "B", "C"), labels: (  
    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("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)
```

10. Conics

10.1. Ellipse — **ctz-def-ellipse** / **ctz-draw-ellipse**

Define and draw ellipses:

- `ctz-def-ellipse(name, center, rx, ry, angle: 0deg)`
- `ctz-draw-ellipse(center, rx, ry, angle: 0deg, stroke: ..., fill: ...)`
- `ctz-draw(name, ...)` for named ellipses

10.2. Parabola (Focus + Directrix) — **ctz-def-parabola** / **ctz-draw-parabola**

Define and draw parabolas from focus and directrix:

- `ctz-def-parabola(name, focus, directrix, extent: auto, steps: 120)`
- `ctz-draw-parabola(focus, directrix, extent: auto, steps: 120, stroke: ...)`

10.3. Parabola (Focus + p + Angle) — **ctz-draw-parabola-focus**

Draw a parabola using the focus, parameter p (vertex-to-focus distance), and opening angle:

- `ctz-draw-parabola-focus(focus, p, angle: 0deg, extent: auto, steps: 120, stroke: ...)`

10.4. Projectile Trajectory — **ctz-def-projectile** / **ctz-draw-projectile**

Define and draw a projectile (parabolic) trajectory:

- `ctz-def-projectile(name, origin, velocity, gravity: 9.81, t-max: auto, steps: 80, y-floor: none, vectors: false, vector-count: 5, vector-scale: 0.12)`
- `ctz-draw-projectile(origin, velocity, gravity: 9.81, t-max: auto, steps: 80, y-floor: none, vectors: false, vector-count: 5, vector-scale: 0.12)`

10.5. Conic Tangents

Draw tangents for conics:

- `ctz-draw-ellipse-tangent(center, rx, ry, t, angle: 0deg, length: auto, stroke: ...)`
- `ctz-draw-parabola-tangent(focus, directrix, t, length: auto, stroke: ...)`
- `ctz-draw-parabola-tangents-from-point(focus, directrix, external, length: auto, stroke: ...)`

10.6. Ellipse Helpers

Descriptive-geometry helpers for ellipse constructions:

- `ctz-ellipse-project-point(center, rx, ry, point, angle: 0deg)` — project a point onto the ellipse along the ray from center
- `ctz-ellipse-tangents-from-point(center, rx, ry, point, angle: 0deg, length: auto)` — tangent lines from an external point
- `ctz-ellipse-circle-intersections(center, rx, ry, circle-center, r, angle: 0deg, steps: 360)` — intersections with a circle (numeric)

10.7. Conics — Visual Example

Code

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

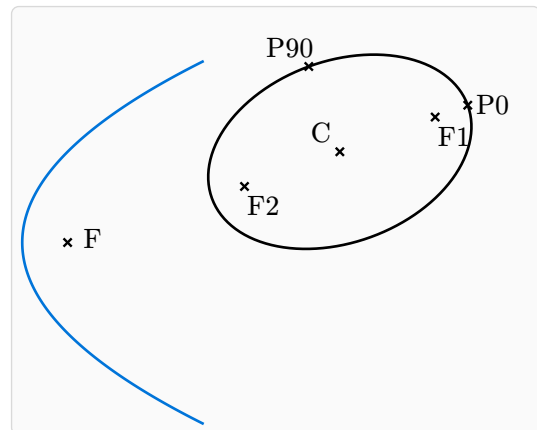
  // Ellipse with labeled points
  let C = (0, 0)
  let rx = 3
  let ry = 2
  let ang = 20deg
  let (F1, F2) = ellipse-foci-raw(C, rx, ry, angle:
ang)
  let P0 = ellipse-angpoint-raw(C, rx, ry, angle:
ang, theta: 0deg)
  let P90 = ellipse-angpoint-raw(C, rx, ry, angle:
ang, theta: 90deg)

  // Register points for labeling
  ctz-def-points(C: C, F1: F1, F2: F2, P0: P0, P90:
P90)

  ctz-draw(ellipse: (C, rx, ry, ang), stroke: black)
  ctz-draw(points: ("C", "F1", "F2", "P0", "P90"),
labels: (
    C: (pos: "above left", offset: (-0.1, 0.1)),
    F1: (pos: "below right", offset: (0.1, -0.1)),
    F2: (pos: "below right", offset: (0.1, -0.1)),
    P0: (pos: "right", offset: (0.1, 0)),
    P90: (pos: "above left", offset: (-0.1, 0.1))
  ))

  // Parabola (focus + directrix)
  let F = (-6, -2)
  let D1 = (-8, -4)
  let D2 = (-8, 1)
  ctz-def-points(F: F)
  ctz-draw(parabola: (F, (D1, D2), 4), stroke: blue)
  ctz-draw(points: ("F",), labels: (F: (pos: "right",
offset: (0.1, 0.1))))
})
```

Figure



Code

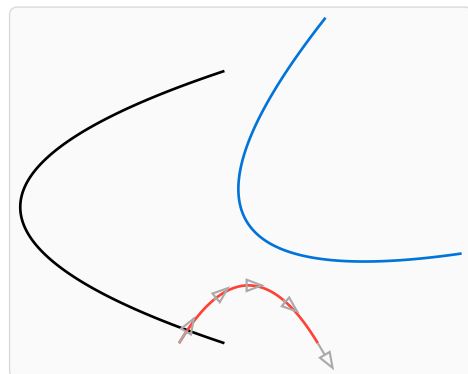
```
#ctz-canvas(length: 0.6cm, {
  import cetz.draw: *
  ctz-init()

  // Parabola from focus + directrix
  ctz-draw(parabola: ((-3, 0), ((-4, -2), (-4, 2)),
3), stroke: black)
  ctz-draw(points: ((-3, 0),), labels: (P: "above"))

  // Parabola from focus + p + angle
  ctz-draw-parabola-focus((2, 0), 0.6, angle: 30deg,
extent: 3, stroke: blue)

  // Projectile with vectors
  ctz-draw(projectile: (origin: (0, -3), velocity:
(3, 5), y-floor: -3, vectors: true, vector-count: 5),
stroke: red)
})
```

Figure



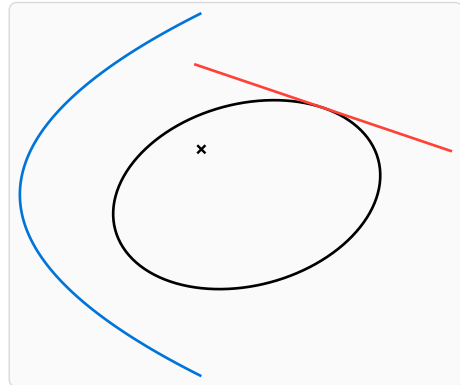
Code

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

  // Ellipse tangents
  ctz-draw(ellipse: ((0, 0), 3, 2, 15deg), stroke:
black)
  ctz-draw-ellipse-tangent((0, 0), 3, 2, 45deg,
angle: 15deg, stroke: red)

  // Parabola tangents from external point
  let F = (-4, 0)
  let D1 = (-6, -3)
  let D2 = (-6, 3)
  let P = (-1, 1)
  ctz-draw(parabola: (F, (D1, D2), 4), stroke: blue)
  ctz-draw-parabola-tangents-from-point(F, (D1, D2),
P, stroke: red)
  ctz-draw(points: (P,), labels: (P: "above"))
})
```

Figure



10.8. 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)
```

10.9. Semicircle — `ctz-draw-semicircle`

Draw a semicircle on a diameter:

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


11. Clipping

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

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

Define a rectangular clip boundary:

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

11.2. 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.

