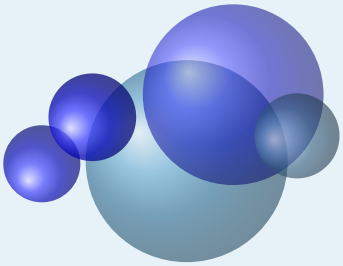
tkz-elements 2.00c

AlterMundus

tkz-elements 2.00c

Euclidean Geometry



Alain Matthes

February 4, 2024 Documentation V.2.00c

http://altermundus.fr

tkz-elements

Alain Matthes

AlterMundus

☞ This document brings together some notes about **tkz-elements**, the first version of a library written in lua, allowing to make all the necessary calculations to define the objects of a Euclidean geometry figure. You need to compile with LuaLaTeX.
With tkz-elements, the definitions and calculations are only done with lua.
The main possibility of programmation proposed is oriented "object programming" with object classes like point, line, triangle, circle and ellipse. For the moment, once the calculations are done, it is tkz-euclide or TikZ which allows the drawings.
I discovered Lua and object-oriented programming when I created this package, so it's highly probable that I've made a few mistakes. If you'd like to participate in the development of this package or give me advice on how to proceed, please contact me via my email.
English is not my native language so there might be some errors.

☞ Acknowledgements : I received much valuable advice, remarks, corrections from Nicolas Kisselhoff, David Carlisle, Roberto Giacomelli and Qrrbrbirlbel.
Thanks to Wolfgang Büchel, for correcting the examples.

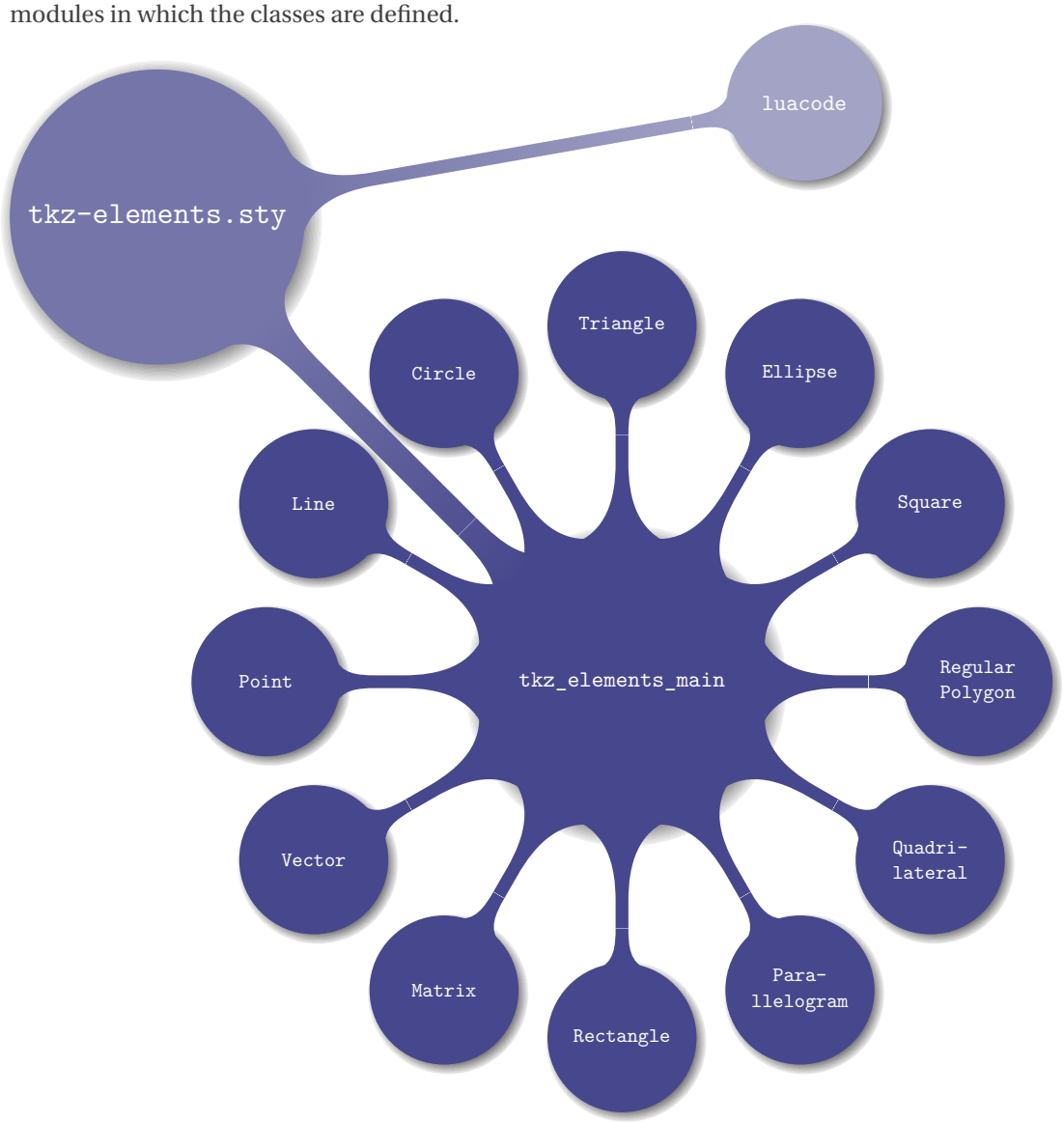
☞ I would also like to thank Eric Weisstein, creator of MathWorld.

☞ You can find some examples on my site: altermundus.fr. under construction!

Contents

1 Structure

tkz-elements.sty loads the luacode package, to create the **tkzelements** environment based on the **luacode** environment.
The **tkzelements** environment initializes scale to 1 and then deletes all the values in the various tables.
The package defines the two macros \tkzGetNodes and \tkzUseLua.
The package loads the file tkz_elements_main.lua. This file initialise all the tables that will be used by the modules in which the classes are defined.



The current classes are (some are still inactive):

- active : point (z) ; line (L) ; circle (C) ; triangle (T) ; ellipse (E) ; quadrilateral (Q) ; square (S) ; rectangle (R) ; parallelogram (P) ; regular_polygon (RP).
- inactive : matrix (M) ; vector (V).

If name is name of a class, you can find its definition in the file tkz_elements_name.lua.

2 Why tkz-elements?

2.1 Calculation accuracy

2.1.1 Calculation accuracy in TikZ

With Ti*k*Z, `veclen(x,y)` calculates the expression $\sqrt{x^2+y^2}$. This calculation is obtained using a polynomial approximation, based on ideas from `Rouben Rostamian`.

```
pgfmathparse{veclen(65,72)} \pgfmathresult
```

☞ $\sqrt{65^2+72^2} \approx 96.9884$.

2.1.2 Calculation accuracy in Lua

A `luaveclen` macro can be defined as follows:

```
\def\luaveclen#1#2{\directlua{tex.print(string.format(
'\percentchar.5f',math.sqrt((#1)*(#1)+(#2)*(#2))))}}
```

and

```
\luaveclen{65}{72}
```

gives
☞ $\sqrt{65^2+72^2} = 97$!!
The error isn't important if it's a hundredth of a `pt` for the placement of an object on a page, but it's unpleasant for the result of a calculation in a mathematical demonstration. What's more, these inaccuracies can combine to produce erroneous constructions.

To remedy this lack of precision, I first introduced the package `fp`, then the package `xfp`. Lately, with the arrival of luaLaTeX, I have been able to add a **Lua** option whose goal was to perform some calculations with **Lua**.
This was the primary reason for creating the package, the second being the introduction of object-oriented programming and easier programming with Lua. Object-oriented programming (oop) convinced me to further develop all the possibilities this method offered.
At that moment, I had received some examples of programming with **Lua** from `Nicolas Kisselhoff`, but I didn't understand its code, so I had to patiently study Lua. Finally, I was able to build tkz-elements, I took many of his ideas I've adapted.

2.1.3 Using objects

Then, I read an article[1] by `Roberto Giacomelli` on object programming based on the **Lua** and Ti*k*Z tools. This was my second source of inspiration. Not only could the programming be done step-by-step, but the introduction of objects allowed the link between the code and the geometry. The code becomes more readable, more explicit and better structured.

2.1.4 Example: Apollonius circle

> Problem The goal is to determine an inner tangent circle to the three exinscribed circles of a triangle.

See MathWorld for more details.

1 Grafica ad oggetti con LuaTEX

This example was my reference for testing the tkz-euclide package. With my first methods and the tools at my disposition, the results lacked precision. Now, with tkz-elements, I can use tools that are more powerful, more precise and easier to create.
The essential principles of figure construction with **tkz-euclide** are kept: definitions, calculations, tracings, labels as well as the step-by-step programmation, corresponding to a construction with a ruler and a compass. This is the version that uses the simplest construction method, made possible by Lua.

```
\begin{tkzelements}
  scale           = .4
  z.A             = point: new (0,0)
  z.B             = point: new (6,0)
  z.C             = point: new (0.8,4)
  T.ABC           = triangle : new ( z.A,z.B,z.C )
  z.N             = T.ABC.eulercenter
  z.S             = T.ABC.spiekercenter
  T.feuerbach     = T.ABC : feuerbach ()
  z.Ea,z.Eb,z.Ec  = get_points ( T.feuerbach )
  T.excentral     = T.ABC : excentral ()
  z.Ja,z.Jb,z.Jc  = get_points ( T.excentral )
  C.JaEa          = circle: new (z.Ja,z.Ea)
  C.ortho         = circle: radius (z.S,math.sqrt(C.JaEa: power(z.S)))
  z.a             = C.ortho.through
  C.euler         = T.ABC: euler_circle ()
  C.apo           = C.ortho : inversion (C.euler)
  z.O             = C.apo.center
  z.xa,z.xb,z.xc  = C.ortho : inversion (z.Ea,z.Eb,z.Ec)
\end{tkzelements}
```

The creation of an object encapsulates its attributes (its characteristics) and methods (i.e. the actions that are specific to it). It is then assigned a reference (a name), which is linked to the object using a table. The table is an associative array that links the reference called key to a value, in this case the object. These notions will be developed later.
T is a table that associates the object triangle with the key ABC. T.ABC is also a table, and its elements are accessed using keys that are attributes of the triangle. These attributes have been defined in the package.

```
z.N = T.ABC.eulercenter
```

N is the name of the point, eulercenter is an attribute of the triangle. [2]

```
T.excentral     = T.ABC : excentral ()
```

Here, excentral is a method linked to the T.ABC object. It defines the triangle formed by the centers of the exinscribed circles.
Two lines are important. The first below shows that the excellent precision provided by Lua makes it possible to define a radius with a complex calculation. The radius of the radical circle is given by $\sqrt{\Pi(S,\mathcal{C}(Ja,Ea))}$ (square root of the power of point S with respect to the exinscribed circle with center Ja passing through Ea).

```
C.ortho  = circle: radius (z.S,math.sqrt(C.JaEa: power(z.S)))
```

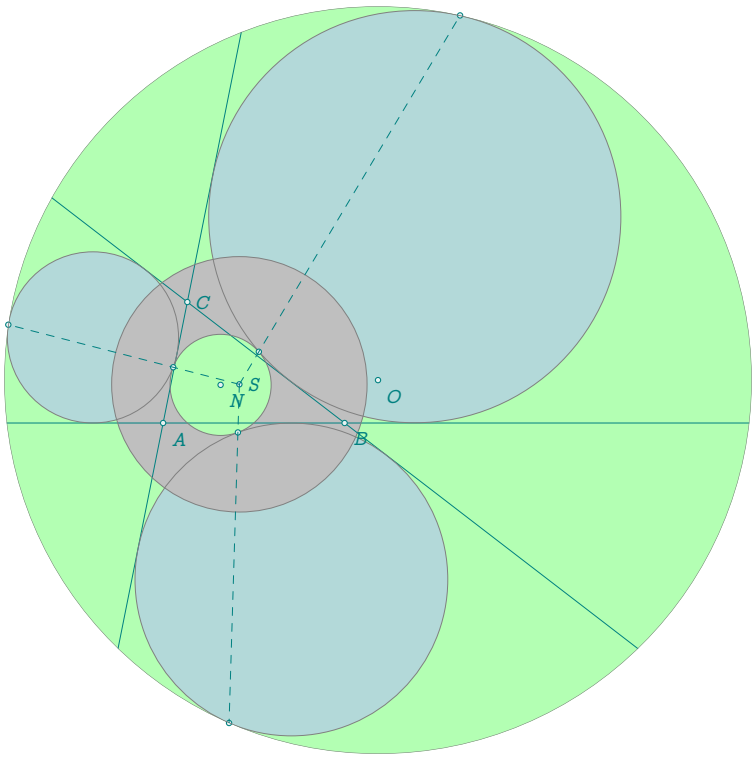
2 The center of the Euler circle, or center of the nine-point circle, is a characteristic of every triangle.

Finally, the inversion of the Euler circle with respect to the radical circle is the Apollonius circle[3]. The transformation has an object as parameter, which is recognized by its type (all objects are typed in the package), and the method determines which algorithm to use according to this type.

```
C.apo   = C.ortho : inversion (C.euler)
```

Now that all the points have been defined, it's time to start drawing the paths. To do this, you need to create the nodes. This is the role of the macro . See 6.1.1
The following section concerns only drawings, and is handled by `tkz-euclide`.

```
\begin{tikzpicture}
   \tkzGetNodes
   \tkzFillCircles[green!30](O,xa)
   \tkzFillCircles[teal!30](Ja,Ea Jb,Eb Jc,Ec)
   \tkzFillCircles[lightgray](S,a)
   \tkzFillCircles[green!30](N,Ea)
   \tkzDrawPoints(xa,xb,xc)
   \tkzClipCircle(O,xa)
   \tkzDrawLines[add=3 and 3](A,B A,C B,C)
   \tkzDrawCircles(Ja,Ea Jb,Eb Jc,Ec S,a O,xa N,Ea)
   \tkzDrawPoints(O,A,B,C,S,Ea,Eb,Ec,N)
   \tkzDrawSegments[dashed](S,xa S,xb S,xc)
   \tkzLabelPoints(O,N,A,B)
   \tkzLabelPoints[right](S,C)
\end{tikzpicture}
```



3 The nine-point circle, or Euler circle, is externally tangent to the three circles. The points of tangency form Feuerbach's triangle.

3 Presentation

3.1 With Lua

The purpose of tkz-elements is simply to calculate dimensions and define points. This is done in Lua. You can think of tkz-elements as a kernel that will be used either by tkz-euclide or by TikZ, see MetaPost. Definitions and calculations are done inside the environment **tkzelements**, this environment is based on **luacode**.

The key points are:

- the source file must be ☞ utf8 encoded;
- compilation is done with ☞ LuaLaTeX;
- you need to load TikZ ou tkz-euclide and tkz-elements;
- definitions and calculations are performed in an orthonormal sytem of reference, using Lua, and are carried out in an environment of tkzelements.

To the right, see the minimum template.
The code is divided into two parts, which are two environments **tkzelements** and **tikzpicture**. In the first environment, you place your Lua code, and in the second, tkz-euclide commands.

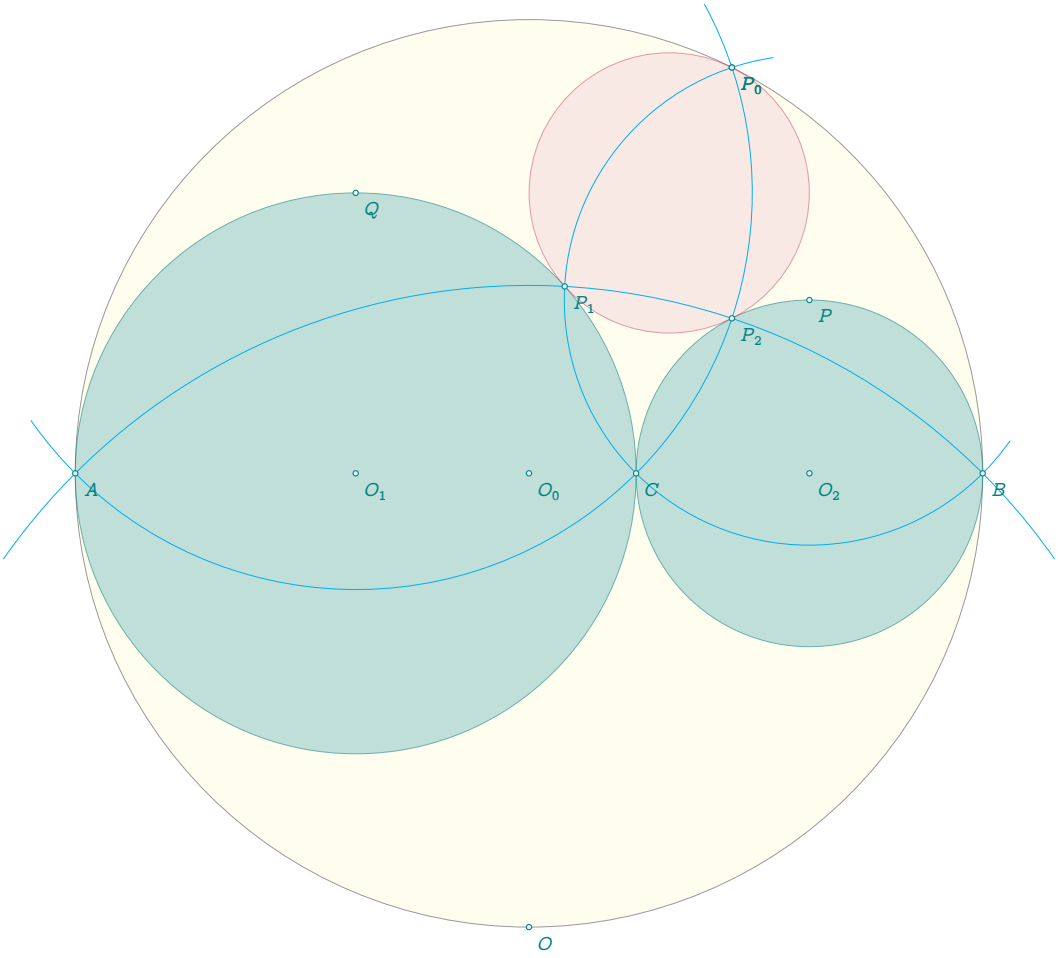
```
% !TEX TS-program = lualatex
% Created by Alain Matthes
\documentclass{standalone}
\usepackage{tkz-euclide}
% or simply TikZ
\usepackage{tkz-elements}
begin{document}

\begin{tkzelements}
   scale = 1
% definition of some points
z.A = point : new (   ,   )
z.B = point : new (   ,   )

 ...code...
\end{tkzelements}

\begin{tikzpicture}
% point transfer to Nodes
\tkzGetNodes

\end{tikzpicture}
\end{document}
```

3.2 The main process

When all the points necessary for the drawing are obtained, they must be transformed into **nodes** so that TikZ or tkz-euclide can draw the figure. This is done through the macro **\tkzGetNodes**. This macro browse all the elements of the table z using the key (in fact the name of the point) and retrieves the values associated with it, i.e. the coordinates of the point (node).

3.3 Complete example: Pappus circle

3.3.1 The figure



3.3.2 The code

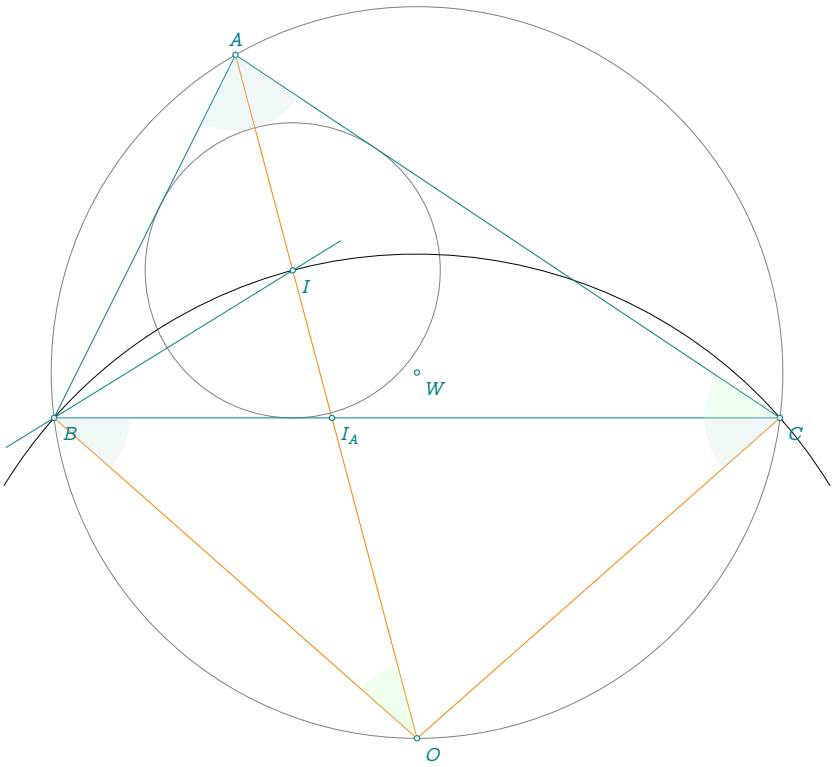
```
% !TEX TS-program = lualatex
\documentclass{article}
\usepackage{tkz-euclide}
\usepackage{tkz-elements}
\begin{document}

\begin{tkzelements}
z.A     = point: new (0 , 0)
z.B     = point: new (10 , 0)        --  creation of two fixed points $A$ and $B$
L.AB    = line:  new ( z.A, z.B)
z.C     = L.AB:  gold_ratio ()       --  use of a method linked to "line"
z.O_0   = line:  new ( z.A, z.B).mid  -- midpoint of segment with an attribute of "line"
z.O_1   = line:  new ( z.A, z.C).mid --  objects are not stored and cannot be reused.
z.O_2   = line:  new ( z.C, z.B).mid
C.AB    = circle: new ( z.O_0, z.B)  --  new object "circle" stored and reused
C.AC    = circle: new ( z.O_1, z.C)
C.CB    = circle: new ( z.O_2, z.B)
z.P     = C.CB.north                 --  no"rth atrributes of a circle
```

```
z.Q     = C.AC.north
z.O     = C.AB.south
z.c     = z.C : north (2)          --   "north" method of a point (needs a parameter)
C.PC    = circle: new ( z.P, z.C)
C.QA    = circle: new ( z.Q, z.A)
z.P_0   = intersection (C.PC,C.AB)   --  search for intersections of two circles.
z.P_1   = intersection (C.PC,C.AC)  --   idem
_,z.P_2 = intersection (C.QA,C.CB)   --  idem
z.O_3   = triangle: new ( z.P_0, z.P_1, z.P_2).circumcenter -- circumcenter attribute of "triangle"
\end{tkzelements}

\begin{tikzpicture}
  \tkzGetNodes
  \tkzDrawCircle[black,fill=yellow!20,opacity=.4](O_0,B)
  \tkzDrawCircles[teal,fill=teal!40,opacity=.6](O_1,C O_2,B)
  \tkzDrawCircle[purple,fill=purple!20,opacity=.4](O_3,P_0)
  \tkzDrawArc[cyan,delta=10](Q,A)(P_0)
  \tkzDrawArc[cyan,delta=10](P,P_0)(B)
  \tkzDrawArc[cyan,delta=10](O,B)(A)
  \tkzDrawPoints(A,B,C,O_0,O_1,O_2,P,Q,P_0,P_0,P_1,P_2,O)
  \tkzLabelPoints(A,B,C,O_0,O_1,O_2,P,Q,P_0,P_0,P_1,P_2,O)
\end{tikzpicture}
\end{document}
```

3.4 Another example with comments: South Pole

Here's another example with comments

```
% !TEX TS-program = lualatex
\documentclass{standalone}
\usepackage{tkz-euclide,tkz-elements}
\begin{document}
\begin{tkzelements}
   z.A      = point: new (2 , 4)          -- we create environment tkzelements
   z.B      = point: new (0 , 0)          -- three fixed points are used
   z.C      = point: new (8 , 0)
   T.ABC    = triangle: new (z.A,z.B,z.C) -- we create a new triangle object
   C.ins    = T.ABC: in_circle ()         -- we get the incircle of this triangle
   z.I      = C.ins.center                -- center is an attribute of the circle
   z.T      = C.ins.through               -- through is also an attribute
   -- z.I,z.T  = get_points (C.ins)       -- get_points is a shortcut
   C.cir    = T.ABC : circum_circle ()    -- we get the  circumscribed circle
   z.W      = C.cir.center                -- we get the center of this circle
   z.O      = C.cir.south                 -- now we get the south pole of this circle
   L.AO     = line: new (z.A,z.O)         -- we create an object "line"
   L.BC     = T.ABC.bc                    -- we get the line (BC)
   z.I_A    = intersection (L.AO,L.BC)    --  we search the intersection of the last lines
\end{tkzelements}
```



Here's the tikzpicture environment to obtain the drawing:

```
\begin{tikzpicture}
\tkzGetNodes
\tkzDrawCircles(W,A I,T)
\tkzDrawArc(O,C)(B)
\tkzDrawPolygon(A,B,C)
\tkzDrawSegments[new](A,O B,O C,O)
\tkzDrawLine(B,I)
\tkzDrawPoints(A,B,C,I,I_A,W,O)
\tkzFillAngles[green!20,opacity=.3](A,O,B A,C,B)
\tkzFillAngles[teal!20,opacity=.3](O,B,C B,C,O B,A,O O,A,C)
\tkzLabelPoints(I,I_A,W,B,C,O)
\tkzLabelPoints[above](A)
\end{tikzpicture}
```

4 Writing Convention

4.1 Miscellaneous

- Numerical variable: the writing conventions for real numbers are the same as for Lua.
- Complex numbers: as for real numbers but to define them you must write `za = point (1,2)`. Mathematically, this corresponds to 1+2i, which you can find with `tex.print(tostring(za))`.(see 21.3)
- Boolean: you can write `bool = true` or `bool = false` then with Lua you can use the code :

```
if bool == ... then ... else ... end
```

and outside the environment **tkzelements** you can use the macro

```
\ifthenelse{\equal{\tkzUseLua{bool}}{true}}{ ... }{ ... }
```

after loading the `ifthen` package.

- String: if st = "Euler's formula" then

```
\tkzUseLua{st} gives Euler's formula
```

4.2 Assigning a Name to a Point

Currently the only obligation is to store the points in the table z [4] if you want to use them in Ti*k*Z or `tkz-euclide`. If it is a point which will not be used, then you can designate it as you wish by respecting the conventions of Lua. The points which occur in the environment **tkzelements** must respect a convention which is `z.name` such that `name` will be the name of the corresponding **node**.
What are the conventions for designating `name`? You have to respect the Lua conventions in particular cases.

1. The use of prime is problematic. If the point name contains more than one symbol and ends with `p` then when passing into TikZ or `tkz-euclide`, the letters `p` will be replaced by `'` using the macro **\tkzGetNodes**;

2. One possibility, however, in order to have a more explicit code is to suppose that you want to designate a point by "euler". It would be possible for example to write `euler = ...`, and at the end of the code for the transfer, `z.E = euler`. It is also possible to use a temporary name `euler` and to replace it in Ti*k*Z. Either at the time of placing the labels, or for example by using `pgfnodealias{E}{euler}`. This possibility also applies in other cases: prime, double prime, etc.

Here are some different ways of naming a point:

- `z.A = point : new (1,2)`
- `z.Bp = point : new (3,4)` –> this gives `B'` in the **tikzpicture**
- `z.H_a = T.ABC : altitude ()` –> this gives `H_a` in the **tikzpicture** code and H_a in the display.

4 To place the point M in the table, simply write `z.M` = …or `z["M"]` = …

4.3 Assigning a Name to Other Objects

You have the choice to give a name to objects other than points. That said, it is preferable to respect certain rules in order to make the code easier to read. I have chosen for my examples the following conventions: first of all I store the objects in tables: `L.name` for lines and segments, `C.name` for circles, `T.name` for triangles, `E.name` for ellipses.
For lines, I use the names of the two points. So if a line passes through points A and B, I name the line `L.AB`.
For circles, I name `C.AB` the circle of center A passing through B, but something like `C.euler` or `C.external` is fine.
For triangles, I name `T.ABC` the triangle whose vertices are A, B and C but `T.feuerbach`.
For ellipses, I name `E.ABC` the ellipse with center A through vertex B and covertex C.

4.4 Writing conventions for attributes, methods.

You must use the conventions of Lua, so

- To obtain an , for all objects, the convention is identical: `object.attribute`. For example, for the point A we access its abscissa with `z.A.re` and its ordinate with `z.A.im`; as for its type we obtain it with `z.A.type`. To get the south pole of the circle `C.OA` you need to write: `C.OA.south`.
- To use a method such as obtaining the incircle of a triangle ABC, just write

  ```
  C.incircle = T.ABC : in_circle ().
  ```

- Some methods need a parameter. For example, to know the distance between a point C to the line (A,B) we will write

  ```
  d = L.AB : distance (z.C).
  ```

- Use the to store a result you don't want to use. If you only need the second point of an intersection between a line and a circle, you would write

  ```
  _,z.J = intersection (L.AB , C.OC).
  ```

5 Work organization

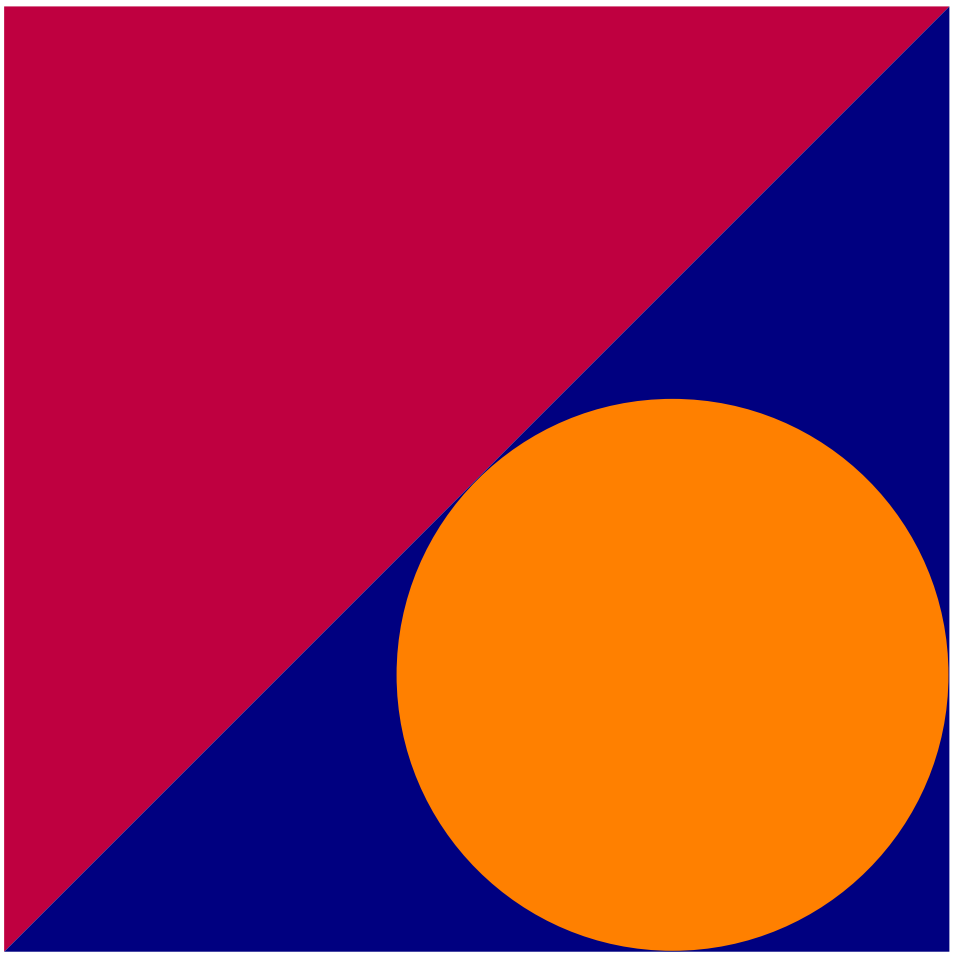
Here's a sample organization.
The line `% !TEX TS-program = lualatex` ensures that you don't forget to compile with LuaLaTeX. The "standalone" class is useful, as all you need to do here is create a figure.
The package `ifthen` is useful if you need to use some Boolean.
The macro **\LuaCodeDebugOn** allows you to try and find errors in Lua code.
It is of course possible to leave the Lua code in the **tkzelements** environment, but externalizing this code has its advantages.
The first advantage, if you use a good editor, is to have a good presentation of the code. Styles are different between "Lua" and LaTeX. This makes the code clearer. This is how I proceeded, then reintegrated the code into the main code.
Another advantage is that you don't have to comment the code incorrectly. For Lua code, you comment lines with `--` (double minus sign), whereas for LaTeX, you comment with `%`.
Third advantage: the code can be reused.

```
% !TEX TS-program = lualatex
% Created by Alain Matthes on 2024-01-09.

\documentclass[margin = 12pt]{standalone}
\usepackage{tkz-euclide}
\usepackage{tkz-elements,ifthen}
```

```
\begin{document}
\LuaCodeDebugOn
\begin{tkzelements}
 scale = 1.25
 dofile ("sangaku.lua")
\end{tkzelements}

\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawCircle(I,F)
   \tkzFillPolygon[color = purple](A,C,D)%
   \tkzFillPolygon[color = blue!50!black](A,B,C)%
   \tkzFillCircle[color = orange](I,F)%
\end{tikzpicture}
\end{document}
```

And here is the code for the "Lua" part: the file `ex_sangaku.lua`

```
z.A         = point : new ( 0,0 )
z.B         = point : new ( 8,0 )
L.AB        = line : new ( z.A , z.B )
S           = L.AB : square ()
_,_,z.C,z.D = get_points (S)
z.F         = S.ac : projection (z.B)
L.BF        = line : new (z.B,z.F)
T.ABC       = triangle : new ( z.A , z.B , z.C )
L.bi        = T.ABC : bisector (2)
z.c         = L.bi.pb
L.Cc        = line : new (z.C,z.c)
z.I         = intersection (L.Cc,L.BF)
```



5.1 Scale problem

If necessary, it's better to do the scaling in the "Lua" section. The reason is that it will be more accurate. There is, however, a problem to be aware of. I've made it a point of honor to avoid using numerical values in my codes whenever possible. In principle, these values only appear in the definition of fixed points. If the "scale" option is used, scaling is applied when points are created. Let's imagine you want to organize your code as follows:

```
scale = 1.5
xB = 8
z.B         = point : new ( xB,0 )
```

Scaling would then be ineffective, as the numerical values are not modified, only the point coordinates. To take scaling into account, use the function `value (v)` .

```
scale = 1.5
xB = value (8)
z.B         = point : new ( xB,0 )
```

5.2 Code presentation

The key point is that, unlike LaTeX or TeX, you can insert spaces absolutely anywhere.

6 Transfers

6.1 Fom Lua to tkz-euclide or TikZ

In this section, we'll look at how to transfer points, Booleans and numerical values.

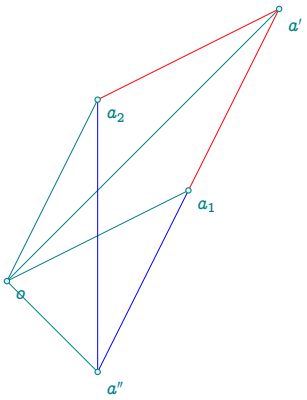
6.1.1 Points transfer

We use an environment **tkzelements** outside an environment **tikzpicture** which allows us to carry out all the necessary calculations, then we launch the macro which transforms the affixes of the table **z** into **Nodes**. It only remains to draw.
Currently the drawing program is either Ti*k*Z or tkz-euclide. You have the possibility to use another package to trace but for that you have to create a macro similar to **\tkzGetNodes**. Of course, this package must be able to store the points as does Ti*k*Z or tkz-euclide.

```
\def\tkzGetNodes{\directlua{%
   for K,V in pairs(z) do
      local n,sd,ft
      n = string.len(K)
      if n >1 then
      _,_,ft, sd = string.find( K , "(.+)(.)" )
     if sd == "p" then   K=ft.."'" end
     _,_,xft, xsd = string.find( ft , "(.+)(.)" )
     if xsd == "p" then  K=xft.."'".."'" end
       end
  tex.print("\\coordinate ("..K..") at ("..V.re..","..V.im..") ;\\\\\")
end}
}
```

See the section In-depth Study 21 for an explanation of the previous code.
The environment **tkzelements** allows to use the underscore _ and the macro **\tkzGetNodes** allows to obtain names of nodes containing **prime** or **double prime**. (see the next example)

```
\begin{tkzelements}
   scale = 1.2
   z.o   = point: new (0,0)
   z.a_1 = point: new (2,1)
   z.a_2 = point: new (1,2)
   z.ap  = z.a_1 + z.a_2
   z.app = z.a_1 - z.a_2
\end{tkzelements}
\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawSegments(o,a_1 o,a_2 o,a' o,a'')
   \tkzDrawSegments[red](a_1,a' a_2,a')
   \tkzDrawSegments[blue](a_1,a'' a_2,a'')
   \tkzDrawPoints(a_1,a_2,a',o,a'')
   \tkzLabelPoints(o,a_1,a_2,a',a'')
\end{tikzpicture}
```

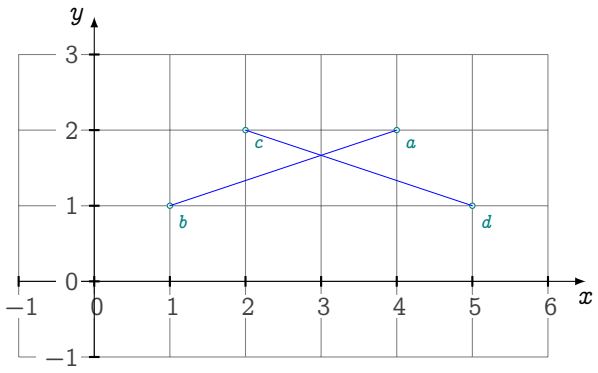


6.1.2 Other transfers

Sometimes it's useful to transfer angle, length measurements or boolean. For this purpose, I have created the macro (see 19.5) `tkzUseLua(value)`

```
\begin{tkzelements}
   z.b = point:  new (1,1)
   z.a = point:  new (4,2)
   z.c = point:  new (2,2)
   z.d = point:  new (5,2)
   L.ab = line : new (z.a,z.b)
   L.cd = line : new (z.c,z.d)
    det = (z.b-z.a)^(z.d-z.c)
    if det == 0 then bool = true
      else bool = false
    end
    x = intersection (L.ab,L.cd)
\end{tkzelements}

The intersection of the two lines lies at
    a point whose affix is:\tkzUseLua{x}

\begin{tikzpicture}
   \tkzGetNodes
    \tkzDrawPoints(a,...,d)
    \ifthenelse{\equal{\tkzUseLua{bool}}{true}}{
    \tkzDrawSegments[red](a,b c,d)}{%
    \tkzDrawSegments[blue](a,b c,d)}
     \tkzLabelPoints(a,...,d)
\end{tikzpicture}
```

The intersection of the two lines lies at a point whose affix is: 3.0+1.66666666666667i



7 Class and object

7.1 Class

Object-oriented programming (OOP) is defined as a programming model built on the concept of objects. An object can be defined as a data table that has unique attributes and methods (operations) that define its behavior.

A class is essentially a user-defined data type. It describes the contents of the objects that belong to it. A class is a blueprint of an object, providing initial values for attributes and implementations of methods[5] common to all objects of a certain kind.

7.2 Object

An Object is an instance of a class. Each object contains attributes and methods. Attributes are information or object characteristics stored in the date table (called field). The methods define behavior.

All objects in the package are typed. The object types currently defined and used are: **point**, **line**, **circle**, **triangle**, **ellipse**, **quadrilateral**, **square**, **rectangle**, **parallelogram** and **regular_polygon**.
They can be created directly using the method `new` by giving points, with the exception of the `class`point class which requires a pair of reals, and `class`regular_polygon which needs two points and an integer.
Objects can also be obtained by applying methods to other objects. For example, `T.ABC : circum_circle ()` creates an object **circle**. Some object attributes are also objects, such as `T.ABC.bc` which creates the object **line**, a straight line passing through the last two points defining the triangle.

7.2.1 Attributes

Attributes are accessed using the classic method, so `T.pc` gives the third point of the triangle and `C.OH.center` gives the center of the circle, but I've added a `get_points` function that returns the points of an object. This applies to straight lines (pa and pc), triangles (pa, pb and pc) and circles (center and through).

Example: `z.O,z.T = get_points (C)` recovers the center and a point of the circle.

7.2.2 Methods

A method is an operation (function or procedure) associated (linked) with an object.
Example: The point object is used to vertically determine a new point object located at a certain distance from it (here 2). Then it is possible to rotate objects around it.

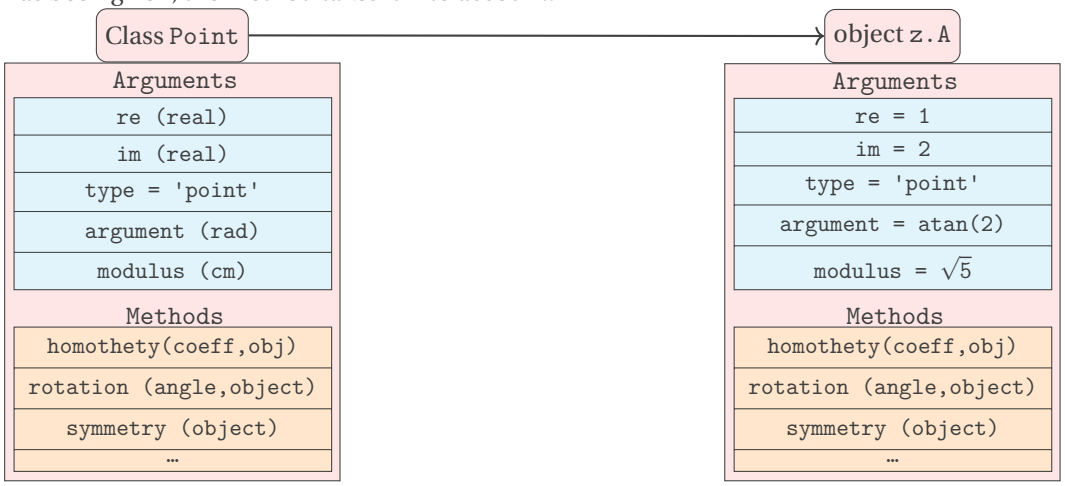
```
\begin{tkzelements}
   z.A = point (1,0)
   z.B = z.A : north (2)
   z.C = z.A : rotation (math.pi/3,z.B)
   tex.print(tostring(z.C))
\end{tkzelements}
```

The coordinates of C are: -0.73205080756888 and 1.0

5 action which an object is able to perform.

8 Class point

The class on which the whole edifice rests, it's the class `point`. This class is hybrid in the sense that it is as much about points of a plane as complex numbers. The principle is the following: the plane is provided with an orthonormal basis which allows us to determine the placement of a point using its abscissa and ordinate coordinates; in the same way any complex number can simply be considered as a pair of real numbers (its real part and its imaginary part). We can then designate the plane as the complex plane, and the complex number $x+iy$ is represented by the point of the plane with coordinates (x,y). Thus the point A will have coordinates stored in the object $z.A$. Coordinates are attributes of the "point" object, like type, argument and modulus.
The creation of a point is done using the following method, but there are other possibilities. If a scaling factor has been given, the method takes it into account.



8.1 Attributes of a point

> Creation `z.A = point: new (1,2)`

The point A has coordinates $x=1$ and $y=2$. If you use the notation `z.A` then A will be the reference of a node in Ti*kZ* or in `tkz-euclide`.
This is the creation of a fixed point with coordinates 1 and 2 and which is named A. The notation `z.A` indicates that the coordinates will be stored in a table noted `z` (reference to the notation of the affixes of the complex numbers) that A is the name of the point and the key allowing access to the values.

Table 1: Point attributes.

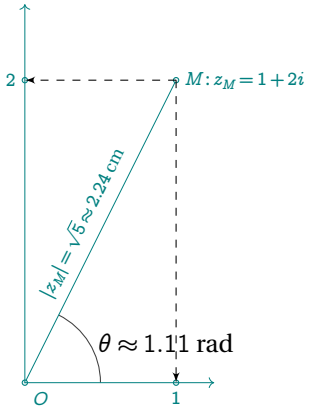
Attributes	Application	Example
re	`z.A.re = 1`	see (7.2.2)
im	`z.A.im = 2`	see (7.2.2)
type	`z.A.type = 'point'`	
argument	`z.A.argument $\approx$ 0.78539816339745`	see (8.1.1)
modulus	`z.A.modulus` $\approx$ 2.2360... $=\sqrt{5}$	see (8.1.1)

8.1.1 Example:point attributes

```
\begin{tkzelements}
   z.M = point: new (1,2)
\end{tkzelements}

\begin{tikzpicture}[scale = 1]
\pgfkeys{/pgf/number format/.cd,std,precision=2}
\let\pmpn\pgfmathprintnumber
\tkzDefPoints{2/4/M,2/0/A,0/0/O,0/4/B}
\tkzLabelPoints(O)
\tkzMarkAngle[fill=gray!30,size=1](A,O,M)
\tkzLabelAngle[pos=1,right](A,O,M){%
$\theta \approx \pmpn{\tkzUseLua{z.M.argument}}$ rad}
\tkzDrawSegments(O,M)
\tkzLabelSegment[above,sloped](O,M){%
$|z_M| =\sqrt{5}\approx \pmpn{\tkzUseLua{z.M.modulus}}$ cm}
\tkzLabelPoint[right](M){$M : z_M = 1 + 2i$}
\tkzDrawPoints(M,A,O,B)
\tkzPointShowCoord(M)
\tkzLabelPoint[below,teal](A){$\tkzUseLua{z.M.re}$}
\tkzLabelPoint[left,teal](B){$\tkzUseLua{z.M.im}$}
\tkzDrawSegments[->,add = 0 and 0.25](O,B O,A)
\end{tikzpicture}
```

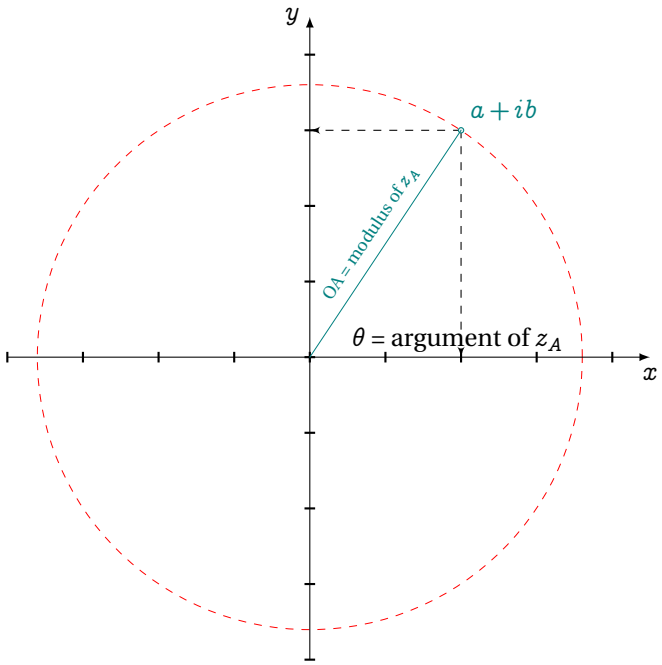
Attributes of `z.M`

- `z.M.re` = 1
- `z.M.im` = 2
- `z.M.type` = 'point'
- `z.M.argument` = $\theta \approx 1.11$ rad
- `z.M.modulus` = $|z_M| = \sqrt{5} \approx 2.24$ cm



8.1.2 Argand diagram

```
\begin{tkzelements}
   z.A = point : new ( 2 , 3 )
   z.O = point : new ( 0 , 0 )
   z.I = point : new ( 1 , 0 )
\end{tkzelements}
\hspace{\fill}\begin{tikzpicture}
   \tkzGetNodes
   \tkzInit[xmin=-4,ymin=-4,xmax=4,ymax=4]
   \tkzDrawCircle[dashed,red](O,A)
   \tkzPointShowCoord(A)
   \tkzDrawPoint(A)
   \tkzLabelPoint[above right](A){\normalsize $a+ib$}
   \tkzDrawX\tkzDrawY
   \tkzDrawSegment(O,A)
   \tkzLabelSegment[above,anchor=south,sloped](O,A){ OA = modulus of $z_A$}
  \tkzLabelAngle[anchor=west,pos=.5](I,O,A){$\theta$ = argument of $z_A$}
\end{tikzpicture}
```



8.2 Methods of the class point

The methods described in the following table are standard. You'll find them in most of the examples at the end of this documentation. The result of the different methods presented in the following table is a **point**. See section (21.3) for the metamethods.

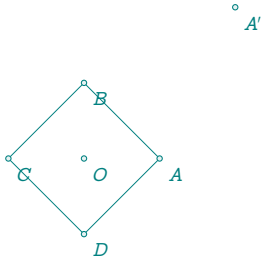
Table 2: Methods of the class point.

Methods	Application	
`new(r, r)`	`z.A = point : new(1,2)`	see (8.2.4)
`polar (d, an)`	`z.A = point : polar(1,math.pi/3)`	see (22.7)
`polar_deg an`	an in deg	polar coordinates an deg
Points		
`north(r)`	`r` distance to the point (1 if empty)	see (22.43) ; 7.2.2)
`south(r)`		
`east(r)`		
`west(r)`		
`normalize()`	`z.b = z.a: normalize ()`	see (8.2.4)
`get_points (obj)`	retrieves points from the object	
`orthogonal (d)`	`z.B=z.A:orthogonal(d)`	$\overrightarrow{OB} \perp \overrightarrow{OA}$ and $OB = d$
`at ()`	`z.X = z.B : at (z.A)`	$\overrightarrow{OB} = \overrightarrow{AX}$ and $OB = d$
Transformations		
`symmetry(obj)`	obj : point, line, etc.	see (8.2.9)
`rotation(an , obj)`	point, line, etc.	see (8.2.8)
`homothety(r,obj)`	`z.c = z.a : homothety (2,z.b)`	see (22.48)

8.2.1 Example: method north (d)

This function defines a point located on a vertical line passing through the given point. This function is useful if you want to report a certain distance (see the following example). If `d` is absent then it is considered equal to 1.

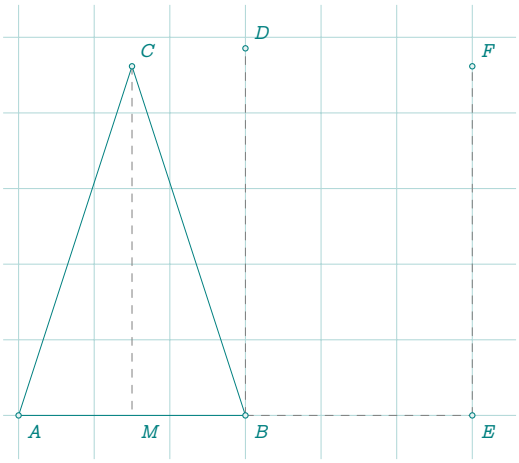
```
\begin{tkzelements}
   z.O   = point : new ( 0, 0 )
   z.A   = z.O : east ()
   z.Ap  = z.O : east (2) : north (2)
   z.B   = z.O : north ()
   z.C   = z.O : west ()
   z.D   = z.O : south ()
\end{tkzelements}
\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawPolygon(A,B,C,D)
   \tkzDrawPoints(A,B,C,D,O,A')
\end{tikzpicture}
```



8.2.2 Length transfer

Use of `north and east` functions linked to points, to transfer lengths, see (19.1)

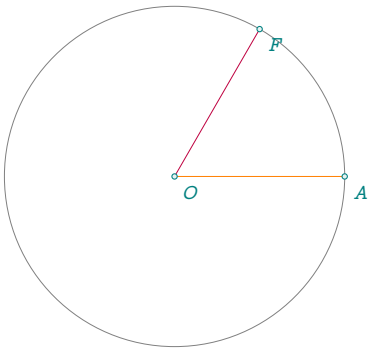
```
\begin{tkzelements}
   z.A = point : new ( 0 , 0 )
   z.B = point : new ( 3 , 0 )
   L.AB = line : new ( z.A , z.B )
   T.ABC =   L.AB : sublime ()
   z.C = T.ABC.pc
   z.D = z.B: north (length(z.B,z.C))
   z.E = z.B: east (L.AB.length)
   z.M = L.AB.mid
   z.F = z.E : north (length(z.C,z.M))
\end{tkzelements}
\begin{tikzpicture}[gridded]
   \tkzGetNodes
   \tkzDrawPolygons(A,B,C)
   \tkzDrawSegments[gray,dashed](B,D B,E E,F C,M)
   \tkzDrawPoints(A,...,F)
   \tkzLabelPoints(A,B,E,M)
   \tkzLabelPoints[above right](C,D,F)
\end{tikzpicture}
```



8.2.3 Example: method polar

This involves defining a point using its modulus and argument.

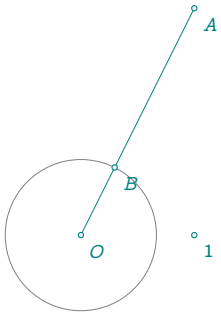
```
\begin{tkzelements}
   z.O     = point:   new  (0, 0)
   z.A     = point:   new  (3, 0)
   z.F     = point:   polar (3, math.pi/3)
\end{tkzelements}
\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawCircle(O,A)
   \tkzDrawSegments[new](O,A)
   \tkzDrawSegments[purple](O,F)
   \tkzDrawPoints(A,O,F)
   \tkzLabelPoints[below right=6pt](A,O,F)
\end{tikzpicture}
```



8.2.4 Method normalize ()

The result is a point located between the origin and the initial point at a distance of 1 from the origin.

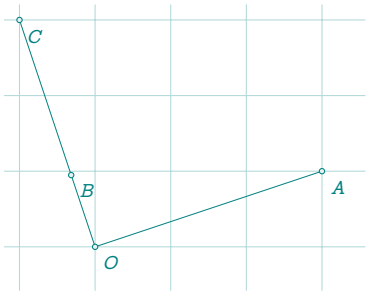
```
\begin{tkzelements}
   scale = 1.5
   z.O = point : new (0,0)
   z.A = point : new (1,2)
   z.B = z.A : normalize ()
   z.I = point : new (1,0)
\end{tkzelements}
\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawSegment(O,A)
   \tkzDrawCircle(O,B)
   \tkzDrawPoints(O,A,B,I)
   \tkzLabelPoints(O,A,B)
   \tkzLabelPoint[below right](I){$1$}
\end{tikzpicture}
```



8.2.5 Orthogonal (d) method

Let O be the origin of the plane. The "orthogonal (d)" method is used to obtain a point B from a point A such that $\overrightarrow{OB} \perp \overrightarrow{OA}$ with $OB = OA$ if d is empty, otherwise $OB = d$.

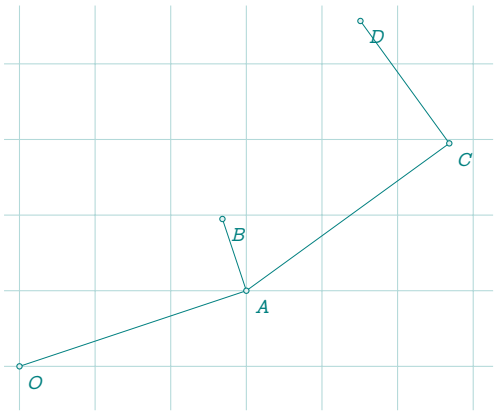
```
\begin{tkzelements}
  z.A = point : new (  3 , 1  )
  z.B = z.A : orthogonal (1)
  z.O = point : new ( 0,0  )
  z.C = z.A : orthogonal ()
\end{tkzelements}
\begin{tikzpicture}[gridded]
  \tkzGetNodes
  \tkzDrawSegments(O,A O,C)
  \tkzDrawPoints(O,A,B,C)
  \tkzLabelPoints[below right](O,A,B,C)
\end{tikzpicture}
```



8.2.6 at method

Cette méthode est complémentaire de la précédente, ainsi on peut souhaiter non pas avoir $\overrightarrow{OB} \perp \overrightarrow{OA}$ mais $\overrightarrow{AB} \perp \overrightarrow{OA}$.

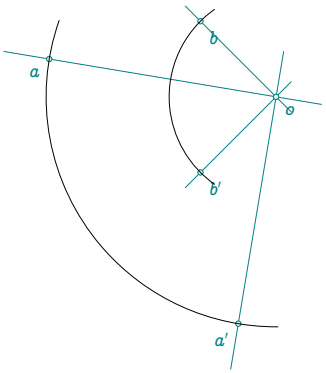
```
\begin{tkzelements}
  z.A = point : new (  3 , 1  )
  z.B = z.A : orthogonal (1)
  z.O = point : new ( 0,0  )
  -- z.B = z.B : at (z.A) -- or
  z.B = z.A : orthogonal (1) : at (z.A)
  z.C = z.A+z.B
  z.D =(z.C-z.A):orthogonal(2) : at (z.C)
\end{tkzelements}
\begin{tikzpicture}[gridded]
  \tkzGetNodes
  \tkzLabelPoints[below right](O,A,B,C,D)
  \tkzDrawSegments(O,A A,B A,C C,D)
  \tkzDrawPoints(O,A,B,C,D)
\end{tikzpicture}
```



8.2.7 Example: rotation of points

The arguments are the angle of rotation in radians, and here a list of points.

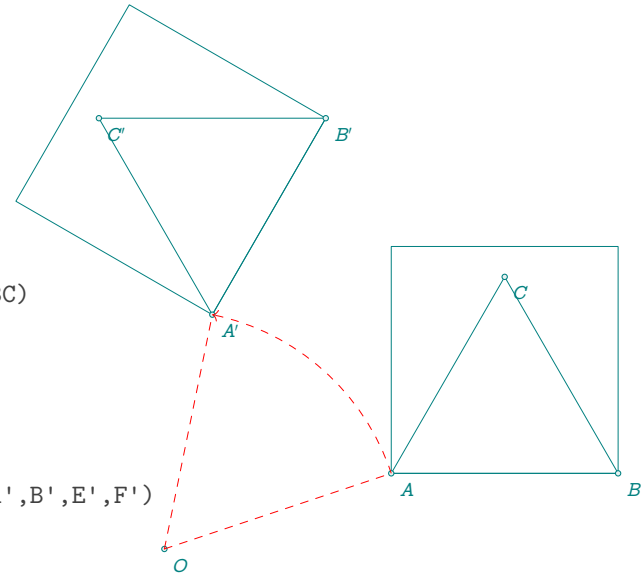
```
\begin{tkzelements}
  z.a       = point:  new(0, -1)
  z.b       = point:  new(4, 0)
  z.o       = point:  new(6, -2)
  z.ap,z.bp = z.o : rotation (math.pi/2,z.a,z.b)
\end{tkzelements}
       \begin{tikzpicture}
       \tkzGetNodes
       \tkzDrawLines(o,a o,a' o,b o,b')
       \tkzDrawPoints(a,a',b,b',o)
       \tkzLabelPoints(b,b',o)
       \tkzLabelPoints[below left](a,a')
       \tkzDrawArc(o,a)(a')
       \tkzDrawArc(o,b)(b')
       \end{tikzpicture}
```



8.2.8 Object rotation

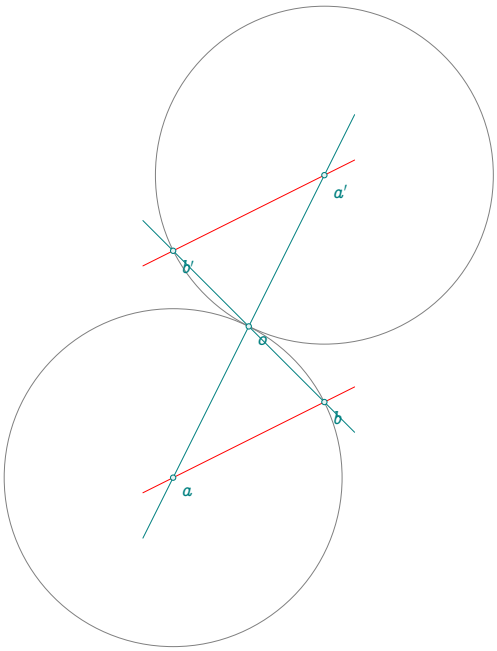
Rotate a triangle by an angle of $\pi/6$ around O.

```
\begin{tkzelements}
   z.O   = point : new ( -1 , -1 )
   z.A   = point : new ( 2 , 0 )
   z.B   = point : new ( 5 , 0 )
   L.AB  = line : new (z.A,z.B)
   T.ABC = L.AB : equilateral ()
   S.fig = L.AB : square ()
   _,_,z.E,z.F = get_points (   S.fig   )
   S.new = z.O : rotation (math.pi/3,S.fig)
   _,_,z.Ep,z.Fp = get_points (   S.new   )
   z.C = T.ABC.pc
   T.ApBpCp = z.O : rotation (math.pi/3,T.ABC)
   z.Ap,z.Bp,z.Cp = get_points ( T.ApBpCp)
\end{tkzelements}

\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawPolygons(A,B,C A',B',C' A,B,E,F A',B',E',F')
   \tkzDrawPoints (A,B,C,A',B',C',O)
    \tkzLabelPoints (A,B,C,A',B',C',O)
    \tkzDrawArc[delta=0,->](O,A)(A')
\end{tikzpicture}
```



8.2.9 Object symmetry

```
\begin{tkzelements}
    z.a = point:  new(0,-1)
    z.b = point:  new(2, 0)
    L.ab = line : new (z.a,z.b)
    C.ab = circle : new (z.a,z.b)
    z.o = point:  new(1,1)
    z.ap,z.bp =  get_points (z.o: symmetry (C.ab))
\end{tkzelements}

\begin{tikzpicture}
\tkzGetNodes
\tkzDrawCircles(a,b a',b')
\tkzDrawLines(a,a' b,b')
\tkzDrawLines[red](a,b a',b')
\tkzDrawPoints(a,a',b,b',o)
\tkzLabelPoints(a,a',b,b',o)
\end{tikzpicture}
```



9 Class line

9.1 Attributes of a line

Writing `L.AB = line: new (z.A,z.B)` creates an object of the class **`line`** (the notation is arbitrary for the moment). Geometrically it is, as much ,the line passing through the points A and B as the segment $[AB]$. Thus we can use the midpoint of `L.AB` which is, of course, the midpoint of the segment $[AB]$. This medium is obtained with `L.AB.mid`. Note that `L.AB.pa = z.A` and `L.AB.pb = z.B`. Finally, if a line L is the result of a method, you can obtain the points with `z.A,z.B = get_points (L)` or with the previous remark.

> Creation `L.AB = line : new ( z.A , z.B )`

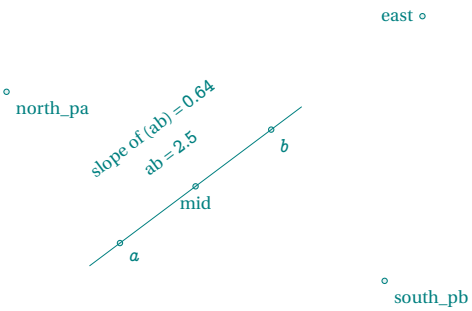
The attributes are :

Table 3: Line attributes.

Attributes	**Application**	
`pa`	First point of the segment	`z.A = L.AB.pa`
`pb`	Second point of the segment	
`type`	Type is 'line'	`L.AB.type = 'line'`
`mid`	Middle of the segment	`z.M = L.AB.mid`
`slope`	Slope of the line	see (9.1.1)
`length`	`l = L.AB.length`	see (19.5 ; 9.1.1)
`north_pa`		See (9.1.1)
`north_pb`		
`south_pa`		
`south_pb`		See (9.1.1)
`east`		
`west`		
`vec`	`V.AB = L.AB.vec`	defines $\overrightarrow{AB}$ See (18)

9.1.1 Example: attributes of class line

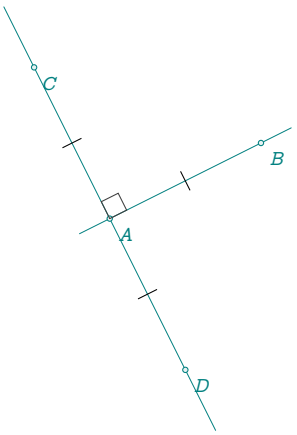
```
\begin{tkzelements}
  scale  = .5
   z.a   = point: new (1, 1)
   z.b   = point: new (5, 4)
   L.ab  = line : new (z.a,z.b)
   z.m   = L.ab.mid
   z.w   = L.ab.west
   z.e   = L.ab.east
   z.r   = L.ab.north_pa
   z.s   = L.ab.south_pb
   sl    = L.ab.slope
   len   = L.ab.length
\end{tkzelements}

\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawPoints(a,b,m,e,r,s,w)
   \tkzLabelPoints(a,b,e,r,s,w)
   \tkzLabelPoints[above](m)
   \tkzDrawLine(a,b)
   \tkzLabelSegment[sloped](a,b){ab = \tkzUseLua{len}}
   \tkzLabelSegment[above=12pt,sloped](a,b){slope of (ab) = \tkzUseLua{sl}}
\end{tikzpicture}
```



9.1.2 Method new and line attributes

Notation `L` or `L.AB` or `L.euler`. The notation is actually free. `L.AB` can also represent the segment. With `L.AB = line : new (z.A,z.B)`, a line is defined.

```
\begin{tkzelements}
   z.A   = point : new (1,1)
   z.B   = point : new (3,2)
   L.AB  = line : new (z.A,z.B)
   z.C   = L.AB.north_pa
   z.D   = L.AB.south_pa
\end{tkzelements}
\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawLines(A,B C,D)
   \tkzDrawPoints(A,...,D)
   \tkzLabelPoints(A,...,D)
   \tkzMarkRightAngle(B,A,C)
   \tkzMarkSegments(A,C A,B A,D)
\end{tikzpicture}
```



9.2 Methods of the class line

Here's the list of methods for the **line** object. The results are either reals, points, lines, circles or triangles. The triangles obtained are similar to the triangles defined below.

Table 4: Methods of the class line.(part 1)

Methods	Comments	
new(pt, pt)	L.AB = line : new(z.A,z.B) line (AB)	see (10.2.1)
Points		
gold_ratio ()	z.C = L.AB : gold_ratio()	see (22.21 ; 3.3 ; 22.8)
normalize ()	z.C = L.AB : normalize()	AC =1 and $C \in (AB)$ see (9.2.6)
normalize_inv ()	z.C = L.AB : normalize_inv()	CB =1 and $C \in (AB)$
barycenter (r,r)	z.C = L.AB : barycenter (1,2)	see (9.2.7)
point (r)	z.C = L.AB : point (2)	$\overrightarrow{AC} = 2\overrightarrow{AB}$ See (22.14 ; 9.2.5)
midpoint ()	z.M = L.AB : midpoint ()	better is z.M = L.AB.mid
harmonic_int (pt)	z.D = L.AB : harmonic_int (z.C)	See (22.8)
harmonic_ext (pt)	z.D = L.AB : harmonic_ext (z.C)	See (22.8)
harmonic_both (r)	z.C,z.D = L.AB : harmonic_both(φ)	19.2
square ()	S.AB = L.AB : square ()	create a square S.AB.[a]
Lines		
ll_from (pt)	L.CD = L.AB : ll_from (z.C)	$(CD) \parallel (AB)$
ortho_from (pt)	L.CD = L.AB : ortho_from (z.C)	$(CD) \perp (AB)$
mediator ()	L.uv = L.AB : mediator ()	(u,v) mediator of (A,B)
Triangles		
equilateral (<swap>)	T.ABC = L.AB : equilateral ()	$(\overrightarrow{AB},\overrightarrow{AC}) > 0$ or < 0 with swap [b]
isosceles (an<,swap>)	T.ABC = L.AB : isosceles (math.pi/6)	
two_angles (an,an)	T.ABC = L.AB : two_angles (an,an)	note [c] see ( )
school ()	Angle measurements are 30°,60° and 90°.	
sss (r,r)	$AC = r\, BC = r$	
as (r,an)	$AC = r\, \widehat{BAC} = an$	
sa (r,an)	$AC = r\, \widehat{ABC} = an$	
Sacred triangles		
gold (<swap>)	T.ABC = L.AB : gold ()	right in B and $AC = \varphi \times AB$
euclide (<swap>)	T.ABC = L.AB : euclide ()	$AB = AC$ and $(\overrightarrow{AB},\overrightarrow{AC}) = \pi/5$
golden (<swap>)	T.ABC = L.AB : golden ()	$(\overrightarrow{AB},\overrightarrow{AC}) = 2 \times \pi/5$
divine ()		
egyptian ()		
cheops ()		

a _,_,z.C,z.D = get_points(S.AB)
b Triangles are defined in the direct sense of rotation, unless the "swap" option is present.
c The given side is between the two angles

Table 5: Methods of the class line.(part 2)

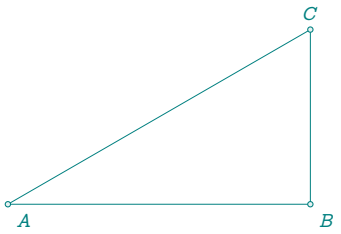
Methods	Comments	
Circles		
`circle ()`	`C.AB = L.AB : circle ()`	center pa through pb
`circle_swap ()`	`C.BA = L.AB : circle_swap ()`	center pb through pa
`apollonius (r)`	`C.apo = L.AB : apollonius (2)`	Ensemble des points tq. `MA/MB = 2`
Transformations		
`reflection ( obj )`	`new obj = L.AB : reflection (obj`	
`translation ( obj )`	`new obj = L.AB : translation (obj)`	
`projection ( obj )`	`z.H = L.AB : projection (z.C)`	$CH \perp (AB)$ and $H \in (AB)$
Miscellaneous		
`distance (pt)`	`d = L.Ab : distance (z.C)`	see 9.2.12
`in_out (pt)`	`b = L.AB: in_out(z.C)`	b=true if $C \in (AB)$
`slope ()`	`a = L.AB : slope()`	better is L.AB.slope
`in_out_segment (pt)`	`b = L.AB : in_out_segment(z.C)`	b=true if $C \in [AB]$

Here are a few examples.

9.2.1 Triangle with two_angles

The angles are on either side of the given segment

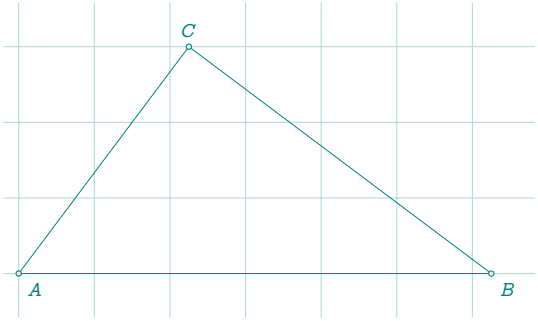
```
\begin{tkzelements}
   z.A   = point : new ( 0 , 0 )
   z.B   = point : new ( 4 , 0 )
   L.AB  = line : new ( z.A , z.B )
   T.ABC = L.AB : two_angles (math.pi/6,math.pi/2)
   z.C   = T.ABC.pc
\end{tkzelements}
\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawPolygons(A,B,C)
   \tkzDrawPoints(A,B,C)
   \tkzLabelPoints(A,B)
    \tkzLabelPoints[above](C)
\end{tikzpicture}
```



9.2.2 Triangle with three given sides

In the following example, a small difficulty arises. The given lengths are not affected by scaling, so it's necessary to use the `value (r)` function, which will modify the lengths according to the scale.

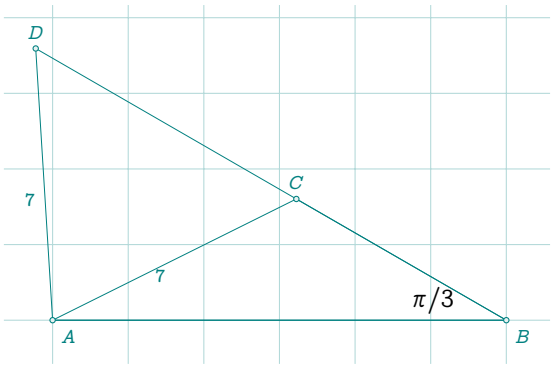
```
\begin{tkzelements}
   scale =1.25
   z.A = point : new ( 0 , 0 )
   z.B = point : new ( 5 , 0 )
   L.AB = line : new ( z.A , z.B )
   T.ABC =   L.AB : sss (value(3),value(4))
   z.C = T.ABC.pc
\end{tkzelements}
\begin{tikzpicture}[gridded]
   \tkzGetNodes
   \tkzDrawPolygons(A,B,C)
   \tkzDrawPoints(A,B,C)
   \tkzLabelPoints(A,B)
   \tkzLabelPoints[above](C)
\end{tikzpicture}
```



9.2.3 Triangle with side between side and angle

In some cases, two solutions are possible.

```
\begin{tkzelements}
   scale =1.2
   z.A = point : new ( 0 , 0 )
   z.B = point : new ( 5 , 0 )
   L.AB = line : new ( z.A , z.B )
   T.ABC,T.ABD =   L.AB : ssa (value(3),math.pi/6)
   z.C = T.ABC.pc
   z.D = T.ABD.pc
\end{tkzelements}
\begin{tikzpicture}[gridded]
   \tkzGetNodes
   \tkzDrawPolygons(A,B,C A,B,D)
   \tkzDrawPoints(A,B,C,D)
   \tkzLabelPoints(A,B)
   \tkzLabelPoints[above](C,D)
   \tkzLabelAngle(C,B,A){$\pi/3$}
   \tkzLabelSegment[below left](A,C){$7$}
   \tkzLabelSegment[below left](A,D){$7$}
\end{tikzpicture}
```



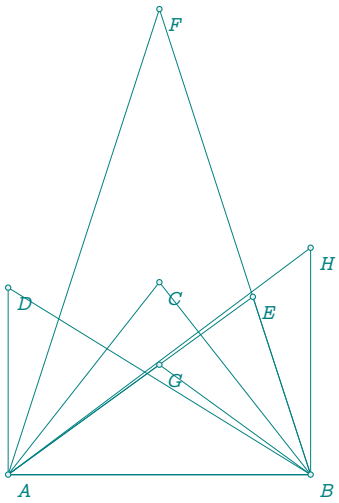
9.2.4 About sacred triangles

The side lengths are proportional to the lengths given in the table. They depend on the length of the initial segment.

Table 6: Sacred triangles.

Name	definition
gold (<swap>)	Right triangle with $a=\varphi$, $b=1$ and $c=\sqrt{\varphi}$
golden (<swap>)	Right triangle $b=\varphi$ $c=1$; half of gold rectangle
divine ()	Isosceles $a=\varphi$, $b=c=1$ and $\beta=\gamma=\pi/5$
pythagoras ()	$a=5$, $b=4$, $c=3$ and other names: isis or egyptian
sublime ()	Isosceles $a=1$, $b=c=\varphi$ and $\beta=\gamma=2\pi/5$; other name: euclid
cheops ()	Isosceles $a=2$, $b=c=\varphi$ and height = $\sqrt{\varphi}$

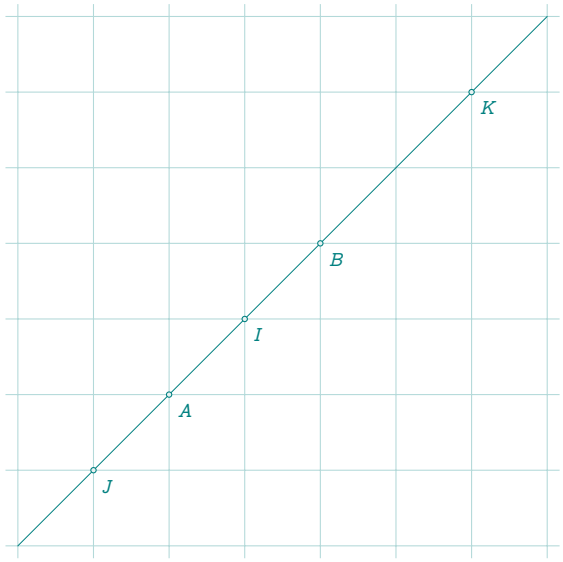
```
\begin{tkzelements}
   z.A = point : new ( 0 , 0 )
   z.B = point : new ( 4 , 0 )
   L.AB = line : new ( z.A , z.B )
   T.ABC = L.AB : cheops ()
   z.C = T.ABC.pc
   T.ABD = L.AB : gold ()
   z.D = T.ABD.pc
   T.ABE = L.AB : euclide ()
   z.E = T.ABE.pc
   T.ABF = L.AB : golden ()
   z.F = T.ABF.pc
   T.ABG = L.AB : divine ()
   z.G = T.ABG.pc
   T.ABH = L.AB : pythagoras ()
   z.H = T.ABH.pc
\end{tkzelements}
\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawPolygons(A,B,C A,B,D A,B,E A,B,F A,B,G A,B,H)
   \tkzDrawPoints(A,...,H)
   \tkzLabelPoints(A,...,H)
\end{tikzpicture}
```



9.2.5 Method point

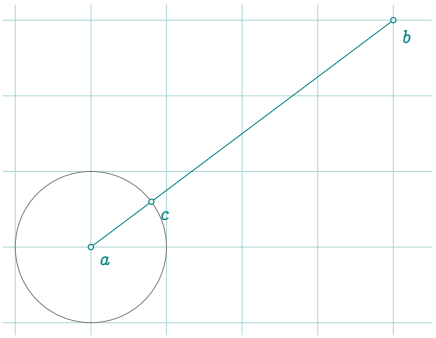
This method is very useful. It allows you to place a point on the line under consideration. If `r = 0` then the point is pa, if `r = 1` it's pb.
If `r = .5` the point obtained is the midpoint of the segment. `r` can be negative or greater than 1.
This method exists for all objects except quadrilaterals.

```
\begin{tkzelements}
   z.A = point : new (-1,-1)
   z.B = point : new (1,1)
   L.AB = line : new (z.A,z.B)
   z.I = L.AB : point (0.5)
   z.J = L.AB : point (-0.5)
   z.K = L.AB : point (2)
\end{tkzelements}
\begin{tikzpicture}[gridded]
\tkzGetNodes
   \tkzDrawLine(J,K)
   \tkzDrawPoints(A,B,I,J,K)
   \tkzLabelPoints(A,B,I,J,K)
 \end{tikzpicture}
```



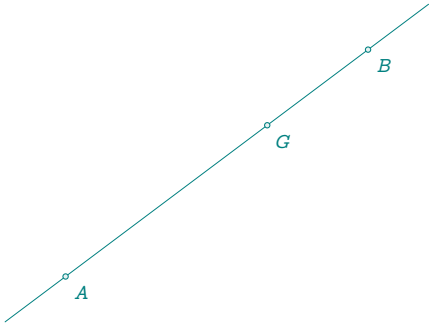
9.2.6 Normalize

```
\begin{tkzelements}
   z.a = point: new (1, 1)
   z.b = point: new (5, 4)
   L.ab = line : new (z.a,z.b)
   z.c   = L.ab : normalize ()
\end{tkzelements}

\begin{tikzpicture}[gridded]
\tkzGetNodes
\tkzDrawSegments(a,b)
\tkzDrawCircle(a,c)
\tkzDrawPoints(a,b,c)
\tkzLabelPoints(a,b,c)
\end{tikzpicture}
```



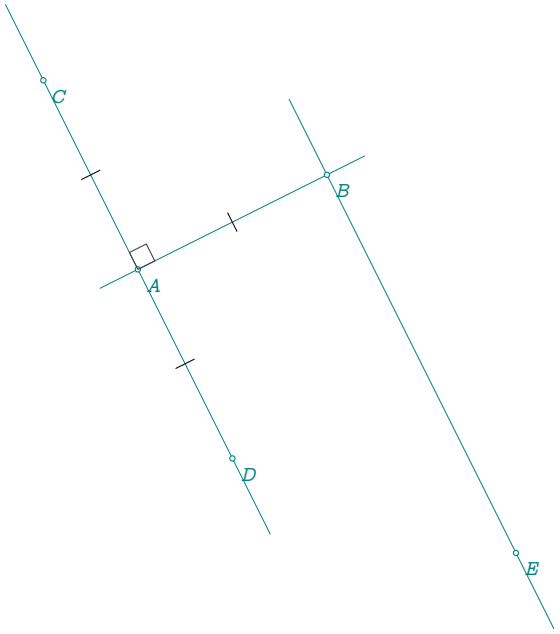
9.2.7 Barycenter with a line

```
\begin{tkzelements}
   z.A = point : new ( 0 , -1 )
   z.B = point : new ( 4 , 2 )
   L.AB = line : new ( z.A , z.B )
   z.G = L.AB : barycenter (1,2)
\end{tkzelements}
\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawLine(A,B)
   \tkzDrawPoints(A,B,G)
   \tkzLabelPoints(A,B,G)
\end{tikzpicture}
```



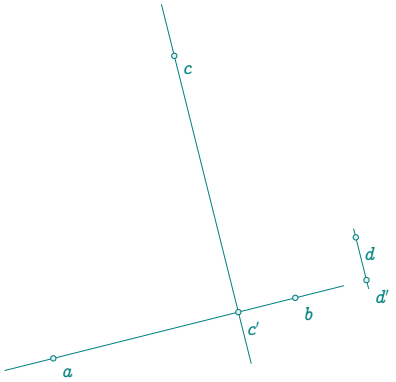
9.2.8 Example: new line from a defined line

```
\begin{tkzelements}
   scale = 1.25
   z.A   = point : new (1,1)
   z.B   = point : new (3,2)
   L.AB  = line : new (z.A,z.B)
   z.C   = L.AB.north_pa
   z.D   = L.AB.south_pa
   L.CD  = line : new (z.C,z.D)
   _,z.E   = get_points ( L.CD: ll_from (z.B))
   -- z.E   = L2.pb
\end{tkzelements}
\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawLines(A,B C,D B,E)
   \tkzDrawPoints(A,...,E)
   \tkzLabelPoints(A,...,E)
   \tkzMarkRightAngle(B,A,C)
   \tkzMarkSegments(A,C A,B A,D)
\end{tikzpicture}
```



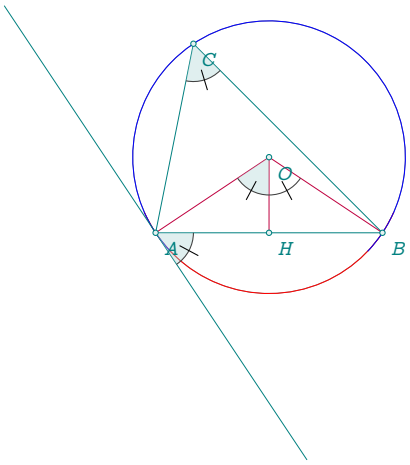
9.2.9 Example: projection of several points

```
\begin{tkzelements}
   scale     = .8
   z.a       = point:  new (0, 0)
   z.b       = point:  new (4, 1)
   z.c       = point:  new (2, 5)
   z.d       = point:  new (5, 2)
   L.ab      = line:   new (z.a,z.b)
   z.cp,z.dp = L.ab:  projection(z.c,z.d)
\end{tkzelements}
 \begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawLines(a,b c,c' d,d')
   \tkzDrawPoints(a,...,d,c',d')
   \tkzLabelPoints(a,...,d,c',d')
 \end{tikzpicture}
```



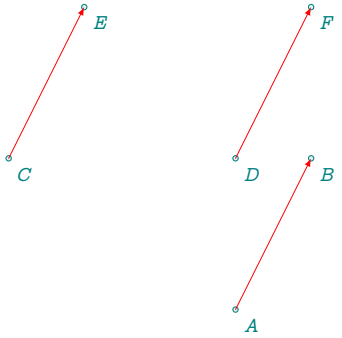
9.2.10 Example: combination of methods

```
\begin{tkzelements}
   z.A     = point: new (0 , 0)
   z.B     = point: new (6 , 0)
   z.C     = point: new (1 , 5)
   T.ABC   = triangle: new (z.A,z.B,z.C)
   L.AB    = T.ABC.ab
   z.O     = T.ABC.circumcenter
   C.OA    = circle: new (z.O,z.A)
   z.H     = L.AB: projection (z.O)
   L.ab    = C.OA: tangent_at (z.A)
   z.a,z.b = L.ab.pa,L.ab.pb
  -- or z.a,z.b  = get_points (L.ab)
\end{tkzelements}
\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawPolygon(A,B,C)
   \tkzDrawCircle(O,A)
   \tkzDrawSegments[purple](O,A O,B O,H)
   \tkzDrawArc[red](O,A)(B)
   \tkzDrawArc[blue](O,B)(A)
   \tkzDrawLine[add = 2 and 1](A,a)
   \tkzFillAngles[teal!30,opacity=.4](A,C,B b,A,B A,O,H)
   \tkzMarkAngles[mark=|](A,C,B b,A,B A,O,H H,O,B)
   \tkzDrawPoints(A,B,C,H,O)
   \tkzLabelPoints(A,B,C,H,O)
\end{tikzpicture}
```



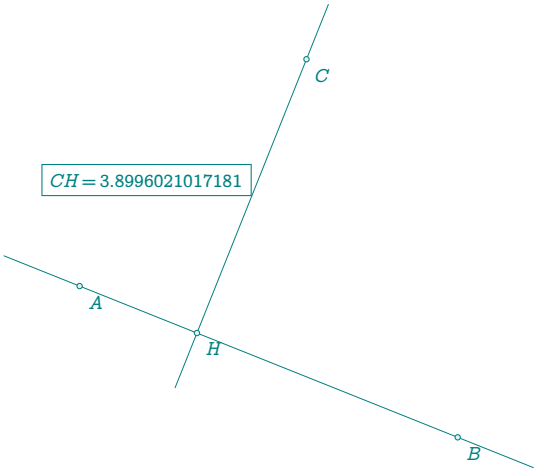
9.2.11 Example: translation

```
\begin{tkzelements}
   z.A  = point:  new (0,0)
   z.B  = point:  new (1,2)
   z.C  = point:  new (-3,2)
   z.D  = point:  new (0,2)
   L.AB = line : new (z.A,z.B)
   z.E,z.F = L.AB : translation (z.C,z.D)
\end{tkzelements}
\begin{tikzpicture}
\tkzGetNodes
\tkzDrawPoints(A,...,F)
\tkzLabelPoints(A,...,F)
\tkzDrawSegments[->,red,> =latex](C,E D,F A,B)
\end{tikzpicture}
```



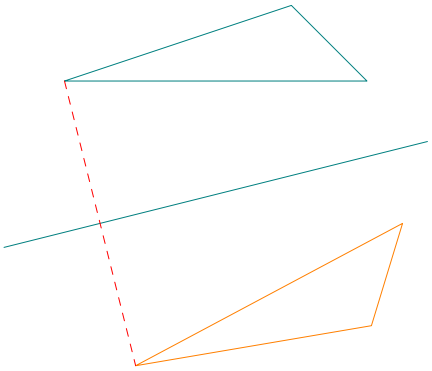
9.2.12 Example: distance and projection

```
\begin{tkzelements}
   z.A    = point : new (0 , 0)
   z.B    = point : new (5 , -2)
   z.C    = point : new (3 , 3)
   L.AB   = line : new (z.A,z.B)
   d      = L.AB : distance (z.C)
   z.H    = L.AB : projection (z.C)
\end{tkzelements}
\begin{tikzpicture}
  \tkzGetNodes
  \tkzDrawLines(A,B C,H)
  \tkzDrawPoints(A,B,C,H)
  \tkzLabelPoints(A,B,C,H)
  \tkzLabelSegment[above left,
  draw](C,H){$CH = \tkzUseLua{d}$}
\end{tikzpicture}
```



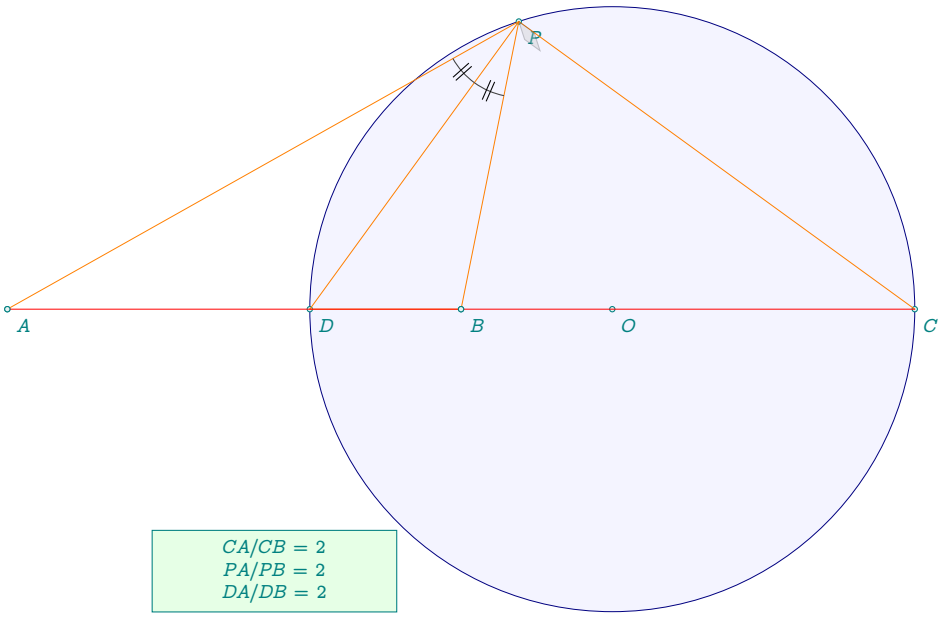
9.2.13 Reflection of object

```
\begin{tkzelements}
   z.A = point : new ( 0 , 0 )
   z.B = point : new ( 4 , 1 )
   z.E = point : new ( 0 , 2 )
   z.F = point : new ( 3 , 3 )
   z.G = point : new ( 4 , 2 )
   L.AB = line : new ( z.A , z.B )
   T.EFG = triangle : new (z.E,z.F,z.G)
   T.new = L.AB : reflection (T.EFG)
   z.Ep,z.Fp,z.Gp = get_points(T.new)
\end{tkzelements}
\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawLine(A,B)
   \tkzDrawPolygon(E,F,G)
   \tkzDrawPolygon[new](E',F',G')
   \tkzDrawSegment[red,dashed](E,E')
\end{tikzpicture}
```



9.3 Apollonius circle MA/MB = k

```
\begin{tkzelements}
   z.A = point : new ( 0 , 0 )
   z.B = point : new ( 6 , 0 )
   L.AB =line: new (z.A,z.B)
   C.apo = L.AB : apollonius (2)
   z.O,z.C = get_points (   C.apo   )
   z.D = C.apo : antipode (z.C)
   z.P = C.apo : point  (0.30)
\end{tkzelements}
\begin{tikzpicture}
   \tkzGetNodes
   \tkzFillCircle[blue!20,opacity=.2](O,C)
   \tkzDrawCircle[blue!50!black](O,C)
   \tkzDrawPoints(A,B,O,C,D,P)
   \tkzLabelPoints[below right](A,B,O,C,D,P)
   \tkzDrawSegments[orange](P,A P,B P,D B,D P,C)
   \tkzDrawSegments[red](A,C)
   \tkzDrawPoints(A,B)
   \tkzLabelCircle[draw,fill=green!10,%
      text width=3cm,text centered,left=24pt](O,D)(60)%
     {$CA/CB=2$\\$PA/PB=2$\\$DA/DB=2$}
   \tkzMarkRightAngle[opacity=.3,fill=lightgray](O,P,C)
   \tkzMarkAngles[mark=||](A,P,D D,P,B)
\end{tikzpicture}
```



Remark: \tkzUseLua{length(z.P,z.A)/length(z.P,z.B)} = 2.0

10 Class circle

10.1 Attributes of a circle

This class is also defined by two points: on the one hand, the center and on the other hand, a point through which the circle passes.