11.3. 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 ctz.draw: *
  ctz-init()

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

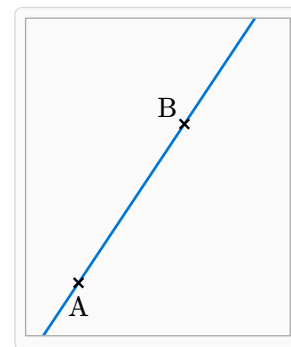
  // Set clip boundary
  ctz-set-clip(-1, -1, 4, 5)

  // 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"), labels: (
    A: "below",
    B: "above left"
  ))
})
```

Figure



12. Grid & Annotation Helpers

This section covers helper functions for creating structured diagrams with grids, highlights, and annotations. These are particularly useful for educational materials like Pascal's triangle, multiplication tables, or any grid-based visualizations.

12.1. Grid Positioning

Three grid layout systems are provided for positioning content.

12.1.1. Triangular Grid

The `triangular-pos(row, col)` function computes positions in a triangular layout where each row has one more element than the previous, centered horizontally. This is the layout used by Pascal's triangle.

Code

```
#cetz.canvas({
  import cetz.draw: *
  import "@local/ctz-euclide:0.1.0": *
  import "@local/ctz-euclide:0.1.0": *

  draw-triangular-grid(
    cetz.draw, 5,
    (row, col) => str(row) + "," + str(col),
    h-spacing: 1.8,
    v-spacing: 1.0,
    text-size: 9pt,
  )
})
```

Figure

		0,0								
		1,0		1,1						
		2,0		2,1		2,2				
		3,0		3,1		3,2		3,3		
		4,0		4,1		4,2		4,3		4,4

Parameters:

- row — Row number (0 = top)
- col — Column within row (0 to row)
- h-spacing — Horizontal spacing between adjacent cells (default: 1.5)
- v-spacing — Vertical spacing between rows (default: 1.2)
- origin — Grid origin point (default: (0, 0))

12.1.2. Pascal's Triangle

For Pascal's triangle binomial coefficients, use `draw-pascal-values`:

Code

```
#cetz.canvas({
  import cetz.draw: *
  import "@local/ctz-euclide:0.1.0": *
  import "@local/ctz-euclide:0.1.0": *

  draw-pascal-values(cetz.draw, 6,
    h-spacing: 1.5, v-spacing: 1.0,
    text-size: 11pt)
})
```

Figure

						1										
						1				1						
						1		2		1						
						1		3		3		1				
						1		4		6		4		1		
						1		5		10		10		5		1

12.1.3. Row and Diagonal Labels

Add row labels (`draw-row-labels`) and diagonal labels (`draw-diagonal-labels`):

Code

```
#cetz.canvas({
  import cetz.draw: *
  import "@local/ctz-euclide:0.1.0": *
  import "@local/ctz-euclide:0.1.0": *

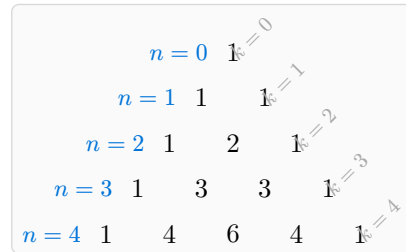
  let sp = (h: 1.4, v: 1.0)

  draw-pascal-values(cetz.draw, 5,
    h-spacing: sp.h, v-spacing: sp.v,
    text-size: 10pt)

  draw-row-labels(cetz.draw, 5,
    h-spacing: sp.h, v-spacing: sp.v,
    offset: -1.2, text-color: blue,
    format: n => $n=#n$)

  draw-diagonal-labels(cetz.draw, 5,
    h-spacing: sp.h, v-spacing: sp.v,
    offset: (0.4, 0.3), text-color: gray,
    format: k => $k=#k$)
})
```

Figure



12.1.4. Rectangular Grid

The `grid-pos(row, col)` function computes positions in a standard rectangular grid:

Code

```
#cetz.canvas({
  import cetz.draw: *
  import "@local/ctz-euclide:0.1.0": *
  import "@local/ctz-euclide:0.1.0": *

  draw-rectangular-grid(
    cetz.draw, 3, 4,
    (r, c) => str(r * 4 + c + 1),
    h-spacing: 1.2, v-spacing: 1.0,
    text-size: 11pt)
})
```

Figure



12.1.5. Hexagonal Grid

The `hex-pos(row, col)` function computes positions in a hexagonal (honeycomb) grid:

Code

```
#cetz.canvas({
  import cetz.draw: *
  import "@local/ctz-euclide:0.1.0": *
  import "@local/ctz-euclide:0.1.0": hex-pos

  for row in range(3) {
    for col in range(4) {
      let p = hex-pos(row, col, size: 0.8)
      cetz.draw.circle(p, radius: 0.35,
        stroke: blue + 0.5pt,
        fill: blue.lighten(90%))
      content(p, text(size: 8pt,
        str(row) + "," + str(col)))
    }
  }
})
```

Figure



12.2. Annotation Helpers

Functions for highlighting cells and drawing annotation arrows.

12.2.1. Cell Highlighting

`highlight-fill` draws a filled circle, `highlight-outline` draws an outlined circle, and `highlight-many` highlights multiple positions:

Code

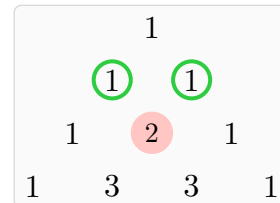
```
#cetz.canvas({
  import cetz.draw: *
  import "@local/ctz-euclide:0.1.0": *
  import "@local/ctz-euclide:0.1.0": *

  draw-pascal-values(cetz.draw, 4,
    h-spacing: 1.5, v-spacing: 1.0)

  // Filled highlight
  let p1 = triangular-pos(2, 1,
    h-spacing: 1.5, v-spacing: 1.0)
  highlight-fill(cetz.draw, p1,
    radius: 0.4, fill: red.lighten(70%))
  content(p1, "2")