> Creation `C.OA = circle: new (z.O,z.A)`

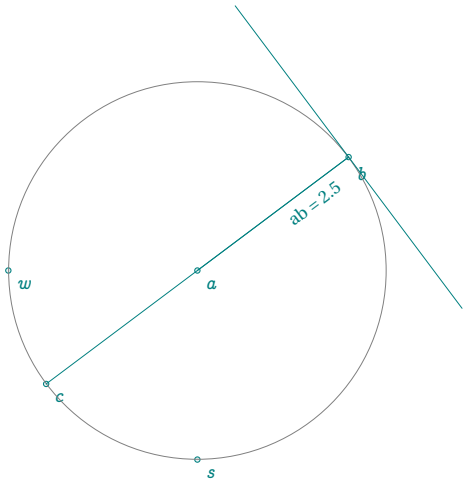
Table 7: Circle attributes.

Attributes	Application	
center	z.A = C.AB.center	
through	z.B = C.AB.through	
type	C.AB.type	C.OA.type = 'circle'
radius	C.AB.radius	r = C.OA.radius *r* real number
north	C.AB.north	z.N = C.OA.north
south	C.AB.south	z.S = C.OA.south
east	C.AB.east	z.E = C.OA.east
west	C.AB.west	z.W = C.OA.west
opp	z.Ap = C.AB.opp	see (10.1.1)
ct	L = C.AB.ct	see (10.1.1)

10.1.1 Example: circle attributes

Three attributes are used (south, west, radius).

```
\begin{tkzelements}
   scale = .5
   z.a   = point: new (1, 1)
   z.b   = point: new (5, 4)
   C.ab  = circle : new (z.a,z.b)
   z.s   = C.ab.south
   z.w   = C.ab.west
   r     = C.ab.radius
   z.c   = C.ab.opp
   z.r,z.t = get_points (C.ab.ct : ortho_from (z.b))
\end{tkzelements}
\begin{tikzpicture}
\tkzGetNodes
\tkzDrawPoints(a,b,c,s,w)
\tkzLabelPoints(a,b,c,s,w)
\tkzDrawCircle(a,b)
\tkzDrawSegments(a,b r,t b,c)
\tkzLabelSegment[sloped](a,b){ab = \tkzUseLua{r}}
\end{tikzpicture}
```



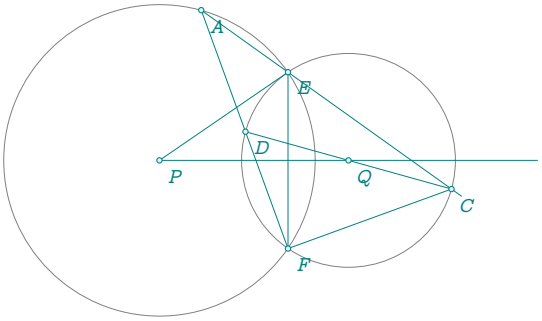
10.2 Methods of the class circle

Table 8: Circle methods.

Methods	Comments	
new(O,A)	C.OA = circle : new (z.O,z.A)	circle center O through A
radius(O,r)	C.OA = circle : radius (z.O,2)	circle center O radius =2 cm
diameter(A,B)	C.OA = circle :diameter(z.A,z.B)	circle diameter $[AB]$
Points		
antipode (pt)	z.C = C.OA: antipode (z.B)	$[BC]$ is a diameter
inversion (pt)	z.Bp = C.AC: inversion (z.B)	
midarc (pt,pt)	z.D = C.AB: midarc (z.B,z.C)	D is the midarc of $\overset{\frown}{BC}$
point (r)	z.E = C.AB: point (0.25)	r between 0 and 1
random_pt(lower, upper)		
internal_similitude (C)	z.I = C.one : internal_similitude (C.two)	
external_similitude (C)	z.J = C.one : external_similitude (C.two)	
radical_center (C1<,C2>)	or only (C1)	see 22.30
Lines		
radical_axis (C)	see (22.2 ; 22.26 ; 22.27 ; 22.28 ; 22.29)	
tangent_at (pt)	z.P = C.OA: tangent_at (z.M)	see (10.2.2 ; 9.2.10)
tangent_from (pt)	z.M,z.N = C.OA: tangent_from (z.P)	see ((iii))
inversion (line)	L or C = C.AC: inversion (L.EF)	see (10.2.5)
common_tangent (C)	z.a,z.b = C.AC: common_tangent (C.EF)	see (10.4 ; 10.5)
Circles		
orthogonal_from (pt)	C = C.OA: orthogonal_from (z.P)	see (10.2.1 ; 10.5 ; 22.36 ; 22.40)
orthogonal_through (pta,ptb)	C = C.OA: orthogonal_through (z.z1,z.z2)	see (22.10)
inversion (...)	C.AC: inversion (pt, pts, L or C)	see 10.2.3, 10.2.4, 10.2.5, 10.2.6
midcircle (C)	C.inv = C.OA: midcircle (C.EF)	see 10.2.7
radical_circle (C1<,C2>)	or only (C1)	see 22.31
Miscellaneous		
power (pt)	p = C.OA: power (z.M)	see (22.42 ; 22.43 ; 22.34)
in_out (pt)	C.OA : in_out (z.M)	see (10.6)
in_out_disk (pt)	C.OA : in_out_disk (z.M)	see (10.6)
draw ()	for further use	
circles_position (C1)	result = string	see (10.3)

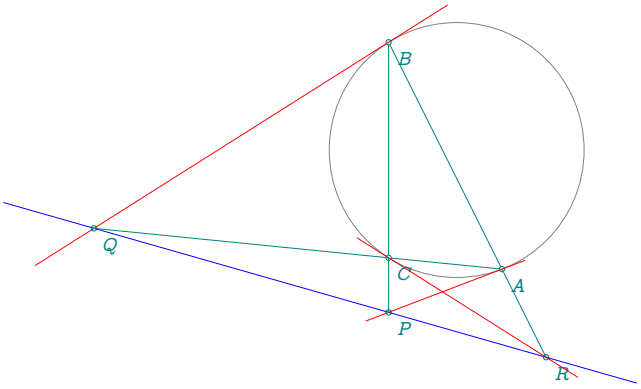
10.2.1 Altshiller

```
\begin{tkzelements}
   z.P   = point : new (0,0)
   z.Q   = point : new (5,0)
   z.I   = point : new (3,2)
   C.QI  = circle :    new (z.Q,z.I)
   C.PE  = C.QI : orthogonal_from (z.P)
   z.E   = C.PE.through
   C.QE  = circle :    new (z.Q,z.E)
   _,z.F = intersection (C.PE,C.QE)
   z.A   = C.PE: point (1/9)
   L.AE  = line : new (z.A,z.E)
   _,z.C = intersection (L.AE,C.QE)
   L.AF  = line : new (z.A,z.F)
   L.CQ  = line : new (z.C,z.Q)
   z.D   = intersection (L.AF,L.CQ)
\end{tkzelements}
\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawCircles(P,E Q,E)
   \tkzDrawLines[add=0 and 1](P,Q)
   \tkzDrawLines[add=0 and 2](A,E)
   \tkzDrawSegments(P,E E,F F,C A,F C,D)
   \tkzDrawPoints(P,Q,E,F,A,C,D)
   \tkzLabelPoints(P,Q,E,F,A,C,D)
\end{tikzpicture}
```



10.2.2 Lemoine

```
\begin{tkzelements}
   scale = 1.6
   z.A   = point: new (1,0)
   z.B   = point: new (5,2)
   z.C   = point: new (1.2,2)
   T     = triangle: new(z.A,z.B,z.C)
   z.O   = T.circumcenter
   C.OA  = circle: new (z.O,z.A)
   L.tA  = C.OA: tangent_at (z.A)
   L.tB  = C.OA: tangent_at (z.B)
   L.tC  = C.OA: tangent_at (z.C)
   z.P   = intersection (L.tA,T.bc)
   z.Q   = intersection (L.tB,T.ca)
   z.R   = intersection (L.tC,T.ab)
\end{tkzelements}
\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawPolygon[teal](A,B,C)
   \tkzDrawCircle(O,A)
   \tkzDrawPoints(A,B,C,P,Q,R)
   \tkzLabelPoints(A,B,C,P,Q,R)
   \tkzDrawLine[blue](Q,R)
   \tkzDrawLines[red](A,P B,Q R,C)
   \tkzDrawSegments(A,R C,P C,Q)
\end{tikzpicture}
```



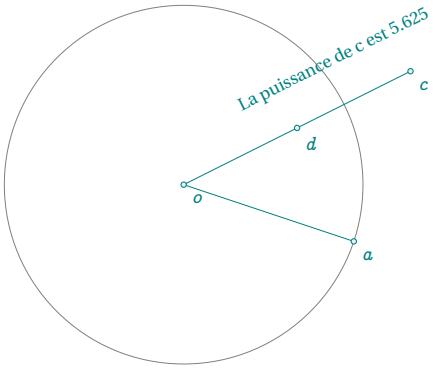
10.2.3 Inversion: point, line and circle

The "inversion" method can be used on a point, a line or a circle. Depending on the type of object, the function determines the correct algorithm to use.

10.2.4 Inversion: point

The "inversion" method can be used on a point, a group of points, a line or a circle. Depending on the type of object, the function determines the correct algorithm to use.

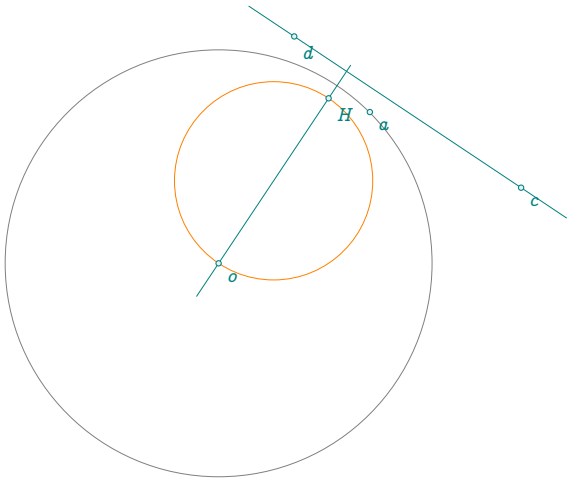
```
\begin{tkzelements}
   z.o   = point:    new (-1,2)
   z.a   = point:    new (2,1)
   C.oa  = circle:   new (z.o,z.a)
   z.c   = point:    new (3,4)
   z.d   = C.oa:     inversion (z.c)
   p     = C.oa:     power (z.c)
\end{tkzelements}
\begin{tikzpicture}
    \tkzGetNodes
    \tkzDrawCircle(o,a)
    \tkzDrawSegments(o,a o,c)
    \tkzDrawPoints(a,o,c,d)
    \tkzLabelPoints(a,o,c,d)
    \tkzLabelSegment[sloped,above=1em](c,d){%
    Power of c with respect to C is \tkzUseLua{p}}
 \end{tikzpicture}
```



10.2.5 Inversion: line

The result is either a straight line or a circle.

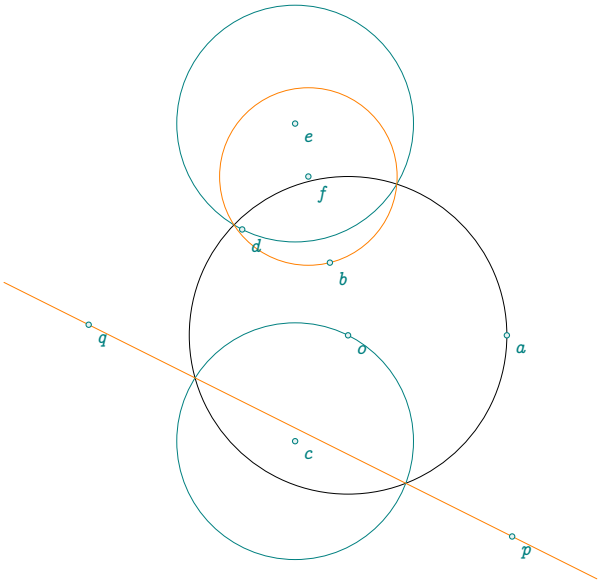
```
\begin{tkzelements}
   z.o      = point:    new (-1,1)
   z.a      = point:    new (1,3)
   C.oa     = circle:   new (z.o,z.a)
   z.c      = point:    new (3,2)
   z.d      = point:    new (0,4)
   L.cd     = line:    new (z.c,z.d)
   C.OH     = C.oa: inversion (L.cd)
   z.O,z.H  = get_points(C.OH)
\end{tkzelements}
\begin{tikzpicture}
    \tkzGetNodes
    \tkzDrawCircles(o,a O,H)
    \tkzDrawLines(c,d o,H)
    \tkzDrawPoints(a,o,c,d,H)
    \tkzLabelPoints(a,o,c,d,H)
 \end{tikzpicture}
```



10.2.6 Inversion: circle

The result is either a straight line or a circle.

```
\begin{tkzelements}
scale = .7
z.o,z.a  = point:  new (-1,3),point:  new (2,3)
z.c      = point:  new (-2,1)
z.e,z.d  = point:  new (-2,7),point: new (-3,5)
C.oa     = circle: new (z.o,z.a)
C.ed     = circle: new (z.e,z.d)
C.co     = circle: new (z.c,z.o)
obj      = C.oa: inversion (C.co)
   if obj.type == "line"
   then z.p,z.q = get_points(obj)
   else z.f,z.b = get_points(obj) end
obj      = C.oa: inversion(C.ed)
if obj.type == "line"
then z.p,z.q = get_points(obj)
else z.f,z.b = get_points(obj) end
color = "orange"
\end{tkzelements}
\begin{tikzpicture}
\tkzGetNodes
\tkzDrawCircles[black](o,a)
\tkzDrawCircles[teal](c,o e,d)
\tkzDrawCircles[\tkzUseLua{color}](f,b)
\tkzDrawLines[\tkzUseLua{color}](p,q)
\tkzDrawPoints(a,...,f,o,p,q)
\tkzLabelPoints(a,...,f,o,p,q)
\end{tikzpicture}
```



10.2.7 midcircle

From Eric Danneels and Floor van Lamoen: A midcircle of two given circles is a circle that swaps the two given circles by inversion. Midcircles are in the same pencil of circles as the given circles. The center of the midcircle(s) is one or both of the centers of similitude. We can distinguish four cases:

(i) The two given circles intersect: there are two midcircles with centers at the centers of similitude of the given circles;

(ii) One given circle is in the interior of the other given circle. Then there is one midcircle with center of similitude at the internal center of similitude of the given circles;

(iii) One given circle is in the exterior of the other given circle. Then there is one midcircle with center at the external center of similitude of the given circles. Clearly the tangency cases can be seen as limit cases of the above;

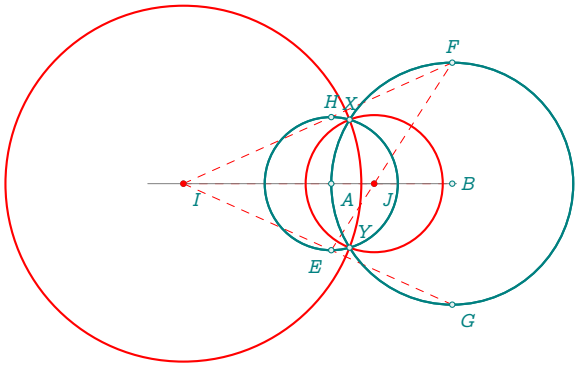
(iv) If the circles intersect in a single point, the unique midcircle has center at the external similitude center or at internal similitude center.

Let's look at each of these cases:

(i) If the two given circles intersect, then there are two circles of inversion through their common points, with centers at the centers of similitudes. The two midcircles bisect their angles and are orthogonal to each other. The centers of the midcircles are the internal center of similitude and the external center of similitude I and J.

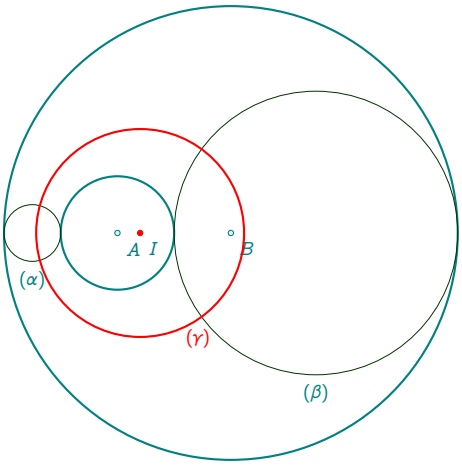
Consider two intersecting circles $(\mathcal{A})$ and $(\mathcal{B})$. We can obtain the centers of similarity of these two circles by constructing EH and FG two diameters parallel of the circles $(\mathcal{A})$ and $(\mathcal{B})$. The line (GE) intercepts the line (AB) in J and the line (EF) intercepts the line (AB) in I. The circles $(\mathcal{I})$ and $(\mathcal{J})$ are orthogonal and are the midcircles of $(\mathcal{A})$ and $(\mathcal{B})$. The division $(A,B;I,J)$ is harmonic.

```
\begin{tkzelements}
scale = .8
z.A = point : new ( 1 , 0 )
z.B = point : new ( 3 , 0 )
z.O = point : new ( 2.1, 0 )
z.P = point : new ( 1 ,0 )
C.AO = circle : new (z.A,z.O)
C.BP = circle : new (z.B,z.P)
z.E = C.AO.south
z.H = C.AO.north
z.F = C.BP.north
z.G = C.BP.south
C.IT,C.JV = C.AO : midcircle (C.BP)
z.I,z.T = get_points (   C.IT    )
z.J,z.V = get_points (   C.JV    )
z.X,z.Y = intersection (C.AO,C.BP)
\end{tkzelements}
```



(ii) One given circle is in the interior of the other given circle.

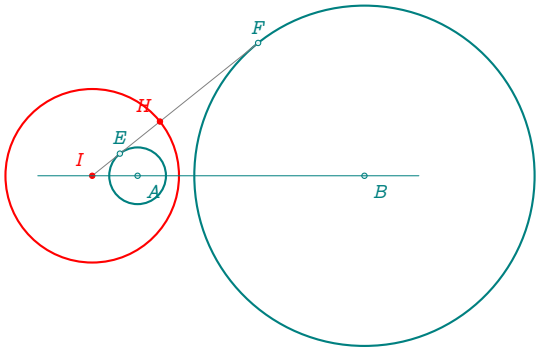
```
\begin{tkzelements}
   scale =.75
z.A = point : new ( 3 , 0 )
z.B = point : new ( 5 , 0 )
z.O = point : new ( 2 , 0 )
z.P = point : new ( 1 , 0 )
L.AB = line :  new (z.A,z.B)
C.AO = circle : new (z.A,z.O)
C.BP = circle : new (z.B,z.P)
z.R,z.S = intersection (L.AB,C.BP)
z.U,z.V = intersection (L.AB,C.AO)
C.SV = circle  :  diameter (z.S,z.V)
C.UR = circle  :  diameter (z.U,z.R)
z.x = C.SV.center
z.y = C.UR.center
C.IT = C.AO : midcircle (C.BP)
z.I,z.T = get_points (   C.IT    )
\end{tkzelements}
```



This case is a little more complicated. We'll construct the two circles (α) and (β) tangent to the two given circles. Then we construct the radical circle orthogonal to the circles (α) and (β). Its center is the radical center as well as the center of internal similtude of circles of center A and B.

(iii) When the two given circles are external to each other, we construct the external center of similitude of the two given circles. I is the center of external similarity of the two given circles. To obtain the inversion circle, simply note that H is such that $IH^2 = IE \times IF$.

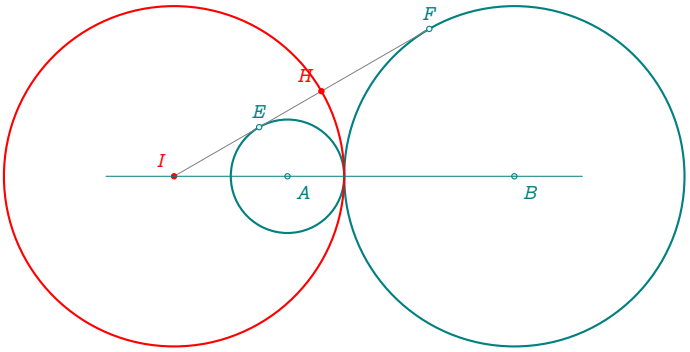
```
\begin{tkzelements}
scale=.75
local a,b,c,d
z.A = point : new ( 0 , 0 )
z.B = point : new ( 4 , 0 )
z.a = point :  new ( .5 , 0)
z.b = point :  new ( 1 , 0)
C.Aa = circle  :  new (z.A,z.a)
C.Bb = circle  :  new (z.B,z.b)
L.AB = line :  new (z.A,z.B)
z.E = C.Aa.north
z.F = C.Bb.north
L.EF = line :  new (z.E,z.F)
C.IT =  C.Aa : midcircle (C.Bb)
z.I,z.T = get_points (   C.IT   )
L.TF = C.Bb : tangent_from (z.I)
z.H = intersection (L.TF,C.IT)
z.E = intersection (L.TF,C.Aa)
z.F=L.TF.pb
\end{tkzelements}
```



(iv) Consider two tangent circles $(\mathcal{A})$ and $(\mathcal{B})$,

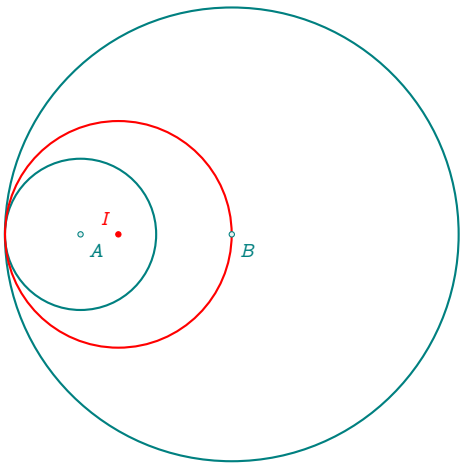
– $(\mathcal{B})$ being external and angent to $(\mathcal{A})$. The construction is identical to the previous one.

```
\begin{tkzelements}
scale=.75
local a,b,c,d
z.A = point : new ( 0 , 0 )
z.B = point : new ( 4 , 0 )
z.a = point :  new ( 1 , 0)
z.b = point :  new ( 1 , 0)
C.Aa = circle  :  new (z.A,z.a)
C.Bb = circle  :  new (z.B,z.b)
L.AB = line :  new (z.A,z.B)
z.E = C.Aa.north
z.F = C.Bb.north
L.EF = line :  new (z.E,z.F)
C.IT =  C.Aa : midcircle (C.Bb)
z.I,z.T = get_points ( C.IT )
L.TF = C.Bb : tangent_from (z.I)
z.H = intersection (L.TF,C.IT)
z.E = intersection (L.TF,C.Aa)
z.F=L.TF.pb
\end{tkzelements}
```



– When one of the given circles is inside and tangent to the other, the construction is easy.

```
\begin{tkzelements}
z.A      = point : new ( 2 , 0 )
z.B      = point : new ( 4 , 0 )
z.a      = point :  new ( 1 , 0)
z.b      = point :  new ( 1 , 0)
C.Aa     = circle  :  new (z.A,z.a)
C.Bb     = circle  :  new (z.B,z.b)
C.IT     =  C.Aa : midcircle (C.Bb)
z.I,z.T  = get_points ( C.IT )
\end{tkzelements}
```

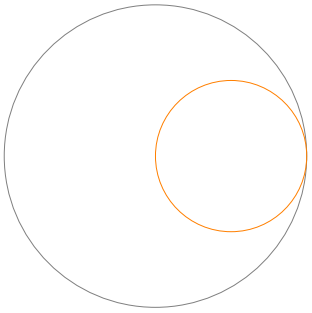


10.3 Circles_position

Cette fonction retourne une chaîne qui indique la position du cercle par rapport à un autre. Utile pour créer une fonction. Les cas sont:

- "outside"
- "outside tangent"
- "inside tangent"
- "inside"
- "intersect"

```
\begin{tkzelements}
   z.A      = point : new ( 0  , 0  )
   z.a      = point : new ( 3  , 0  )
   z.B      = point : new ( 2  , 0  )
   z.b      = point : new ( 3  , 0  )
   C.Aa     = circle: new (z.A,z.a)
   C.Bb     = circle: new (z.B,z.b)
   position = C.Aa : circles_position (C.Bb)
   if position == "inside tangent"
   then color = "orange"
   else color = "blue" end
\end{tkzelements}

\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawCircle(A,a)
   \tkzDrawCircle[color=\tkzUseLua{color}](B,b)
\end{tikzpicture}
```

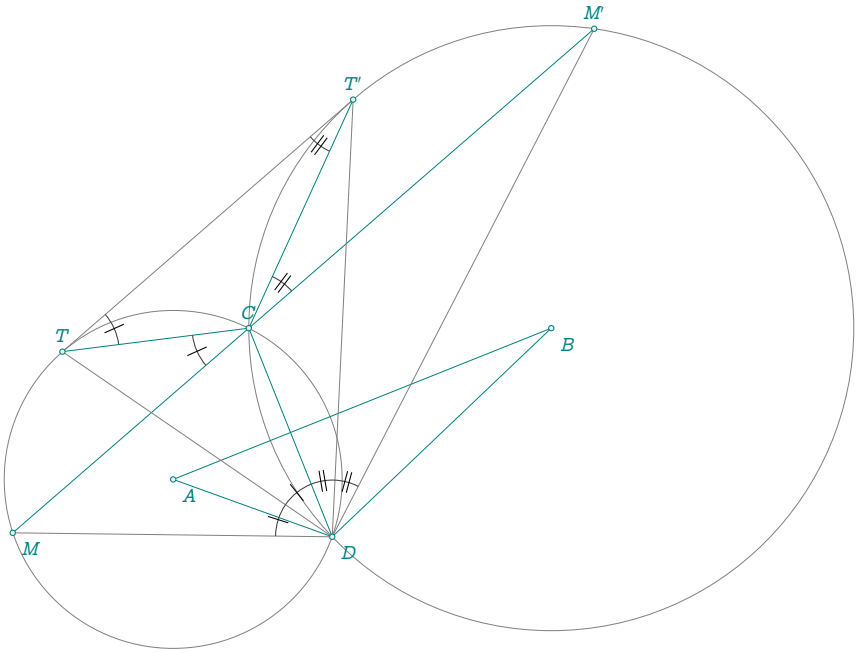


10.4 Common tangent: Angle of two intersecting circles

Let be a tangent common to both circles at T and T' (closest to C). Let a secant parallel to this tangent pass through C. Then the segment $[TT']$ is seen from the other common point D at an angle equal to half the angle of the two circles.

```
\begin{tkzelements}
   z.A   = point : new ( 0  , 0  )
```

```
   z.B   = point : new ( 5  , 2  )
   L.AB = line : new ( z.A , z.B )
   z.C   = point : new ( 1 , 2 )
   C.AC  = circle : new (z.A,z.C)
   C.BC  = circle : new (z.B,z.C)
   z.T,z.Tp = C.AC : common_tangent (C.BC)
   L.TTp = line : new (z.T,z.Tp)
   z.M   = C.AC : point (0.45)
   L.MC  =line : new (z.M,z.C)
   z.Mp  = intersection (L.MC, C.BC)
   L.mm = L.TTp : ll_from (z.C)
   _,z.M = intersection (L.mm, C.AC)
   z.Mp = intersection (L.mm, C.BC)
   _,z.D = intersection (C.AC,C.BC)
\end{tkzelements}
\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawCircles(A,C B,C)
   \tkzDrawSegments(M,M' A,D B,D A,B C,D T,C T',C)
   \tkzDrawSegments[gray](D,M D,M' T,T' D,T D,T')
   \tkzDrawPoints(A,B,C,D,M,M',T,T')
   \tkzLabelPoints(A,B,D,M)
   \tkzLabelPoints[above](C,M',T,T')
   \tkzMarkAngles[mark=|,size=.75](T,C,M C,T,T' C,D,T T,D,M)
   \tkzMarkAngles[mark=||,size=.75](M',C,T' T,T',C T',D,C M',D,T')
\end{tikzpicture}
```

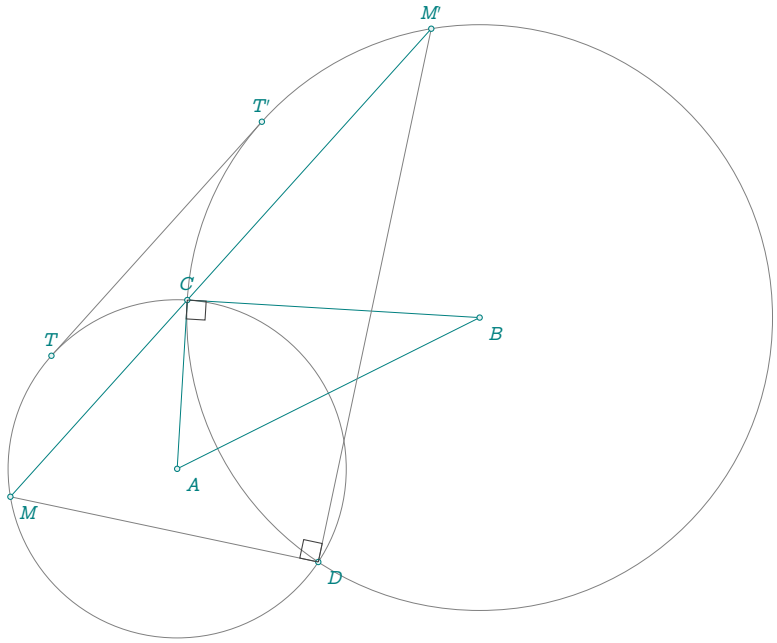


10.5 Common tangent: orthogonality

For two circles to be orthogonal, it is necessary and sufficient for a secant passing through one of their common points to be seen from the other common point at a right angle.

```
\begin{tkzelements}
```

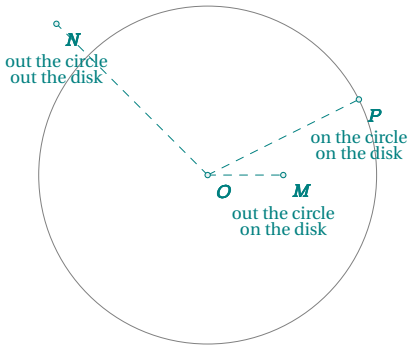
```
   z.A   = point : new ( 0  , 0  )
   z.B   = point : new ( 4  , 2  )
   L.AB = line : new ( z.A , z.B )
   z.a   = point : new ( 1 , 2 )
   C.Aa  = circle : new (z.A,z.a)
   C.BC = C.Aa : orthogonal_from (z.B)
   z.C,z.D = intersection (C.Aa,C.BC)
   C.AC  = circle : new (z.A,z.C)
   z.T,z.Tp = C.AC : common_tangent (C.BC)
   L.TTp = line : new (z.T,z.Tp)
   z.M   = C.AC : point (0.45)
   L.MC  =line : new (z.M,z.C)
   z.Mp  = intersection (L.MC, C.BC)
   L.mm = L.TTp : ll_from (z.C)
   _,z.M = intersection (L.mm, C.AC)
   z.Mp = intersection (L.mm, C.BC)
\end{tkzelements}

\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawCircles(A,C B,C)
   \tkzDrawSegments(M,M' A,C B,C C A,B)
   \tkzDrawSegments[gray](D,M D,M' T,T')
   \tkzDrawPoints(A,B,C,D,M,M',T,T')
   \tkzLabelPoints(A,B,D,M)
   \tkzLabelPoints[above](C,M',T,T')
   \tkzMarkRightAngles(M',D,M A,C,B)
\end{tikzpicture}
```



10.6 In_out for circle and disk

```
\begin{tkzelements}
z.O = point : new (0,0)
```

```
z.A = point : new (1,2)
C.OA = circle : new (z.O,z.A)
z.N = point : new (-2,2)
z.M = point : new (1,0)
z.P = point : new (2,1)
BCm = C.OA : in_out (z.M)
BDm = C.OA : in_out_disk (z.M)
BCn = C.OA : in_out (z.N)
BDn = C.OA : in_out_disk (z.N)
BCp = C.OA : in_out (z.P)
BDp = C.OA : in_out_disk (z.P)
\end{tkzelements}
\def\tkzPosPoint#1#2#3#4{%
\tkzLabelPoints(O,M,N,P)
   \ifthenelse{\equal{\tkzUseLua{#1}}{true}}{
   \tkzLabelPoint[below=#4pt,font=\scriptsize](#2){on  the #3}}{%
   \tkzLabelPoint[below=#4pt,font=\scriptsize](#2){out  the #3}}}
\begin{tikzpicture}
\tkzGetNodes
\tkzDrawSegments[dashed](O,M O,N O,P)
\tkzDrawCircle(O,A)
\tkzDrawPoints(O,M,N,P)
\tkzPosPoint{BCm}{M}{circle}{8}
\tkzPosPoint{BCn}{N}{circle}{8}
\tkzPosPoint{BCp}{P}{circle}{8}
\tkzPosPoint{BDm}{M}{disk}{14}
\tkzPosPoint{BDn}{N}{disk}{14}
\tkzPosPoint{BDp}{P}{disk}{14}
\end{tikzpicture}
```



11 Classe triangle

11.1 Attributes of a triangle

The triangle object is created using the new method, for example with

Creation `T.ABC = triangle : new ( z.A , z.B , z.C )`

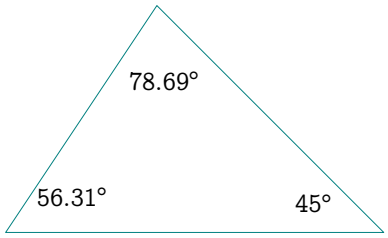
(See examples: 22.3; 22.4; 22.9). Multiple attributes are then created.

Table 9: Triangle attributes.

Attributes	Application
pa	T.ABC.pa
pb	T.ABC.pb
pc	T.ABC.pc
type	'triangle'
circumcenter	T.ABC.circumcenter
centroid	T.ABC.centroid
incenter	T.ABC.incenter
orthocenter	T.ABC.orthocenter
eulercenter	T.ABC.eulercenter
spiekercenter	T.ABC.spiekercenter
a	It's the length of the side opposite the first vertex
b	It's the length of the side opposite the second verte
c	It's the length of the side opposite the third vertex
alpha	Vertex angle of the first vertex
beta	Vertex angle of the second vertex
gamma	Vertex angle of the third vertex
ab	Line defined by the first two points of the triangle
bc	Line defined by the last two points
ca	Line defined by the last and the first points of the triangle

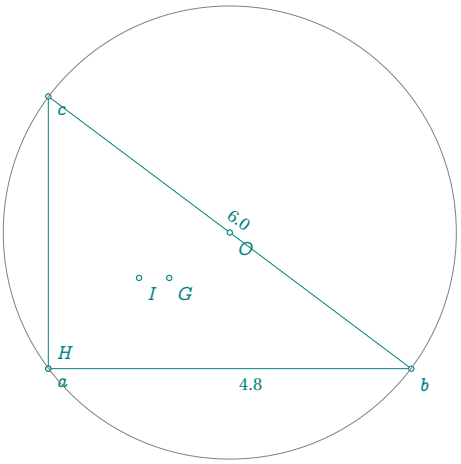
11.2 Triangle attributes: angles

```
\begin{tkzelements}
  z.A       = point: new(0,0)
  z.B       = point: new(5,0)
  z.C       = point: new(2,3)
  T.ABC     = triangle: new (z.A,z.B,z.C)
\end{tkzelements}
\def\wangle#1{\tkzDN[2]{%
  \tkzUseLua{math.deg(T.ABC.#1)}}}
\begin{tikzpicture}
\tkzGetNodes
  \tkzDrawPolygons(A,B,C)
  \tkzLabelAngle(B,A,C){$\wangle{alpha}^\circ$}
  \tkzLabelAngle(C,B,A){$\wangle{beta}^\circ$}
  \tkzLabelAngle(A,C,B){$\wangle{gamma}^\circ$}
\end{tikzpicture}
```



11.2.1 Example: triangle attributes

```
\begin{tkzelements}
   z.a   = point: new (0 , 0)
   z.b   = point: new (4 , 0)
   z.c   = point: new (0 , 3)
   T.abc = triangle : new (z.a,z.b,z.c)
   z.O   = T.abc.circumcenter
   z.I   = T.abc.incenter
   z.H   = T.abc.orthocenter
   z.G   = T.abc.centroid
   a     = T.abc.a
   b     = T.abc.b
   c     = T.abc.c
   alpha =  T.abc.alpha
   beta  =  T.abc.beta
   gamma =  T.abc.gamma
\end{tkzelements}
\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawPolygon(a,b,c)
   \tkzDrawPoints(a,b,c,O,G,I,H)
   \tkzLabelPoints(a,b,c,O,G,I)
   \tkzLabelPoints[above right](H)
   \tkzDrawCircles(O,a)
   \tkzLabelSegment[sloped](a,b){\tkzUseLua{c}}
   \tkzLabelSegment[sloped,above](b,c){\tkzUseLua{a}}
\end{tikzpicture}
```



11.3 Methods of the class triangle

Table 10: triangle methods.

Methods	Comments
new (a, b ,c)	T.ABC = triangle : new (z.A,z.B,z.C)
...	T or T.name with what you want for name, is possible.
Points	
lemoine_point ()	T.ABC : lemoine_point () intersection os the symmedians
symmedian_point ()	Lemoine point or the Grebe point
bevan_point ()	Circumcenter of the excentral triangle
mittenpunkt_point ()	Symmedian point of the excentral triangle
gergonne_point ()	Intersection of the three cevians that lead to the contact points
nagel_point ()	Intersection of the three cevians that lead to the extouch points
feuerbach_point ()	The point at which the incircle and euler circle are tangent.
spieker_center ()	Incenter of the medial triangle
barycenter (ka,kb,kc)	T.ABC: barycenter (2,1,1) barycenter of ({A,2},{B,1},{C,1})
base (u,v)	z.D = T.ABC: base(1,1) –> ABDC is a parallelogram
projection (p)	Projection of a point on the sides
euler_points ()	Euler points of euler circle
nine_points ()	9 Points of the euler circle
parallelogram ()	z.D = T.ABC : parallelogram () –> ABCD is a parallelogram
Lines	
altitude (n)	L.AHa = T.ABC : altitude () n empty or 0 line from A [a]
bisector (n)	L.Bb = T.ABC : bisector (1) n = 1 line from B [b]
bisector_ext(n)	n=2 line from the third vertex.
symmedian_line (n)	Cevian with respect to Lemoine point.
euler_line ()	the line through N ,G, H and O if the triangle is not equilateral [c]
antiparallel(pt,n)	n=0 antiparallel through pt to (BC), n=1 to (AC) etc.
Circles	
euler_circle ()	C.NP = T.ABC : euler_circle () –> N euler point [d]
circum_circle ()	C.OA = T.ABC : circum () Triangle's circumscribed circle
in_circle ()	Inscribed circle of the triangle
ex_circle (n)	Circle tangent to the three sides of the triangle ; n =1 swap ; n=2 2 swap
first_lemoine_circle ()	The center is the midpoint between Lemoine point and the circumcenter.[e]
second_lemoine_circle ()	see example 22.58
spieker_circle ()	The incircle of the medial triangle

a z.Ha = L.AHa.pb recovers the common point of the opposite side and altitude. The method orthic is usefull.
b _,z.b = get_points(L.Bb) recovers the common point of the opposite side and bisector.
c N center of nine points circle, G centroid, H orthocenter , O circum center
d The midpoint of each side of the triangle, the foot of each altitude, the midpoint of the line segment from each vertex of the triangle to the orthocenter.
e Through the Lemoine point draw lines parallel to the triangle's sides. The points where the parallel lines intersect the sides of ABC then lie on a circle known as the first Lemoine circle.

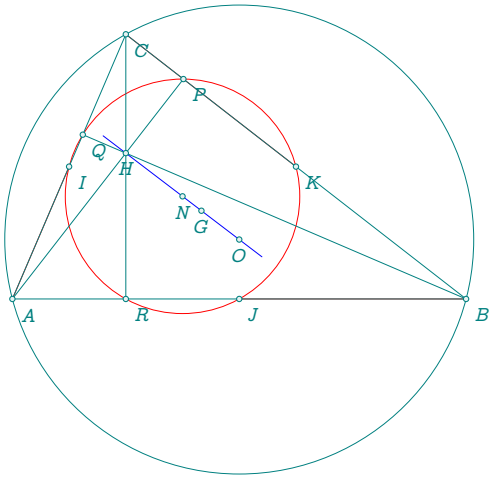
Remark: If you don't need to use the triangle object several times, you can obtain a bisector or a altitude with the next functions

bisector (z.A,z.B,z.C) and altitude (z.A,z.B,z.C) See (25)

Methods	Comments
Triangles	
orthic ()	T = T.ABC : orthic () triangle joining the feet of the altitudes
medial ()	T = T.ABC : medial () triangle with vertices at the midpoints
incentral ()	Cevian triangle of the triangle with respect to its incenter
excentral ()	Triangle with vertices corresponding to the excenters
extouch ()	Triangle formed by the points of tangency with the excircles
intouch ()	Contact triangle formed by the points of tangency of the incircle
tangential ()	Triangle formed by the lines tangent to the circumcircle at the vertices
feuerbach ()	Triangle formed by the points of tangency of the euler circle with the excircles
anti ()	Anticomplementary Triangle The given triangle is its medial triangle.
cevian (pt)	Triangle formed with the endpoints of the three cevians with respect to pt.
symmedian ()	Triangle formed with the intersection points of the symmedians.
euler ()	Triangle formed with the euler points
Miscellaneous	
area ()	$\mathcal{A}$ = T.ABC: area ()
barycentric_coordinates (pt)	Triples of numbers corresponding to masses placed at the vertices
in_out (pt)	Boolean. Test if pt is inside the triangle
check_equilateral ()	Boolean. Test if the triangle is equilateral

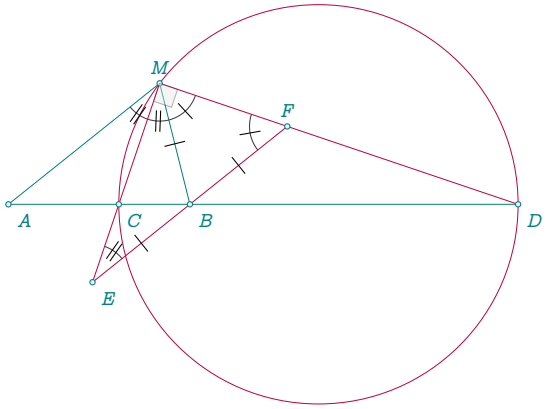
11.3.1 Euler line

```
\begin{tkzelements}
   z.A           = point: new (0 , 0)
   z.B           = point: new (6 , 0)
   z.C           = point: new (1.5 , 3.5)
   T.ABC         = triangle: new (z.A,z.B,z.C)
   z.O           = T.ABC.circumcenter
   z.G           = T.ABC.centroid
   z.N           = T.ABC.eulercenter
   z.H           = T.ABC.orthocenter
   z.P,z.Q,z.R   = get_points (T.ABC: orthic())
   z.K,z.I,z.J   = get_points (T.ABC: medial ())
\end{tkzelements}
\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawLines[blue](O,H)
   \tkzDrawCircle[red](N,I)
   \tkzDrawCircles[teal](O,A)
   \tkzDrawSegments(A,P B,Q C,R)
   \tkzDrawSegments[red](A,I B,J C,K)
   \tkzDrawPolygons(A,B,C)
   \tkzDrawPoints(A,B,C,N,I,J,K,O,P,Q,R,H,G)
   \tkzLabelPoints(A,B,C,I,J,K,P,Q,R,H)
   \tkzLabelPoints[below](N,O,G)
\end{tikzpicture}
```



11.4 Harmonic division and bisector

```
\begin{tkzelements}
   scale    =  .4
   z.A      = point: new (0 , 0)
   z.B      = point: new (6 , 0)
   z.M      = point: new (5 , 4)
   T.AMB    = triangle : new (z.A,z.M,z.B)
   L.AB     = T.AMB.ca
   L.bis    = T.AMB : bisector (1)
   z.C      = L.bis.pb
   L.bisext = T.AMB : bisector_ext (1)
   z.D      = intersection (L.bisext,L.AB)
   L.CD     = line: new (z.C,z.D)
   z.O      = L.CD.mid
   L.AM     = line: new (z.A,z.M)
   L.LL     = L.AM : ll_from (z.B)
   L.MC     = line: new (z.M,z.C)
   L.MD     = line: new (z.M,z.D)
   z.E      = intersection (L.LL,L.MC)
   z.F      = intersection (L.LL,L.MD)
\end{tkzelements}
```




```
\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawPolygon(A,B,M)
   \tkzDrawCircle[purple](O,C)
   \tkzDrawSegments[purple](M,E M,D E,F)
   \tkzDrawSegments(D,B)
   \tkzDrawPoints(A,B,M,C,D,E,F)
   \tkzLabelPoints[below right](A,B,C,D,E)
   \tkzLabelPoints[above](M,F)
   \tkzMarkRightAngle[opacity=.4,fill=gray!20](C,M,D)
   \tkzMarkAngles[mark=||,size=.5](A,M,E E,M,B B,E,M)
   \tkzMarkAngles[mark=|,size=.5](B,M,F M,F,B)
   \tkzMarkSegments(B,E B,M B,F)
\end{tikzpicture}
```

12 Classe ellipse

12.1 Attributes of an ellipse

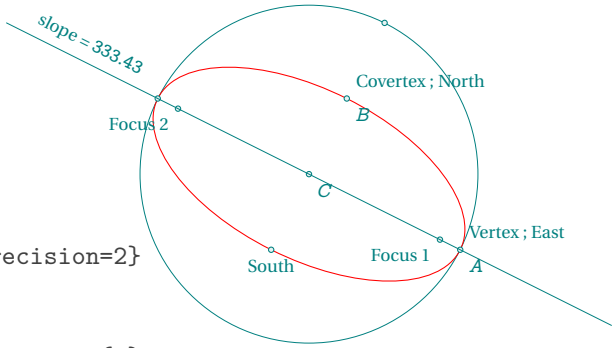
The first attributes are the three points that define the ellipse : `center` , `vertex` and `covertex`. The first method to define an ellipse is to give its center, then the point named **`vertex`** which defines the major axis and finally the point named **`covertex`** which defines the minor axis.

Table 11: Ellipse attributes.

Attributes	Application
`center`	center of the ellipse
`vertex`	point of the major axis and of the ellipse
`covertex`	point of the minor axis and of the ellipse
`type`	The type is 'ellipse'
`Rx`	Radius from center to vertex
`Ry`	Radius from center to covertex
`slope`	Slope of the line passes through the foci
`Fa`	First focus
`Fb`	Second focus
`south`	See next example 12.1.1
`north`	
`west`	
`east`	

12.1.1 Atributes of an ellipse: example

```
\begin{tkzelements}
   z.C   = point: new (3 , 2)
   z.A   = point: new (5 , 1)
   L.CA  = line : new (z.C,z.A)
   z.b   = L.CA.north_pa
   L     = line : new (z.C,z.b)
   z.B   = L : point (0.5)
   E     = ellipse: new (z.C,z.A,z.B)
   a     = E.Rx
   b     = E.Ry
   z.F1  = E.Fa
   z.F2  = E.Fb
   slope = math.deg(E.slope)
   z.E   = E.east
   z.N   = E.north
   z.W   = E.west
   z.S   = E.south
   z.Co  = E.covertex
   z.Ve  = E.vertex
\end{tkzelements}
\begin{tikzpicture}
   \pgfkeys{/pgf/number format/.cd,fixed,precision=2}
   \tkzGetNodes
   \tkzDrawCircles[teal](C,A)
   \tkzDrawEllipse[red](C,\tkzUseLua{a},\tkzUseLua{b},
   \tkzUseLua{slope})
   \tkzDrawPoints(C,A,B,b,W,S,F1,F2)
   \tkzLabelPoints(C,A,B)
   \tkzDrawLine[add = .5 and .5](A,W)
   \tkzLabelSegment[pos=1.5,above,sloped](A,W){%
    slope = \pgfmathprintnumber{\tkzUseLua{slope}}}
   \tkzLabelPoint[below](S){South}
   \tkzLabelPoint[below left](F1){Focus 1}
   \tkzLabelPoint[below left](F2){Focus 2}
   \tkzLabelPoint[above right](Ve){Vertex ; East}
   \tkzLabelPoint[above right](Co){Covertex ; North}
\end{tikzpicture}
```



12.2 Methods of the class ellipse

Before reviewing the methods and functions related to ellipses, let's take a look at how you can draw ellipses with `tkz-elements`. The **`\tkzDrawEllipse`** macro requires 4 arguments: the center of the ellipse, the long radius (on the focus axis), the short radius and the angle formed by the focus axis. The last three arguments must be transferred from **`tkzelements`** to **`tikzpicture`**. To do this, you'll need to use a `tkz-elements` function: `set_lua_to_tex`. See 19 or the next examples.

☞ You need to proceed with care, because unfortunately at the moment, the macros you create are global and you can overwrite existing macros. One solution is either to choose a macro name that won't cause any problems, or to save the initial macro.

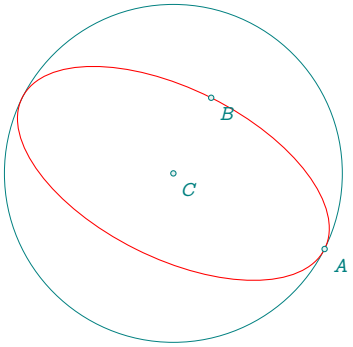
Table 12: Ellipse methods.

Methods	Example
new (pc, pa ,pb)	E = ellipse: new (center, vertex, covertex)
foci (f1,f2,v)	E = ellipse: foci (focus 1, focus 2, vertex)
radii (c,a,b,sl)	E = ellipse: radii (center, radius a, radius b, slope)
in_out (pt)	pt in/out of the ellipse
tangent_at (pt)	see example 9.2.5
tangent_from (pt)	see example 9.2.5
point (t)	vertex = point (0) covertex = point (0.25) ex see 9.2.5

12.2.1 Method new

The main method for creating a new ellipse is new. The arguments are three: center, vertex and covertex For attributes see 12

```
\begin{tkzelements}
   z.C      = point: new (3 , 2)
   z.A      = point: new (5 , 1)
   z.B      = z.C : homothety(0.5,
              z.C : rotation (math.pi/2,z.A))
   E        = ellipse: new (z.C,z.A,z.B)
   a        = E.Rx
   b        = E.Ry
   slope = math.deg(E.slope)
\end{tkzelements}
\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawCircles[teal](C,A)
   \tkzDrawEllipse[red](C,\tkzUseLua{a},
         \tkzUseLua{b},\tkzUseLua{slope})
   \tkzDrawPoints(C,A,B)
   \tkzLabelPoints(C,A,B)
\end{tikzpicture}
```

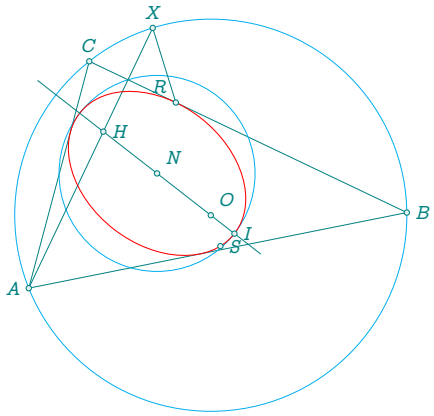


The function tkzUseLua (variable) is used to transfer values to Ti*k*Z or tkz-euclide.

12.2.2 Method foci

The first two points are the foci of the ellipse. The third one is the vertex. We can deduce all the other characteristics.
The function launches the new *method, all the characteristics of the ellipse are defined.*

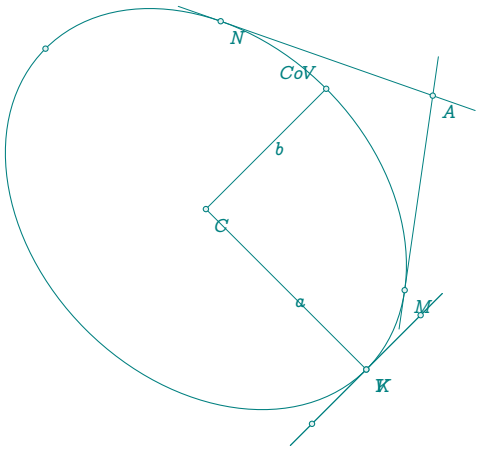
```
\begin{tkzelements}
   z.A      = point: new (0 , 0)
   z.B      = point: new (5 , 1)
   L.AB     = line : new (z.A,z.B)
   z.C      = point: new (.8 , 3)
   T.ABC    = triangle: new (z.A,z.B,z.C)
   z.N      = T.ABC.eulercenter
   z.H      = T.ABC.orthocenter
   z.O      = T.ABC.circumcenter
   _,_,z.Mc = get_points (T.ABC: medial ())
   L.euler  = line: new (z.H,z.O)
   C.circum = circle: new (z.O,z.A)
   C.euler  = circle: new (z.N,z.Mc)
   z.i,z.j  = intersection (L.euler,C.circum)
   z.I,z.J  = intersection (L.euler,C.euler)
   E        = ellipse: foci (z.H,z.O,z.I)
   L.AH     = line: new (z.A,z.H)
   z.X      = intersection (L.AH,C.circum)
   L.XO     = line: new (z.X,z.O)
   z.R,z.S  = intersection (L.XO,E)
   a,b      = E.Rx,E.Ry
   ang      = math.deg(E.slope)
\end{tkzelements}
\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawPolygon(A,B,C)
   \tkzDrawCircles[cyan](O,A N,I)
   \tkzDrawSegments(X,R A,X)
   \tkzDrawEllipse[red](N,\tkzUseLua{a},
      \tkzUseLua{b},\tkzUseLua{ang})
   \tkzDrawLines[add=.2 and .5](I,H)
   \tkzDrawPoints(A,B,C,N,O,X,H,R,S,I)
   \tkzLabelPoints[above](C,X)
   \tkzLabelPoints[above right](N,O)
   \tkzLabelPoints[above left](R)
   \tkzLabelPoints[left](A)
   \tkzLabelPoints[right](B,I,S,H)
   \end{tikzpicture}
```



12.2.3 Method point and radii

The method `point` defines a point M of the ellipse whose coordinates are $(a \times cos(phi), b \times sin(phi))$. `phi` angle between (center,vertex) and (center,M)
The environment **`tkzelements`** *uses as* **`lua`** *the radian as unit for angles.*

```
\begin{tkzelements}
   z.C         = point: new (2 , 3)
   z.A         = point: new (6 , 5)
   a           = value(4)
   b           = value(3)
   ang         = math.deg(-math.pi/4)
   E           = ellipse: radii (z.C,a,b,-math.pi/4)
   z.V         = E : point (0)
   z.K         = E : point (1)
   z.CoV       = E : point (0.25)
   z.X         = E : point (0.5)
   L           = E :tangent_at (z.V)
   z.x,z.y     = get_points(L)
   L.ta,L.tb   = E :tangent_from (z.A)
   z.M         = L.ta.pb
   z.N         = L.tb.pb
   L.K         = E :tangent_at (z.K)
   z.ka,z.kb   = get_points(L.K)
\end{tkzelements}
\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawSegments(C,V C,CoV)
   \tkzDrawLines(x,y A,M A,N ka,kb)
   \tkzLabelSegment(C,V){$a$}
   \tkzLabelSegment[right](C,CoV){$b$}
   \tkzDrawEllipse[teal](C,\tkzUseLua{a},\tkzUseLua{b},\tkzUseLua{ang})
   \tkzDrawPoints(C,V,CoV,X,x,y,M,N,A,K)
   \tkzLabelPoints(C,V,A,M,N,K)
   \tkzLabelPoints[above left](CoV)
\end{tikzpicture}
```



13 Classe Quadrilateral

13.1 Quadrilateral Attributes

Points are created in the direct direction. A test is performed to check whether the points form a rectangle, otherwise compilation is blocked.

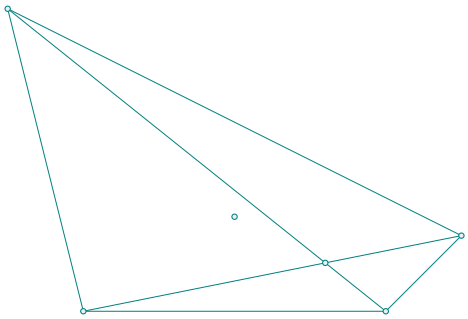
Creation `Q.new = rectangle : new (z.A,z.B,z.C,z.D)`

Table 13: rectangle attributes.

Attributes	Application	
pa	z.A = Q.new.pa	
pb	z.B = Q.new.pb	
pc	z.C = Q.new.pc	
pd	z.D = Q.new.pd	
type	Q.new.type= 'quadrilateral'	
i	z.I = Q.new.i	intersection of diagonals
g	z.G = Q.new.g	barycenter
a	AB = Q.new.a	barycenter
b	BC = Q.new.b	barycenter
c	CD = Q.new.c	barycenter
d	DA = Q.new.d	barycenter
ab	Q.new.ab	line passing through two vertices
ac	Q.new.ca	idem.
ad	Q.new.ad	idem.
bc	Q.new.bc	idem.
bd	Q.new.bd	idem.
cd	Q.new.cd	idem.

13.1.1 Quadrilateral attributes

```
\begin{tkzelements}
z.A      = point : new ( 0 , 0 )
z.B      = point : new ( 4 , 0 )
z.C      = point : new ( 5 , 1 )
z.D      = point : new ( -1 , 4 )
Q.ABCD   = quadrilateral : new ( z.A , z.B , z.C , z.D )
z.I      = Q.ABCD.i
z.G      = Q.ABCD.g
\end{tkzelements}

\begin{tikzpicture}
\tkzGetNodes
\tkzDrawPolygon(A,B,C,D)
\tkzDrawSegments(A,C B,D)
\tkzDrawPoints(A,B,C,D,I,G)
\end{tikzpicture}
```



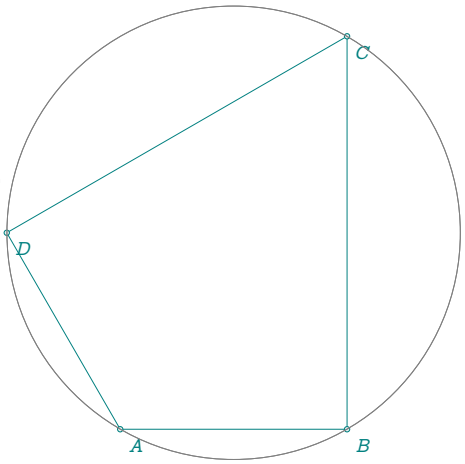
13.2 Quadrilateral methods

Table 14: Quadrilateral methods.

Methods	Comments
iscyclic ()	inscribed ? (see next example)

13.2.1 Inscribed quadrilateral

```
\begin{tkzelements}
z.A      = point : new ( 0 , 0 )
z.B      = point : new ( 4 , 0 )
z.D      = point : polar ( 4 , 2*math.pi/3 )
L.DB     = line : new (z.D,z.B)
T.equ    = L.DB : equilateral ()
z.C      = T.equ.pc
Q.new    = quadrilateral : new (z.A,z.B,z.C,z.D)
bool     = Q.new : iscyclic ()
if bool == true then
C.cir    = triangle : new (z.A,z.B,z.C): circum_circle ()
z.O      = C.cir.center
end
\end{tkzelements}

\begin{tikzpicture}
\tkzGetNodes
\tkzDrawPolygon(A,B,C,D)
\tkzDrawPoints(A,B,C,D)
\tkzLabelPoints(A,B,C,D)
\tkzDrawCircle(O,A)
\ifthenelse{\equal{\tkzUseLua{bool}}{true}}{
\tkzDrawCircle(O,A)}{}
\end{tikzpicture}
```



14 Classe square

14.1 Square attributes

Points are created in the direct direction. A test is performed to check whether the points form a square, otherwise compilation is blocked.

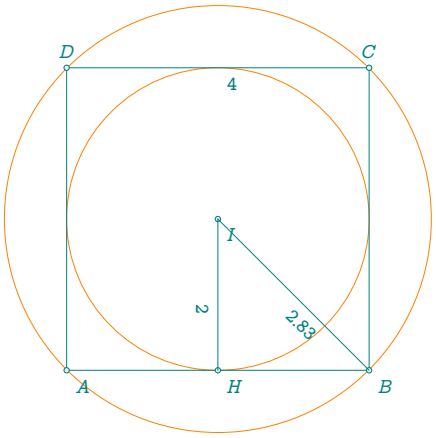
Creation `S.AB = square : new (z.A,z.B,z.C,z.D)`

Table 15: Square attributes.

Attributes	Application	
pa	z.A = S.AB.pa	
pb	z.B = S.AB.pb	
pc	z.C = S.AB.pc	
pd	z.D = S.AB.pd	
type	S.AB.type= 'square'	
side	s = S.AB.center	s = length of side
center	z.I = S.AB.center	center of the square
exradius	S.AB.exradius	radius of the circumscribed circle
inradius	S.AB.inxradius	radius of the inscribed circle
proj	S.AB.proj	projection of the center on one side
ab	S.AB.ab	line passing through two vertices
ac	S.AB.ca	idem.
ad	S.AB.ad	idem.
bc	S.AB.bc	idem.
bd	S.AB.bd	idem.
cd	S.AB.cd	idem.

14.1.1 Example: square attributes

```
\begin{tkzelements}
z.A         = point  : new ( 0 , 0 )
z.B         = point  : new ( 4 , 0 )
z.C         = point  : new ( 4 , 4)
z.D         = point  : new ( 0 , 4)
S.new       = square : new ( z.A , z.B ,z.C,z.D)
z.I         = S.new.center
z.H         = S.new.proj
\end{tkzelements}
\begin{tikzpicture}
\tkzGetNodes
\tkzDrawCircles[orange](I,A I,H)
\tkzDrawPolygon(A,B,C,D)
\tkzDrawPoints(A,B,C,D,H,I)
\tkzLabelPoints(A,B,H,I)
\tkzLabelPoints[above](C,D)
\tkzDrawSegments(I,B I,H)
\tkzLabelSegment[sloped](I,B){\pmpn{\tkzUseLua{S.new.exradius}}}
\tkzLabelSegment[sloped](I,H){\pmpn{\tkzUseLua{S.new.inradius}}}
\tkzLabelSegment[sloped](D,C){\pmpn{\tkzUseLua{S.new.side}}}
\end{tikzpicture}
```



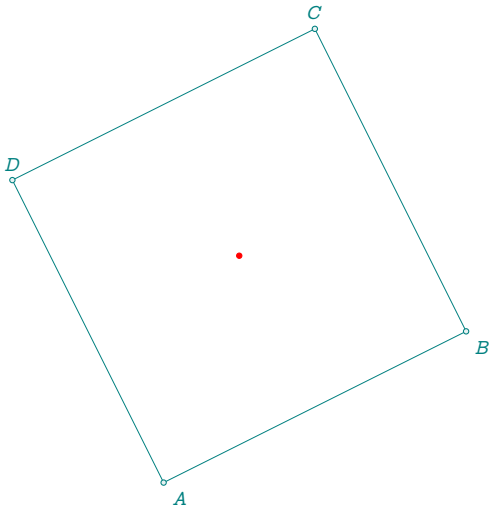
14.2 Square methods

Table 16: Square methods.

Methods	Comments	
rotation (zi,za)	S.IA = square : rotation (z.I,z.A)	*I* square center *A* first vertex
side (za,zb)	S.AB = square : side (z.A,z.B)	AB is the first side (direct)

14.2.1 Square with side method

```
\begin{tkzelements}
   scale       = 2
   z.A         = point : new ( 0 , 0 )
   z.B         = point : new ( 2 , 1 )
   S.side      = square : side (z.A,z.B)
   z.B         = S.side.pb
   z.C         = S.side.pc
   z.D         = S.side.pd
   z.I         = S.side.center
\end{tkzelements}

\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawPolygon(A,B,C,D)
   \tkzDrawPoints(A,B,C,D)
   \tkzLabelPoints(A,B)
   \tkzLabelPoints[above](C,D)
   \tkzDrawPoints[red](I)
\end{tikzpicture}
```



15 Classe rectangle

15.1 Rectangle attributes

Points are created in the direct direction. A test is performed to check whether the points form a rectangle, otherwise compilation is blocked.

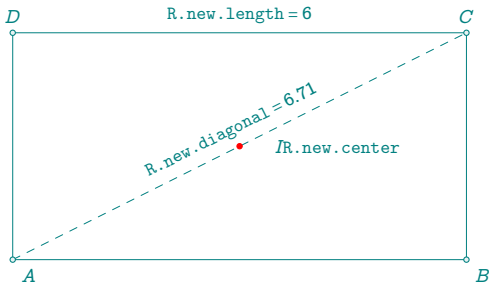
Creation `R.ABCD = rectangle : new (z.A,z.B,z.C,z.D)`

Table 17: rectangle attributes.

Attributes	Application	
pa	z.A = R.ABCD.pa	
pb	z.B = R.ABCD.pb	
pc	z.C = R.ABCD.pc	
pd	z.D = R.ABCD.pd	
type	R.ABCD.type= 'rectangle'	
center	z.I = R.ABCD.center	center of the rectangle
length	R.ABCD.length	the length
width	R.ABCD.width	the width
diagonal	R.ABCD.diagonal	diagonal length
ab	R.ABCD.ab	line passing through two vertices
ac	R.ABCD.ca	idem.
ad	R.ABCD.ad	idem.
bc	R.ABCD.bc	idem.
bd	R.ABCD.bd	idem.
cd	R.ABCD.cd	idem.

15.1.1 Example

```
\begin{tkzelements}
z.A   = point : new ( 0 , 0 )
z.B   = point : new ( 4 , 0 )
z.C   = point  : new ( 4 , 4)
z.D   = point  : new ( 0 , 4)
R.new = rectangle : new (z.A,z.B,z.C,z.D)
z.I   = R.new.center
\end{tkzelements}

\begin{tikzpicture}
\tkzGetNodes
\tkzDrawPolygon(A,B,C,D)
\tkzDrawPoints(A,B,C,D)
\tkzLabelPoints(A,B)
\tkzLabelPoints[above](C,D)
\tkzDrawPoints[red](I)
\end{tikzpicture}
```



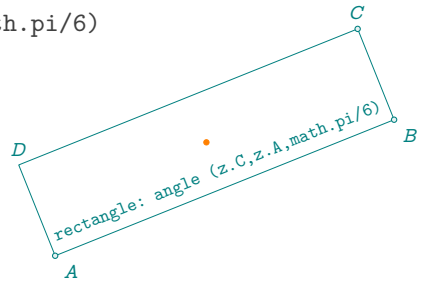
15.2 Rectangle methods

Table 18: Rectangle methods.

Methods	Comments	
angle (zi,za,angle)	R.ang = rectangle : angle (z.I,z.A);z.A	vertex ; ang angle between 2 vertices
gold (za,zb)	R.gold = rectangle : gold (z.A,z.B)	length/width = ϕ
diagonal (za,zc)	R.diag = rectangle : diagonal (z.I,z.A)	*I* square center *A* first vertex
side (za,zb,d)	S.IA = rectangle : side (z.I,z.A)	*I* square center *A* first vertex
get_lengths ()	S.IA = rectangle : get_lengths ()	*I* square center *A* first vertex

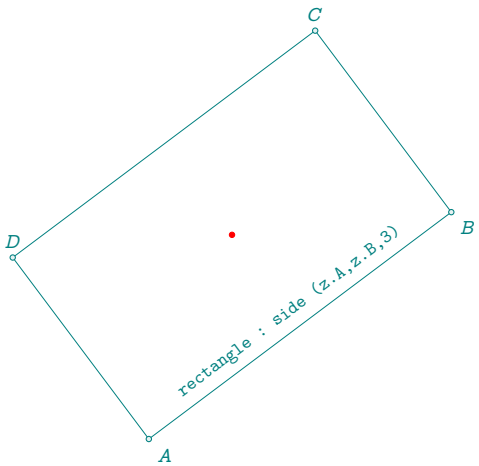
15.2.1 Angle method

```
\begin{tkzelements}
scale   = .5
z.A     = point : new ( 0 , 0 )
z.B     = point : new ( 4 , 0 )
z.I     = point : new ( 4 , 3 )
P.ABCD  = rectangle : angle ( z.I , z.A , math.pi/6)
z.B     = P.ABCD.pb
z.C     = P.ABCD.pc
z.D     = P.ABCD.pd
\end{tkzelements}

\begin{tikzpicture}
\tkzGetNodes
\tkzDrawPolygon(A,B,C,D)
\tkzDrawPoints(A,B,C)
\tkzLabelPoints(A,B,C,D)
\tkzDrawPoints[new](I)
\end{tikzpicture}
```



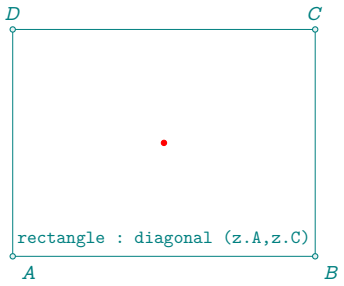
15.2.2 Side method

```
\begin{tkzelements}
z.A    = point : new ( 0 , 0 )
z.B    = point : new ( 4 , 3 )
R.side = rectangle : side (z.A,z.B,3)
z.C    = R.side.pc
z.D    = R.side.pd
z.I    = R.side.center
\end{tkzelements}
\begin{tikzpicture}
\tkzGetNodes
\tkzDrawPolygon(A,B,C,D)
\tkzDrawPoints(A,B,C,D)
\tkzLabelPoints(A,B)
\tkzLabelPoints[above](C,D)
\tkzDrawPoints[red](I)
\end{tikzpicture}
```



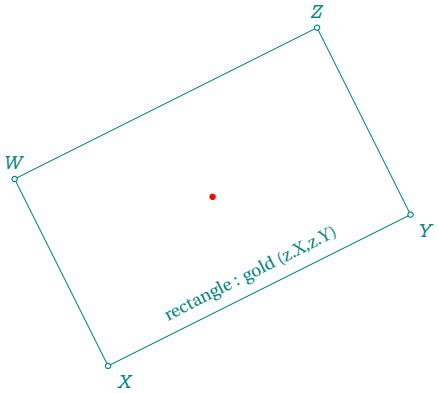
15.2.3 Diagonal method

```
\begin{tkzelements}
z.A         = point : new ( 0 , 0 )
z.C         = point : new ( 4 , 3 )
R.diag      = rectangle : diagonal (z.A,z.C)
z.B         = R.diag.pb
z.D         = R.diag.pd
z.I         = R.diag.center
\end{tkzelements}

\begin{tikzpicture}
\tkzGetNodes
\tkzDrawPolygon(A,B,C,D)
\tkzDrawPoints(A,B,C,D)
\tkzLabelPoints(A,B)
\tkzLabelPoints[above](C,D)
\tkzDrawPoints[red](I)
\tkzLabelSegment[sloped,above](A,B){|rectangle : diagonal (z.A,z.C)|}
\end{tikzpicture}
```



15.2.4 Gold method

```
\begin{tkzelements}
z.X    = point : new ( 0 , 0 )
z.Y    = point : new ( 4 , 2 )
R.gold = rectangle : gold (z.X,z.Y)
z.Z    = R.gold.pc
z.W    = R.gold.pd
z.I    = R.gold.center
\end{tkzelements}

\begin{tikzpicture}
\tkzGetNodes
\tkzDrawPolygon(X,Y,Z,W)
\tkzDrawPoints(X,Y,Z,W)
\tkzLabelPoints(X,Y)
\tkzLabelPoints[above](Z,W)
\tkzDrawPoints[red](I)
\tkzLabelSegment[sloped,above](X,Y){rectangle :  gold (z.X,z.Y)}
\end{tikzpicture}
```



16 Classe parallelogram

16.1 Parallelogram attributes

Points are created in the direct direction. A test is performed to check whether the points form a parallelogram, otherwise compilation is blocked.

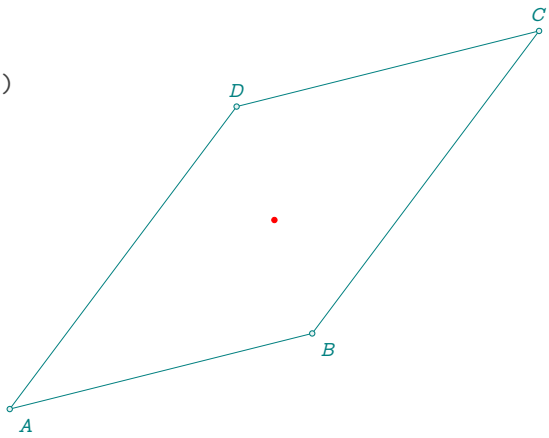
Creation `P.new = parallelogram : new (z.A,z.B,z.C,z.D)`

Table 19: Parallelogram attributes.

Attributes	Application	
pa	z.A = P.new.pa	
pb	z.B = P.new.pb	
pc	z.C = P.new.pc	
pd	z.D = P.new.pd	
type	P.new.type= 'parallelogram'	
i	z.I = P.new.i	intersection of diagonals
ab	P.new.ab	line passing through two vertices
ac	P.new.ca	idem.
ad	P.new.ad	idem.
bc	P.new.bc	idem.
bd	P.new.bd	idem.
cd	P.new.cd	idem.

16.1.1 Example: attributes

```
\begin{tkzelements}
z.A          = point : new ( 0 , 0 )
z.B          = point : new ( 4 , 1 )
z.C          = point : new ( 7 , 5 )
z.D          = point : new ( 3 , 4 )
P.new        = parallelogram : new (z.A,z.B,z.C,z.D)
z.B          = P.new.pb
z.C          = P.new.pc
z.D          = P.new.pd
z.I          = P.new.center
\end{tkzelements}
\begin{tikzpicture}
\tkzGetNodes
\tkzDrawPolygon(A,B,C,D)
\tkzDrawPoints(A,B,C,D)
\tkzLabelPoints(A,B)
\tkzLabelPoints[above](C,D)
\tkzDrawPoints[red](I)
\end{tikzpicture}
```



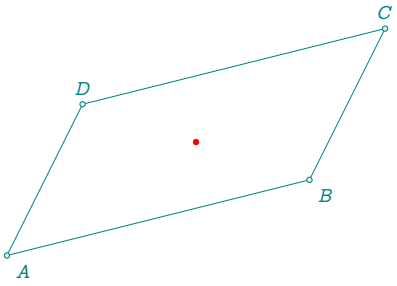
16.2 Parallelogram methods

Table 20: Parallelogram methods.

Methods	Comments
`fourth (za,zb,zc)`	completes a triangle by parallelogram (see next example)

16.2.1 parallelogram with fourth method

```
\begin{tkzelements}
   scale = .75
z.A      = point : new ( 0 , 0 )
z.B      = point : new ( 4 , 1 )
z.C      = point : new ( 5 , 3 )
P.four   = parallelogram : fourth (z.A,z.B,z.C)
z.D      = P.four.pd
z.I      = P.four.center
\end{tkzelements}
\begin{tikzpicture}
\tkzGetNodes
\tkzDrawPolygon(A,D,B,C)
\tkzDrawPoints(A,B,C,D)
\tkzLabelPoints(A,B)
\tkzLabelPoints[above](C,D)
\tkzDrawPoints[red](I)
\end{tikzpicture}
```



17 Classe Regular Polygon

17.1 regular_polygon attributes

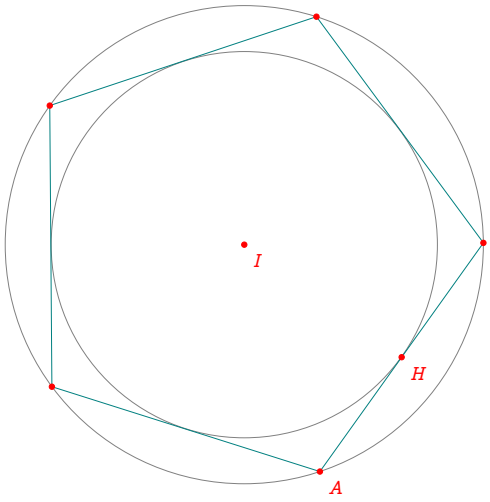
Creation `RP.IA = regular_polygon : new (z.I,z.A,6)`

Table 21: Regular_polygon attributes.

Attributes	Application
center	z.I = RP.IA.center
table	array containing all vertex affixes
through	first vertex
circle	defines the circle with center I passing through A
type	RP.IA.type= 'regular_polygon'
side	s = RP.IA.side ; s = length of side
exradius	S.AB.exradius ; radius of the circumscribed circle
inradius	S.AB.inxradius ; radius of the inscribed circle
proj	RP.IA.proj ; projection of the center on one side
angle	RP.IA.angle ; angle formed by the center and 2 consecutive vertices

17.1.1 Pentagon

```
\begin{tkzelements}
z.O     = point:    new (0,0)
z.I     = point:    new (1,3)
z.A     = point:    new (2,0)
RP.five  = regular_polygon : new (z.I,z.A,5)
RP.five : name ("P_")
C.ins  = circle: radius (z.I,RP.five.inradius)
z.H = RP.five.proj
\end{tkzelements}
\begin{tikzpicture}
\def\nb{\tkzUseLua{RP.five.nb}}
\tkzGetNodes
\tkzDrawCircles(I,A I,H)
\tkzDrawPolygon(P_1,P_...,P_\nb)
\tkzDrawPoints[red](P_1,P_...,P_\nb,H,I)
\tkzLabelPoints[red](I,A,H)
\end{tikzpicture}
```



17.2 regular_polygon methods

Table 22: Circle methods.

Methods	Comments
new(O,A,n)	RP.five = regular_polygon : new (z.I,z.A,5) ; I center A first vertex 5 sides
Circle	
incircle ()	C.IH = RP.five : incircle ()
Points	
name (string)	see17.1.1

18 Class vector

In fact, they are more a class of oriented segments than vectors in the strict mathematical sense.
A vector is defined by giving two points (i.e. two affixes). `V.AB = vector : new (z.A,z.B)` creates the vector $\vec{(}AB)$, i.e. the oriented segment with origin A representing a vector. A few rudimentary operations are defined, such as sum, subtraction and multiplication by a scalar.
The sum is defined as follows:
Let V.AB + V.CD result in a vector V.AE defined as follows
If $\overrightarrow{CD} = \overrightarrow{BE}$ then $\overrightarrow{AB} + \overrightarrow{CD} = \overrightarrow{AB} + \overrightarrow{BE} = \vec{(}AE)$

> Creation `V.AB = vector: new (z.A,z.B)`

```
z.A = ...
z.B = ...
z.C = ...
z.D = ...
V.AB = vector : new (z.A,z.B)
V.CD = vector : new (z.C,z.D)
V.AE = V.AB + V.CD  -- possible V.AB : add (V.CD)
z.E  = V.AE.h -- we recover the final point (h = head)
```

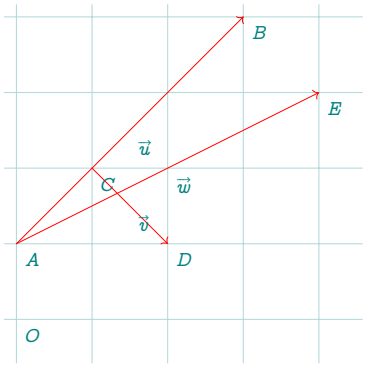
18.1 Attributes of a vector

Table 23: Vector attributes.

Attributes	Application	Example
pa	`V.AB.t = z.A` t for tail	see (7.2.2)
pb	`V.AB.h = z.B` h for head	see (7.2.2)
type	`V.AB.type = 'vector'`	
slope	`V.AB.slope`	see (18.1.1)
length	`V.AB.norm`	see (18.1.1)

18.1.1 Example vector attributes

```
\begin{tkzelements}
  z.O          = point: new (0,0)
  z.A          = point: new (0,1)
  z.B          = point: new (3,4)
  L.AB         = line : new ( z.A , z.B )
  z.C          = point: new (1,2)
  z.D          = point: new (2,1)
  u            = vector : new (z.A,z.B)
  v            = vector : new (z.C,z.D)
  w =u+v
  z.E = w.h
\end{tkzelements}
\begin{tikzpicture}[gridded]
    \tkzGetNodes
    \tkzLabelPoints(A,B,C,D,O,E)
    \tkzDrawSegments[->,red](A,B C,D A,E)
    \tkzLabelSegment(A,B){$\overrightarrow{u}$}
    \tkzLabelSegment(C,D){$\overrightarrow{v}$}
    \tkzLabelSegment(A,E){$\overrightarrow{w}$}
  \end{tikzpicture}
$\overrightarrow{w}$ has slope :
$\tkzDN{\tkzUseLua{math.deg(w.slope)}}^\circ$
```



$\overrightarrow{w}$ has slope : 26.57°

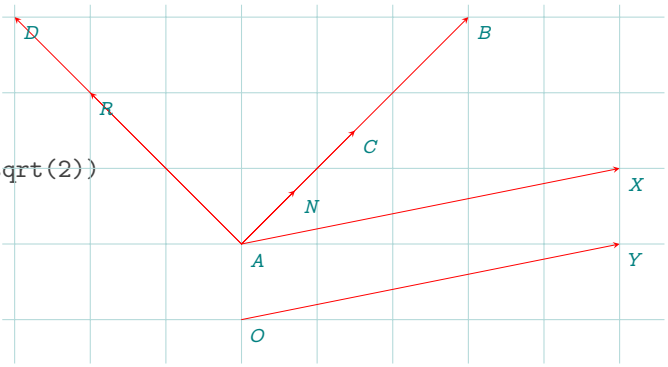
18.2 Methods of the class vector

Table 24: Methods of the class vector.

Metamethods	**Application**	
`__add (u,v)`	`V.AB + V.CD`	
`__sub (u,v)`	`V.AB - V.CD`	
`__unm (u)`	`V.CD = -V.AB`	
`__mul (k,u)`	`V.CD = k*V.AB`	
Methods	**Application**	
`new(pt, pt)`	`V.AB = vector: new (z.A,z.B)`	
`normalize(V)`	`V.AB : normalize ()`	
`orthogonal(d)`	`V.AB : orthogonal (d)`	
`scale(d)`	`V.CD = V.AB : scale (2)`	$\overrightarrow{CD}=2\overrightarrow{AB}$
`at (V)`	`V.DB = V.AC : at (z.D)`	$\overrightarrow{DB}=\overrightarrow{AC}$

18.2.1 Example of methods

```
\begin{tkzelements}
  z.O          = point: new (0,0)
  z.A          = point: new (0,1)
  z.B          = point: new (3,4)
  V.AB         = vector: new (z.A,z.B)
  V.AC         = V.AB : scale (.5)
  z.C          = V.AC.h
  V.AD         = V.AB : orthogonal ()
  z.D          = V.AD.h
  V.AN         = V.AB : normalize ()
  z.N          = V.AN.h
  V.AR         = V.AB : orthogonal (2*math.sqrt(2))
  z.R          = V.AR.h
  V.AX         = 2*V.AC - V.AR
  z.X          = V.AX.h
  V.OY         = V.AX : at (z.O)
  z.Y          = V.OY.h
\end{tkzelements}
\begin{tikzpicture}[gridded]
  \tkzGetNodes
  \tkzDrawSegments[>=stealth,->,red](A,B A,C A,D A,N A,R A,X O,Y)
  \tkzLabelPoints(A,B,C,D,O,N,R,X,Y)
\end{tikzpicture}
```



19 Math constants and functions

Table 25: Math constants and functions.

contants or functions	Comments
tkzphi	constant $\varphi = (1 + math.sqrt(5))/2$
tkzinvphi	constant $1/\varphi = 1/tkzphi$
tkzsqrtphi	constant $\sqrt{\varphi} = math.sqrt(tkzphi)$
length (a,b)	point.abs(a-b) See (8.2.2)
islinear (z1,z2,z3)	Are the points aligned? (z2-z1) ‖ (z3-z1) ?
isortho (z1,z2,z3)	(z2-z1) ⊥ (z3-z1) ? boolean
get_angle (z1,z2,z3)	the vertex is z1 See (19.7)
bisector (z1,z2,z3)	L.Aa = bisector (z.A,z.B,z.C) from A (19.7)
bisector_ext (z1,z2,z3)	L.Aa = bisector_ext (z.A,z.B,z.C) from A
altitude (z1,z2,z3)	altitude from z1
set_lua_to_tex (list)	set_lua_to_tex('a','n') defines \a and \n
tkzUseLua (variable)	\textbackslash\tkzUseLua{a} prints the value of a
value (v)	apply scale * value
real (v)	apply value /scale
angle_normalize (an)	to get a value between 0 and 2π
barycenter ({z1,n1},{z2,n2}, ...)	barycenter of list of points

19.1 Length of a segment

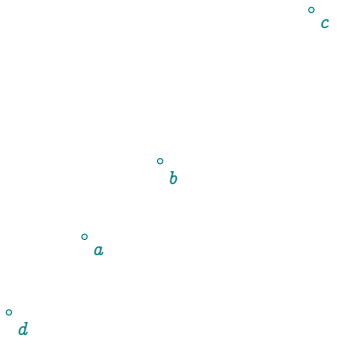
length(z.A,z.B) is a shortcut for point.abs(z.A-z.B). This avoids the need to use complexes.

19.2 Harmonic division with tkzphi

```
\begin{tkzelements}
   scale =.5
   z.a = point: new(0,0)
   z.b = point: new(8,0)
   L.ab = line: new (z.a,z.b)
   z.m,z.n = L.ab: harmonic_both (tkzphi)
\end{tkzelements}
\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawLine[add= .2 and .2](a,n)
   \tkzDrawPoints(a,b,n,m)
   \tkzLabelPoints(a,b,n,m)
\end{tikzpicture}
```

19.3 Function islinear

```
\begin{tkzelements}
   z.a = point: new (1, 1)
   z.b = point: new (2, 2)
   z.c = point: new (4, 4)
   if islinear (z.a,z.b,z.c) then
       z.d = point: new (0, 0)
    else
        z.d = point: new (-1, -1)
   end
\end{tkzelements}
\begin{tikzpicture}
    \tkzGetNodes
    \tkzDrawPoints(a,...,d)
    \tkzLabelPoints(a,...,d)
\end{tikzpicture}
```



19.4 Function value

value to apply scaling if necessary
If `scale = 1.2` with a = value(5) the actual value of a will be $5 \times 1.2 = 6$.

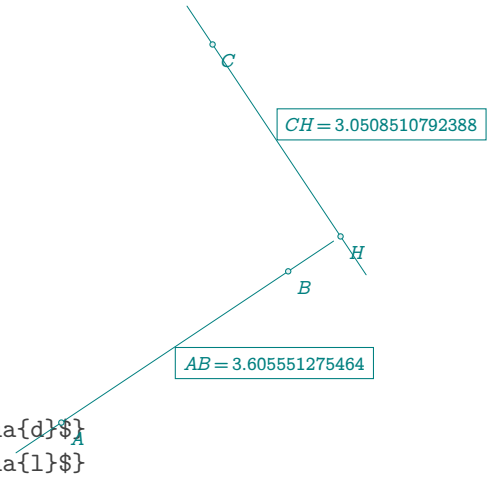
19.4.1 Function real

If `scale = 1.2` with a = 6 then real(a) = $6/1.2 = 5$.

19.5 Transfer from lua to TEX

It's possible to transfer variable from Lua to TEX with `\tkzUseLua`.

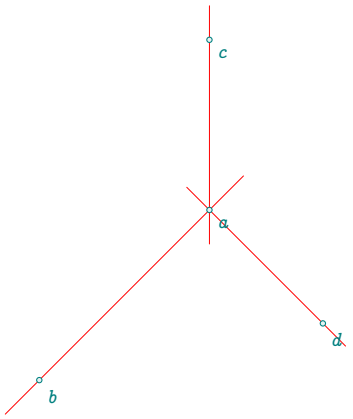
```
\begin{tkzelements}
   z.A            = point : new (0 , 0)
   z.B            = point : new (3 , 2)
   z.C            = point : new (2 , 5)
   L.AB           = line : new (z.A,z.B)
   d              = L.AB : distance (z.C)
   l              = L.AB.length
   z.H            = L.AB : projection (z.C)
\end{tkzelements}
\begin{tikzpicture}
\tkzGetNodes
\tkzDrawLines(A,B C,H)
\tkzDrawPoints(A,B,C,H)
\tkzLabelPoints(A,B,C,H)
\tkzLabelSegment[above right,draw](C,H){$CH = \tkzUseLua{d}$}
\tkzLabelSegment[below right,draw](A,B){$AB = \tkzUseLua{l}$}
\end{tikzpicture}
```



19.6 Normalized angles : Slope of lines (ab), (ac) and (ad)

```
\begin{tkzelements}
   z.a      = point: new(0, 0)
   z.b      = point: new(-3, -3)
   z.c      = point: new(0, 3)
   z.d      = point: new(2, -2)
   angle    = point.arg (z.b-z.a)
   tex.print('slope of (ab) : '..tostring(angle)..'\\\\')
   tex.print('slope normalized of (ab) : '..tostring(angle\_normalize(angle))..'\\\\')
   angle    = point.arg (z.c-z.a)
   tex.print('slope of (ac) : '..tostring(angle)..'\\\\')
   tex.print('slope normalized of (ac) : '..tostring(angle\_normalize(angle))..'\\\\')
   angle    = point.arg (z.d-z.a)
   tex.print('slope of (ad) : '..tostring(angle)..'\\\\')
   tex.print('slope normalized of (acd) : '..tostring(angle\_normalize(angle))..'\\\\')
\end{tkzelements}
\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawLines[red](a,b a,c a,d)
   \tkzDrawPoints(a,b,c,d)
   \tkzLabelPoints(a,b,c,d)
\end{tikzpicture}
```

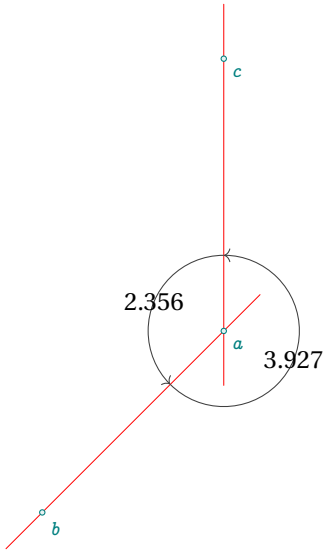
slope of (ab) : -2.3561944901923
slope normalized of (ab) : 3.9269908169872
slope of (ac) : 1.5707963267949
slope normalized of (ac) : 1.5707963267949
slope of (ad) : -0.78539816339745
slope normalized of (ad) : 5.4977871437821



19.7 Get angle

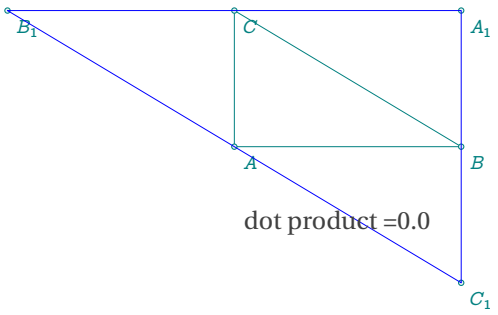
The function `get_angle (a,b,c)` gives the angle normalized of $(\overrightarrow{ab},\overrightarrow{ac})$.

```
\begin{tkzelements}
   z.a   = point: new(0, 0)
   z.b   = point: new(-2, -2)
   z.c   = point: new(0, 3)
   angcb = tkzround ( get_angle (z.a,z.c,z.b),3)
   angbc = tkzround ( get_angle (z.a,z.b,z.c),3)
\end{tkzelements}

\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawLines[red](a,b a,c)
   \tkzDrawPoints(a,b,c)
   \tkzLabelPoints(a,b,c)
   \tkzMarkAngle[->](c,a,b)
   \tkzLabelAngle(c,a,b){\tkzUseLua{angcb}}
   \tkzMarkAngle[->](b,a,c)
   \tkzLabelAngle(b,a,c){\tkzUseLua{angbc}}
\end{tikzpicture}
```



19.8 Dot or scalar product

```
\begin{tkzelements}
   z.A    = point: new(0,0)
   z.B    = point: new(5,0)
   z.C    = point: new(0,3)
   T.ABC  = triangle: new (z.A,z.B,z.C)
   z.A_1,
   z.B_1,
   z.C_1  = get_points (T.ABC: anti ())
   x  = dot_product (z.A,z.B,z.C)
\end{tkzelements}
\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawPolygon(A,B,C)
   \tkzDrawPoints(A,B,C,A_1,B_1,C_1)
   \tkzLabelPoints(A,B,C,A_1,B_1,C_1)
   \tkzDrawPolygon[blue](A_1,B_1,C_1)
   \tkzText[right](0,-
1){dot product =\tkzUseLua{x}}
\end{tikzpicture}
```



The scalar product of the vectors $\overrightarrow{AC}$ and $\overrightarrow{AB}$ is equal to 0.0, so these vectors are orthogonal.