  // Outline highlights
  let p2 = triangular-pos(1, 0,
    h-spacing: 1.5, v-spacing: 1.0)
  let p3 = triangular-pos(1, 1,
    h-spacing: 1.5, v-spacing: 1.0)
  highlight-many(cetz.draw, (p2, p3),
    radius: 0.35, stroke: green + 1.5pt)
})
```

Figure



12.2.2. Curved Arrows

`curved-arrow` draws a curved annotation arrow. The `bend` parameter controls curvature direction and amount (positive = bend right, negative = bend left):

Code

```
#cetz.canvas({
  import cetz.draw: *
  import "@local/ctz-euclide:0.1.0": *
  import "@local/ctz-euclide:0.1.0": *

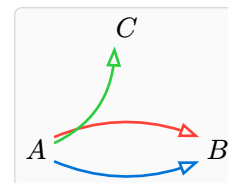
  content((0, 0), $A$)
  content((3, 0), $B$)
  content((1.5, 2), $C$)

  curved-arrow(cetz.draw,
    (0.3, 0.2), (2.7, 0.2),
    bend: 0.5, stroke: red + 1pt)

  curved-arrow(cetz.draw,
    (0.3, -0.2), (2.7, -0.2),
    bend: -0.5, stroke: blue + 1pt)

  curved-arrow(cetz.draw,
    (0.3, 0.1), (1.3, 1.7),
    bend: -0.6, stroke: green + 1pt)
})
```

Figure



12.2.3. Smooth Arrows

`smooth-arrow` draws a Catmull-Rom spline through waypoints for complex curved paths:

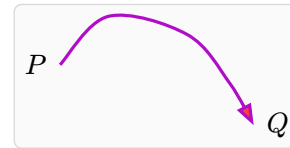
Code

```
#cetz.canvas({
  import cetz.draw: *
  import "@local/ctz-euclide:0.1.0": *
  import "@local/ctz-euclide:0.1.0": *

  content((0, 0), $P$)
  content((4, -1), $Q$)

  smooth-arrow(cetz.draw,
    (0.4, 0), (3.6, -1),
    waypoints: (
      (1.2, 0.8),
      (2.5, 0.5),
      (3.2, -0.3),
    ),
    stroke: purple + 1.2pt)
})
```

Figure



12.2.4. Addition Indicator

`draw-addition-indicator` shows two values combining into a result (useful for Pascal's triangle):

Code

```
#cetz.canvas({
  import cetz.draw: *
  import "@local/ctz-euclide:0.1.0": *
  import "@local/ctz-euclide:0.1.0": *

  let a = (0, 0)
  let b = (1.5, 0)
  let sum = (0.75, -1.2)

  content(a, $3$)
  content(b, $3$)
  content(sum, $6$)

  draw-addition-indicator(cetz.draw,
    a, b, sum, color: green)
})
```

Figure



12.2.5. Braces

`draw-brace-h` and `draw-brace-v` draw horizontal and vertical braces with optional labels:

Code

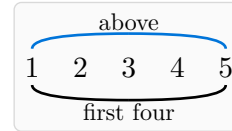
```
#cetz.canvas({
  import cetz.draw: *
  import "@local/ctz-euclide:0.1.0": *
  import "@local/ctz-euclide:0.1.0": *

  for i in range(5) {
    content((i * 0.8, 0),
      text(size: 11pt, str(i + 1)))
  }

  draw-brace-h(cetz.draw,
    (0, -0.3), (3.2, -0.3),
    label: text(size: 9pt, "first four"),
    side: "below", amplitude: 0.25)

  draw-brace-h(cetz.draw,
    (0, 0.3), (3.2, 0.3),
    label: text(size: 9pt, "above"),
    side: "above", amplitude: 0.25,
    stroke: blue + 1pt)
})
```

Figure



Code

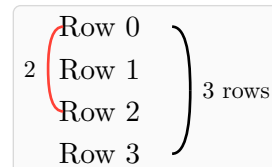
```
#cetz.canvas({
  import cetz.draw: *
  import "@local/ctz-euclide:0.1.0": *
  import "@local/ctz-euclide:0.1.0": *

  for i in range(4) {
    content((0, -i * 0.7),
      text(size: 11pt, "Row " + str(i)))
  }

  draw-brace-v(cetz.draw,
    (1.2, 0), (1.2, -2.1),
    label: text(size: 9pt, "3 rows"),
    side: "right", amplitude: 0.3)