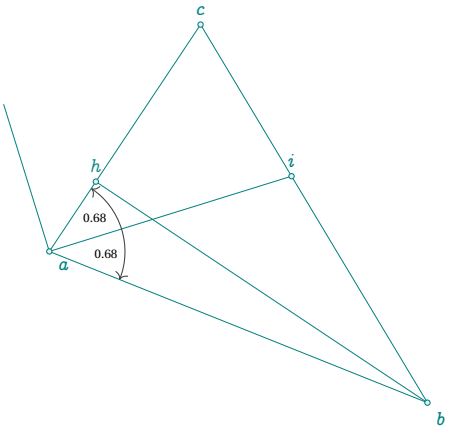
19.9 Alignment or orthogonality

With the functions `islinear` and `isortho`. `islinear(z.a,z.b,z.c)` gives `true` idf the points a, b and c are aligned.
`isortho(z.a,z.b,z.c)` gives `true` if the line (ab) is orthogonal to the line (ac).

19.10 Bisector and altitude

These functions are useful if you don't need to create a useful triangle object for the rest of your code.

```
\begin{tkzelements}
   z.a   = point: new (0, 0)
   z.b   = point: new (5, -2)
   z.c   = point: new (2, 3)
   z.i   = bisector (z.a,z.c,z.b).pb
   z.h   = altitude (z.b,z.a,z.c).pb
   angic = tkzround ( get_angle (z.a,z.i,z.c),2)
   angci = tkzround ( get_angle (z.a,z.b,z.i),2)
   z.e = bisector_ext (z.a,z.b,z.c).pb
\end{tkzelements}

\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawPolygon(a,b,c)
   \tkzDrawSegments(a,i b,h a,e)
   \tkzDrawPoints(a,b,c,i,h)
   \tkzLabelPoints(a,b)
   \tkzLabelPoints[above](c,i,h)
   \tkzMarkAngle[->](i,a,c)
   \tkzLabelAngle[font=\tiny,pos=.75](i,a,c){\tkzUseLua{angci}}
   \tkzMarkAngle[<-](b,a,i)
   \tkzLabelAngle[font=\tiny,pos=.75](b,a,i){\tkzUseLua{angic}}
\end{tikzpicture}
```



19.11 Other functions

Not documented because still in beta version: parabola, Cramer22, Cramer33.

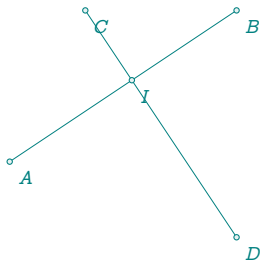
20 Intersections

It's an essential tool. For the moment, the classes concerned are lines, circles and ellipses, with the following combinations: line-line; line-circle; circle-circle and line-ellipse. The argument is a pair of objects, in any order. Results consist of one or two values, either points, boolean **false** or underscore _.

20.1 Line-line

The result is of the form: `point` or `false`.

```
\begin{tkzelements}
   z.A   = point : new (1,-1)
   z.B   = point : new (4,1)
   z.C   = point : new (2,1)
   z.D   = point : new (4,-2)
    z.I  = point : new (0,0)
   L.AB  = line : new (z.A,z.B)
   L.CD  = line : new (z.C,z.D)
   x     = intersection (L.AB,L.CD)
   if x  == false then
   tex.print('error')
   else
   z.I   = x
   end
\end{tkzelements}

\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawSegments(A,B C,D)
   \tkzDrawPoints(A,B,C,D,I)
   \tkzLabelPoints(A,B,C,D,I)
\end{tikzpicture}
```

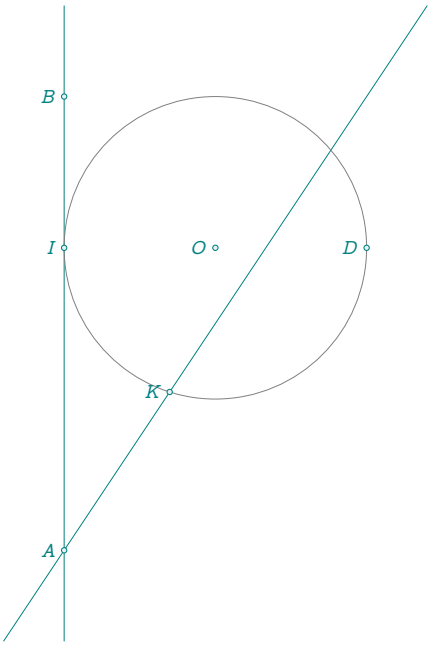


Other examples: 10.2.1, 10.2.2, 22.3

20.2 Line-circle

The result is of the form : `point,point` or `false,false`. If the line is tangent to the circle, then the two points are identical. You can ignore one of the points by using the underscore: `_,point` or `point,_`. When the intersection yields two solutions, the order of the points is determined by the argument of `(z.p - z.c)` with `c` center of the circle and `p` point of intersection. The first solution corresponds to the smallest argument (arguments are between 0 and 2π).

```
\begin{tkzelements}
   z.A   = point : new (1,-1)
   z.B   = point : new (1,2)
   L.AB  = line : new (z.A,z.B)
   z.O   = point : new (2,1)
   z.D   = point : new (3,1)
   z.E   = point : new (3,2)
   L.AE  = line : new (z.A,z.E)
   C.OD  = circle : new (z.O,z.D)
   z.I,_ = intersection (L.AB,C.OD)
   _,z.K = intersection (C.OD,L.AE)
\end{tkzelements}

\begin{tikzpicture}
\tkzGetNodes
   \tkzDrawLines(A,B A,E)
   \tkzDrawCircle(O,D)
   \tkzDrawPoints(A,B,O,D,I,K)
   \tkzLabelPoints[left](A,B,O,D,I,K)
\end{tikzpicture}
```

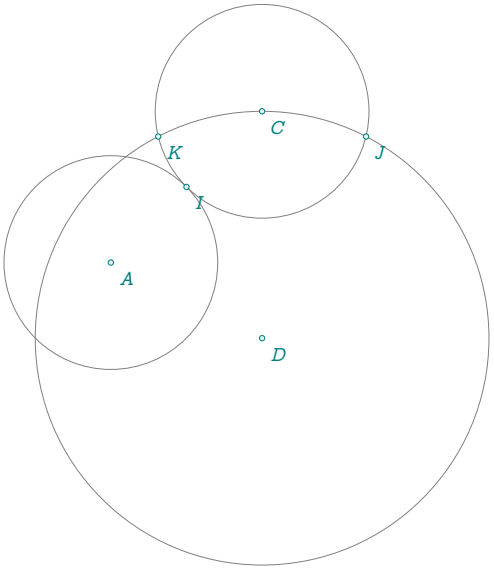


Other examples: 10.2.1

20.3 Circle-circle

The result is of the form : `point,point` or `false,false`. If the circles are tangent, then the two points are identical. You can ignore one of the points by using the underscore: `_ , point` or `point , _`. As for the intersection of a line and a circle, consider the argument of `z.p-z.c` with `c` center of the first circle and `p` point of intersection. The first solution corresponds to the smallest argument (arguments are between 0 and 2π).

```
\begin{tkzelements}
   z.A      = point : new (1,1)
   z.B      = point : new (2,2)
   z.C      = point : new (3,3)
   z.D      = point : new (3,0)
   C.AB     = circle : new (z.A,z.B)
   C.CB     = circle : new (z.C,z.B)
   z.I,_    = intersection (C.AB,C.CB)
   C.DC     = circle : new (z.D,z.C)
   z.J,z.K  = intersection (C.DC,C.CB)
\end{tkzelements}
\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawCircles(A,B C,B D,C)
   \tkzDrawPoints(A,I,C,D,J,K)
   \tkzLabelPoints(A,I,C,D,J,K)
\end{tikzpicture}
```

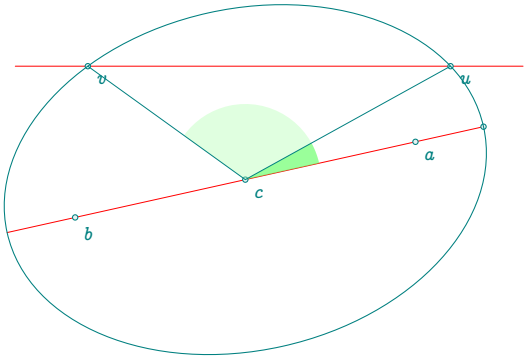


Other examples: 10.2.1, 3.3

20.4 Line-ellipse

The following example is complex, but it shows the possibilities of Lua. The designation of intersection points is a little more complicated than the previous one, as the argument characterizing the major axis must be taken into account. The principle is the same, but this argument must be subtracted. In concrete terms, you need to consider the slopes of the lines formed by the center of the ellipse and the points of intersection, and the slope of the major axis.

```
\begin{tkzelements}
   scale    = .5
   z.a      = point: new (5 , 2)
   z.b      = point: new (-4 , 0)
   z.m      = point: new (2 , 4)
   z.n      = point: new (4 , 4)
   L.ab     = line : new (z.a,z.b)
   L.mn     = line : new (z.m,z.n)
   z.c      = L.ab. mid
   z.e      = L.ab: point (-.2)
   E        = ellipse: foci (z.a,z.b,z.e)
   z.u,z.v  = intersection (E,L.mn)
  -- transfer to tex
   a        = E.Rx
   b        = E.Ry
   ang      = math.deg(E.slope)
\end{tkzelements}
\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawLines[red](a,b u,v) % p,s p,t
   \tkzDrawPoints(a,b,c,e,u,v) %
   \tkzLabelPoints(a,b,c,u,v)
   \tkzDrawEllipse[teal](c,\tkzUseLua{a},\tkzUseLua{b},\tkzUseLua{ang})
   \tkzDrawSegments(c,u c,v)
   \tkzFillAngles[green!30,opacity=.4](e,c,v)
   \tkzFillAngles[green!80,opacity=.4](e,c,u)
\end{tikzpicture}
```



Other examples: 12.2.2, 22.32

21 In-depth study

21.1 The tables

21.1.1 General tables

Tables are the only data structure ”container” integrated in Lua. They are associative arrays which associates a key (reference or index) with a value in the form of a field (set) of key/value pairs. Moreover, tables have no fixed size and can grow based on our need dynamically.
Tables are created using table constructors, the simplest of which is the use of braces, e.g. {}. This defines an empty table.

```
F = {"banana", "apple", "cherry"}
```

print(F[2]) –> pomme
qui peut être également définit par

```
FR = {[1] = "banana", [3] = "cherry", [2] = "apple"}
```

print(FR[3]) –> cherry
FR[4]=”orange”

```
print(#FR)
-- I for Index
for I,V in ipairs(FR) do
   print(I,V)
end
```

1 banana
2 apple
3 cherry
4 orange

```
C = {["banana"] = "yellow" , ["apple"] = "green" , ["cherry"] = "red" }
C.orange = "orange"
```

```
for K,V in pairs (C) do
   print(K,V)
end
```

banana = yellow cherry = red orange = orange apple = green
Another useful feature is the ability to create a table to store an unknown number of function parameters, for example:

```
function ReturnTable (...)
 return table.pack (...)
 end
```

```
function ParamToTable (...)
   mytab =  ReturnTable(...)
   for i=1,mytab.n do
      print(mytab[i])
   end
end
ParamToTable("cherry","apple","orange")
```

Using tables with table[key] syntax:
`C["banana"] and F[1]`
But with string constants as keys we have the sugar syntax: C.banana but this syntax does not accept numbers.
It's possible to erase a key/value pair from a table, with :

```
C.banana = nil
```

21.1.2 Table z

This is the most important table in the package. It stores all points and enables them to be transferred to Ti*k*Z.
It is defined with `z = {}`, then each time we write

```
z.name = point : new (a , b)
```

a point object is stored in the table. The key is `name`, the value is an object. We have seen that `z.name.re = a` and that `z.name.im = b`.
However, the elements of this table have essential properties.
For example, if you wish to display an element, then `tex.print(tostring(z.name)) = a+ib` the `tostring` operation displays the affix corresponding to the point.
In addition, we'll see that it's possible to perform operations with the elements of the `z` table.

21.2 Transferts

We've seen (sous-section 6.1.1) that the macro transfers point coordinates to Ti*k*Z. Let's take a closer look at this macro:

```
\def\tkzGetNodes{\directlua{%
   for K,V in pairs(z) do
      local K,n,sd,ft
      n = string.len(KS)
      if n >1 then
      _,_,ft, sd = string.find( K , "(.+)(.)" )
     if sd == "p" then   K=ft.."'" end
       end
  tex.print("\\coordinate ("..K..") at ("..V.re..","..V.im..") ;\\\\\")
end}
}
```

It consists mainly of a loop. The variables used are K (for keys) and V (for Values). To take pairs (key/value) from the `z` table, use the `pairs` function. K becomes the name of a node whose coordinates are `V.re` and `V.im`. Meanwhile, we search for keys with more than one symbol ending in `p`, in order to associate them with the symbol "'" valid in Ti*k*Z.

21.3 Complex numbers library and point

Unless you want to create your own functions, you won't need to know and use complex numbers. However, in some cases it may be useful to implement some of their properties.
`z.A = point : new (1,2 )` and `z.B = point : new (1,-1)` define two affixes which are $z_A = 1+2i$ and $z_B = 1-i$. Note the difference in notations `z.A` and z_A for two distinct entities: a Lua object and an affix.

If you want to use only complex numbers then you must choose the following syntax :`za =point (1,2)`. The difference between `z.A = point : new (1,2)` and `za = point (1,2)` is that the first function takes into account the scale. If `scale = 2` then $z_A = 2+4i$. In addition, the object referenced by A is stored in table `z` and not za.
The notation may come as a surprise, as I used the term "point". The aim here was not to create a complete library on complex numbers, but to be able to use their main properties in relation to points. I didn't want to have two different levels, and since a unique connection can be established between the points of the plane and the complexes, I decided not to mention the complex numbers! But they are there.

Table 26: Point or complex metamethods.

Metamethods	Application	
__add(z1,z2)	`z.a + z.b`	affix
__sub(z1,z2)	`z.a - z.b`	affix
__unm(z)	`- z.a`	affix
__mul(z1,z2)	`z.a * z.b`	affix
__concat(z1,z2)	`z.a .. z.b`	dot product = real number [a]
__pow(z1,z2)	`z.a ^ z.b`	determinant = real number
__div(z1,z2)	`z.a / z.b`	affix
__tostring(z)	tex.print(tostring(z))	displays the affix
__tonumber(z)	tonumber(z)	affix or nil
__eq(z1,z2)	eq (z.a,z.b)	boolean

a If O is the origin of the complex plan, then we get the dot product of the vectors $\overrightarrow{Oa}$ and $\overrightarrow{Ob}$

Table 27: Point (complex) class methods.

Methods	Application	
`conj(z)`	`z.a : conj()`	affix (conjugate)
`mod(z)`	`z.a : mod()`	real number = modulus `z.a`
`abs (z)`	`z.a : abs()`	real number = modulus
`norm (z)`	`z.a : norm()`	norm (real number)
`arg (z).`	`z.a : arg()`	real number = argument of z.a (in rad)
`get(z)`	`z.a : get()`	re and im (two real numbers)
`sqrt(z).`	`z.a : sqrt()`	affix

The class is provided with two specific metamethods.

- Since concatenation makes little sense here, the operation associated with `..` is the scalar or dot product. If `z1 = a+ib` and `z2 = c+id` then

  `z1..z2 = (a+ib) .. (c+id) = (a+ib) (c-id) = ac+bd + i(bc-ad)`

  There's also a mathematical function, `dot_product`, which takes three arguments. See example 19.8

- With the same idea, the operation associated with `^` is the determinant i.e.

  `z1 ^ z2 = (a+ib) ^ (c+id) = ad - bc From (a-ib) (c+id) = ac+bd + i(ad - bc)` we take the imaginary part.

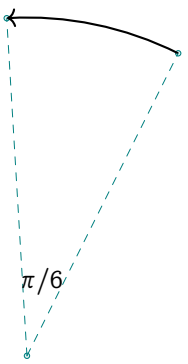
21.3.1 Example of complex use

Let `za = math.cos(a) + i math.sin(a)` . This is obtained from the library by writing

```
za = point(math.cos(a),math.sin(a)).
```

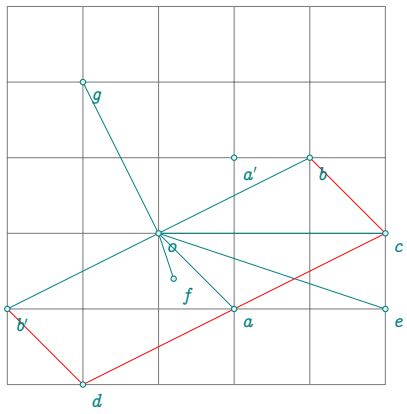
Then `z.B = z.A * za` describes a rotation of point A by an angle `a`.

```
\begin{tkzelements}
   z.O = point : new (0,0)
   z.A = point : new (1,2)
   a = math.pi/6
   za = point(math.cos(a),math.sin(a))
   z.B = z.A * za
\end{tkzelements}
\begin{tikzpicture}
\tkzGetNodes
\tkzDrawPoints(O,A,B)
\tkzDrawArc[->,delta=0](O,A)(B)
\tkzDrawSegments[dashed](O,A O,B)
\tkzLabelAngle(A,O,B){$\pi/6$}
\end{tikzpicture}
```



21.3.2 Point operations(complex)

```
\begin{tkzelements}
   z.o  = point: new(0,0)
   z.a  = point: new(1,-1)
   z.b  = point: new(2,1)
   z.bp = -z.b
   z.c  = z.a + z.b
   z.d  = z.a - z.b
   z.e  = z.a * z.b
    z.f  = z.a / z.b
    z.ap = point.conj (z.a)
    -- = z.a : conj ()
   z.g = z.b* point(math.cos(math.pi/2),
                    math.sin(math.pi/2))
\end{tkzelements}

\hspace*{\fill}
\begin{tikzpicture}
 \tkzGetNodes
 \tkzInit[xmin=-2,xmax=3,ymin=-2,ymax=3]
 \tkzGrid
 \tkzDrawSegments(o,a o,b o,c o,e o,b' o,f o,g)
 \tkzDrawSegments[red](a,c b,c b',d a,d)
 \tkzDrawPoints(a,...,g,o,a',b')
 \tkzLabelPoints(o,a,b,c,d,e,f,g,a',b')
\end{tikzpicture}
```



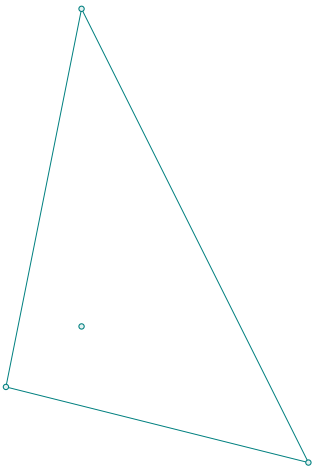
21.4 Barycenter

Here's the definition of the barycenter, which is used some forty times in the package.
`table.pack` builds a table from a list.
`tp.n` gives the number of pairs.
`tp[i][1]` is an affix and `tp[i][2]` the associated weight (real value). 5se the example.

```
function barycenter_ (...)
local tp = table.pack(...)
local i
local sum = 0
local weight=0
for i=1,tp.n do
   sum = sum + tp[i][1]*tp[i][2]
   weight = weight + tp[i][2]
end
return sum/weight
end
```

21.4.1 Using the barycentre

```
\begin{tkzelements}
 z.A =  point: new (1,0)
 z.B =  point: new (5,-1)
 z.C =  point: new (2,5)
 z.G =  barycenter ({z.A,3},{z.B,1},{z.C,1})
\end{tkzelements}

\begin{tikzpicture}
\tkzGetNodes
\tkzDrawPolygon(A,B,C)
\tkzDrawPoints(A,B,C,G)
\end{tikzpicture}
```



21.4.2 Incenter of a triangle

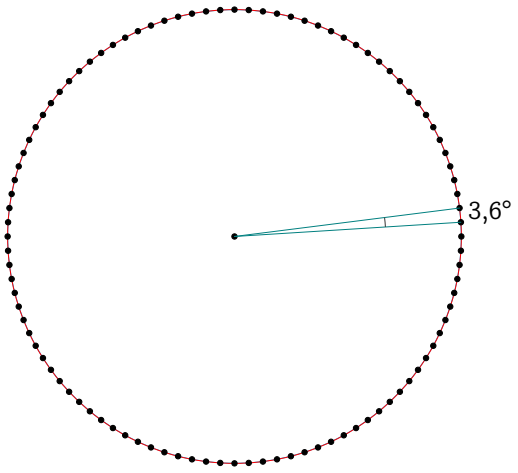
The calculation of the weights ka, kb and kc is precise, and the result obtained with the barycenter is excellent. Note the presence of the underscore _ for certain functions. These functions are internal (developer). Each external (user) function is associated with its internal counterpart.
Here's how to determine the center of the inscribed circle of a triangle:

```
function in_center_ ( a,b,c )
   local ka = point.abs (b-c)
   local kc = point.abs (b-a)
   local kb = point.abs (c-a)
   return    barycenter_ ( {a,ka} , {b,kb} , {c,kc} )
end
```

21.5 Loop and table notation

The problem encountered in this example stems from the notation of the point names. Since it's not possible to write in simplified form, we have to resort to table[key] notation.

```
\begin{tkzelements}
  local r  = 3
  z.O      = point : new (0,0)
  max      = 100
  for i    = 1,max
  do
     z["A_"..i] = point : polar(r,2*i*math.pi/max)
  end
  a = math.deg(get_angle (z.O,z.A_1,z.A_2))
\end{tkzelements}
```



21.6 In_out method

This function can be used for the following objects

- line
- circle
- triangle
- ellipse

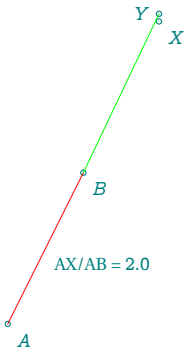
The disk object doesn't exist, so with `in\_out\_disk` it's possible to determine whether a point is in a disk.

21.6.1 In_out for a line

```
function line: in_out (pt)
local sc,epsilon
epsilon = 10^(-12)
sc = math.abs ((pt-self.pa)^(pt-self.pb))
if sc <= epsilon
   then
      return true
   else
      return false
   end
end
```

The `ifthen` package is required for the code below.

```
\begin{tkzelements}
z.A     = point: new (0,0)
z.B     = point: new (1,2)
z.X     = point: new (2,4.000)
z.Y     = point: new (2,4.1)
L.AB = line :  new (z.A,z.B)
if L.AB : in_out (z.X)
  then
   inline = true  k = (z.X-z.A)/(z.B-z.A)
  else
   inline = false
  end
 inline_bis = L.AB : in_out (z.Y)
\end{tkzelements}

\begin{tikzpicture}
\tkzGetNodes
\tkzDrawPoints(A,B,X,Y)
\tkzLabelPoints(A,B,X)
\tkzLabelPoints[left](Y)
\ifthenelse{\equal{\tkzUseLua{inline}}{true}}{
   \tkzDrawSegment[red](A,B)
   \tkzLabelSegment(A,B){AX/AB = $\tkzUseLua{k}$}}{%
   \tkzDrawSegment[blue](A,B)}
\ifthenelse{\equal{\tkzUseLua{inline_bis}}{false}}{%
 \tkzDrawSegment[green](B,Y)}{}
\end{tikzpicture}
```



21.7 Determinant and dot product

21.7.1 Determinant

We've just seen how to use ^ to obtain the determinant associated with two vectors.
`in_out` is simply a copy of `islinear` .
Here's the definition and transformation of the power of a complex number.

```
   -- determinant  is '^'   ad - bc
   function point.__pow(z1,z2)
       local z
       z = point.conj(z1) * z2   -- (a-ib) (c+id) = ac+bd + i(ad - bc)
      return z.im
   end
```

21.7.2 Dot product

Here's the definition of the dot product between two affixes and the concatenation transformation.

```
-- dot product is '..'         result ac + bd
function point.__concat(z1,z2)
    local z
    z = z1 * point.conj(z2)         -- (a+ib) (c-id) = ac+bd + i(bc-ad)
  return z.re
end
```

21.7.3 Dot product: orthogonality test

Here's a function `isortho` to test orthogonality between two vectors.

```
function isortho (z1,z2,z3)
   local epsilon
   local dp
   epsilon = 10^(-8)
   dp = (z2-z1) .. (z3-z1)
   if math.abs(dp) < epsilon
    then
        return true
    else
        return false
    end
end
```

21.7.4 Dot product: projection

The projection of a point onto a straight line is a fundamental function, and its definition is as follows:

```
function projection_ ( pa,pb,pt )
   local v
   local z
   if aligned ( pa,pb,pt ) then
   return pt
   else
    v = pb - pa
    z = ((pt - pa)..v)/(point.norm(v)) -- .. dot product
   return pa + z * v
   end
end
```

The function `aligned` is equivalent to `islinear` but does not use a determinant. It will be replaced in a future version.

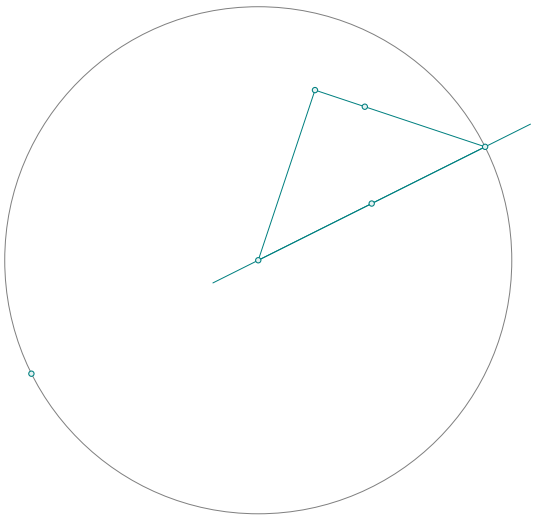
21.8 Point method

The point method is a method for many objects:

- line ,
- circle,
- ellipse,
- triangle.

You obtain a point on the object by entering a real number between 0 and 1.

```
\begin{tkzelements}
   z.A   = point : new ( 0 , 0 )
   z.B   = point : new ( 4 , 2 )
   z.C   = point : new ( 1 , 3 )
   L.AB  = line : new (z.A,z.B)
   C.AB  = circle  : new (z.A,z.B)
   T.ABC = triangle  : new  (z.A,z.B,z.C)
   z.I   = L.AB : point (0.5)
   z.J   = C.AB : point (0.5)
   z.K   = T.ABC : point (0.5)
\end{tkzelements}
\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawLine(A,B)
   \tkzDrawCircle(A,B)
   \tkzDrawPolygon(A,B,C)
   \tkzDrawPoints(A,B,C,I,J,K)
\end{tikzpicture}
```



21.9 Behind the objects

Before introducing objects, I only used functions whose parameters were points (comlexes).
For example, `z.m = midpoint_ (z.a,z.b)` defines the midpoint of points a and b. With objects, first define the line/sgment `L.ab` and then obtain the middle with `z.m = L.ab.mid`.
I've kept the functions (which I'll call "primary") whose only arguments are points. They are distinguished from the others by a terminal underscore. In fact, all (almost) object-related functions depend on a primary function. We've just seen the case of the midpoint of a point, so let's look at two other cases:

– Rotation around a point. `c` is the center of rotation, `a` the angle and `pt` the point to be affected. For example: `z.Mp = rotation (z.A,math.pi/6,z.M)`

```
function rotation_ (c,a,pt)
local z = point( math.cos(a) , math.sin(a) )
return z*(pt-c)+c
end
```

With objects, this gives `z.Mp = z.A : rotation (math.pi/6,z.M)`

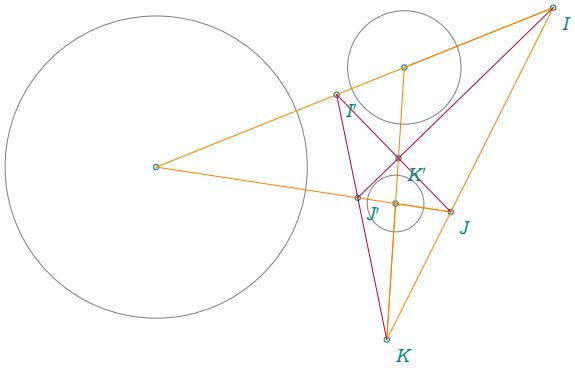
– The intersection of a line and a circle is obtained using `intersection_lc_ (z.A,z.B,z.O,z.T)`. using the straight line (A,B) and the circle $C(O,T)$.

This will result in the objects: `intersection (L.AB,C.OT)`

The difference is that programming is more direct with primary functions and a little more efficient, but loses visibility.

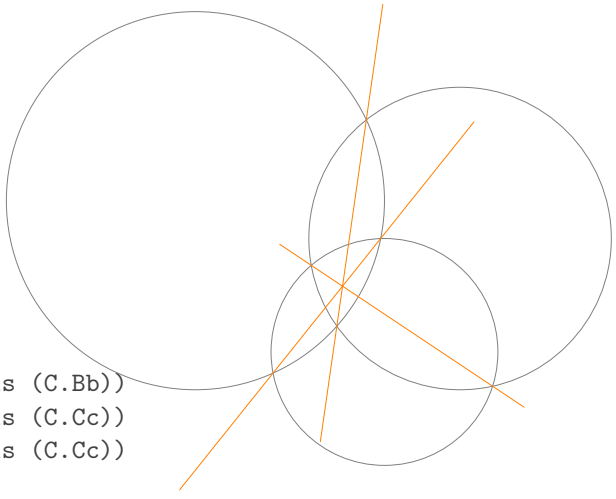
22 Examples

22.1 D'Alembert 1




```
\begin{tkzelements}
   z.A = point : new (0,0)
   z.a = point : new (4,0)
   z.B = point : new (7,-1)
   z.b = point : new (5.5,-1)
   z.C = point : new (5,-4)
   z.c = point : new (4.25,-4)
   C.Aa    = circle :    new (z.A,z.a)
   C.Bb    = circle :    new (z.B,z.b)
   C.Cc    = circle :    new (z.C,z.c)
   z.I = C.Aa : external_similitude (C.Bb)
   z.J = C.Aa : external_similitude (C.Cc)
   z.K = C.Cc : external_similitude (C.Bb)
   z.Ip    = C.Aa : internal_similitude (C.Bb)
   z.Jp    = C.Aa : internal_similitude (C.Cc)
   z.Kp    = C.Cc : internal_similitude (C.Bb)
\end{tkzelements}
\begin{tikzpicture}[rotate=-60]
   \tkzGetNodes
   \tkzDrawCircles(A,a B,b C,c)
   \tkzDrawPoints(A,B,C,I,J,K,I',J',K')
   \tkzDrawSegments[new](I,K A,I A,J B,I B,K C,J C,K)
   \tkzDrawSegments[purple](I,J' I',J I',K)
   \tkzLabelPoints(I,J,K,I',J',K')
\end{tikzpicture}
```

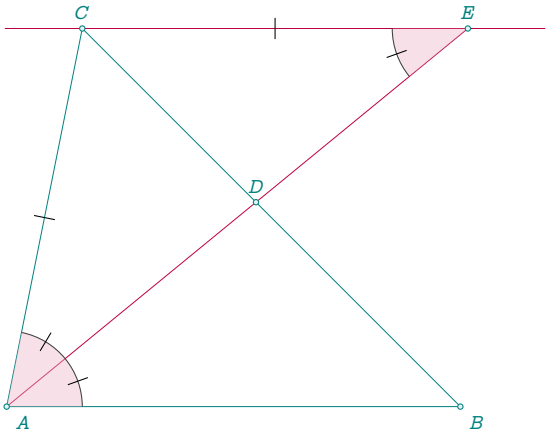
22.2 D'Alembert 2



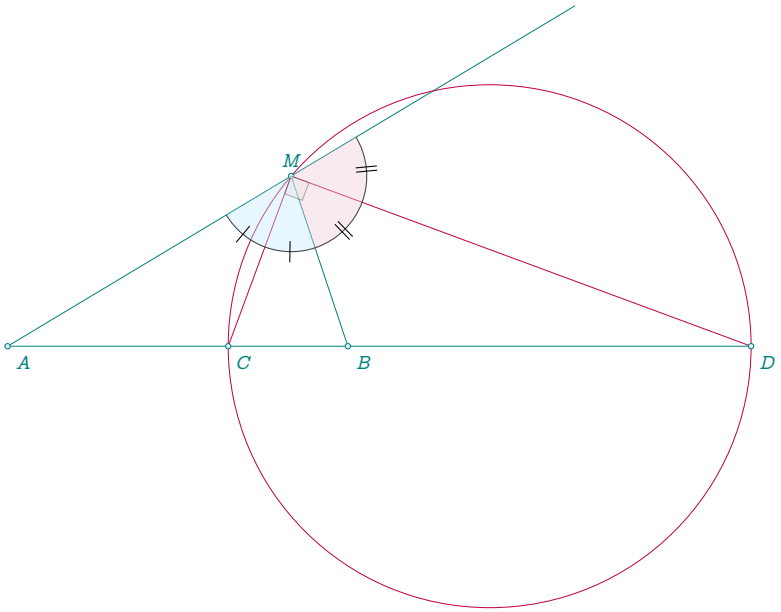
```
\begin{tkzelements}
   scale = .75
   z.A     = point : new (0,0)
   z.a     = point : new (5,0)
   z.B     = point : new (7,-1)
   z.b     = point : new (3,-1)
   z.C     = point : new (5,-4)
   z.c     = point : new (2,-4)
   C.Aa    = circle :    new (z.A,z.a)
   C.Bb    = circle :    new (z.B,z.b)
   C.Cc    = circle :    new (z.C,z.c)
   z.i,z.j = get_points (C.Aa : radical_axis (C.Bb))
   z.k,z.l = get_points (C.Aa : radical_axis (C.Cc))
   z.m,z.n = get_points (C.Bb : radical_axis (C.Cc))
\end{tkzelements}
\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawCircles(A,a B,b C,c)
   \tkzDrawLines[new](i,j k,l m,n)
\end{tikzpicture}
```

22.3 Alternate

```
\begin{tkzelements}
   z.A  = point: new (0 , 0)
   z.B  = point: new (6 , 0)
   z.C  = point: new (1 , 5)
   T    = triangle: new (z.A,z.B,z.C)
   z.I  = T.incenter
   L.AI = line: new (z.A,z.I)
   z.D  = intersection (L.AI,T.bc)
   L.LLC    = T.ab: ll_from (z.C)
   z.E  = intersection (L.AI,L.LLC)
\end{tkzelements}
\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawPolygon(A,B,C)
   \tkzDrawLine[purple](C,E)
   \tkzDrawSegment[purple](A,E)
   \tkzFillAngles[purple!30,opacity=.4](B,A,C C,E,D)
   \tkzMarkAngles[mark=|](B,A,D D,A,C C,E,D)
   \tkzDrawPoints(A,...,E)
   \tkzLabelPoints(A,B)
   \tkzLabelPoints[above](C,D,E)
   \tkzMarkSegments(A,C C,E)
\end{tikzpicture}
```



22.4 Apollonius circle

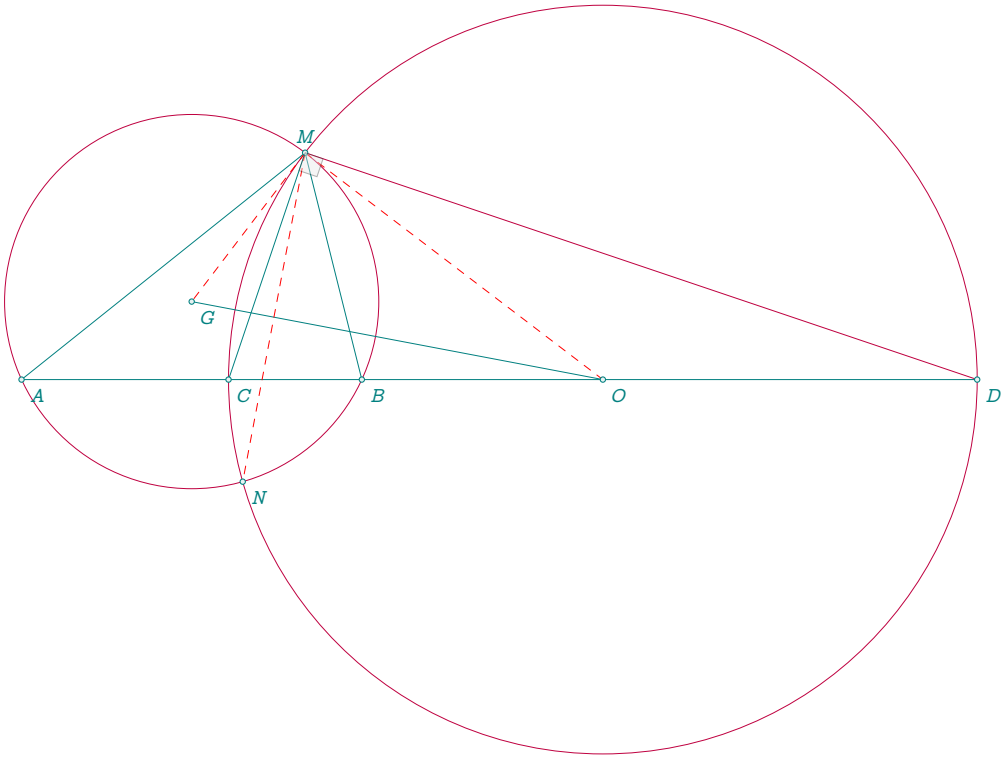


```
\begin{tkzelements}
scale=.75
   z.A      = point: new (0 , 0)
   z.B      = point: new (6 , 0)
   z.M      = point: new (5 , 3)
   T.MAB    = triangle : new (z.M,z.A,z.B)
   L.bis    = T.MAB : bisector ()
   z.C      = L.bis.pb
   L.bisext = T.MAB : bisector_ext ()
   z.D      = intersection (T.MAB.bc, L.bisext)
   L.CD     = line: new (z.C,z.D)
   z.O      = L.CD.mid
   L.AM     = T.MAB.ab
   z.E      = z.M : symmetry (z.A)
\end{tkzelements}
\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawSegment[add=0 and 1](A,M)
   \tkzDrawSegments[purple](M,C M,D)
   \tkzDrawCircle[purple](O,C)
   \tkzDrawSegments(A,B B,M D,B)
   \tkzDrawPoints(A,B,M,C,D)
   \tkzLabelPoints[below right](A,B,C,D)
   \tkzLabelPoints[above](M)
   \tkzFillAngles[opacity=.4,cyan!20](A,M,B)
   \tkzFillAngles[opacity=.4,purple!20](B,M,E)
   \tkzMarkRightAngle[opacity=.4,fill=gray!20](C,M,D)
   \tkzMarkAngles[mark=|](A,M,C C,M,B)
   \tkzMarkAngles[mark=||](B,M,D D,M,E)
\end{tikzpicture}
```

Remark : The circle can be obtained with:
`C.AB = T.MAB.bc : apollonius (length(z.M,z.A)/length(z.M,z.B))`

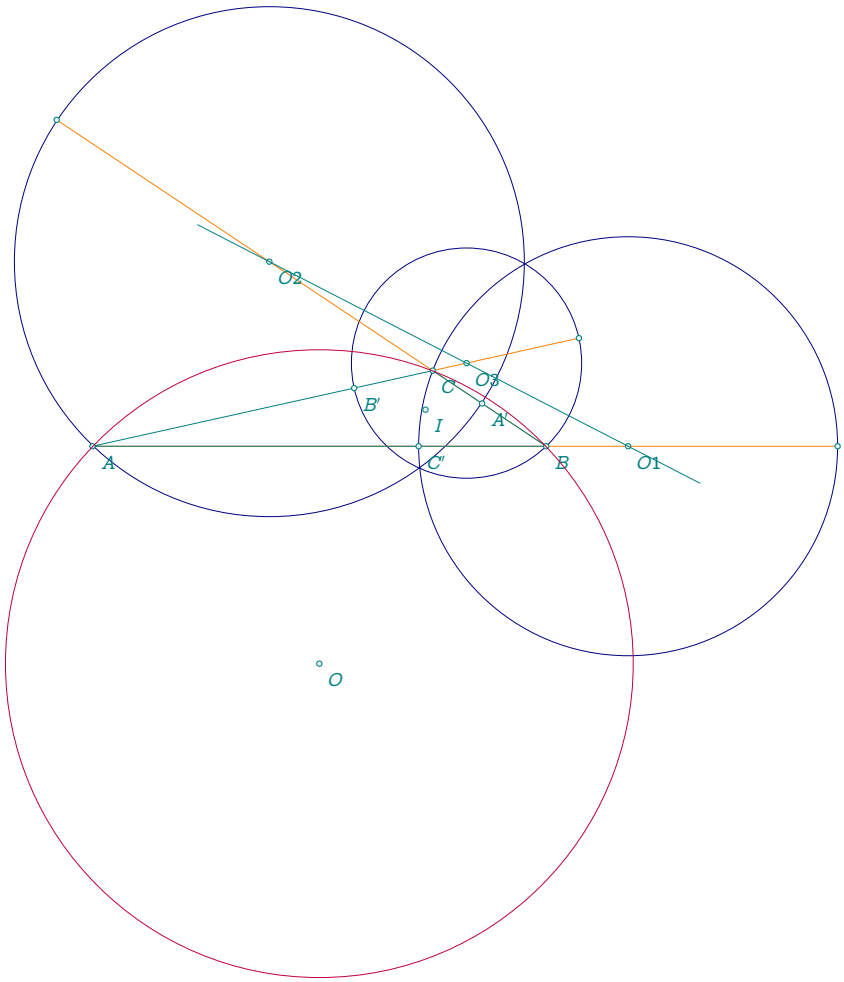
22.5 Apollonius and circle circumscribed

```
\begin{tkzelements}
   scale =.75
   z.A   = point: new (0 , 0)
   z.B   = point: new (6 , 0)
   z.M   = point: new (5 , 4)
   T.AMB = triangle: new (z.A,z.M,z.B)
   L.AB  = T.AMB.ca
   z.I   = T.AMB.incenter
   L.MI  = line: new (z.M,z.I)
   z.C   = intersection (L.AB , L.MI)
   L.MJ  = L.MI: ortho_from (z.M)
   z.D   = intersection (L.AB , L.MJ)
   L.CD  = line: new (z.C,z.D)
   z.O   = L.CD.mid
   z.G   = T.AMB.circumcenter
   C.GA  = circle: new (z.G,z.A)
   C.OC  = circle: new (z.O,z.C)
   _,z.N = intersection (C.GA , C.OC)
\end{tkzelements}
\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawPolygon(A,B,M)
   \tkzDrawCircles[purple](O,C G,A)
   \tkzDrawSegments[purple](M,D)
   \tkzDrawSegments(D,B O,G M,C)
   \tkzDrawSegments[red,dashed](M,N M,O M,G)
   \tkzDrawPoints(A,B,M,C,D,N,O,G)
   \tkzLabelPoints[below right](A,B,C,D,N,O,G)
   \tkzLabelPoints[above](M)
   \tkzMarkRightAngle[opacity=.4,fill=gray!20](C,M,D)
\end{tikzpicture}
```



22.6 Apollonius circles in a triangle

```
\begin{tkzelements}
   z.A   = point: new (0 , 0)
   z.B   = point: new (6 , 0)
   z.C   = point: new (4.5 , 1)
   T.ABC = triangle: new (z.A,z.B,z.C)
   z.I   = T.ABC.incenter
   z.O   = T.ABC.circumcenter
   L.CI  = line: new (z.C,z.I)
   z.Cp  = intersection (T.ABC.ab , L.CI)
   z.x   = L.CI.north_pa
   L.Cx  = line: new (z.C,z.x)
   z.R   = intersection (L.Cx,T.ABC.ab)
   L.CpR = line: new (z.Cp,z.R)
   z.O1  = L.CpR.mid
   L.AI  = line: new (z.A,z.I)
   z.Ap  = intersection (T.ABC.bc , L.AI)
   z.y   = L.AI.north_pa
   L.Ay  = line: new (z.A,z.y)
   z.S   = intersection (L.Ay,T.ABC.bc)
   L.ApS = line: new (z.Ap,z.S)
   z.O2  = L.ApS.mid
   L.BI  = line: new (z.B,z.I)
   z.Bp  = intersection (T.ABC.ca , L.BI)
   z.z   = L.BI.north_pa
   L.Bz  = line: new (z.B,z.z)
   z.T   = intersection (L.Bz,T.ABC.ca)
   L.Bpt = line: new (z.Bp,z.T)
   z.O3  = L.Bpt.mid
\end{tkzelements}
\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawCircles[blue!50!black](O1,C' O2,A' O3,B')
   \tkzDrawSegments[new](B,S C,T A,R)
   \tkzDrawPolygon(A,B,C)
   \tkzDrawPoints(A,B,C,A',B',C',O,I,R,S,T,O1,O2,O3)
   \tkzLabelPoints(A,B,C,A',B',C',O,I)
   \tkzLabelPoints(O1,O2,O3)
   \tkzDrawCircle[purple](O,A)
   \tkzDrawLine(O1,O2)
\end{tikzpicture}
```