  draw-brace-v(cetz.draw,
    (-0.6, 0), (-0.6, -1.4),
    label: text(size: 9pt, "2"),
    side: "left", amplitude: 0.25,
    stroke: red + 1pt)
})
```

Figure



13. 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, ctz-def-inversion
- **Utilities:** ctz-def-midpoint, dist, angle-at-vertex, triangle-area, etc.

14. Function Reference

14.1. 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

14.2. 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

14.3. 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

14.4. 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`

14.5. 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
- `ctz-def-thales-triangle(na, nb, nc, center, radius, ...)` — Right triangle via Thales' theorem

14.6. Transformations

- `ctz-def-rotation(name, source, center, angle)` — Rotation (works on all object types)
- `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
- `ctz-def-inversion(name, source, center, radius)` — Inversion (works on points, lines, circles, polygons)
- `ctz-duplicate(target, source, points: auto)` — Duplicate any geometric object

14.7. Drawing

- `ctz-draw(name, ...)` or `ctz-draw(points: ..., labels: ...)` — Draw any object type (polymorphic) or draw/label points using unified API
- `ctz-draw-points(names...)` — Draw points (legacy, prefer unified `ctz-draw()`)
- `ctz-draw-labels(names, placements)` — Label points (legacy, prefer unified `ctz-draw()`)
- `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

14.8. 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

14.9. Grid Positioning

- `triangular-pos(row, col, ...)` — Position in triangular grid (Pascal layout)
- `grid-pos(row, col, ...)` — Position in rectangular grid
- `hex-pos(row, col, ...)` — Position in hexagonal grid
- `binomial(n, k)` — Binomial coefficient $C(n,k)$

14.10. Grid Drawing

- `draw-triangular-grid(cetz-draw, rows, content-fn, ...)` — Draw triangular grid with custom content
- `draw-rectangular-grid(cetz-draw, rows, cols, content-fn, ...)` — Draw rectangular grid
- `draw-pascal-values(cetz-draw, rows, ...)` — Draw Pascal's triangle values
- `draw-row-labels(cetz-draw, rows, ...)` — Draw row labels
- `draw-diagonal-labels(cetz-draw, count, ...)` — Draw diagonal labels

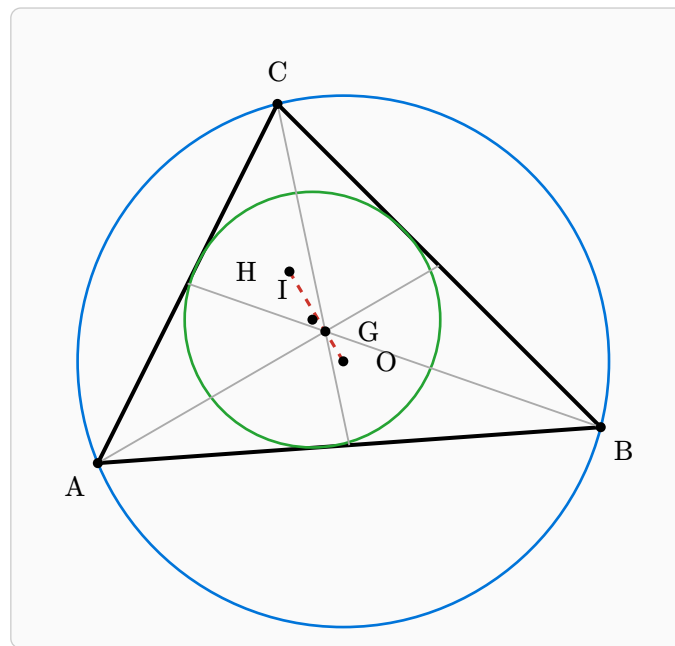
14.11. Annotations

- `highlight-fill(cetz-draw, pos, ...)` — Filled circle highlight
- `highlight-outline(cetz-draw, pos, ...)` — Outlined circle highlight
- `highlight-many(cetz-draw, positions, ...)` — Highlight multiple positions
- `curved-arrow(cetz-draw, from, to, ...)` — Curved annotation arrow
- `smooth-arrow(cetz-draw, from, to, waypoints, ...)` — Smooth spline arrow
- `draw-addition-indicator(cetz-draw, a, b, result, ...)` — Show addition relationship
- `draw-brace-h(cetz-draw, start, end, ...)` — Horizontal brace
- `draw-brace-v(cetz-draw, start, end, ...)` — Vertical brace

15. 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")

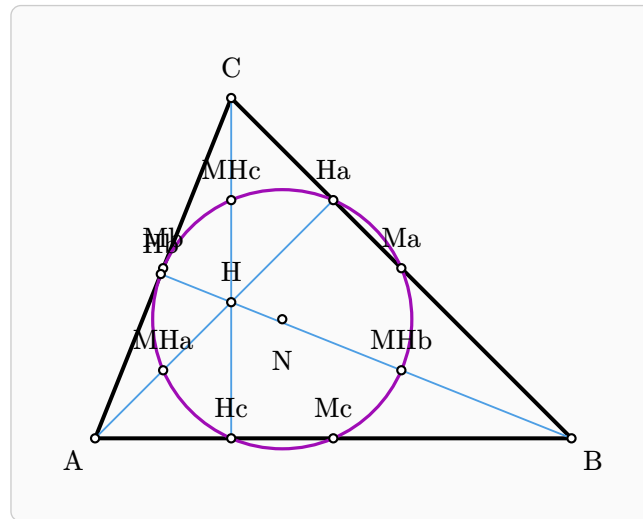
  // Euler line (H, G, O are collinear)
  ctz-draw(segment: ("H", "O"), stroke: (
    paint: red.darken(20%),
    dash: "dashed",
    thickness: 1.2pt
  ))

  // 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(segment: ("A", "Ma"), stroke: gray + 0.7pt)
  ctz-draw(segment: ("B", "Mb"), stroke: gray + 0.7pt)
  ctz-draw(segment: ("C", "Mc"), stroke: gray + 0.7pt)

  ctz-draw(points: ("A", "B", "C", "G", "O", "I", "H"), labels: (
    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")

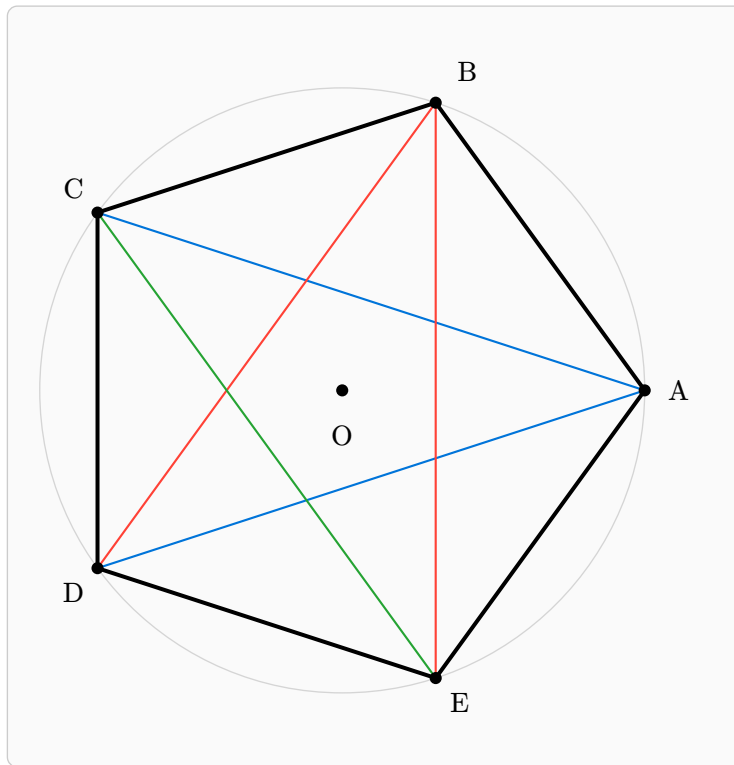
  // Orthocenter and midpoints to vertices
  ctz-def-orthocenter("H", "A", "B", "C")
  ctz-def-midpoint("MHa", "H", "A")
  ctz-def-midpoint("MHb", "H", "B")
  ctz-def-midpoint("MHc", "H", "C")

  // 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(segment: ("A", "Ha"), stroke: blue.lighten(30%) + 0.7pt)
  ctz-draw(segment: ("B", "Hb"), stroke: blue.lighten(30%) + 0.7pt)
  ctz-draw(segment: ("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"), labels: (
    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(0: (0, 0), V1: (4, 0))
  ctz-def-regular-polygon("Pent", ("A", "B", "C", "D", "E"), "O", "V1")

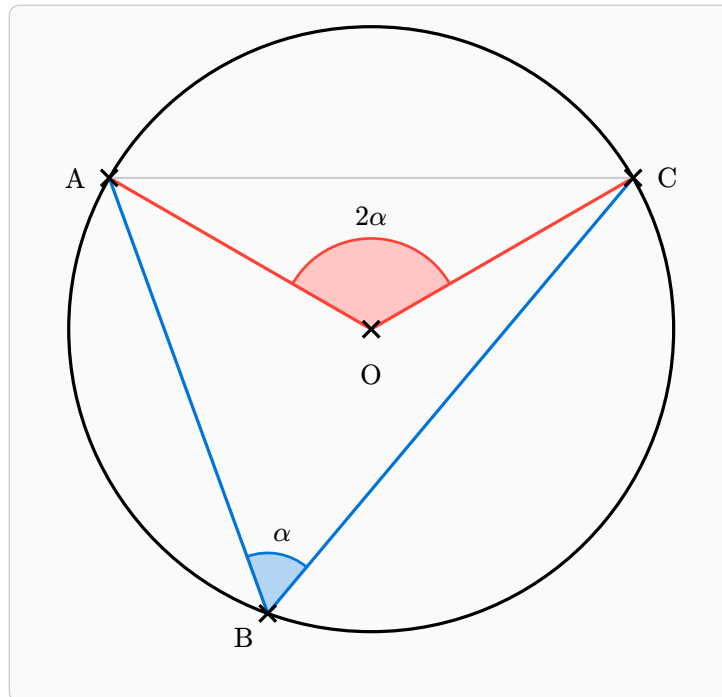
  // Pentagon
  ctz-draw("Pent", stroke: black + 1.5pt)

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

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

  ctz-draw(points: ("A", "B", "C", "D", "E", "O"), labels: (
    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 ctz.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: (_pt("O"), 4), stroke: black + 1.2pt)

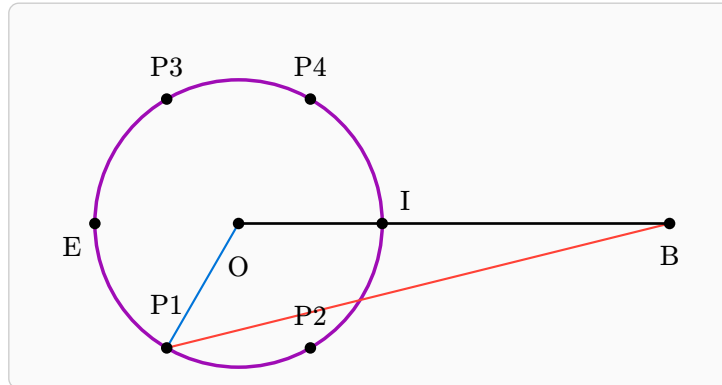
  // Place points on circle
  ctz-def-rotation("A", "R", "O", 150)
  ctz-def-rotation("C", "R", "O", 30)
  ctz-def-rotation("B", "R", "O", 250)

  // Inscribed angle at B
  ctz-draw(segment: ("A", "B"), stroke: blue + 1.2pt)
  ctz-draw(segment: ("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(segment: ("O", "A"), stroke: red + 1.2pt)
  ctz-draw(segment: ("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(segment: ("A", "C"), stroke: gray.lighten(40%) + 0.8pt)
  ctz-draw(points: ("O", "A", "B", "C"), labels: (
    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))

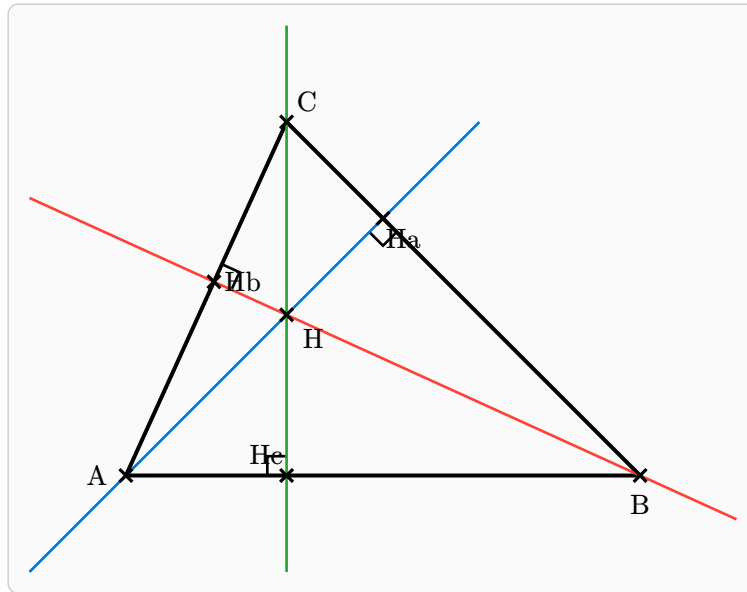
  // Center is midpoint of E and I
  ctz-def-midpoint("O", "E", "I")
  ctz-draw(circle-through: ("O", "E"), stroke: purple.darken(10%) + 1.3pt)

  // 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(segment: ("P1", "A"), stroke: blue + 0.8pt)
  ctz-draw(segment: ("P1", "B"), stroke: red + 0.8pt)
  ctz-draw(segment: ("A", "B"), stroke: black + 1pt)

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

Orthocenter and Altitudes



```
#ctz-canvas(length: 0.85cm, {
  import ctz.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)

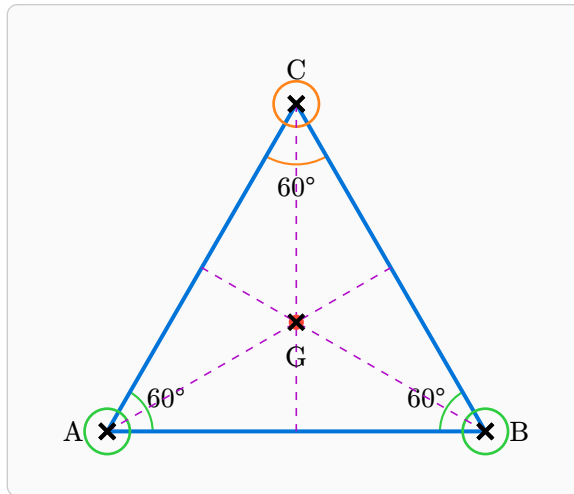
  // Orthocenter (intersection of altitudes)
  ctz-def-orthocenter("H", "A", "B", "C")

  // 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"), labels: (
    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)

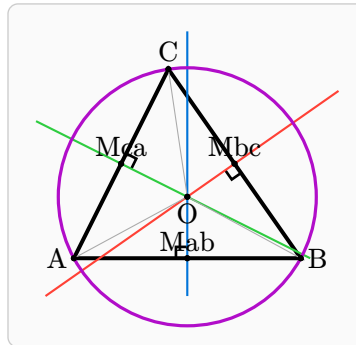
  // Draw circles at vertices
  ctz-draw(circle-r: (_pt("A"), 0.3), stroke: green + 1pt)
  ctz-draw(circle-r: (_pt("B"), 0.3), stroke: green + 1pt)
  ctz-draw(circle-r: (_pt("C"), 0.3), stroke: orange + 1pt)