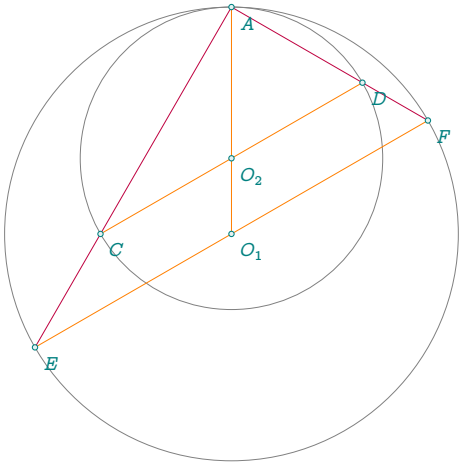


Same result using the function `T.ABC.ab : apollonius (k)`

```
\begin{tkzelements}
scale       = .75
z.A         = point: new (0 , 0)
z.B         = point: new (6 , 0)
z.C         = point: new (4.5 , 1)
T.ABC       = triangle : new (z.A,z.B,z.C)
z.O         = T.ABC.circumcenter
C.AB        = T.ABC.ab : apollonius (length(z.C,z.A)/length(z.C,z.B))
z.w1,z.t1   = get_points ( C.AB )
C.AC        = T.ABC.ca : apollonius (length(z.B,z.C)/length(z.B,z.A))
z.w2,z.t2   = get_points ( C.AC )
C.BC        = T.ABC.bc : apollonius (length(z.A,z.B)/length(z.A,z.C))
z.w3,z.t3   = get_points ( C.BC )
\end{tkzelements}
```

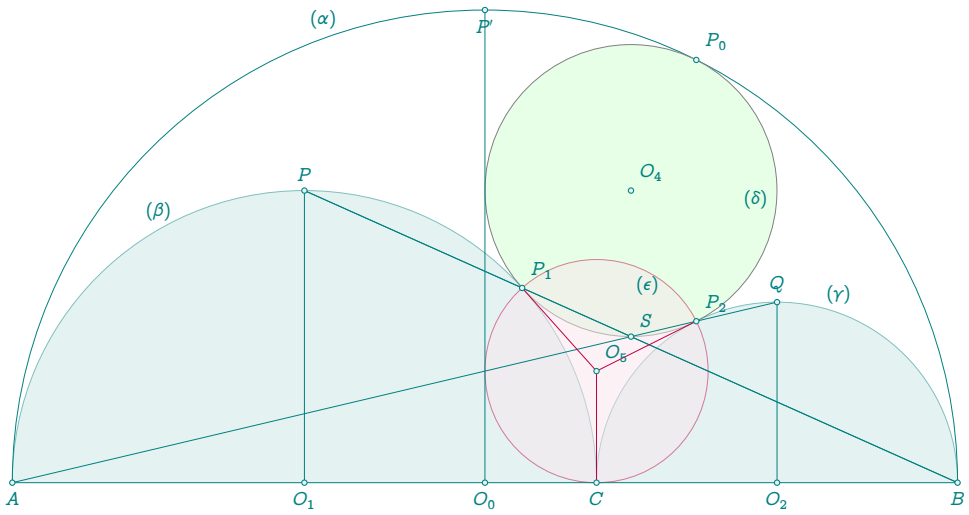
22.7 Archimedes

```
\begin{tkzelements}
   z.O_1    = point:   new  (0, 0)
   z.O_2    = point:   new  (0, 1)
   z.A      = point:   new  (0, 3)
   z.F      = point:   polar (3, math.pi/6)
   L        = line:    new   (z.F,z.O_1)
   C        = circle:  new  (z.O_1,z.A)
   z.E      = intersection (L,C)
   T        = triangle: new (z.F,z.E,z.O_2)
   z.x      = T: parallelogram ()
   L        = line: new    (z.x,z.O_2)
   C        = circle: new  (z.O_2,z.A)
   z.C,z.D  = intersection (L ,C)
\end{tkzelements}
\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawCircles(O_1,A O_2,A)
   \tkzDrawSegments[new](O_1,A E,F C,D)
   \tkzDrawSegments[purple](A,E A,F)
   \tkzDrawPoints(A,O_1,O_2,E,F,C,D)
   \tkzLabelPoints(A,O_1,O_2,E,F,C,D)
\end{tikzpicture}
```



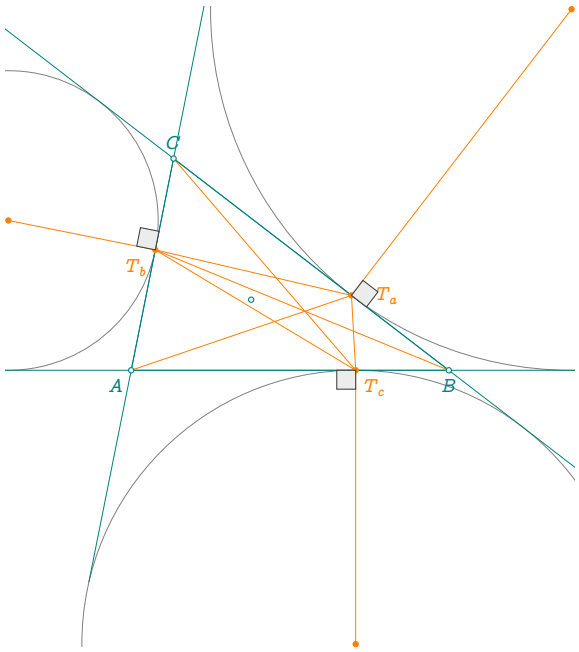
22.8 Bankoff circle

```
\begin{tkzelements}
   z.A      = point: new (0 , 0)
   z.B      = point: new (10 , 0)
   L.AB     = line : new (z.A,z.B)
   z.C      = L.AB: gold_ratio ()
   L.AC     = line : new (z.A,z.C)
   L.CB     = line : new (z.C,z.B)
   z.O_0    = L.AB.mid
   z.O_1    = L.AC.mid
   z.O_2    = L.CB.mid
   C.O0B    = circle : new (z.O_0,z.B)
   C.O1C    = circle : new (z.O_1,z.C)
   C.O2C    = circle : new (z.O_2,z.B)
   z.Pp     = C.O0B : midarc (z.B,z.A)
   z.P      = C.O1C : midarc (z.C,z.A)
   z.Q      = C.O2C : midarc (z.B,z.C)
   L.O1O2   = line : new (z.O_1,z.O_2)
   L.O0O1   = line : new (z.O_0,z.O_1)
   L.O0O2   = line : new (z.O_0,z.O_2)
   z.M_0    = L.O1O2 : harmonic_ext (z.C)
   z.M_1    = L.O0O1 : harmonic_int (z.A)
   z.M_2    = L.O0O2 : harmonic_int (z.B)
   L.BP     = line : new (z.B,z.P)
   L.AQ     = line : new (z.A,z.Q)
   z.S      = intersection (L.BP,L.AQ)
   L.PpO0   = line : new (z.Pp,z.O_0)
   L.PC     = line : new (z.P,z.C)
   z.Ap     = intersection (L.PpO0,L.PC)
   L.CS     = line : new (z.C,z.S)
   C.M1A    = circle : new (z.M_1,z.A)
   C.M2B    = circle : new (z.M_2,z.B)
   z.P_0    = intersection (L.CS,C.O0B)
   z.P_1    = intersection (C.M2B,C.O1C)
   z.P_2    = intersection (C.M1A,C.O2C)
   T.P0P1P2 = triangle : new (z.P_0,z.P_1,z.P_2)
   z.O_4    = T.P0P1P2.circumcenter
   T.CP1P2  = triangle : new (z.C,z.P_1,z.P_2)
   z.O_5    = T.CP1P2.circumcenter
\end{tkzelements}
\begin{tikzpicture}
\tkzGetNodes
\tkzDrawSemiCircles[teal](O_0,B)
\tkzDrawSemiCircles[teal,fill=teal!20,opacity=.5](O_1,C O_2,B)
\tkzDrawCircle[fill=green!10](O_4,P_0)
\tkzDrawCircle[purple,fill=purple!10,opacity=.5](O_5,C)
\tkzDrawSegments(A,B O_0,P' B,P A,Q)
\tkzDrawSegments(P,B Q,O_2 P,O_1)
\tkzDrawSegments[purple](O_5,P_2 O_5,P_1 O_5,C)
\tkzDrawPoints(A,B,C,P_0,P_2,P_1,O_0,O_1,O_2,O_4,O_5,Q,P,P',S)
\tkzLabelPoints[below](A,B,C,O_0,O_1,O_2,P')
\tkzLabelPoints[above](Q,P)
\tkzLabelPoints[above right](P_0,P_2,P_1,O_5,O_4,S)
\begin{scope}[font=\scriptsize]
  \tkzLabelCircle[above](O_1,C)(120){$(\beta)$}
  \tkzLabelCircle[above](O_2,B)(70){$(\gamma)$}
  \tkzLabelCircle[above](O_0,B)(110){$(\alpha)$}
  \tkzLabelCircle[left](O_4,P_2)(60){$(\delta)$}
  \tkzLabelCircle[left](O_5,C)(140){$(\epsilon)$}
\end{scope}
\end{tikzpicture}
```



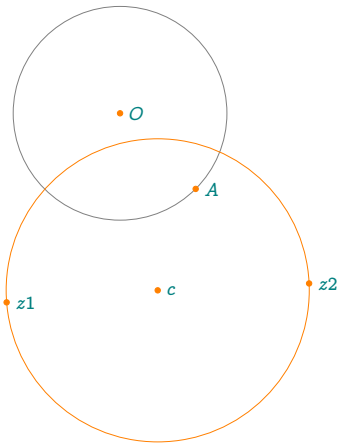
22.9 Excircles

```
\begin{tkzelements}
   scale                = 0.7
   z.A                  = point: new (0,0)
   z.B                  = point: new (6,0)
   z.C                  = point: new (.8,4)
   T                    = triangle: new ( z.A, z.B, z.C)
   z.K                  = T.centroid
   z.J_a,z.J_b,z.J_c    = get_points (T: excentral())
   z.T_a,z.T_b,z.T_c    = get_points (T: extouch())
   la                   = line: new ( z.A, z.T_a)
   lb                   = line: new ( z.B, z.T_b)
   z.G                  = intersection (la,lb)
\end{tkzelements}
\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawPoints[new](J_a,J_b,J_c)
   \tkzClipBB
   \tkzDrawCircles[gray](J_a,T_a J_b,T_b J_c,T_c)
   \tkzDrawLines[add=1 and 1](A,B B,C C,A)
   \tkzDrawSegments[new](A,T_a B,T_b C,T_c)
   \tkzDrawSegments[new](J_a,T_a J_b,T_b J_c,T_c)
   \tkzDrawPolygon(A,B,C)
   \tkzDrawPolygon[new](T_a,T_b,T_c)
   \tkzDrawPoints(A,B,C,K)
   \tkzDrawPoints[new](T_a,T_b,T_c)
   \tkzLabelPoints[below left](A)
   \tkzLabelPoints[below](B)
   \tkzLabelPoints[above](C)
   \tkzLabelPoints[new,below left](T_b)
   \tkzLabelPoints[new,below right](T_c)
   \tkzLabelPoints[new,right=6pt](T_a)
   \tkzMarkRightAngles[fill=gray!15](J_a,T_a,B J_b,T_b,C J_c,T_c,A)
\end{tikzpicture}
```



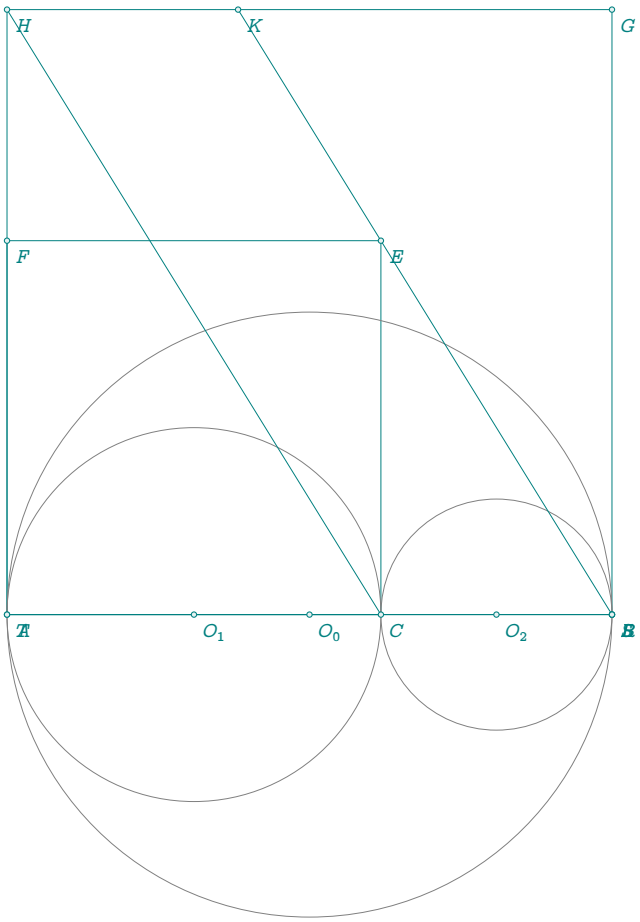
22.10 Orthogonal circle through

```
\begin{tkzelements}
   z.O   = point: new (0,1)
   z.A   = point: new (1,0)
   z.z1  = point: new (-1.5,-1.5)
   z.z2  = point: new (2.5,-1.25)
   C.OA  = circle: new (z.O,z.A)
   C     = C.OA: orthogonal_through (z.z1,z.z2)
   z.c   = C.center
\end{tkzelements}
\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawCircle(O,A)
   \tkzDrawCircle[new](c,z1)
   \tkzDrawPoints[new](O,A,z1,z2,c)
   \tkzLabelPoints[right](O,A,z1,z2,c)
\end{tikzpicture}
```



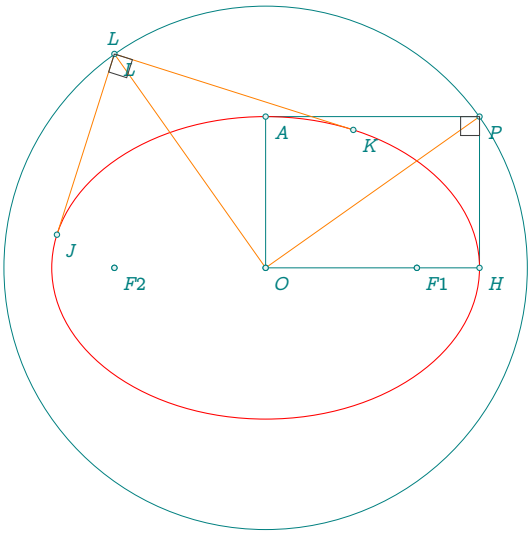
22.11 Divine ratio

```
\begin{tkzelements}
z.A         = point: new (0 , 0)
z.B         = point: new (8 , 0)
L.AB        = line: new (z.A,z.B)
z.C         = L.AB: gold_ratio ()
L.AC        = line: new (z.A,z.C)
z.O_1       = L.AC.mid
_,_,z.G,z.H = get_points(L.AB: square ())
_,_,z.E,z.F = get_points(L.AC: square ())
L.CB        = line: new (z.C,z.B)
z.O_2       = L.CB.mid
z.O_0       = L.AB.mid
L.BE        = line: new (z.B,z.E)
L.GH        = line: new (z.G,z.H)
z.K         = intersection (L.BE,L.GH)
C0          = circle: new (z.O_0,z.B)
z.R,_       = intersection (L.BE,C0)
C2          = circle: new (z.O_2,z.B)
z.S,_       = intersection (L.BE,C2)
L.AR        = line:   new (z.A,z.R)
C1          = circle: new (z.O_1,z.C)
_,z.T       = intersection (L.AR,C1)
L.BG        = line: new (z.B,z.G)
z.L         = intersection (L.AR,L.BG)
\end{tkzelements}
\begin{tikzpicture}
\tkzGetNodes
\tkzDrawPolygons(A,C,E,F A,B,G,H)
\tkzDrawCircles(O_1,C O_2,B O_0,B)
\tkzDrawSegments(H,C B,K A,L)
\tkzDrawPoints(A,B,C,K,E,F,G,H,O_0,O_1,O_2,R,S,T,L)
\tkzLabelPoints(A,B,C,K,E,F,G,H,O_0,O_1,O_2,R,S,T,L)
\end{tikzpicture}
```


H
K
G
F
E
A
O_1
O_0
C
O_2
B

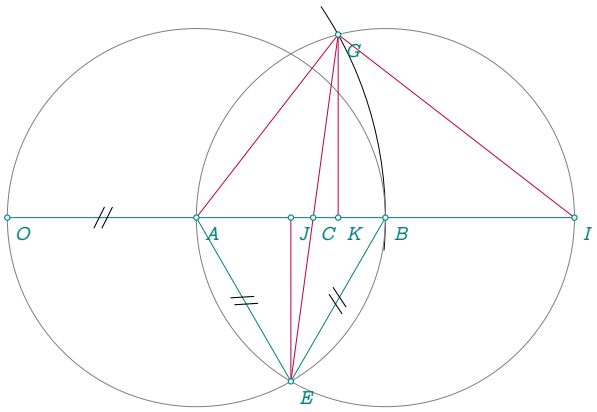
22.12 Director circle

```
\begin{tkzelements}
  scale    = .5
  z.O      = point: new (0 , 0)
  z.F1     = point: new (4 , 0)
  z.F2     = point: new (-4 , 0)
  z.H      = point: new (4*math.sqrt(2) , 0)
  E        = ellipse: foci (z.F2,z.F1,z.H)
  a,b      = E.Rx, E.Ry
  z.A      = E.covertex
  T        = triangle: new (z.H,z.O,z.A)
  z.P      = T: parallelogram ()
  C        = circle: new (z.O,z.P)
  z.L      = C: point (0.25)
  L.J,L.K  = E: tangent_from (z.L)
  z.J      = L.J.pb
  z.K      = L.K.pb
\end{tkzelements}
\begin{tikzpicture}
  \tkzGetNodes
  \tkzDrawPoints(F1,F2,O)
  \tkzDrawCircles[teal](O,P)
  \tkzDrawPolygon(H,O,A,P)
  \tkzDrawEllipse[red](O,\tkzUseLua{a},\tkzUseLua{b},0)
  \tkzDrawSegments[orange](O,P O,L L,J L,K)
  \tkzDrawPoints(F1,F2,O,H,A,P,L,J,K)
  \tkzLabelPoints(F1,F2,O,H,A,P,L,J,K)
  \tkzLabelPoints[above](L)
  \tkzMarkRightAngles(A,P,H J,L,K)
\end{tikzpicture}
```



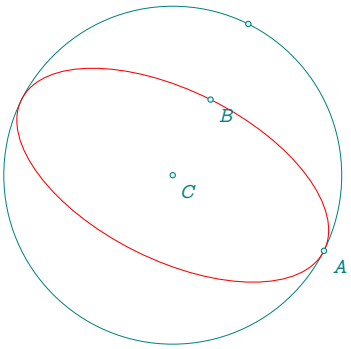
22.13 Gold division

```
\begin{tkzelements}
z.A          = point:  new (0,0)
z.B          = point:  new (2.5,0)
L.AB         = line:   new (z.A,z.B)
C.AB         = circle: new (z.A,z.B)
C.BA         = circle: new (z.B,z.A)
z.J          = L.AB: midpoint ()
L.JB         = line:new (z.J,z.B)
z.F,z.E      = intersection (C.AB , C.BA)
z.I,_        = intersection (L.AB , C.BA)
z.K          = L.JB : midpoint ()
L.mediator   = L.JB: mediator ()
z.G          = intersection (L.mediator,C.BA)
L.EG         = line:new (z.E,z.G)
z.C          = intersection  (L.EG,L.AB)
z.O          = C.AB: antipode (z.B)
\end{tkzelements}
   \begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawArc[delta=5](O,B)(G)
   \tkzDrawCircles(A,B B,A)
   \tkzDrawSegments(A,E B,E O,I)
   \tkzDrawSegments[purple](J,E A,G G,I K,G E,G)
   \tkzMarkSegments[mark=s||](A,E B,E O,A)
   \tkzDrawPoints(A,B,C,E,I,J,G,O,K)
   \tkzLabelPoints(A,B,C,E,I,J,G,O,K)
   \end{tikzpicture}
```



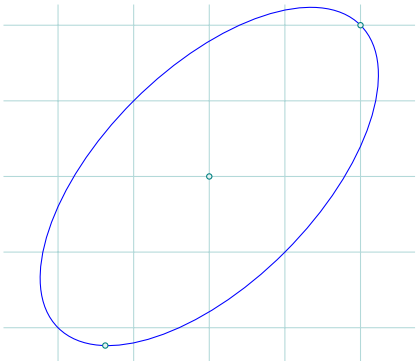
22.14 Ellipse

```
\begin{tkzelements}
   z.C       = point: new (3 , 2)
   z.A       = point: new (5 , 1)
   L.CA      = line : new (z.C,z.A)
   z.b       = L.CA.north_pa
   L         = line : new (z.C,z.b)
   z.B       = L : point (0.5)
   E         = ellipse: new (z.C,z.A,z.B)
   a         = E.Rx
   b         = E.Ry
   slope     = math.deg(E.slope)
\end{tkzelements}
\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawCircles[teal](C,A)
   \tkzDrawEllipse[red](C,\tkzUseLua{a},\tkzUseLua{b},\tkzUseLua{slope})
   \tkzDrawPoints(C,A,B,b)
   \tkzLabelPoints(C,A,B)
\end{tikzpicture}
```



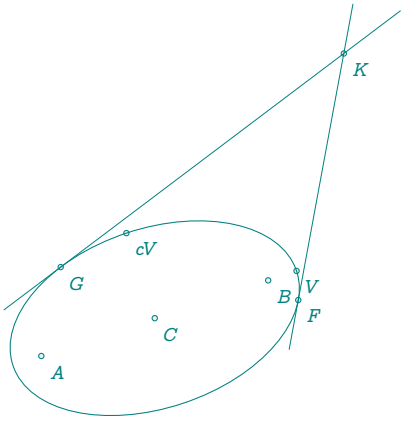
22.15 Ellipse with radii

```
\begin{tkzelements}
scale=.5
z.C    = point: new (0 , 4)
b      = value(math.sqrt(8))
a      = value(math.sqrt(32))
ang    = math.deg(math.pi/4)
E      = ellipse: radii (z.C,a,b,math.pi/4)
z.V    = E : point (0)
z.CoV  = E : point (math.pi/2)
\end{tkzelements}
\begin{tikzpicture}[gridded]
\tkzGetNodes
\tkzDrawEllipse[blue](C,\tkzUseLua{a},
             \tkzUseLua{b},\tkzUseLua{ang})
\tkzDrawPoints(C,V,CoV)
\end{tikzpicture}
```



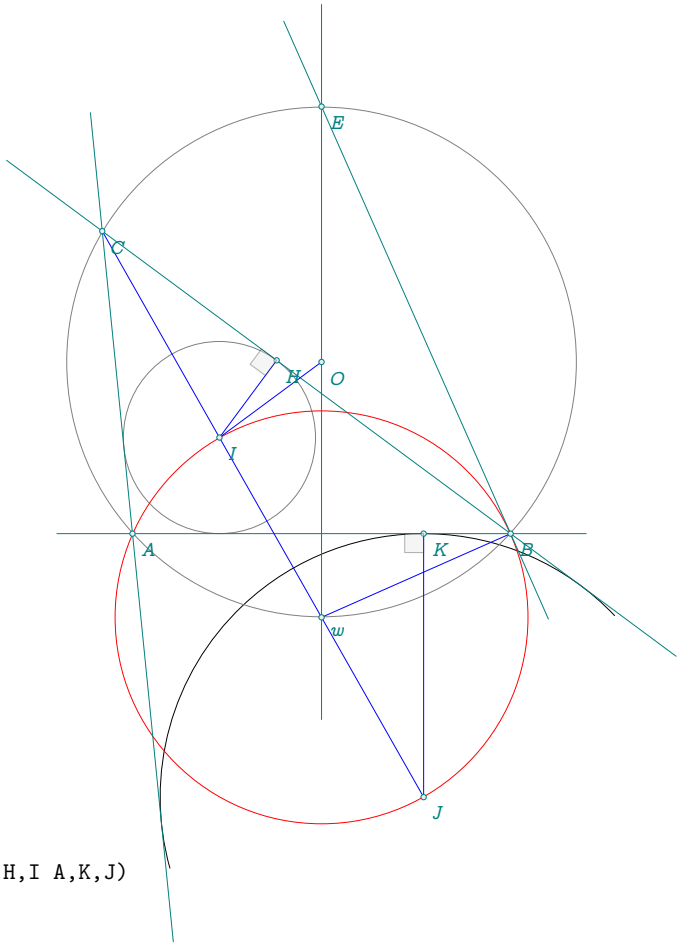
22.16 Ellipse_with_foci

```
\begin{tkzelements}
   local e
   e           = .8
   z.A         = point: new (2 , 3)
   z.B         = point: new (5 , 4)
   z.K         = point: new (6, 7)
   L.AB        = line: new (z.A,z.B)
   z.C         = L.AB.mid
   c           = point.abs(z.B-z.C)
   a           = c/e
   b           = math.sqrt (a^2-c^2)
   z.V         = z.C + a*(z.B-
z.C)/point.abs(z.B-z.C)
   E           = ellipse: foci (z.A,z.B,z.V)
   z.cV        = E.covertex
   ang         = math.deg(E.slope)
   L.ta,L.tb   = E: tangent_from (z.K)
   z.F         = L.ta.pb
   z.G         = L.tb.pb
\end{tkzelements}
\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawPoints(A,B,C,K,F,G,V,cV)
   \tkzLabelPoints(A,B,C,K,F,G,V,cV)
   \tkzDrawEllipse[teal](C,\tkzUseLua{a},\tkzUseLua{b},\tkzUseLua{ang})
   \tkzDrawLines(K,F K,G)
\end{tikzpicture}
```

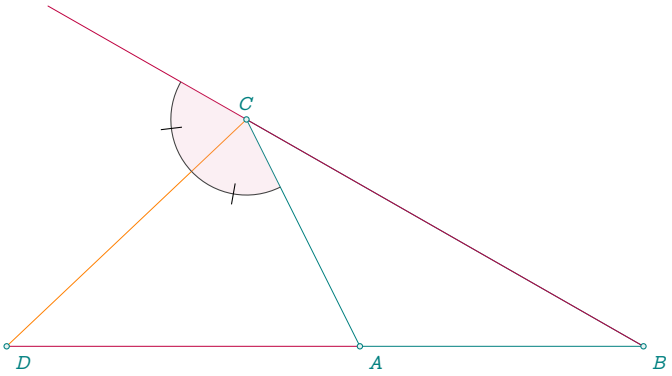


22.17 Euler relation

```
\begin{tkzelements}
 scale      =  .75
 z.A        = point: new (0 , 0)
 z.B        = point: new (5 , 0)
 z.C        = point: new (-.4 , 4)
 T.ABC      = triangle: new (z.A,z.B,z.C)
 z.J,z.K    = get_points(T.ABC: ex_circle (2))
 z.X,z.Y,z.K= T.ABC : projection (z.J)
 z.I,z.H    = get_points(T.ABC : in_circle())
 z.O        = T.ABC.circumcenter
 C.OA       = circle : new (z.O,z.A)
 T.IBA      = triangle: new (z.I,z.B,z.A)
 z.w        = T.IBA.circumcenter
 L.Ow       = line : new (z.O,z.w)
 _,z.E      = intersection (L.Ow, C.OA)
\end{tkzelements}
   \begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawArc(J,X)(Y)
   \tkzDrawCircles(I,H O,A)
   \tkzDrawCircle[red](w,I)
   \tkzDrawLines(Y,C A,B X,C E,w E,B)
   \tkzDrawSegments[blue](J,C J,K I,H I,O w,B)
   \tkzDrawPoints(A,B,C,I,J,E,w,H,K,O)
   \tkzLabelPoints(A,B,C,J,I,w,H,K,E,O)
   \tkzMarkRightAngles[fill=gray!20,opacity=.4](C,H,I A,K,J)
   \end{tikzpicture}
```

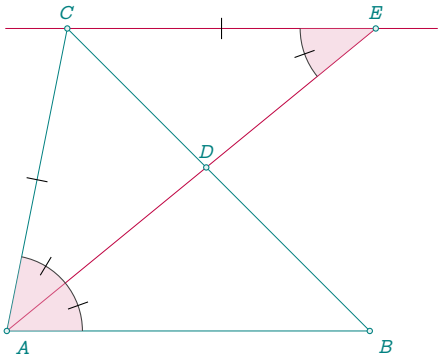


22.18 External angle




```
\begin{tkzelements}
  scale  = .75
  z.A    = point: new (0 , 0)
  z.B    = point: new (5 , 0)
  z.C    = point: new (-2 , 4)
  T.ABC  = triangle: new (z.A,z.B,z.C)
  T.ext  = T.ABC: excentral ()
  z.O    = T.ABC.circumcenter
  z.D    = intersection (T.ext.ab,T.ABC.ab)
  z.E    = z.C: symmetry (z.B)
\end{tkzelements}
\begin{tikzpicture}
  \tkzGetNodes
  \tkzDrawPolygon(A,B,C)
  \tkzDrawLine[purple,add=0 and .5](B,C)
  \tkzDrawSegment[purple](A,D)
  \tkzDrawSegment[orange](C,D)
  \tkzFillAngles[purple!30,opacity=.2](D,C,A E,C,D)
  \tkzMarkAngles[mark=|](D,C,A E,C,D)
  \tkzDrawPoints(A,...,D)
  \tkzLabelPoints[above](C)
  \tkzLabelPoints(A,B,D)
\end{tikzpicture}
```

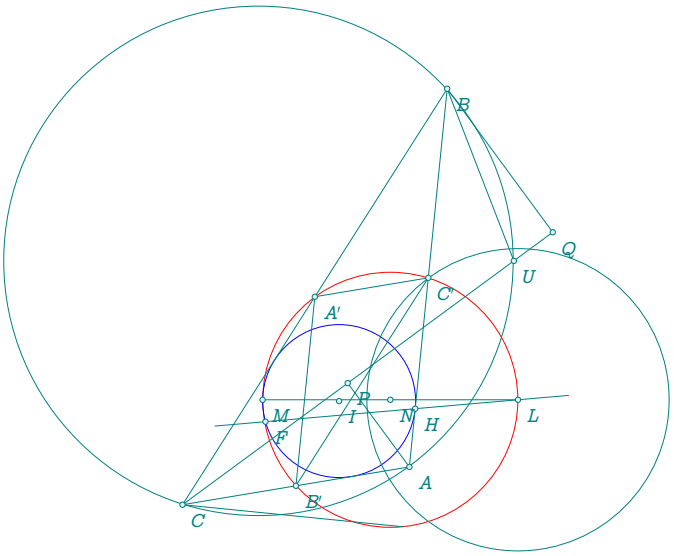
22.19 Internal angle




```
\begin{tkzelements}
   scale =  .8
   z.A   = point: new (0 , 0)
   z.B   = point: new (6 , 0)
   z.C   = point: new (1 , 5)
   T     = triangle: new (z.A,z.B,z.C)
   z.I   = T.incenter
   L.AI  = line: new (z.A,z.I)
   z.D   = intersection (L.AI, T.bc)
   L.LL  = T.ab: ll_from (z.C)
   L.AD  = line: new (z.A,z.D)
   z.E   = intersection (L.LL,L.AD)
\end{tkzelements}
\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawPolygon(A,B,C)
   \tkzDrawLine[purple](C,E)
   \tkzDrawSegment[purple](A,E)
   \tkzFillAngles[purple!30,opacity=.4](B,A,C C,E,D)
   \tkzMarkAngles[mark=|](B,A,D D,A,C C,E,D)
   \tkzDrawPoints(A,...,E)
   \tkzLabelPoints(A,B)
   \tkzLabelPoints[above](C,D,E)
   \tkzMarkSegments(A,C C,E)
\end{tikzpicture}
```

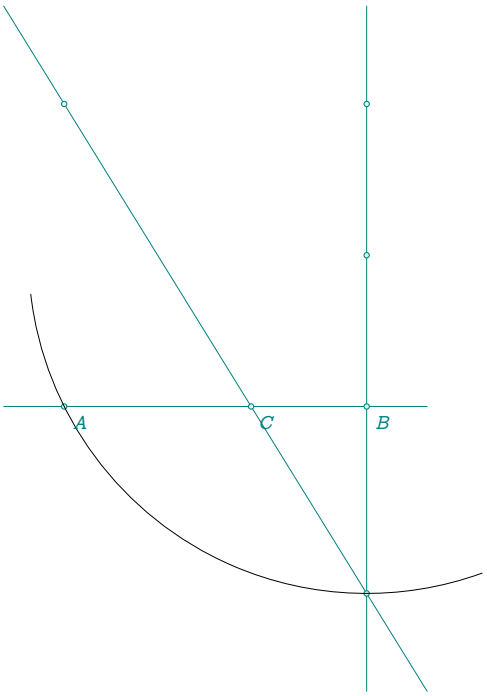
22.20 Feuerbach theorem

```
\begin{tkzelements}
   scale     =  1.5
   z.A       = point: new (0 , 0)
   z.B       = point: new (5 , -.5)
   z.C       = point: new (-.5 , 3)
   T.ABC     = triangle: new (z.A,z.B,z.C)
   z.O       = T.ABC.circumcenter
   z.N       = T.ABC.eulercenter
   z.I,z.K   = get_points(T.ABC: in_circle())
   z.H       = T.ABC.ab : projection (z.I)
   z.Ap,
   z.Bp,
   z.Cp      = get_points (T.ABC : medial ())
   C.IH      = circle:new (z.I,z.H)
   C.NAp     = circle:new (z.N,z.Ap)
   C.OA      = circle:new (z.O,z.A)
   z.U       = C.OA.south
   z.L       = C.NAp.south
   z.M       = C.NAp.north
   z.X       = T.ABC.ab: projection (z.C)
   L.CU      = line: new (z.C,z.U)
   L.ML      = line: new (z.M,z.L)
   z.P       = L.CU: projection (z.A)
   z.Q       = L.CU: projection (z.B)
   L.LH      = line: new (z.L,z.H)
   z.F       = intersection (L.LH,C.IH) -- feuerbach
\end{tkzelements}

\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawLine(L,F)
   \tkzDrawCircle[red](N,A')
   \tkzDrawCircle[blue](I,H)
   \tkzDrawCircles[teal](O,A L,C')
   \tkzDrawSegments(M,L B,U Q,C C,X A,P B,Q)
   \tkzDrawPolygons(A,B,C A',B',C')
   \tkzDrawPoints(A,B,C,N,H,A',B',C',U,L,M,P,Q,F,I)
   \tkzLabelPoints(A,B,C,N,H,A',B',C',U,L,M,P,Q,F,I)
\end{tikzpicture}
```



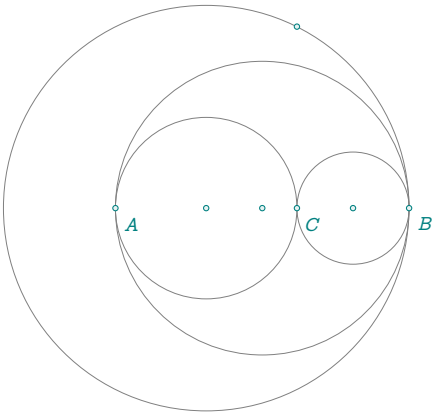
22.21 Gold ratio with segment

```
\begin{tkzelements}
   z.A      = point: new (0 , 0)
   z.B      = point: new (8 , 0)
   L.AB     = line: new (z.A,z.B)
   _,_,z.X,z.Y = get_points(L.AB: square ())
   L.BX     = line: new (z.B,z.X)
   z.M      = L.BX.mid
   C.MA     = circle: new (z.M,z.A)
   _,z.K    = intersection (L.BX,C.MA)
   L.AK     = line: new (z.Y,z.K)
   z.C      = intersection (L.AK,L.AB)
\end{tkzelements}
\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawLines(A,B X,K)
   \tkzDrawLine[teal](Y,K)
   \tkzDrawPoints(A,B,C,X,Y,M,K)
   \tkzDrawArc[delta=20](M,A)(K)
   \tkzLabelPoints(A,B,C)
\end{tikzpicture}
```

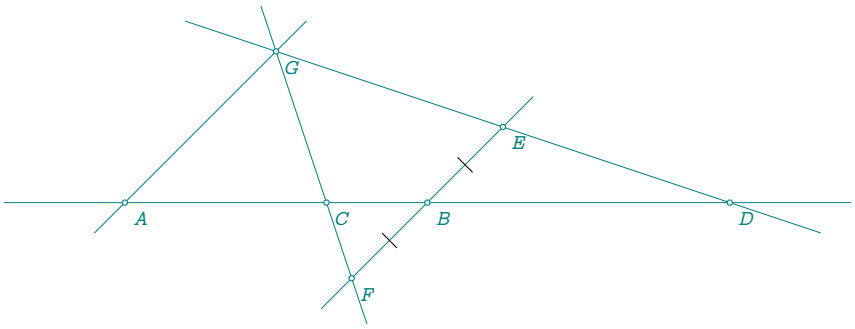


22.22 Gold Arbelos

```
\begin{tkzelements}
   scale    = .6
   z.A      = point: new (0 , 0)
   z.C      = point: new (6 , 0)
   L.AC     = line: new (z.A,z.C)
   _,_,z.x,z.y  = get_points(L.AC: square ())
   z.O_1    = L.AC . mid
   C        = circle: new (z.O_1,z.x)
   z.B      = intersection (L.AC,C)
   L.CB     = line: new (z.C,z.B)
   z.O_2    = L.CB.mid
   L.AB     = line: new (z.A,z.B)
   z.O_0    = L.AB.mid
\end{tkzelements}
\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawCircles(O_1,C O_2,B O_0,B)
   \tkzDrawPoints(A,C,B,O_1,O_2,O_0)
   \tkzLabelPoints(A,C,B)
\end{tikzpicture}
```

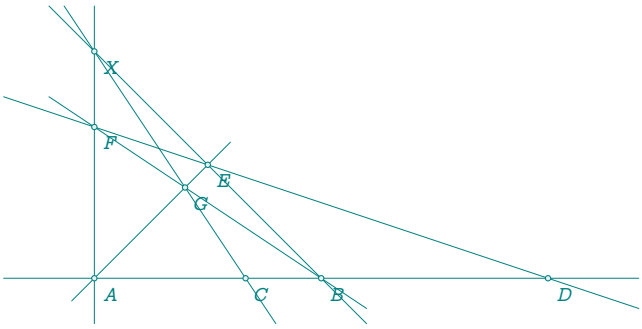


22.23 Harmonic division v1



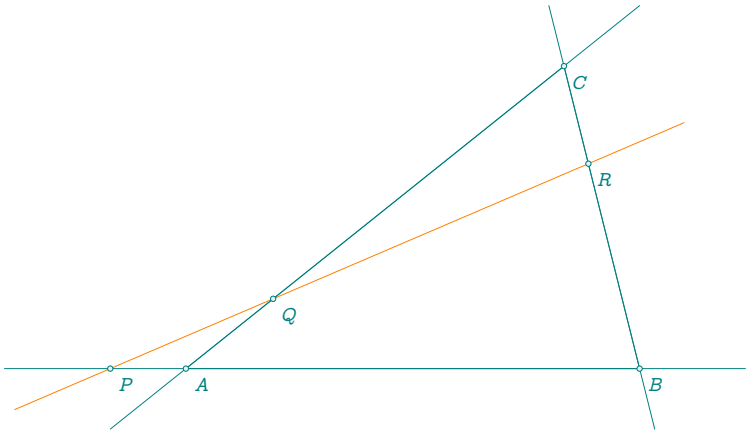

```
\begin{tkzelements}
   scale=.75
   z.A  = point: new (0 , 0)
   z.B  = point: new (4 , 0)
   z.D  = point: new (12,0)
   L.AB = line : new (z.A,z.B)
   z.X  = L.AB.north_pa
   L.XB = line : new (z.X,z.B)
   z.E  = L.XB.mid
   L.DE = line : new (z.D,z.E)
   L.XA = line : new (z.X,z.A)
   z.F  = intersection (L.DE,L.XA)
   L.AE = line : new (z.A,z.E)
   L.BF = line : new (z.B,z.F)
   z.G  = intersection (L.AE,L.BF)
   L.XG = line : new (z.X,z.G)
z.C  = intersection (L.XG,L.AB)
\end{tkzelements}
\begin{tikzpicture}
   \tkzGetNodes
   \tkzDefPoints{0/0/A,4/0/B}
   \tkzDefPoints{2/2/G}
   \tkzDefLine[parallel=through B,K=.5](A,G) \tkzGetPoint{E}
   \tkzInterLL(G,E)(A,B)   \tkzGetPoint{D}
   \tkzDefPointBy[symmetry= center B](E) \tkzGetPoint{F}
   \tkzInterLL(G,F)(A,B)   \tkzGetPoint{C}
   \tkzDrawLines(A,D A,G F,E G,F G,D)
   \tkzDrawPoints(A,B,G,E,F,C,D)
   \tkzLabelPoints(A,B,G,E,F,C,D)
   \tkzMarkSegments(F,B B,E)
\end{tikzpicture}
```

22.24 Harmonic division v2




```
\begin{tkzelements}
   scale    = .5
   z.A      = point: new (0 , 0)
   z.B      = point: new (6 , 0)
   z.D      = point: new (12 , 0)
   L.AB     = line: new (z.A,z.B)
   z.X      = L.AB.north_pa
   L.XB     = line: new (z.X,z.B)
   z.E      = L.XB.mid
   L.ED     = line: new (z.E,z.D)
   L.AX     = line: new (z.A,z.X)
   L.AE     = line: new (z.A,z.E)
   z.F      = intersection (L.ED,L.AX)
   L.BF     = line: new (z.B,z.F)
   z.G      = intersection (L.AE,L.BF)
   L.GX     = line: new (z.G,z.X)
   z.C      = intersection (L.GX,L.AB)
\end{tkzelements}
\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawLines(A,D A,E B,F D,F X,A X,B X,C)
   \tkzDrawPoints(A,...,G,X)
   \tkzLabelPoints(A,...,G,X)
\end{tikzpicture}
```

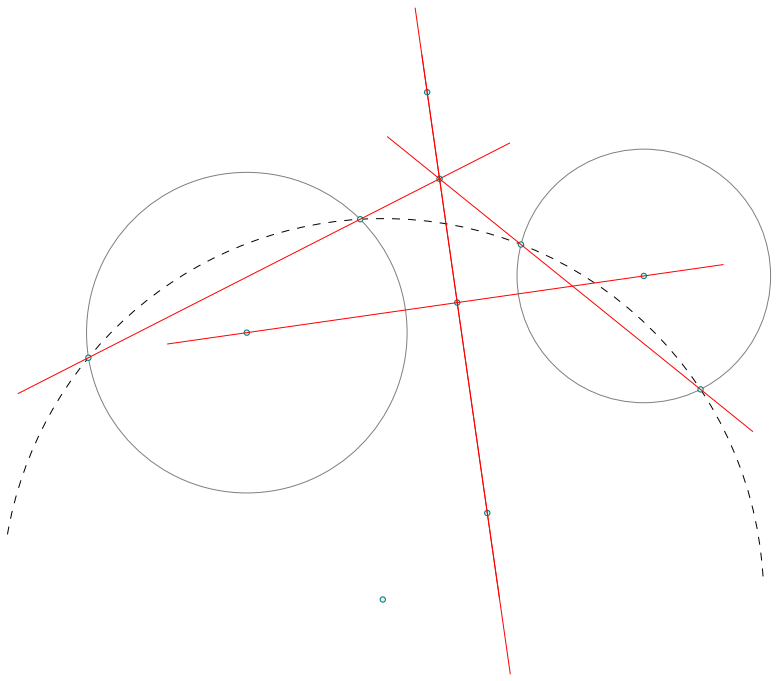
22.25 Menelaus




```
\begin{tkzelements}
   z.A  = point: new (0 , 0)
   z.B  = point: new (6 , 0)
   z.C  = point: new (5 , 4)
   z.P  = point: new (-1 , 0)
   z.X  = point: new (6 , 3)
   L.AC = line: new (z.A,z.C)
   L.PX = line: new (z.P,z.X)
   L.BC = line: new (z.B,z.C)
   z.Q  = intersection (L.AC,L.PX)
   z.R  = intersection (L.BC,L.PX)
\end{tkzelements}
\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawPolygon(A,B,C)
   \tkzDrawLine[new](P,R)
   \tkzDrawLines(P,B A,C B,C)
   \tkzDrawPoints(P,Q,R,A,B,C)
   \tkzLabelPoints(A,B,C,P,Q,R)
\end{tikzpicture}
```