  // 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(segment: ("A", "Ma"), stroke: (paint: purple, thickness: 0.6pt, dash: "dashed"))
  ctz-draw(segment: ("B", "Mb"), stroke: (paint: purple, thickness: 0.6pt, dash: "dashed"))
  ctz-draw(segment: ("C", "Mc"), stroke: (paint: purple, thickness: 0.6pt, dash: "dashed"))

  ctz-draw(points: ("A", "B", "C", "G"), labels: (
    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)

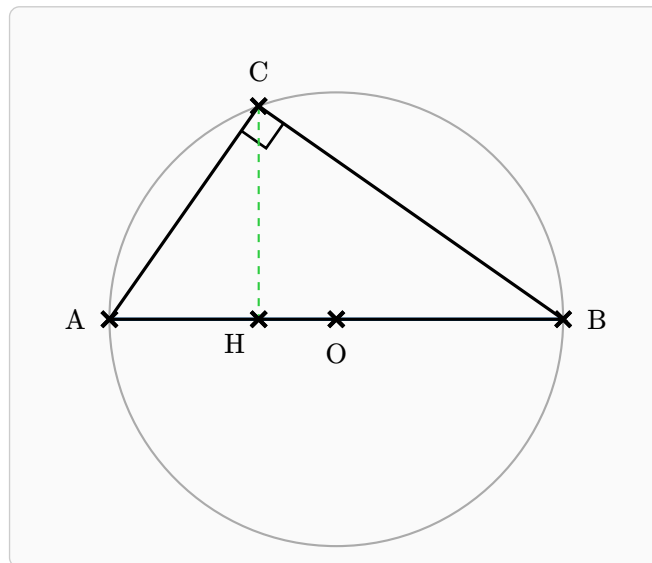
  // Draw circumcircle
  ctz-draw(circumcircle: ("A", "B", "C"), stroke: purple + 1.2pt)

  // Radii (equal length)
  ctz-draw(segment: ("O", "A"), stroke: gray + 0.4pt)
  ctz-draw(segment: ("O", "B"), stroke: gray + 0.4pt)
  ctz-draw(segment: ("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"), labels: (
    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: (_pt("O"), 3), stroke: gray + 0.8pt)
  ctz-draw(segment: ("A", "B"), stroke: blue + 1.2pt)

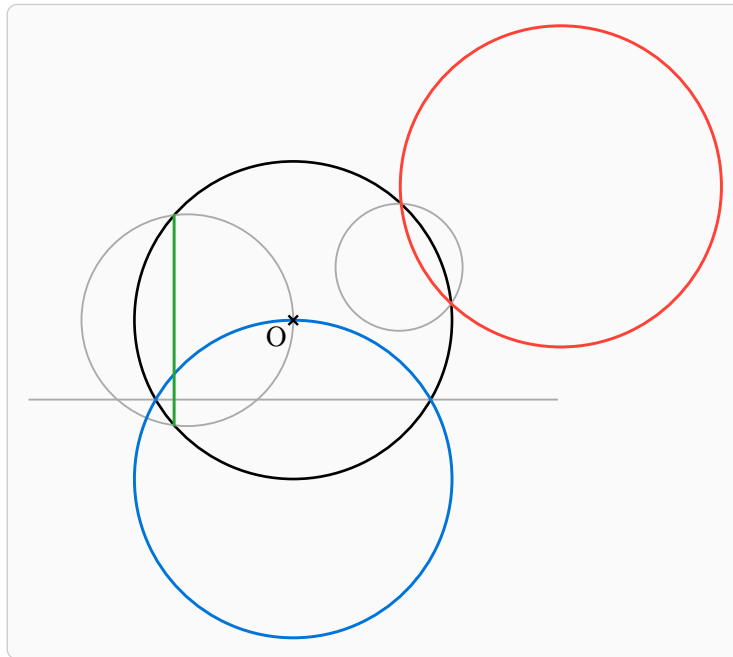
  // Draw triangle ACB
  // By Thales' theorem: angle ACB = 90° (inscribed in semicircle)
  ctz-draw(line: ("A", "C", "B", "A"), stroke: black + 1.2pt)

  // 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(segment: ("C", "H"), stroke: (paint: green, thickness: 0.8pt, dash: "dashed"))

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

Inversion



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

  ctz-set-clip(-6, -5, 6, 6)

  ctz-def-points(
    O: (0, 0),
    A: (-5, -1.5),
    B: (5, -1.5),
    C: (2, 1),
    D: (-2, 0),
  )

  ctz-def-line("l1", "A", "B")
  ctz-def-circle("c1", "C", radius: 1.2)
  ctz-def-circle("c2", "D", radius: 2)

  ctz-def-inversion("lli", "l1", "O", 3)
  ctz-def-inversion("cli", "c1", "O", 3)
  ctz-def-inversion("c2i", "c2", "O", 3)

  ctz-draw(circle-r: (_pt("O"), 3), stroke: black + 1pt)

  ctz-draw("l1", stroke: gray + 0.7pt)
  ctz-draw("c1", stroke: gray + 0.7pt)
  ctz-draw("c2", stroke: gray + 0.7pt)

  ctz-draw("lli", stroke: blue + 1.1pt)
  ctz-draw("cli", stroke: red + 1.1pt)
  ctz-draw("c2i", stroke: green.darken(20%) + 1.1pt)

  ctz-draw(points: ("O"), labels: (0: "below left"))
})
```

Inversion Packing



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

  let n = 24

  // Inversion circle setup
  ctz-def-points(0: (-4.5, 0))
  ctz-def-line("l1", (-1, 0), (-1, 1))
  ctz-def-line("l2", (1, 0), (1, 1))

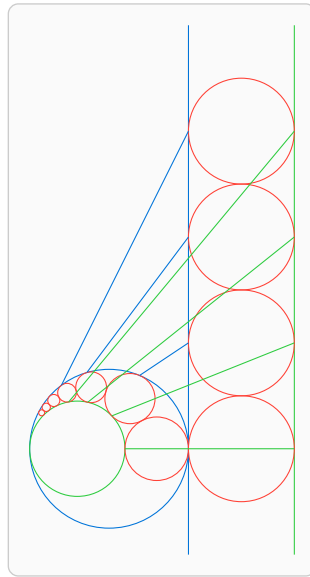
  ctz-def-inversion("l1i", "l1", "0", 1)
  ctz-def-inversion("l2i", "l2", "0", 1)

  // Draw inverted lines (outer boundary)
  ctz-draw("l1i", stroke: black + 0.5pt)
  ctz-draw("l2i", stroke: black + 0.5pt)

  // Invert a stack of circles to create the packing
  for i in range(-n, n + 1) {
    let name = "c" + str(i)
    let invname = "ci" + str(i)
    ctz-def-circle(name, (0, 2 * i), radius: 1)
    ctz-def-inversion(invname, name, "0", 1)

    ctz-draw(invname, fill: yellow, stroke: red + 0.4pt)
  }
})
```

Inversion Ladder



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

  let 0 = (-4, 0)
  let R = 3
  let n = 6

  ctz-def-line("l1", (-1, 0), (-1, 1))
  ctz-def-line("l2", (1, 0), (1, 1))
  ctz-def-inversion("l1i", "l1", 0, R)
  ctz-def-inversion("l2i", "l2", 0, R)

  ctz-set-clip(-2, -2, 4, 8)

  ctz-draw-line-global-clip((-1, 0), (-1, 1), add: (10, 10), stroke: blue + 0.4pt)
  ctz-draw-line-global-clip((1, 0), (1, 1), add: (10, 10), stroke: green + 0.4pt)
  ctz-draw("l1i", stroke: blue + 0.4pt)
  ctz-draw("l2i", stroke: green + 0.4pt)

  for i in range(0, n + 1) {
    let cname = "C" + str(i)
    let cpname = "Cp" + str(i)
    ctz-def-circle(cname, (0, 2 * i), radius: 1)
    ctz-def-inversion(cpname, cname, 0, R)
    ctz-draw(cpname, stroke: red + 0.4pt)

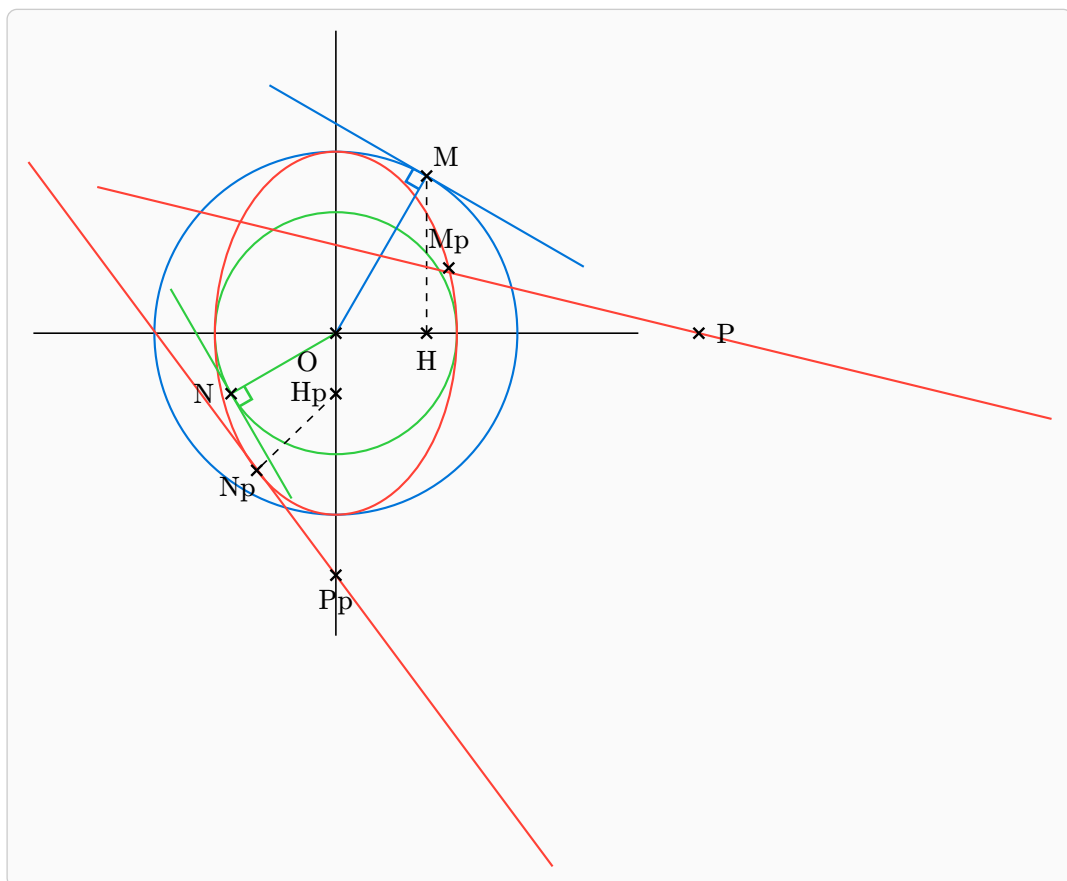
    if calc.abs(i) < 4 {
      ctz-draw(cname, stroke: red + 0.4pt)

      ctz-def-points(P1: (1, 2 * i), P2: (-1, 2 * i))
      ctz-def-inversion("P1i", "P1", 0, R)
      ctz-def-inversion("P2i", "P2", 0, R)

      ctz-draw(segment: ("P1", "P1i"), stroke: green + 0.4pt)
      ctz-draw(segment: ("P2", "P2i"), stroke: blue + 0.4pt)
    }
  }
})
```

16. Advanced Examples

Ellipse and Circles Tangency



```

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

  let a = 3
  let b = 2

  let blue-stroke = (paint: blue, thickness: 0.8pt)
  let green-stroke = (paint: green, thickness: 0.8pt)
  let red-stroke = (paint: red, thickness: 0.8pt)

  // Axes
  ctz-def-points(0: (0, 0), X1: (-5, 0), X2: (5, 0), Y1: (0, -5), Y2: (0, 5))
  ctz-draw(line: ("X1", "X2"), stroke: (paint: black, thickness: 0.6pt))
  ctz-draw(line: ("Y1", "Y2"), stroke: (paint: black, thickness: 0.6pt))

  // Circles and ellipse
  ctz-def-circle("C", "0", radius: a)
  ctz-def-circle("Cp", "0", radius: b)
  ctz-draw("C", stroke: blue-stroke)
  ctz-draw("Cp", stroke: green-stroke)
  ctz-draw(ellipse: ("0", b, a, 0deg), stroke: red-stroke)

  // Key points H, H'
  ctz-def-points(H: (a / 2, 0), Hp: (0, -b / 2))

  // Lines through H and H'
  ctz-def-perp("L1a", "L1b", ("X1", "X2"), "H")
  ctz-def-para("L2a", "L2b", ("X1", "X2"), "Hp")

  // Intersections with circles
  ctz-def-lc(("M", "M2"), ("L1a", "L1b"), "C", near: "H")
  ctz-def-lc(("N", "N2"), ("L2a", "L2b"), "Cp", near: "Hp")