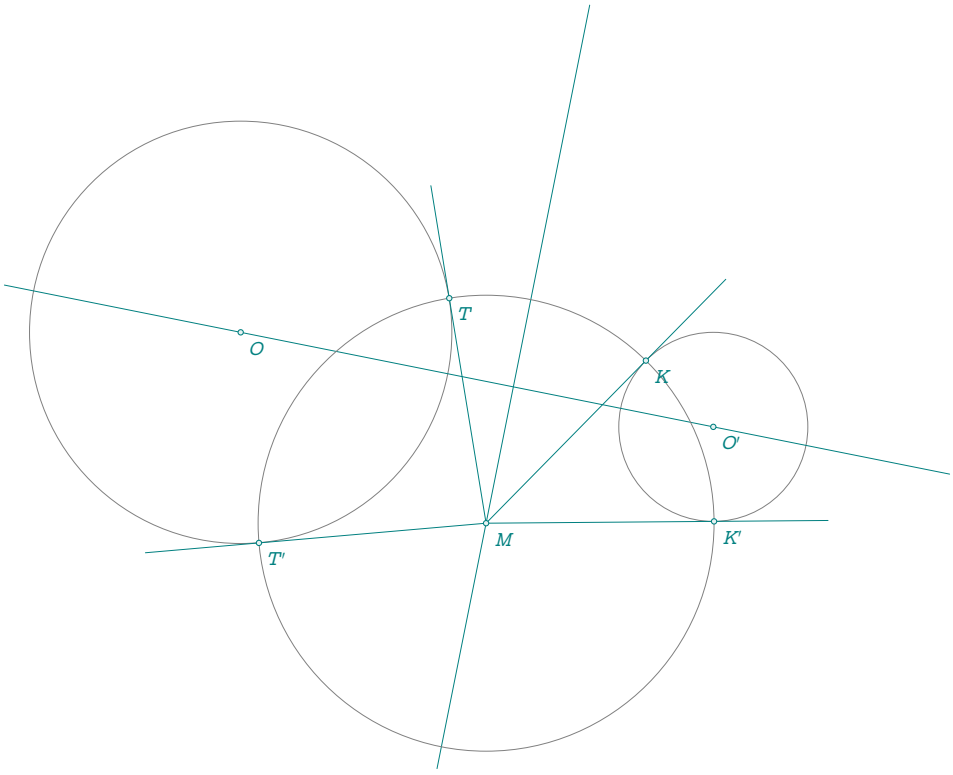
22.26 Radical axis v1

```
\begin{tkzelements}
scale    = .75
z.X      = point : new (0,0)
z.B      = point : new (2,2)
z.Y      = point : new (7,1)
z.Ap     = point : new (8,-1)
L.XY     = line :    new (z.X,z.Y)
C.XB     = circle : new (z.X,z.B)
C.YAp    = circle : new (z.Y,z.Ap)
z.E,z.F  = get_points (C.XB : radical_axis (C.YAp))
z.A      = C.XB : point (0.4)
T.ABAp   = triangle: new (z.A,z.B,z.Ap)
z.O      = T.ABAp.circumcenter
C.OAp    = circle : new (z.O,z.Ap)
_,z.Bp   = intersection (C.OAp,C.YAp)
L.AB     = line : new (z.A,z.B)
L.ApBp   = line : new (z.Ap,z.Bp)
z.M      = intersection (L.AB,L.ApBp)
z.H      = L.XY : projection (z.M)
\end{tkzelements}
\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawCircles(X,B Y,A')
   \tkzDrawArc[dashed,delta=30](O,A')(A)
   \tkzDrawPoints(A,B,A',B',M,H,X,Y,O,E,F)
   \tkzDrawLines[red](A,M A',M X,Y E,F)
   \tkzDrawLines[red,add=1 and 3](M,H)
\end{tikzpicture}
```



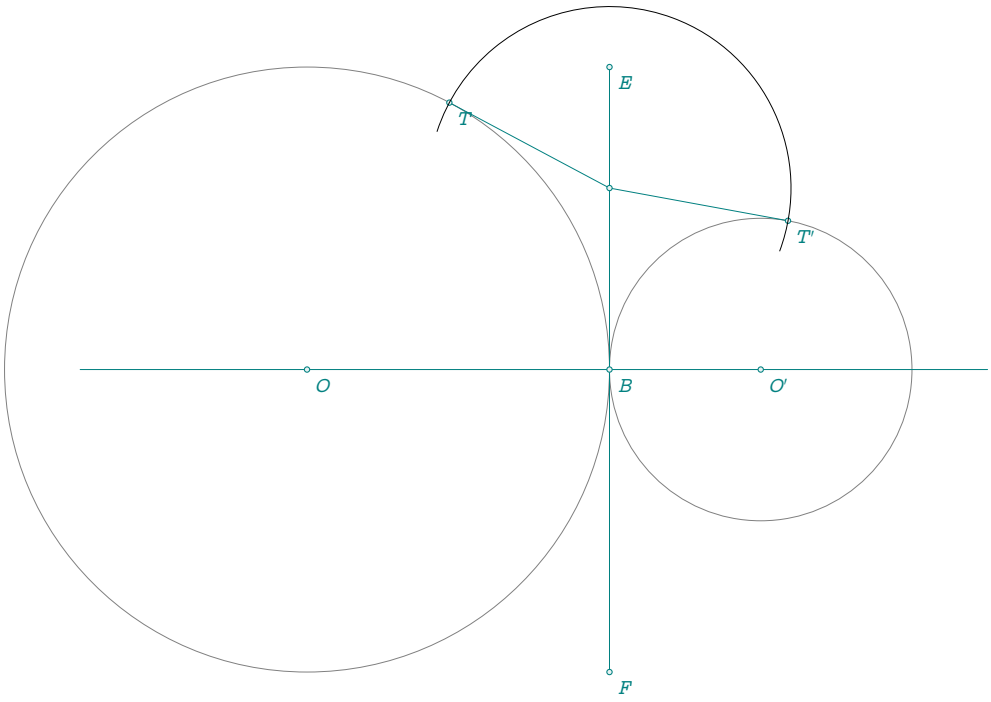
22.27 Radical axis v2

```
\begin{tkzelements}
scale       = 1.25
z.O         = point : new (-1,0)
z.Op        = point : new (4,-1)
z.B         = point : new (0,2)
z.D         = point : new (4,0)
C.OB        = circle :    new (z.O,z.B)
C.OpD       = circle :    new (z.Op,z.D)
L.EF        = C.OB : radical_axis (C.OpD)
z.E,z.F     = get_points (L.EF)
z.M         = L.EF : point (.75)
L.MT,L.MTp  = C.OB : tangent_from (z.M)
_,z.T       = get_points (L.MT)
_,z.Tp      = get_points (L.MTp)
L.MK,L.MKp  = C.OpD : tangent_from (z.M)
_,z.K       = get_points (L.MK)
_,z.Kp      = get_points (L.MKp)
\end{tkzelements}
\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawCircles(O,B O',D)
   \tkzDrawLine(E,F)
   \tkzDrawLine[add=.5 and .5](O,O')
   \tkzDrawLines[add = 0 and .5](M,T M,T' M,K M,K')
   \tkzDrawCircle(M,T)
   \tkzDrawPoints(O,O',T,M,T',K,K')
   \tkzLabelPoints(O,O',T,T',K,K',M)
\end{tikzpicture}
```



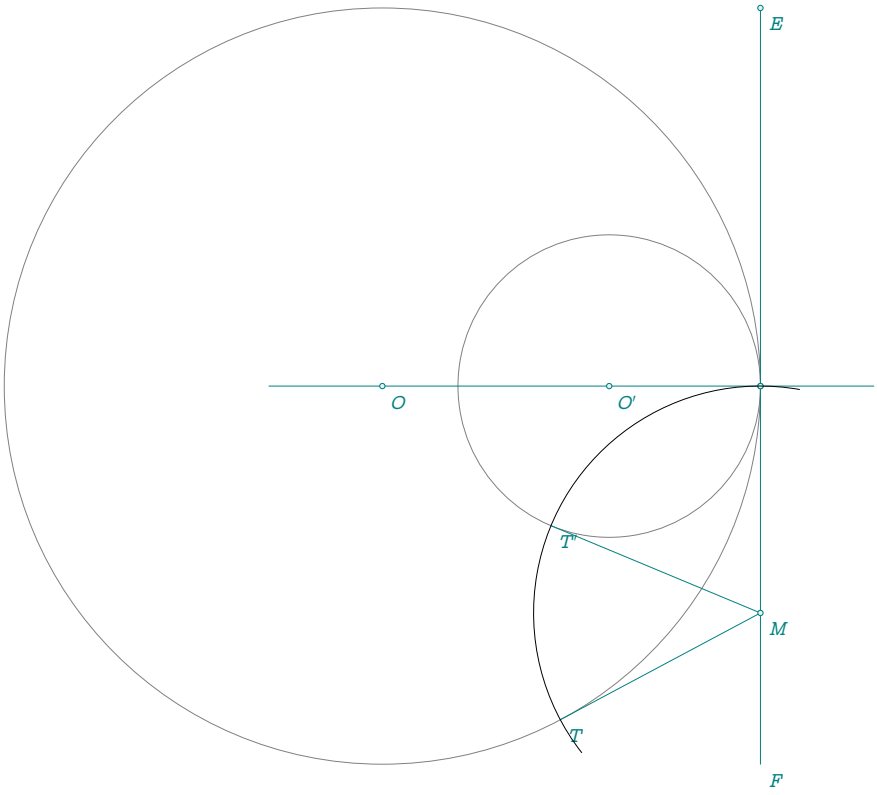
22.28 Radical axis v3

```
   \begin{tkzelements}
   z.O      = point : new (0,0)
   z.B      = point : new (4,0)
   z.Op     = point : new (6,0)
   C.OB     = circle :    new (z.O,z.B)
   C.OpB    = circle :    new (z.Op,z.B)
   L.EF     = C.OB : radical_axis (C.OpB)
   z.E,z.F  = get_points(L.EF)
   z.M      = L.EF : point (0.2)
   L        = C.OB : tangent_from (z.M)
   _,z.T    = get_points (L)
   L        = C.OpB : tangent_from (z.M)
   _,z.Tp   = get_points (L)
\end{tkzelements}
\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawCircles(O,B O',B)
   \tkzDrawSegments(M,T M,T')
   \tkzDrawSegments(E,F)
   \tkzDrawLine[add=.5 and .5](O,O')
   \tkzDrawPoints(O,B,O',E,F,M,T,T')
   \tkzLabelPoints(O,O',B,E,F,T,T')
   \tkzDrawArc(M,T')(T)
\end{tikzpicture}
```



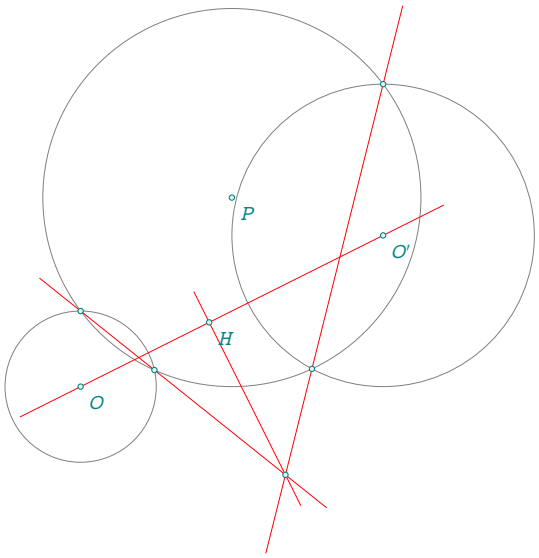
22.29 Radical axis v4

```
\begin{tkzelements}
   z.O     = point : new (0,0)
   z.B     = point : new (5,0)
   z.Op    = point : new (3,0)
   C.OB    = circle :    new (z.O,z.B)
   C.OpB   = circle :    new (z.Op,z.B)
   L.EF    = C.OB : radical_axis (C.OpB)
   z.E,z.F = get_points(L.EF)
   z.H     = L.EF.mid
   z.M     = L.EF : point (.8)
   _,L     = C.OB : tangent_from (z.M)
   _,z.T   = get_points (L)
   _,L     = C.OpB : tangent_from (z.M)
   _,z.Tp  = get_points (L)
\end{tkzelements}
\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawCircles(O,B O',B)
   \tkzDrawSegments(M,T M,T')
   \tkzDrawSegments(E,F)
   \tkzDrawLine[add=.3 and .3](O,H)
   \tkzDrawPoints(O,O',B,E,H,M)
   \tkzLabelPoints[below right](O,O',E,F,M,T,T')
   \tkzDrawArc(M,B)(T)
\end{tikzpicture}
```



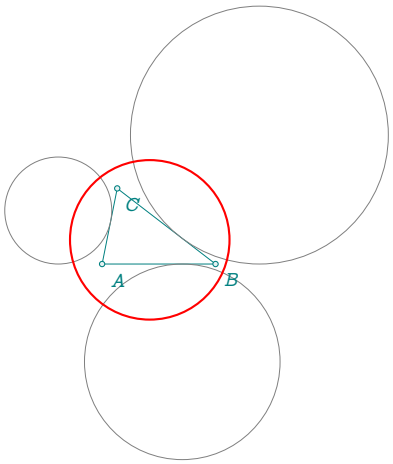
22.30 Radical center

```
\begin{tkzelements}
   z.O      = point : new (0,0)
   z.x      = point : new (1,0)
   z.y      = point : new (4,0)
   z.z      = point : new (2,0)
   z.Op     = point : new (4,2)
   z.P      = point : new (2,2.5)
   C.Ox     = circle :    new (z.O,z.x)
   C.Pz     = circle :    new (z.P,z.z)
   C.Opy    = circle :    new (z.Op,z.y)
   z.ap,z.a = intersection (C.Ox,C.Pz)
   z.bp,z.b = intersection (C.Opy,C.Pz)
   L.aap    = line : new (z.a,z.ap)
   L.bbp    = line : new (z.b,z.bp)
   z.X      = intersection (L.aap,L.bbp)
-- or z.X   = radical_center(C.Ox,C.Pz,C.Opy)
   L.OOp    = line : new (z.O,z.Op)
   z.H      = L.OOp : projection (z.X)
\end{tkzelements}
\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawCircles(O,a O',b P,z)
   \tkzDrawLines[red](a,X b',X H,X O,O')
   \tkzDrawPoints(O,O',P,a,a',b,b',X,H)
   \tkzLabelPoints[below right](O,O',P,H)
\end{tikzpicture}
```



22.31 Radical circle

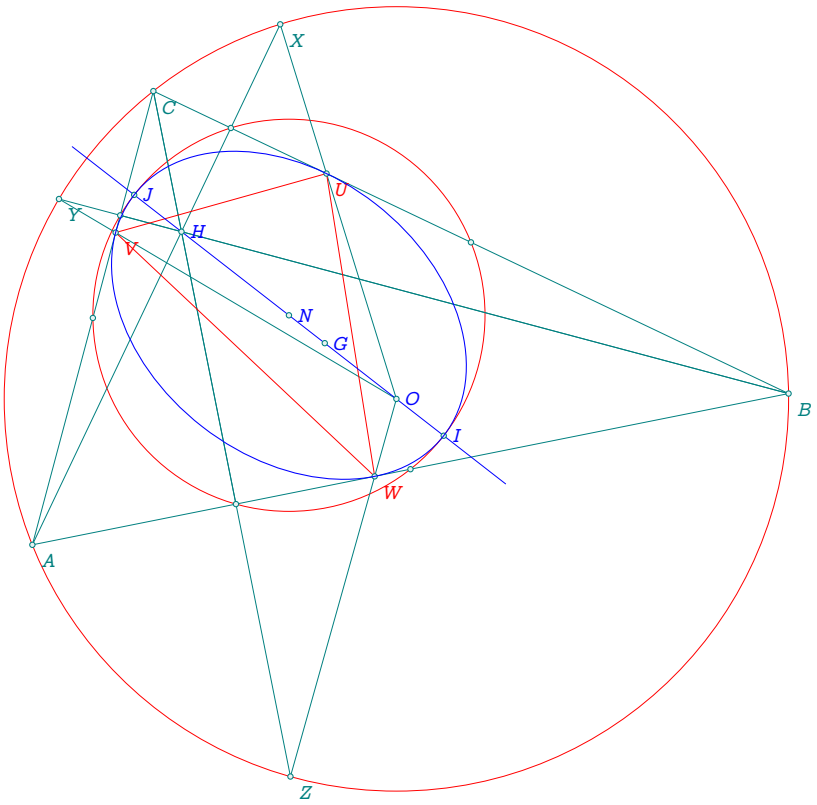
```
\begin{tkzelements}
   scale       = .25
   z.A         = point: new (0,0)
   z.B         = point: new (6,0)
   z.C         = point: new (0.8,4)
   T.ABC       = triangle : new ( z.A,z.B,z.C )
   C.exa       = T.ABC : ex_circle ()
   z.I_a,z.Xa  = get_points (C.exa)
   C.exb       = T.ABC : ex_circle (1)
   z.I_b,z.Xb  = get_points (C.exb)
   C.exc       = T.ABC : ex_circle (2)
   z.I_c,z.Xc  = get_points (C.exc)
   C.ortho     = C.exa : radical_circle (C.exb,C.exc)
   z.w,z.a     = get_points (C.ortho)
\end{tkzelements}
\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawPolygon(A,B,C)
   \tkzDrawCircles(I_a,Xa I_b,Xb I_c,Xc)
   \tkzDrawCircles[red,thick](w,a)
   \tkzDrawPoints(A,B,C)
   \tkzLabelPoints(A,B,C)
\end{tikzpicture}
```



22.32 Euler ellipse

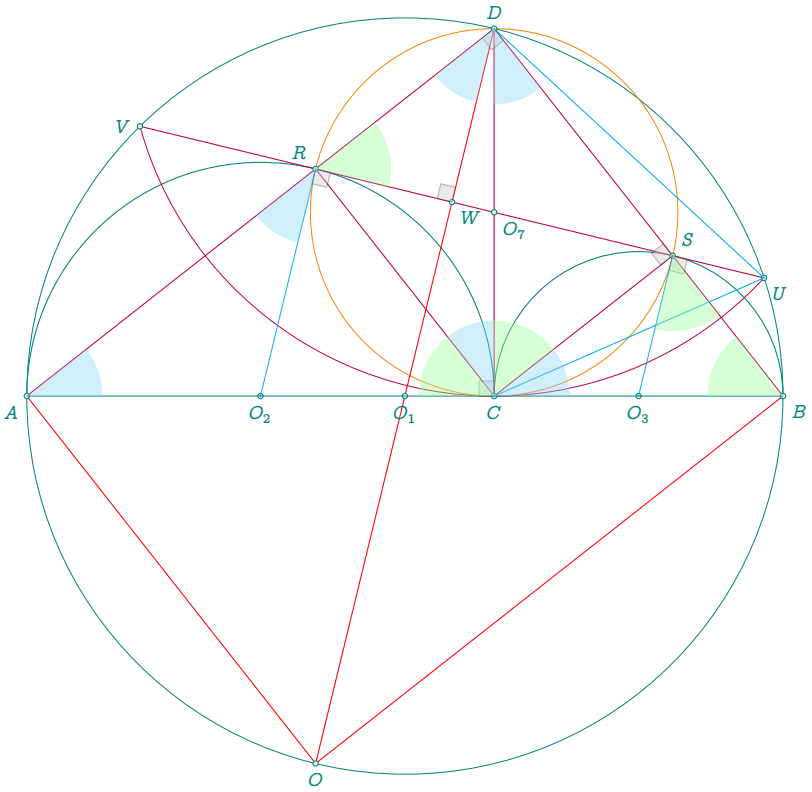
```
\begin{tkzelements}
   z.A               = point: new (0 , 0)
   z.B               = point: new (5 , 1)
   L.AB              = line : new (z.A,z.B)
   z.C               = point: new (.8 , 3)
   T.ABC             = triangle: new (z.A,z.B,z.C)
   z.N               = T.ABC.eulercenter
   z.G               = T.ABC.centroid
   z.O               = T.ABC.circumcenter
   z.H               = T.ABC.orthocenter
   z.Ma,z.Mb,z.Mc    = get_points (T.ABC : medial ())
   z.Ha,z.Hb,z.Hc    = get_points (T.ABC : orthic ())
   z.Ea,z.Eb,z.Ec    = get_points (T.ABC: extouch())
   L.euler           = T.ABC : euler_line ()
   C.circum          = T.ABC : circum_circle ()
   C.euler           = T.ABC : euler_circle ()
   z.I,z.J           = intersection (L.euler,C.euler)
   E                 = ellipse: foci (z.H,z.O,z.I)
   a                 = E.Rx
   b                 = E.Ry
   ang               = math.deg(E.slope)
   L.AH              = line: new (z.A,z.H)
   L.BH              = line: new (z.B,z.H)
   L.CH              = line: new (z.C,z.H)
   z.X               = intersection (L.AH,C.circum)
   _,z.Y             = intersection (L.BH,C.circum)
   _,z.Z             = intersection (L.CH,C.circum)
   L.BC              = line: new (z.B,z.C)
   L.XO              = line: new (z.X,z.O)
   L.YO              = line: new (z.Y,z.O)
   L.ZO              = line: new (z.Z,z.O)
   z.x               = intersection (L.BC,L.XO)
   z.U               = intersection (L.XO,E)
  _,z.V              = intersection (L.YO,E)
  _,z.W              = intersection (L.ZO,E)
 \end{tkzelements}
```

```
\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawPolygon(A,B,C)
   \tkzDrawCircles[red](N,Ma O,A)
   \tkzDrawSegments(A,X B,Y C,Z B,Hb C,Hc X,O Y,O Z,O)
   \tkzDrawPolygon[red](U,V,W)
   \tkzLabelPoints[red](U,V,W)
   \tkzLabelPoints(A,B,C,X,Y,Z)
   \tkzDrawLine[blue](I,J)
   \tkzLabelPoints[blue,right](O,N,G,H,I,J)
   \tkzDrawPoints(I,J,U,V,W)
   \tkzDrawPoints(A,B,C,N,G,H,O,X,Y,Z,Ma,Mb,Mc,Ha,Hb,Hc)
   \tkzDrawEllipse[blue](N,\tkzUseLua{a},\tkzUseLua{b},\tkzUseLua{ang})
\end{tikzpicture}
```



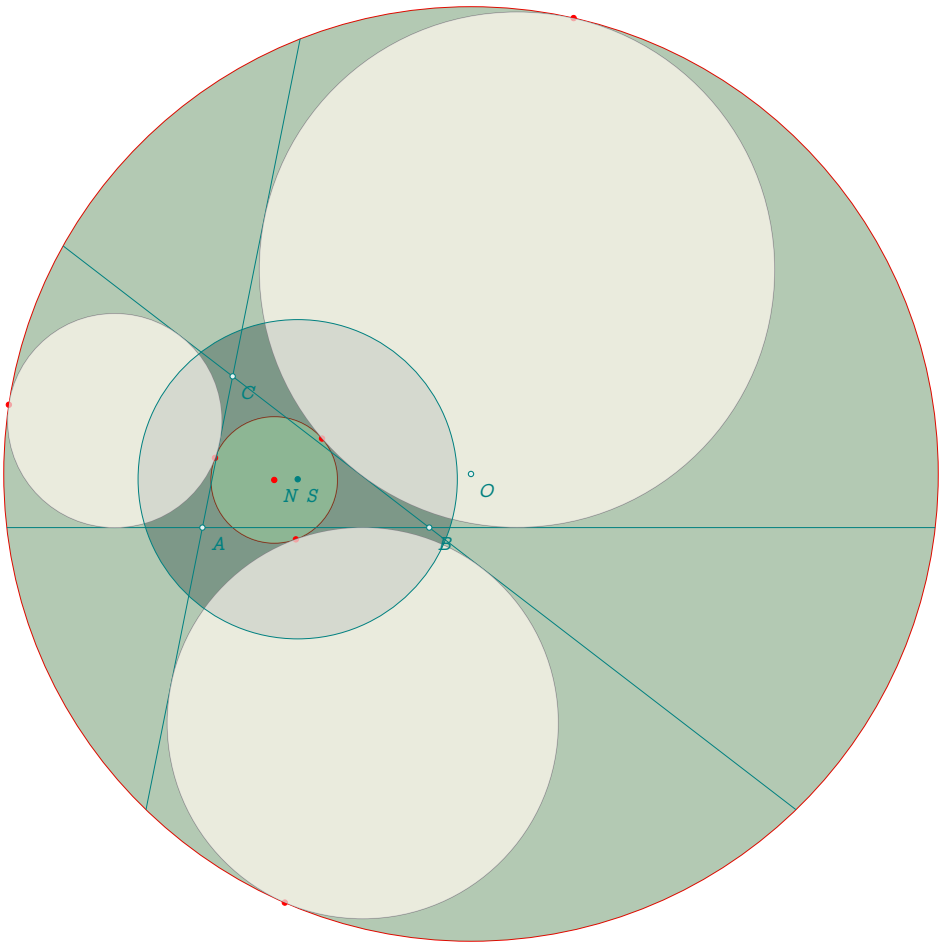
22.33 Gold Arbelos properties

```
\begin{tkzelements}
   z.A      = point : new(0,0)
   z.B      = point : new(10,0)
   z.C      = gold_segment_ (z.A,z.B)
   L.AB     = line:new (z.A,z.B)
   z.O_1    = L.AB.mid
   L.AC     = line:new (z.A,z.C)
   z.O_2    = L.AC.mid
   L.CB     = line:new (z.C,z.B)
   z.O_3    = L.CB.mid
   C1       = circle:new (z.O_1,z.B)
   C2       = circle:new (z.O_2,z.C)
   C3       = circle:new (z.O_3,z.B)
   z.Q      = C2.north
   z.P      = C3.north
   L1       = line:new (z.O_2,z.O_3)
   z.M_0    = L1:harmonic_ext (z.C)
   L2       = line:new (z.O_1,z.O_2)
   z.M_1    = L2:harmonic_int (z.A)
   L3       = line:new (z.O_1,z.O_3)
   z.M_2    = L3:harmonic_int (z.B)
   Lbq      = line:new (z.B,z.Q)
   Lap      = line:new (z.A,z.P)
   z.S      = intersection (Lbq,Lap)
   z.x      = z.C: north ()
   L        = line : new (z.C,z.x)
   z.D,_    = intersection (L,C1)
   L.CD     = line :new (z.C,z.D)
   z.O_7    = L.CD.mid
   C.DC     = circle: new (z.D,z.C)
   z.U,z.V  = intersection (C.DC,C1)
   L.UV     = line :new (z.U,z.V)
   z.R ,z.S = L.UV : projection (z.O_2,z.O_3)
   L.O1D    = line : new (z.O_1,z.D)
   z.W      = intersection (L.UV,L.O1D)
   z.O      = C.DC : inversion (z.W)
\end{tkzelements}
\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawCircles[teal](O_1,B)
   \tkzDrawSemiCircles[thin,teal](O_2,C O_3,B)
   \tkzDrawArc[purple,delta=0](D,V)(U)
   \tkzDrawCircle[new](O_7,C)
   \tkzDrawSegments[thin,purple](A,D D,B C,R C,S C,D U,V)
   \tkzDrawSegments[thin,red](O,D A,O O,B)
   \tkzDrawPoints(A,B,C,D,O_7) %,
   \tkzDrawPoints(O_1,O_2,O_3,U,V,R,S,W,O)
   \tkzDrawSegments[cyan](O_3,S O_2,R)
   \tkzDrawSegments[very thin](A,B)
   \tkzDrawSegments[cyan,thin](C,U U,D)
   \tkzMarkRightAngles[size=.2,fill=gray!40,opacity=.4](D,C,A A,D,B
     D,S,C D,W,V O_3,S,U O_2,R,U)
   \tkzFillAngles[cyan!40,opacity=.4](B,A,D A,D,O_1
     C,D,B D,C,R B,C,S A,R,O_2)
   \tkzFillAngles[green!40,opacity=.4](S,C,D W,R,D
     D,B,C R,C,A O_3,S,B)
   \tkzLabelPoints[below](C,O_2,O_3,O_1)
   \tkzLabelPoints[above](D)
   \tkzLabelPoints[below](O)
   \tkzLabelPoints[below left](A)
   \tkzLabelPoints[above left](R)
   \tkzLabelPoints[above right](S)
   \tkzLabelPoints[left](V)
   \tkzLabelPoints[below right](B,U,W,O_7)
\end{tikzpicture}
```



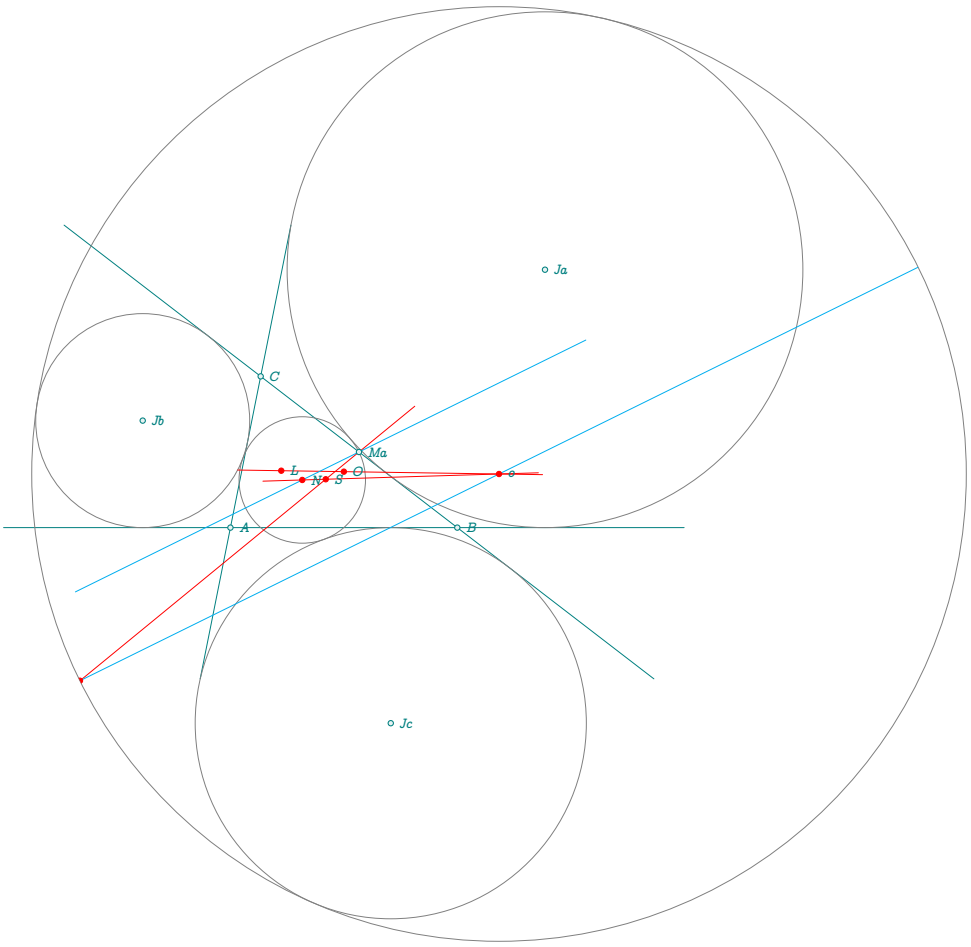
22.34 Apollonius circle v1 with inversion

```
\begin{tkzelements}
   scale             = .7
   z.A               = point: new (0,0)
   z.B               = point: new (6,0)
   z.C               = point: new (0.8,4)
   T.ABC             = triangle : new ( z.A,z.B,z.C )
   z.N               = T.ABC.eulercenter
   z.Ea,z.Eb,z.Ec    = get_points ( T.ABC : feuerbach () )
   z.Ja,z.Jb,z.Jc    = get_points ( T.ABC : excentral () )
   z.S               = T.ABC : spieker_center ()
   C.JaEa            = circle : new (z.Ja,z.Ea)
   C.ortho           = circle : radius (z.S,math.sqrt(C.JaEa : power (z.S) ))
   z.a               = C.ortho.south
   C.euler           = T.ABC: euler_circle ()
   C.apo             = C.ortho : inversion (C.euler)
   z.O               = C.apo.center
   z.xa,z.xb,z.xc    = C.ortho : inversion (z.Ea,z.Eb,z.Ec)
\end{tkzelements}
\begin{tikzpicture}
   \tkzGetNodes
\tkzDrawCircles[red](O,xa N,Ea)
\tkzFillCircles[green!30!black,opacity=.3](O,xa)
\tkzFillCircles[yellow!30,opacity=.7](Ja,Ea Jb,Eb Jc,Ec)
\tkzFillCircles[teal!30!black,opacity=.3](S,a)
\tkzFillCircles[green!30,opacity=.3](N,Ea)
\tkzDrawPoints[red](Ea,Eb,Ec,xa,xb,xc,N)
\tkzClipCircle(O,xa)
\tkzDrawLines[add=3 and 3](A,B A,C B,C)
\tkzDrawCircles(Ja,Ea Jb,Eb Jc,Ec)
\tkzFillCircles[lightgray!30,opacity=.7](Ja,Ea Jb,Eb Jc,Ec)
\tkzDrawCircles[teal](S,a)
\tkzDrawPoints(A,B,C,O)
\tkzDrawPoints[teal](S)
\tkzLabelPoints(A,B,C,O,S,N)
\end{tikzpicture}
```



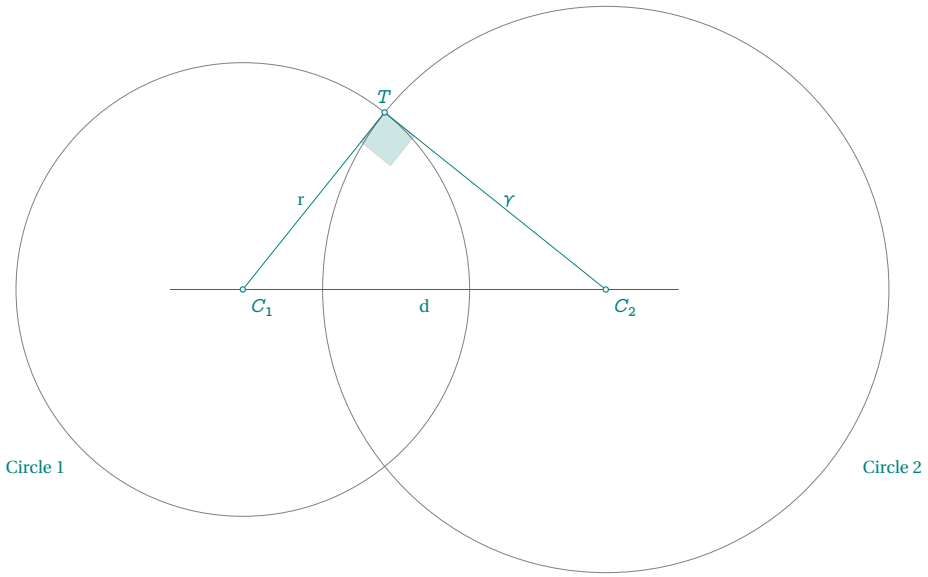
22.35 Apollonius circle v2

```
\begin{tkzelements}
   scale       = .5
   z.A         = point: new (0,0)
   z.B         = point: new (6,0)
   z.C         = point: new (0.8,4)
   T.ABC       = triangle: new(z.A,z.B,z.C)
   z.O         = T.ABC.circumcenter
   z.H         = T.ABC.orthocenter
   z.G         = T.ABC.centroid
   z.L         = T.ABC: lemoine_point ()
   z.S         = T.ABC: spieker_center ()
   C.euler     = T.ABC: euler_circle ()
   z.N,z.Ma    = get_points (C.euler)
   C.exA       = T.ABC : ex_circle ()
   z.Ja,z.Xa   = get_points (C.exA)
   C.exB       = T.ABC : ex_circle (1)
   z.Jb,z.Xb   = get_points (C.exB)
   C.exC       = T.ABC : ex_circle (2)
   z.Jc,z.Xc   = get_points (C.exC)
   L.OL        = line: new (z.O,z.L)
   L.NS        = line: new (z.N,z.S)
   z.o         = intersection (L.OL,L.NS) -- center of Apollonius circle
   L.NMa       = line: new (z.N,z.Ma)
   L.ox        = L.NMa: ll_from (z.o)
   L.MaS       = line: new (z.Ma,z.S)
   z.t         = intersection (L.ox,L.MaS) -- through
\end{tkzelements}

\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawLines[add=1 and 1](A,B A,C B,C)
   \tkzDrawCircles(Ja,Xa Jb,Xb Jc,Xc o,t N,Ma) %
   \tkzClipCircle(o,t)
   \tkzDrawLines[red](o,L N,o Ma,t)
   \tkzDrawLines[cyan,add=4 and 4](Ma,N o,t)
   \tkzDrawPoints(A,B,C,Ma,Ja,Jb,Jc)
   \tkzDrawPoints[red](N,O,L,S,o,t)
   \tkzLabelPoints[right,font=\tiny](A,B,C,Ja,Jb,Jc,O,N,L,S,Ma,o)
\end{tikzpicture}
```



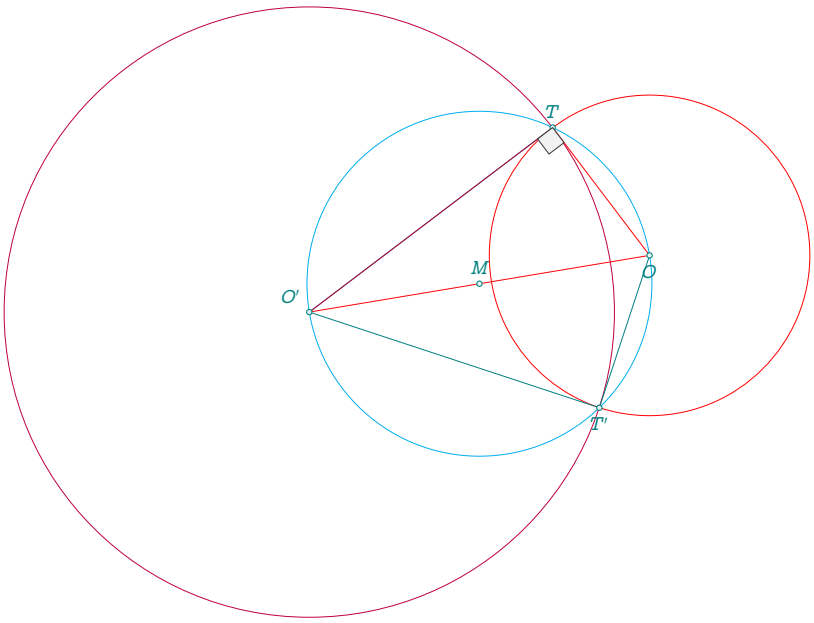
22.36 Orthogonal circles v1

```
\begin{tkzelements}
   scale    = .6
   z.C_1    = point: new (0,0)
   z.C_2    = point: new (8,0)
   z.A      = point: new (5,0)
   C        = circle: new (z.C_1,z.A)
   z.S,z.T  = get_points (C: orthogonal_from (z.C_2))
\end{tkzelements}
\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawCircles(C_1,T C_2,T)
   \tkzDrawSegments(C_1,T C_2,T)
   \tkzDrawLine(C_1,C_2)
   \tkzMarkRightAngle[fill=teal,%
  opacity=.2,size=1](C_1,T,C_2)
   \tkzDrawPoints(C_1,C_2,T)
   \tkzLabelPoints(C_1,C_2)
   \tkzLabelPoints[above](T)
   \tkzLabelSegment[left](C_1,T){r}
   \tkzLabelSegments[right](C_2,T){$\gamma$}
   \tkzLabelSegment[below](C_1,C_2){d}
   \tkzLabelCircle[left=10pt](C_1,T)(180){Circle 1}
   \tkzLabelCircle[right=10pt](C_2,T)(180){Circle 2}
\end{tikzpicture}
```



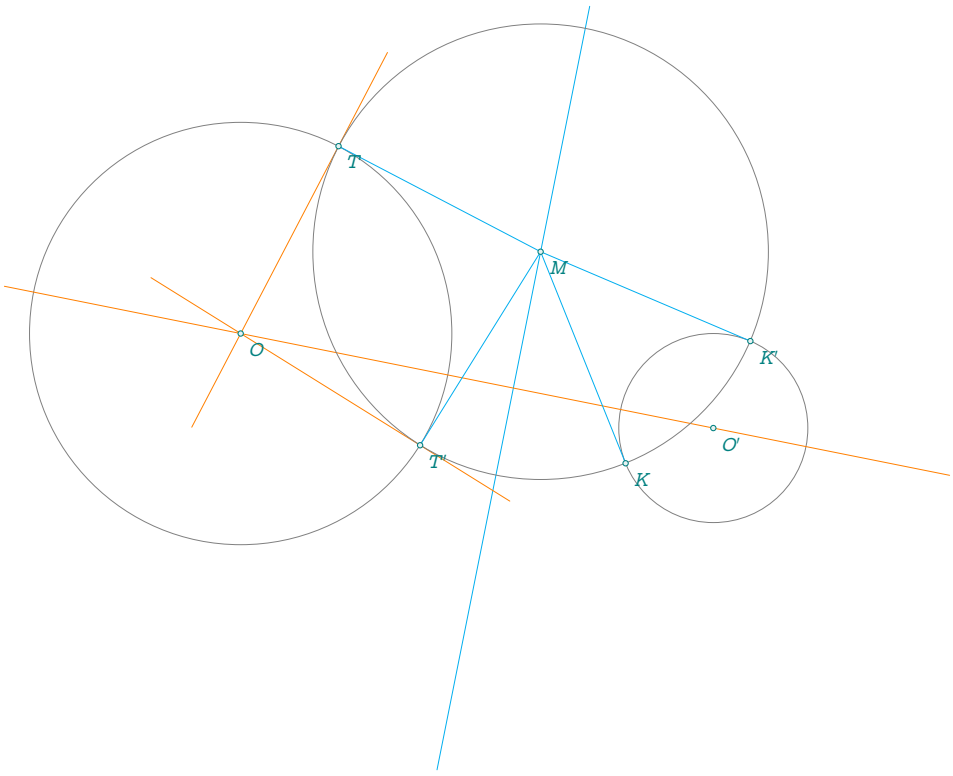
22.37 Orthogonal circles v2

```
\begin{tkzelements}
scale    = .75
z.O      = point: new (2,2)
z.Op     = point: new (-4,1)
z.P      = point: polar (4,0)
C.OP     = circle: new (z.O,z.P)
C.Oz1    =  C.OP : orthogonal_from (z.Op)
z.z1     = C.Oz1.through
L.OP     = line : new (z.O,z.P)
C.Opz1   = circle: new (z.Op,z.z1)
L.T,L.Tp = C.Opz1 : tangent_from (z.O)
z.T      = L.T.pb
z.Tp     = L.Tp.pb
L.OOp    = line : new (z.O,z.Op)
z.M      = L.OOp.mid
\end{tkzelements}
\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawCircle[red](O,P)
   \tkzDrawCircle[purple](O',z1)
   \tkzDrawCircle[cyan](M,T)
   \tkzDrawSegments(O',T O,T' O',T')
   \tkzDrawSegment[purple](O',T)
   \tkzDrawSegments[red](O,T O,O')
   \tkzDrawPoints(O,O',T,T',M)
   \tkzMarkRightAngle[fill=gray!10](O',T,O)
   \tkzLabelPoint[below](O){$O$}
   \tkzLabelPoint[above](T){$T$}
   \tkzLabelPoint[above](M){$M$}
   \tkzLabelPoint[below](T'){$T'$}
   \tkzLabelPoint[above left](O'){$O'$}
\end{tikzpicture}
```

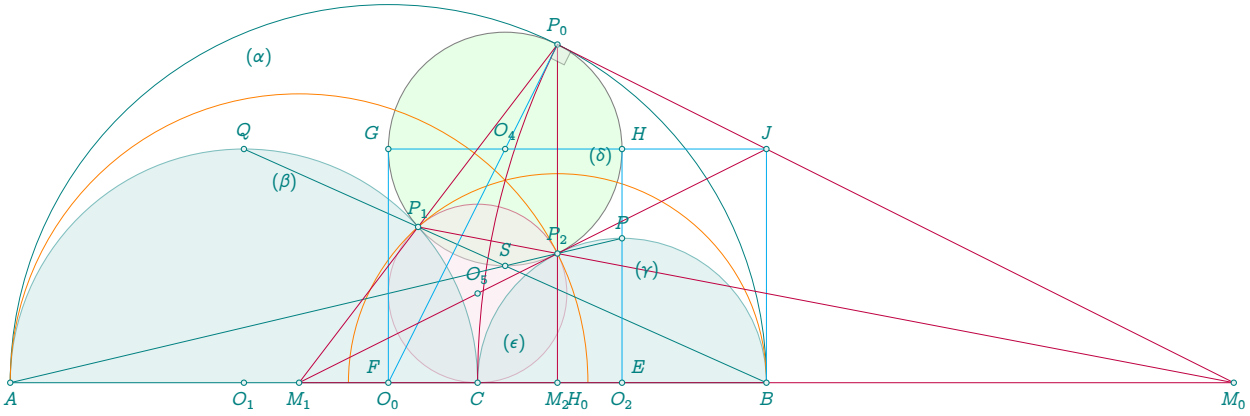


22.38 Orthogonal circle to two circles

```
\begin{tkzelements}
   z.O          = point :   new (-1,0)
   z.B          = point :   new (0,2)
   z.Op         = point :   new (4,-1)
   z.D          = point :   new (4,0)
   C.OB         = circle :  new (z.O,z.B)
   C.OpD        = circle :  new (z.Op,z.D)
   z.E,z.F      = get_points (C.OB : radical_axis (C.OpD))
   L.EF         = line : new (z.E,z.F)
   z.M          = L.EF : point (.25)
   L.T,L.Tp     = C.OB : tangent_from (z.M)
   L.K,L.Kp     = C.OpD : tangent_from (z.M)
   z.T          = L.T.pb
   z.K          = L.K.pb
   z.Tp         = L.Tp.pb
   z.Kp         = L.Kp.pb
\end{tkzelements}
\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawCircles(O,B O',D)
   \tkzDrawLine[cyan](E,F)
   \tkzDrawLines[add=.5 and .5,orange](O,O' O,T O,T')
   \tkzDrawSegments[cyan](M,T M,T' M,K M,K')
   \tkzDrawCircle(M,T)
   \tkzDrawPoints(O,O',T,M,T',K,K')
   \tkzLabelPoints(O,O',T,T',M,K,K')
\end{tikzpicture}
```