  // Project M, N onto ellipse
  ctz-def-ellipse-project("Mp", "0", b, a, "M")
  ctz-def-ellipse-project("Np", "0", b, a, "N")

  // Dashed construction
  ctz-draw(line: ("H", "M"), stroke: (paint: black, thickness: 0.6pt, dash: "dashed"))
  ctz-draw(line: ("Hp", "Np"), stroke: (paint: black, thickness: 0.6pt, dash: "dashed"))

  // Radii
  ctz-draw(line: ("0", "M"), stroke: blue-stroke)
  ctz-draw(line: ("0", "N"), stroke: green-stroke)

  // Tangents at M and N
  ctz-def-tangent-at("TgM1", "TgM2", "M", "0")
  ctz-def-tangent-at("TgN1", "TgN2", "N", "0")
  ctz-draw(line: ("TgM1", "TgM2"), stroke: blue-stroke)
  ctz-draw(line: ("TgN1", "TgN2"), stroke: green-stroke)

  // Intersections with axes
  ctz-def-ll("P", ("TgM1", "TgM2"), ("X1", "X2"))
  ctz-def-ll("Pp", ("TgN1", "TgN2"), ("Y1", "Y2"))

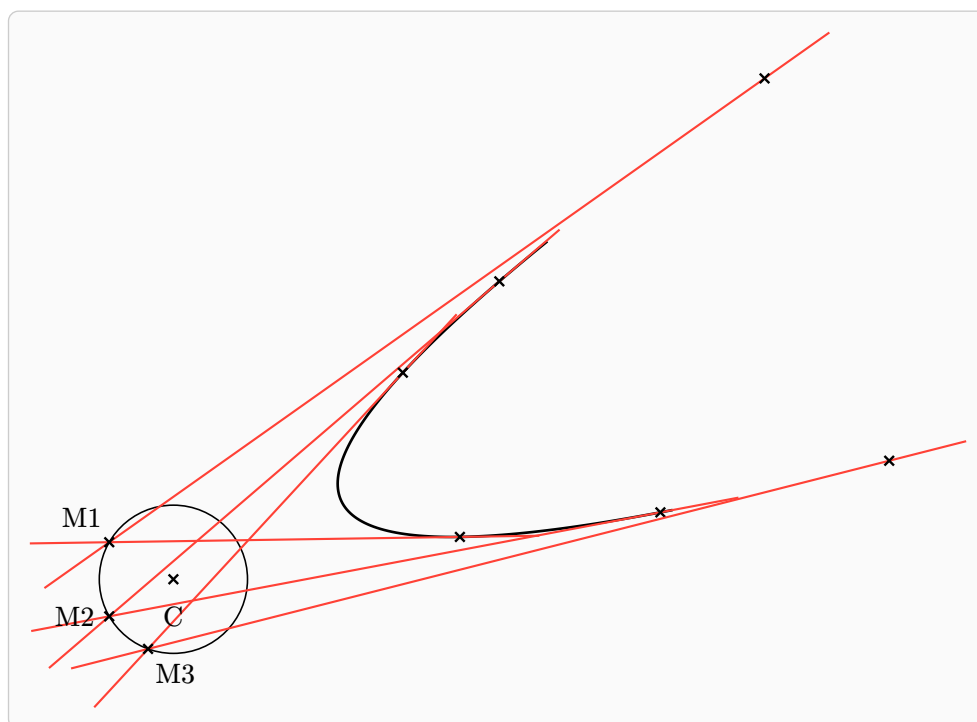
  // Perpendicular marks
  ctz-draw-mark-right-angle("TgM1", "M", "0", color: blue)
  ctz-draw-mark-right-angle("TgN1", "N", "0", color: green)

  // Ellipse tangents from P, P'
  ctz-def-ellipse-tangent-from("Tp1", "Tp2", "0", b, a, "P", "Mp")
  ctz-def-ellipse-tangent-from("Tpp1", "Tpp2", "0", b, a, "Pp", "Np")
  ctz-draw(line: ("Tp1", "Tp2"), stroke: red-stroke, mark: (end: ">", start: ">"))
  ctz-draw(line: ("Tpp1", "Tpp2"), stroke: red-stroke, mark: (end: ">", start: ">"))

  // Labels
  ctz-draw(points: ("0", "H", "Hp", "M", "Mp", "N", "Np", "P", "Pp"), labels: (
    0: (pos: "below left", offset: (-0.15, -0.15)),
    H: "below", Hp: "left", M: "above right", Mp: "above",
    N: "left", Np: "below left", P: "right", Pp: "below"
  ))
})

```


Parabola Tangents from Circle



```

#ctz-canvas(length: 0.7cm, {
  import cetz.draw: *
  import "@local/ctz-euclide:0.1.0": *
  ctz-init()

  let F = (0, 0)
  let p = 0.35
  let theta = 25deg

  ctz-set-clip(-4, -4, 12, 12)

  // Draw parabola
  ctz-draw-parabola-focus(F, p, angle: theta, extent: 2.8,
    stroke: (paint: black, thickness: 1pt))

  let (D1, D2, V) = parabola-directrix-raw(F, p, angle: theta)

  // Circle positioned to be tangent to parabola
  let C = (V.at(0) + 10 * (V.at(0) - F.at(0)),
    V.at(1) + 10 * (V.at(1) - F.at(1)))
  let r = 1.4
  ctz-draw(circle-r: (C, r), stroke: (paint: black, thickness: 0.6pt))

  // Three points on circle
  let M1 = (C.at(0) + r * calc.cos(150deg), C.at(1) + r * calc.sin(150deg))
  let M2 = (C.at(0) + r * calc.cos(210deg), C.at(1) + r * calc.sin(210deg))
  let M3 = (C.at(0) + r * calc.cos(250deg), C.at(1) + r * calc.sin(250deg))

  // Draw tangent lines from each M point to parabola
  for M in (M1, M2, M3) {
    let tangents = parabola-tangents-from-point-raw(
      F, D1, D2, M, length: 1.5)
    for t in tangents {
      let (p1, p2, tp) = t
      ctz-draw(line: (p1, p2), stroke: (paint: red, thickness: 0.8pt))
      ctz-draw(points: (tp,), style: (shape: "dot", size: 2pt, fill: blue))
    }
  }

  // Label points
  ctz-def-points(C: C, M1: M1, M2: M2, M3: M3)
  ctz-draw(points: ("C", "M1", "M2", "M3"), labels: (
    C: (pos: "below", offset: (0, -0.25)),
    M1: (pos: "above left", offset: (-0.2, 0.15)),
    M2: (pos: "left", offset: (-0.2, 0)),
    M3: (pos: "below right", offset: (0.2, -0.15)),
  ))
})

```