22.39 Midcircles

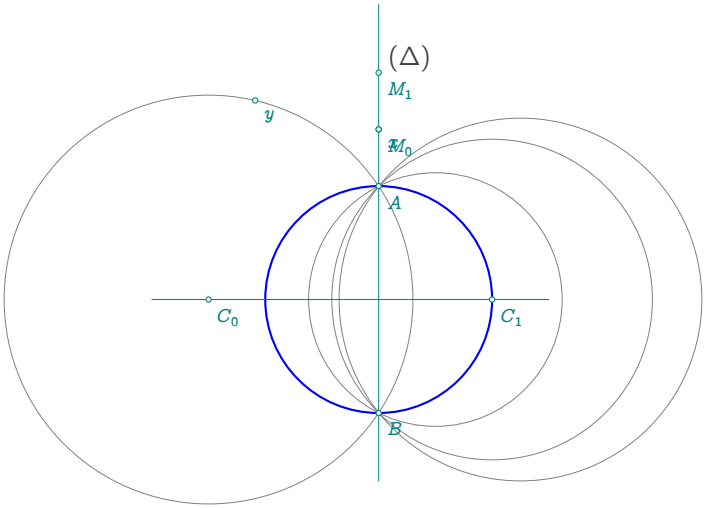


```
\begin{tkzelements}
   z.A      = point: new (0 , 0)
   z.B      = point: new (10 , 0)
   L.AB     = line : new (z.A,z.B)
   z.C      = L.AB: gold_ratio ()
   L.AC     = line : new (z.A,z.C)
   L.CB     = line : new (z.C,z.B)
   z.O_0    = L.AB.mid
   z.O_1    = L.AC.mid
   z.O_2    = L.CB.mid
   C.O0B    = circle : new (z.O_0,z.B)
   C.O1C    = circle : new (z.O_1,z.C)
   C.O2C    = circle : new (z.O_2,z.B)
   z.Q      = C.O1C : midarc (z.C,z.A)
   z.P      = C.O2C : midarc (z.B,z.C)
   L.O1O2   = line : new (z.O_1,z.O_2)
   L.O0O1   = line : new (z.O_0,z.O_1)
   L.O0O2   = line : new (z.O_0,z.O_2)
   z.M_0    = L.O1O2 : harmonic_ext (z.C)
   z.M_1    = L.O0O1 : harmonic_int (z.A)
   z.M_2    = L.O0O2 : harmonic_int (z.B)
   L.BQ     = line : new (z.B,z.Q)
   L.AP     = line : new (z.A,z.P)
   z.S      = intersection (L.BQ,L.AP)
   L.CS     = line : new (z.C,z.S)
   C.M1A    = circle : new (z.M_1,z.A)
   C.M2B    = circle : new (z.M_2,z.B)
   z.P_0    = intersection (L.CS,C.O0B)
   z.P_1    = intersection (C.M2B,C.O1C)
   z.P_2    = intersection (C.M1A,C.O2C)
   T.P012   = triangle : new (z.P_0,z.P_1,z.P_2)
   z.O_4    = T.P012.circumcenter
   T.CP12   = triangle : new (z.C,z.P_1,z.P_2)
   z.O_5    = T.CP12.circumcenter
   z.BN     = z.B : north ()
   L.BBN    = line : new (z.B,z.BN)
   L.M1P2   = line : new (z.M_1,z.P_2)
   z.J      = intersection (L.BBN,L.M1P2)
   L.AP0    = line : new (z.A,z.P_0)
   L.BP0    = line : new (z.B,z.P_0)
   C.O4P0   = circle : new (z.O_4,z.P_0)
   _,z.G    = intersection (L.AP0,C.O4P0)
   z.H      = intersection (L.BP0,C.O4P0)
   z.Ap     = z.M_1: symmetry (z.A)
   z.H_4,z.F,z.E,z.H_0 = L.AB : projection (z.O_4,z.G,z.H,z.P_0)
\end{tkzelements}
```

```
\begin{tikzpicture}
\tkzGetNodes
\tkzDrawCircle[thin,fill=green!10](O_4,P_0)
\tkzDrawCircle[purple,fill=purple!10,opacity=.5](O_5,C)
\tkzDrawSemiCircles[teal](O_0,B)
\tkzDrawSemiCircles[thin,teal,fill=teal!20,opacity=.5](O_1,C O_2,B)
\tkzDrawSemiCircles[color = orange](M_2,B)
\tkzDrawSemiCircles[color = orange](M_1,A')
\tkzDrawArc[purple,delta=0](M_0,P_0)(C)
\tkzDrawSegments[very thin](A,B A,P B,Q)
\tkzDrawSegments[color=cyan](O_0,P_0 B,J G,J G,O_0 H,O_2)
\tkzDrawSegments[ultra thin,purple](M_1,P_0 M_2,P_0 M_1,M_0 M_0,P_1 M_0,P_0 M_1,J)
\tkzDrawPoints(A,B,C,P_0,P_2,P_1,M_0,M_1,M_2,J,P,Q,S)
\tkzDrawPoints(O_0,O_1,O_2,O_4,O_5,G,H)
\tkzMarkRightAngle[size=.2,fill=gray!20,opacity=.4](O_0,P_0,M_0)
\tkzLabelPoints[below](A,B,C,M_0,M_1,M_2,O_1,O_2,O_0)
\tkzLabelPoints[above](P_0,O_5,O_4)
\tkzLabelPoints[above](P_1,J)
\tkzLabelPoints[above](P_2,P,Q,S)
\tkzLabelPoints[above right](H,E)
\tkzLabelPoints[above left](F,G)
\tkzLabelPoints[below right](H_0)
\tkzLabelCircle[below=4pt,font=\scriptsize](O_1,C)(80){$(\beta)$}
\tkzLabelCircle[below=4pt,font=\scriptsize](O_2,B)(80){$(\gamma)$}
\tkzLabelCircle[below=4pt,font=\scriptsize](O_0,B)(110){$(\alpha)$}
\tkzLabelCircle[left,font=\scriptsize](O_4,P_2)(60){$(\delta)$}
\tkzLabelCircle[above left,font=\scriptsize](O_5,C)(40){$(\epsilon)$}
\end{tikzpicture}
```

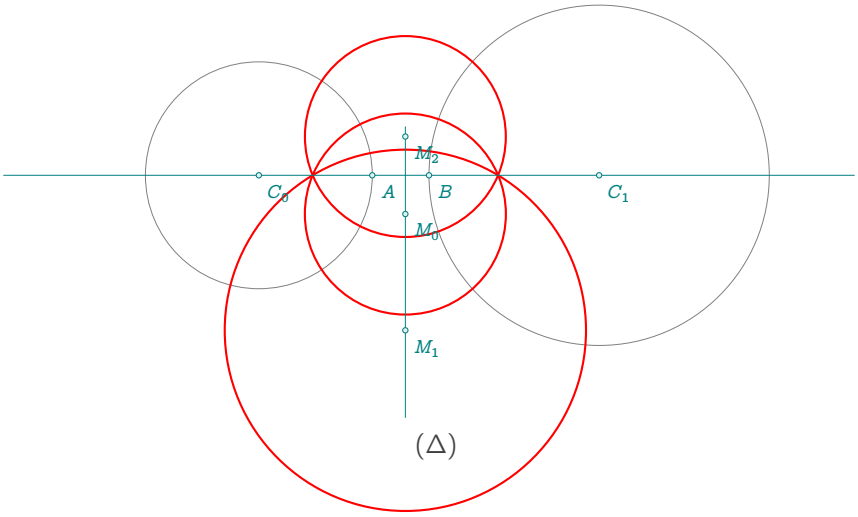
22.40 Pencil v1

```
\begin{tkzelements}
   scale       = .75
   z.A         = point : new (0,2)
   z.B         = point : new (0,-2)
   z.C_0       = point : new (-3,0)
   z.C_1       = point : new (2,0)
   z.C_3       = point : new (2.5,0)
   z.C_5       = point : new (1,0)
   L.BA        = line : new (z.B,z.A)
   z.M_0       = L.BA : point (1.25)
   z.M_1       = L.BA : point (1.5)
   C.C0A       = circle :    new (z.C_0,z.A)
   z.x,z.y     = get_points (C.C0A : orthogonal_from (z.M_0))
   z.xp,z.yp   = get_points (C.C0A : orthogonal_from (z.M_1))
   z.O         = L.BA.mid
\end{tkzelements}
\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawCircles(C_0,A C_1,A C_3,A C_5,A)
   \tkzDrawCircles[thick,color=red](M_0,x M_1,x')
   \tkzDrawCircles[thick,color=blue](O,A)
   \tkzDrawLines(C_0,C_1 B,M_1)
   \tkzDrawPoints(A,B,C_0,C_1,M_0,M_1,x,y)
   \tkzLabelPoints[below right](A,B,C_0,C_1,M_0,M_1,x,y)
   \tkzLabelLine[pos=1.25,right]( M_0,M_1){$(\Delta)$}
\end{tikzpicture}
```



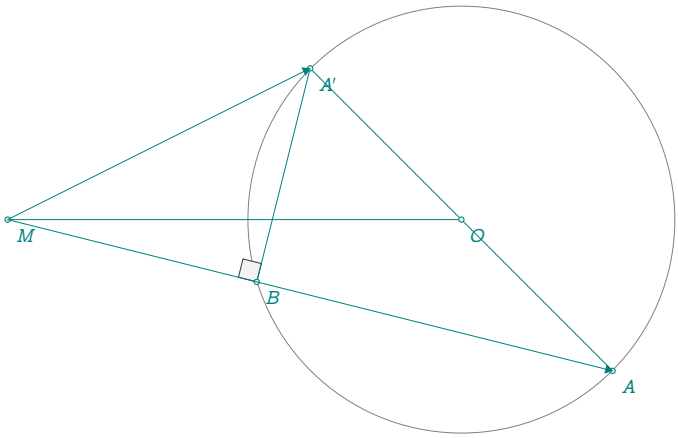
22.41 Pencil v2

```
\begin{tkzelements}
   scale=.75
   z.A     = point : new (0,0)
   z.B     = point : new (1,0)
   z.C_0   = point : new (-2,0)
   z.C_1   = point : new (4,0)
   C.C0A   = circle : new (z.C_0,z.A)
   C.C1B   = circle : new (z.C_1,z.B)
   L.EF    = C.C0A : radical_axis (C.C1B)
   z.M_0   = L.EF : point (.4)
   z.M_1   = L.EF : point (.1)
   z.M_2   = L.EF : point (.6)
   C.orth0     = C.C0A : orthogonal_from (z.M_0)
   C.orth1     = C.C0A : orthogonal_from (z.M_1)
   C.orth2     = C.C0A : orthogonal_from (z.M_2)
   z.u         = C.orth0.through
   z.v         = C.orth1.through
   z.t         = C.orth2.through
\end{tkzelements}
\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawCircles(C_0,A C_1,B)
   \tkzDrawCircles[thick,color=red](M_0,u M_1,v M_2,t)
   \tkzDrawLines[add= .75 and .75](C_0,C_1 M_0,M_1)
   \tkzDrawPoints(A,B,C_0,C_1,M_0,M_1,M_2)
   \tkzLabelPoints[below right](A,B,C_0,C_1,M_0,M_1,M_2)
   \tkzLabelLine[pos=2,right]( M_0,M_1){$(\Delta)$}
\end{tikzpicture}
```



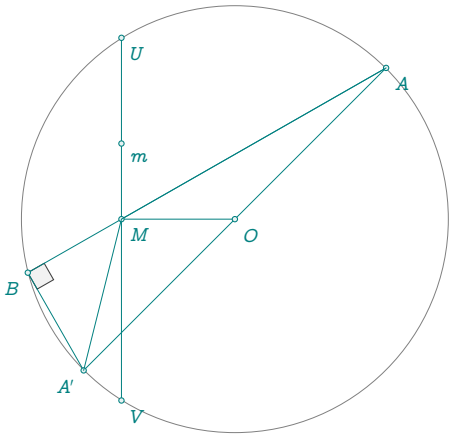
22.42 Power v1

```
\begin{tkzelements}
   z.O     = point : new (0,0)
   z.A     = point : new (2,-2)
   z.M     = point : new (-6,0)
   L.AM    = line : new (z.A,z.M)
   C.OA    = circle :    new (z.O,z.A)
   z.Ap    = C.OA : antipode (z.A)
   z.B     = intersection (L.AM, C.OA)
\end{tkzelements}
\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawCircle(O,A)
   \tkzMarkRightAngle[fill=gray!10](A',B,M)
   \tkzDrawSegments(M,O A,A' A',B)
   \tkzDrawPoints(O,A,A',M,B)
   \tkzLabelPoints(O,A,A',M,B)
   \tkzDrawSegments[-Triangle](M,A M,A')
\end{tikzpicture}
```



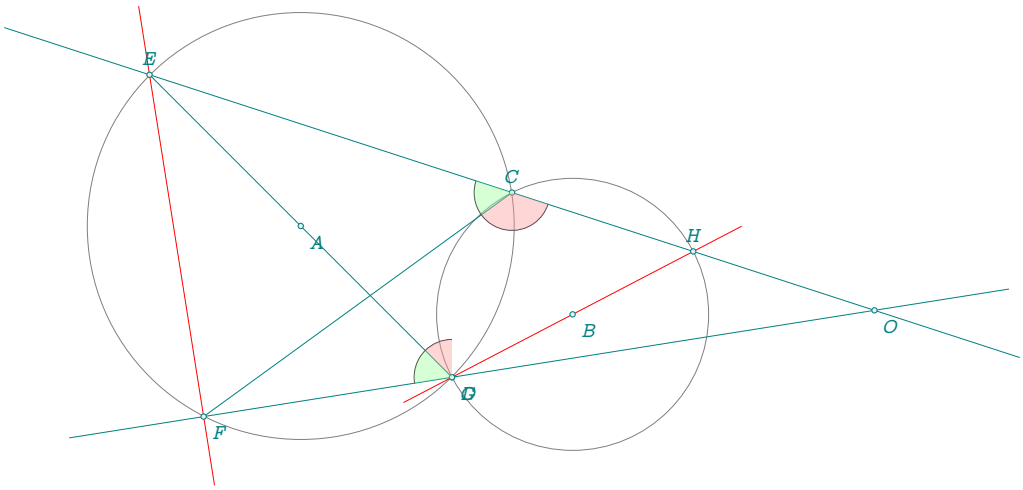
22.43 Power v2

```
\begin{tkzelements}
   z.O     = point : new (0,0)
   z.A     = point : new (2,2)
   z.M     = point : new (-1.5,0)
   L.AM    = line : new (z.A,z.M)
   C.OA    = circle :    new (z.O,z.A)
   z.Ap    = C.OA : antipode (z.A)
   _,z.B   = intersection (L.AM, C.OA)
   z.m     = z.M : north(1)
   L.mM    = line : new (z.m,z.M)
   z.U,z.V = intersection (L.mM,C.OA)
\end{tkzelements}
\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawCircle(O,A)
   \tkzMarkRightAngle[fill=gray!10](A',B,M)
   \tkzDrawSegments(M,O A,A' A',B A,B U,V)
   \tkzDrawPoints(O,A,A',M,B,U,V,m)
   \tkzLabelPoints(O,A,M,U,V,m)
   \tkzLabelPoints[below left](A',B)
   \tkzDrawSegments(M,A M,A')
\end{tikzpicture}
```



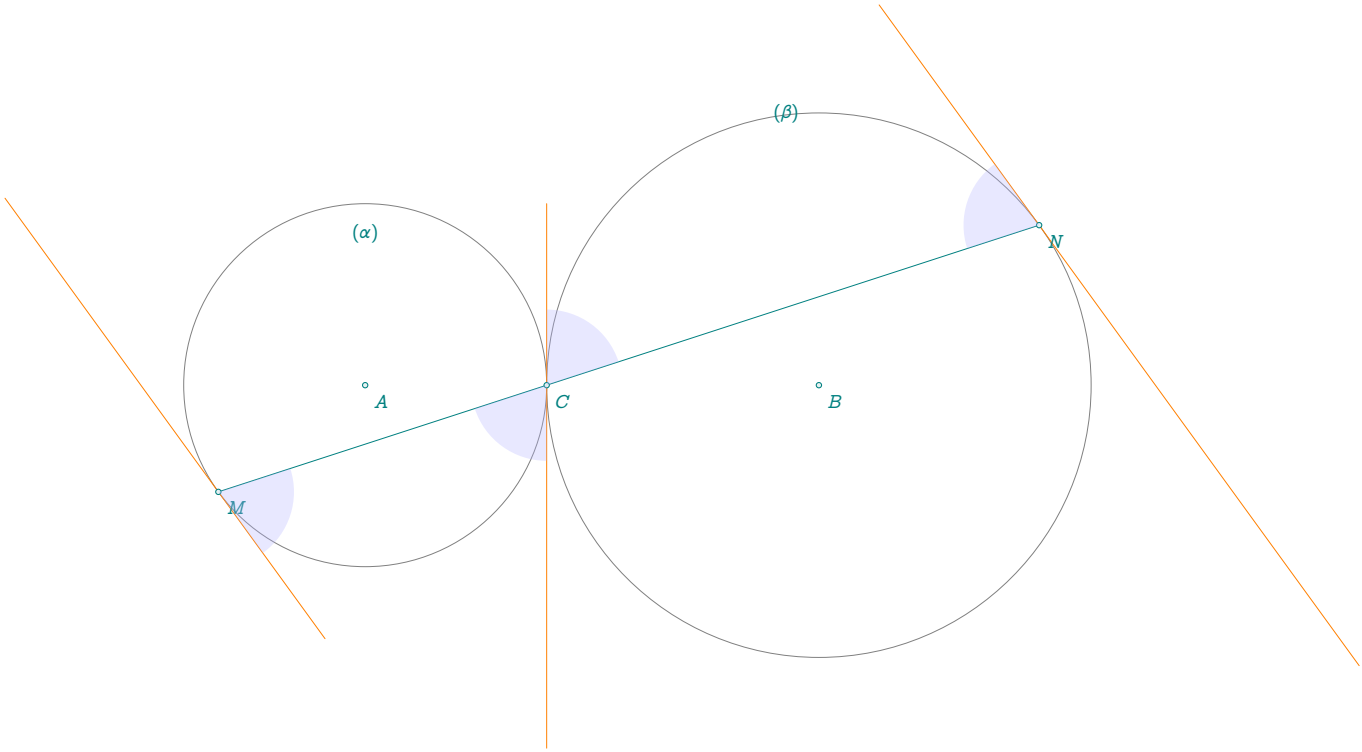
22.44 Reim v1

```
\begin{tkzelements}
   z.A     = point: new (0,0)
   z.E     = point: new (-2,2)
   C.AE    = circle :   new (z.A,z.E)
   z.C     = C.AE : point (0.65)
   z.D     = C.AE : point (0.5)
   z.F     = C.AE : point (0.30)
   L.EC    = line: new (z.E,z.C)
   z.H     = L.EC : point (1.5)
   T.CDH   = triangle : new (z.C,z.D,z.H)
   z.B     = T.CDH.circumcenter
   C.BD    = circle : new (z.B,z.D)
   L.FD    = line: new (z.F,z.D)
   z.G     = intersection (L.FD,C.BD)
   z.O     = intersection (L.EC,L.FD)
\end{tkzelements}
\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawCircles(A,E B,H)
   \tkzDrawSegments(E,D C,F)
   \tkzDrawLines(E,O F,O)
   \tkzDrawLines[red](E,F H,G)
   \tkzDrawPoints(A,...,H,O)
   \tkzLabelPoints(A,B,D,F,G,O)
   \tkzLabelPoints[above](E,C,H)
   \tkzMarkAngles[size=.5](E,C,F E,D,F)
   \tkzFillAngles[green!40,opacity=.4,size=.5](E,C,F E,D,F)
   \tkzMarkAngles[size=.5](F,C,H G,D,E)
   \tkzFillAngles[red!40,opacity=.4,size=.5](F,C,H G,D,E)
\end{tikzpicture}
```



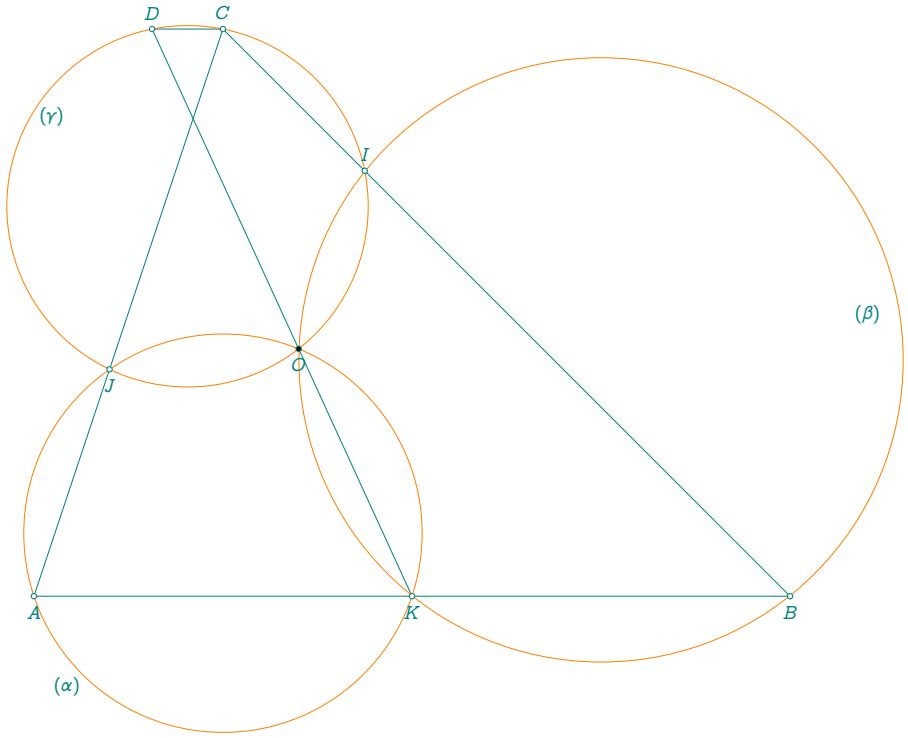
22.45 Reim v2

```
\begin{tkzelements}
   scale    = .6
   z.A      = point: new (0,0)
   z.B      = point: new (10,0)
   z.C      = point: new (4,0)
   C.AC     = circle: new (z.A,z.C)
   z.c,z.cp = get_points (C.AC: tangent_at (z.C))
   z.M      = C.AC: point (0.6)
   L.MC     = line: new (z.M,z.C)
   C.BC     = circle: new (z.B,z.C)
   z.N      = intersection (L.MC,C.BC)
   z.m,z.mp = get_points (C.AC: tangent_at (z.M))
   z.n,z.np = get_points (C.BC: tangent_at (z.N))
\end{tkzelements}
\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawCircles(A,C B,C)
   \tkzDrawLines[new,add=1 and 1](M,m N,n C,c)
   \tkzDrawSegment(M,N)
   \tkzDrawPoints(A,B,C,M,N)
   \tkzLabelPoints[below right](A,B,C,M,N)
   \tkzFillAngles[blue!30,opacity=.3](m',M,C N,C,c' M,C,c n',N,C)
   \tkzLabelCircle[below=4pt,font=\scriptsize](A,C)(90){$(\alpha)$}
   \tkzLabelCircle[left=4pt,font=\scriptsize](B,C)(-90){$(\beta)$}
\end{tikzpicture}
```



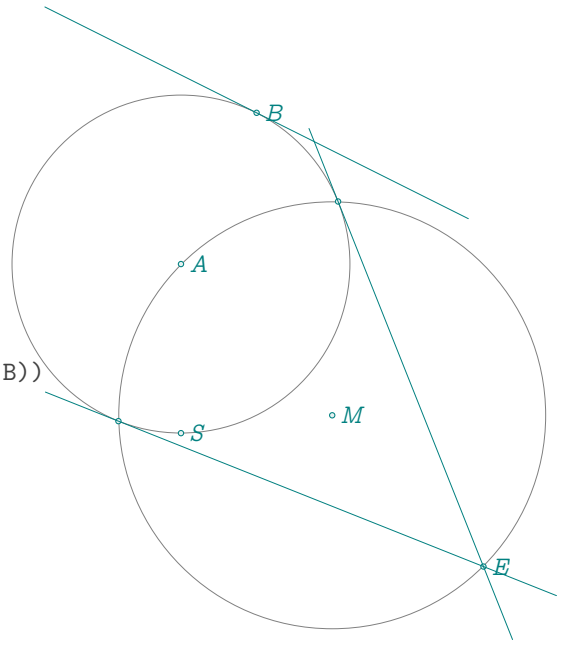
22.46 Reim v3

```
\begin{tkzelements}
   z.A     = point: new (0,0)
   z.B     = point: new (8,0)
   z.C     = point: new (2,6)
   L.AB    = line : new (z.A,z.B)
   L.AC    = line : new (z.A,z.C)
   L.BC    = line : new (z.B,z.C)
   z.I     = L.BC : point (0.75)
   z.J     = L.AC : point (0.4)
   z.K     = L.AB : point (0.5)
   T.AKJ   = triangle : new (z.A,z.K,z.J)
   T.BIK   = triangle : new (z.B,z.I,z.K)
   T.CIJ   = triangle : new (z.C,z.I,z.J)
   z.x     = T.AKJ.circumcenter
   z.y     = T.BIK.circumcenter
   z.z     = T.CIJ.circumcenter
   C.xK    = circle: new (z.x,z.K)
   C.yK    = circle: new (z.y,z.K)
   z.O,_   = intersection (C.xK,C.yK)
   C.zO    = circle: new (z.z,z.O)
   L.KO    = line: new (z.K,z.O)
   z.D     = intersection (L.KO,C.zO)
\end{tkzelements}

\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawSegments(K,D D,C)
   \tkzDrawPolygon[teal](A,B,C)
   \tkzDrawCircles[orange](x,A y,B z,C)
   \tkzDrawPoints[fill=white](A,B,C,I,J,K,D)
   \tkzLabelPoints[below](A,B,J,K,O)
   \tkzLabelPoints[above](C,D,I)
   \tkzDrawPoints[fill=black](O)
   \tkzLabelCircle[below=4pt,font=\scriptsize](x,A)(20){$(\alpha)$}
   \tkzLabelCircle[left=4pt,font=\scriptsize](y,B)(60){$(\beta)$}
   \tkzLabelCircle[below=4pt,font=\scriptsize](z,C)(60){$(\gamma)$}
\end{tikzpicture}
```



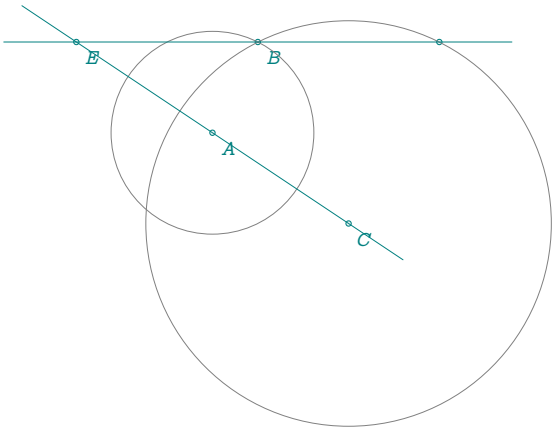
22.47 Tangent and circle

```
\begin{tkzelements}
   z.A          = point:    new (1,0)
   z.B          = point:    new (2,2)
   z.E          = point:    new (5,-4)
   L.AE         = line :    new (z.A,z.E)
   C.AB         = circle:   new (z.A , z.B)
   z.S          = C.AB.south
   z.M          = L.AE.mid
   L.Ti,L.Tj    = C.AB:   tangent_from (z.E)
   z.i          = L.Ti.pb
   z.j          = L.Tj.pb
   z.k,z.l      = get_points (C.AB:  tangent_at (z.B))
\end{tkzelements}
\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawCircles(A,B M,A)
   \tkzDrawPoints(A,B,E,i,j,M,S)
   \tkzDrawLines(E,i E,j k,l)
   \tkzLabelPoints[right,font=\small](A,B,E,S,M)
\end{tikzpicture}
```



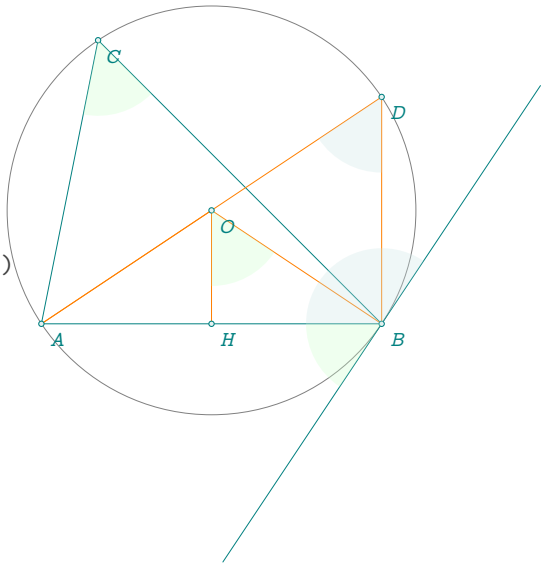
22.48 Homothety

```
\begin{tkzelements}
   z.A      = point:  new (0,0)
   z.B      = point:  new (1,2)
   z.E      = point:  new (-3,2)
   z.C,z.D  = z.E : homothety(2,z.A,z.B)
\end{tkzelements}
\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawPoints(A,B,C,E,D)
   \tkzLabelPoints(A,B,C,E)
   \tkzDrawCircles(A,B C,D)
   \tkzDrawLines(E,C E,D)
\end{tikzpicture}
```



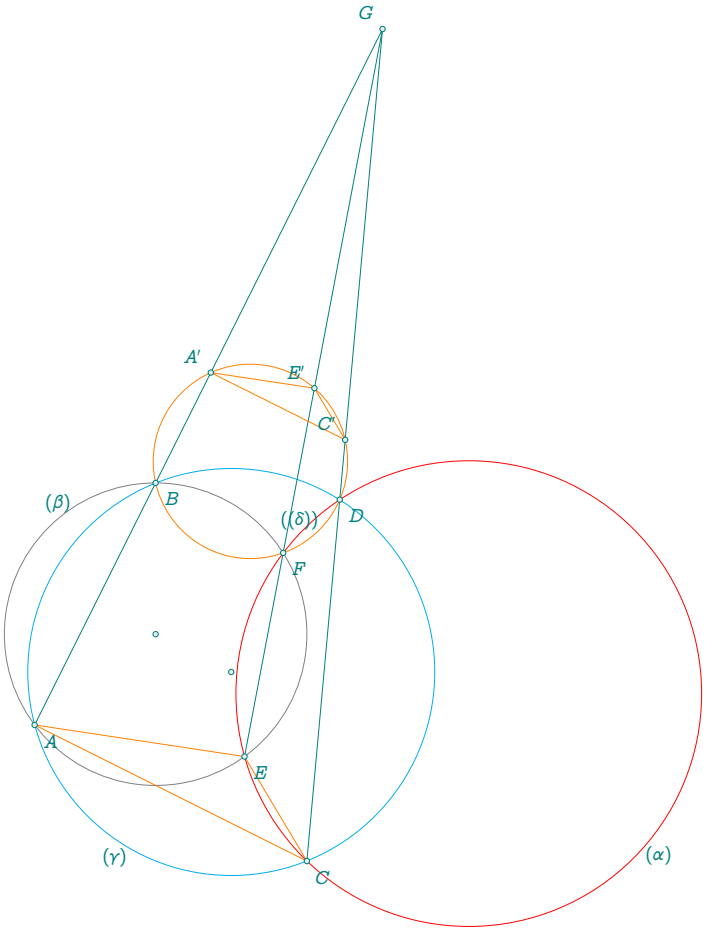
22.49 Tangent and chord

```
\begin{tkzelements}
   scale       = .8
   z.A         = point: new (0 , 0)
   z.B         = point: new (6 , 0)
   z.C         = point: new (1 , 5)
   z.Bp        = point: new (2 , 0)
   T.ABC       = triangle: new (z.A,z.B,z.C)
   L.AB        = line: new (z.A,z.B)
   z.O         = T.ABC.circumcenter
   C.OA        = circle: new (z.O,z.A)
   z.D         = C.OA: point (4.5)
   L.AO        = line: new (z.A,z.O)
   z.b1,z.b2   = get_points (C.OA: tangent_at (z.B))
   z.H         = L.AB: projection (z.O)
\end{tkzelements}
\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawCircle(O,A)
   \tkzDrawPolygon(A,B,C)
   \tkzDrawSegments[new](A,O B,O O,H A,D D,B)
   \tkzDrawLine(b1,b2)
   \tkzDrawPoints(A,B,C,D,H,O)
   \tkzFillAngles[green!20,opacity=.3](H,O,B A,C,B  A,B,b1)
   \tkzFillAngles[teal!20,opacity=.3](A,D,B b2,B,A)
   \tkzLabelPoints(A,B,C,D,H,O)
\end{tikzpicture}
```



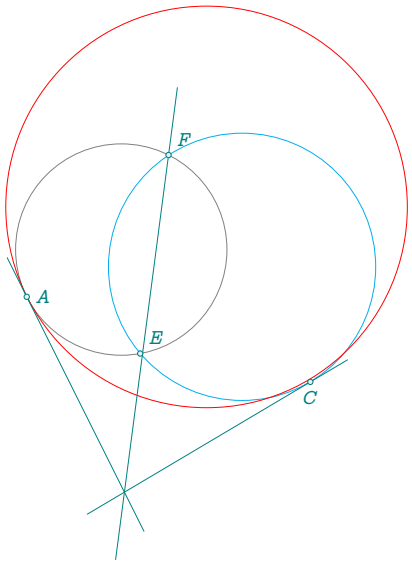
22.50 Three chords

```
\begin{tkzelements}
z.O  = point: new (0 , 0)
z.B  = point: new (0 , 2)
z.P  = point: new (1 , -.5)
C.OB = circle : new (z.O,z.B)
C.PB = circle : new (z.P,z.B)
_,z.A    = intersection (C.OB,C.PB)
z.D  = C.PB: point(0.85)
z.C  = C.PB: point(0.5)
z.E  = C.OB: point(0.6)
L.AB = line : new (z.A,z.B)
L.CD = line : new (z.C,z.D)
z.G  = intersection (L.AB,L.CD)
L.GE = line : new (z.G,z.E)
z.F,_    = intersection (L.GE,C.OB)
T.CDE    = triangle: new (z.C,z.D,z.E)
T.BFD    = triangle: new (z.B,z.F,z.D)
z.w  = T.CDE.circumcenter
z.x  = T.BFD.circumcenter
L.GB = line : new (z.G,z.B)
L.GE = line : new (z.G,z.E)
L.GD = line : new (z.G,z.D)
C.xB = circle : new (z.x,z.B)
C.xF = circle : new (z.x,z.F)
C.xD = circle : new (z.x,z.D)
z.Ap = intersection (L.GB,C.xB)
z.Ep,_   = intersection (L.GE,C.xF)
z.Cp,_   = intersection (L.GD,C.xD)
\end{tkzelements}
\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawCircles(O,B)
   \tkzDrawCircles[cyan](P,B)
   \tkzDrawCircles[red](w,E)
   \tkzDrawCircles[new](x,F)
   \tkzDrawSegments(A,G E,G C,G)
   \tkzDrawPolygons[new](A,E,C A',E',C')
   \tkzDrawPoints(A,...,G,A',E',C',O,P)
   \begin{scope}[font=\scriptsize]
   \tkzLabelPoints(A,...,F)
   \tkzLabelPoints[above left](G,A',E',C')
   \tkzLabelCircle[left](O,B)(30){$(\beta)$}
   \tkzLabelCircle[below](P,A)(40){$(\gamma)$}
   \tkzLabelCircle[right](w,C)(90){$(\alpha)$}
   \tkzLabelCircle[left](x,B)(-230){$((\delta))$}
   \end{scope}
\end{tikzpicture}
```


G
A'
E'
C'
(β)
B
(δ)
D
F
A
E
(γ)
C
(α)

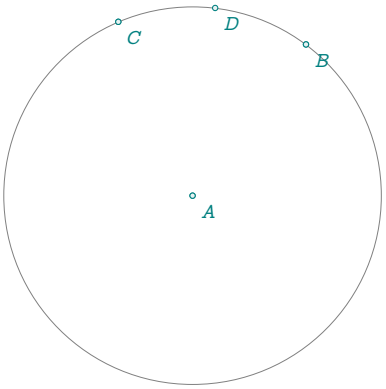
22.51 Three tangents

```
\begin{tkzelements}
   z.A    = point: new (-1 , 0)
   z.C    = point: new (4 , -1.5)
   z.E    = point: new (1 , -1)
   z.F    = point: new (1.5 , 2.5)
   T.AEF  = triangle : new (z.A,z.E,z.F)
   T.CEF  = triangle : new (z.C,z.E,z.F)
   z.w    = T.AEF.circumcenter
   z.x    = T.CEF.circumcenter
   C.wE   = circle : new (z.w,z.E)
   C.xE   = circle : new (z.x,z.E)
   L.Aw   = line : new (z.A,z.w)
   L.Cx   = line : new (z.C,z.x)
   z.G    = intersection (L.Aw,L.Cx)
   L.TA   = C.wE : tangent_at (z.A)
   L.TC   = C.xE : tangent_at (z.C)
   z.I    = intersection (L.TA,L.TC)
\end{tkzelements}
\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawCircles(w,E)
   \tkzDrawCircles[cyan](x,E)
   \tkzDrawCircles[red](G,A)
   \tkzDrawLines(A,I C,I F,I)
   \tkzDrawPoints(A,C,E,F)
   \tkzLabelPoints[right](A)
   \tkzLabelPoints[above right](E,F)
   \tkzLabelPoints[below](C)
\end{tikzpicture}
```



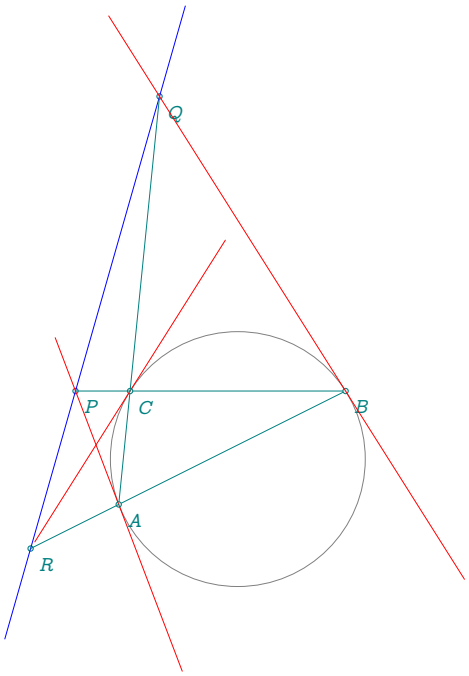
22.52 Midarc

```
\begin{tkzelements}
   z.A   = point:  new (-1,0)
   z.B   = point:  new (2,4)
   C.AB  = circle: new (z.A,z.B)
   z.C   =  z.A: rotation (math.pi/3,z.B)
   z.D   = C.AB: midarc (z.B,z.C)
\end{tkzelements}
\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawPoints(A,B,C)
   \tkzDrawCircles(A,B)
   \tkzDrawPoints(A,...,D)
   \tkzLabelPoints(A,...,D)
\end{tikzpicture}
```



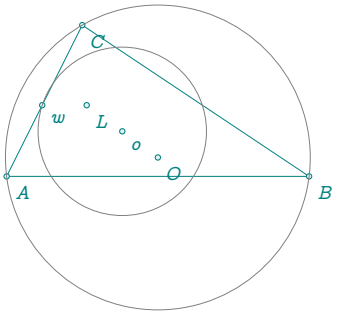
22.53 Lemoine Line without macro

```
\begin{tkzelements}
   scale       = 1.6
   z.A         = point: new (1,0)
   z.B         = point: new (5,2)
   z.C         = point: new (1.2,2)
   T           = triangle: new(z.A,z.B,z.C)
   z.O         = T.circumcenter
   L.AB        = line: new (z.A,z.B)
   L.AC        = line: new (z.A,z.C)
   L.BC        = line: new (z.B,z.C)
   C.OA        = circle: new (z.O,z.A)
   z.Ar,z.Al   = get_points (C.OA: tangent_at (z.A))
   z.Br,z.Bl   = get_points (C.OA: tangent_at (z.B))
   z.Cr,z.Cl   = get_points (C.OA: tangent_at (z.C))
   L.tA        = line: new (z.Ar,z.Al)
   L.tB        = line: new (z.Br,z.Bl)
   L.tC        = line: new (z.Cr,z.Cl)
   z.P         = intersection (L.tA,L.BC)
   z.Q         = intersection (L.tB,L.AC)
   z.R         = intersection (L.tC,L.AB)
\end{tkzelements}
\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawPolygon[teal](A,B,C)
   \tkzDrawCircle(O,A)
   \tkzDrawPoints(A,B,C,P,Q,R)
   \tkzLabelPoints(A,B,C,P,Q,R)
   \tkzDrawLine[blue](Q,R)
   \tkzDrawLines[red](Ar,Al Br,Q Cr,Cl)
   \tkzDrawSegments(A,R C,P C,Q)
\end{tikzpicture}
```



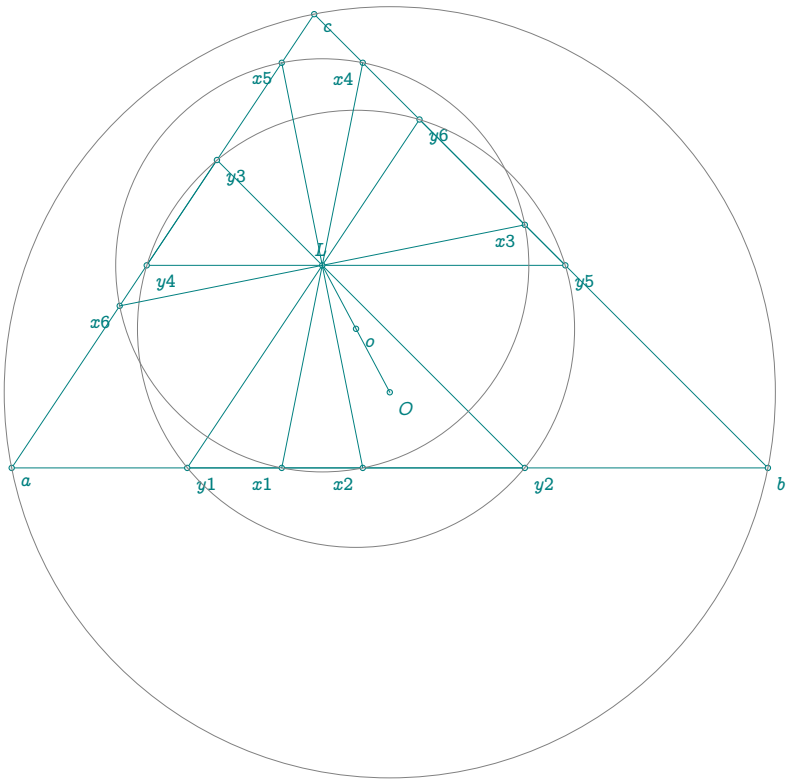
22.54 First Lemoine circle

```
\begin{tkzelements}
   z.A      = point:  new (1,1)
   z.B      = point:  new (5,1)
   z.C      = point:  new (2,3)
   T        = triangle: new (z.A,z.B,z.C)
   z.O      = T.circumcenter
   z.o,z.w  = get_points (T : first_lemoine_circle ())
   z.L      = T : lemoine_point ()
\end{tkzelements}

\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawPolygons(A,B,C)
   \tkzDrawPoints(A,B,C,o,w,O,L)
   \tkzLabelPoints(A,B,C,o,w,O,L)
   \tkzDrawCircles(o,w  O,A)
\end{tikzpicture}
```



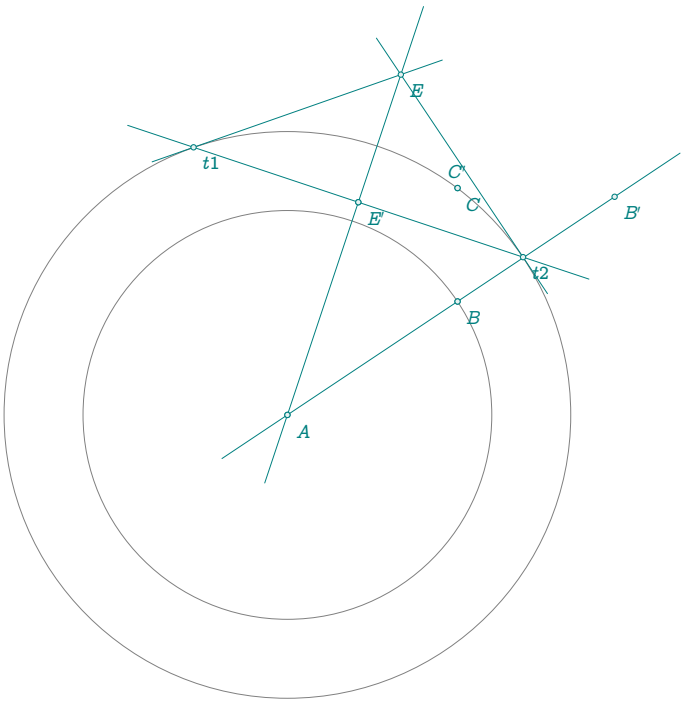
22.55 First and second Lemoine circles

```
\begin{tkzelements}
   scale             = 2
   z.a               = point:  new (0,0)
   z.b               = point:  new (5,0)
   z.c               = point:  new (2,3)
   T                 = triangle: new (z.a,z.b,z.c)
   z.O               = T.circumcenter
   z.o,z.p           = get_points (T : first_lemoine_circle ())
   L.ab              = line : new (z.a,z.b)
   L.ca              = line : new (z.c,z.a)
   L.bc              = line : new (z.b,z.c)
   z.L,z.x           = get_points (T : second_lemoine_circle ())
   C.first_lemoine   = circle : new (z.o,z.p)
   z.y1,z.y2         = intersection (L.ab,C.first_lemoine)
   z.y5,z.y6         = intersection (L.bc,C.first_lemoine)
   z.y3,z.y4         = intersection (L.ca,C.first_lemoine)
   C.second_lemoine  = circle : new (z.L,z.x)
   z.x1,z.x2         = intersection (L.ab,C.second_lemoine)
   z.x3,z.x4         = intersection (L.bc,C.second_lemoine)
   z.x5,z.x6         = intersection (L.ca,C.second_lemoine)
   L.y1y6            = line : new (z.y1,z.y6)
   L.y4y5            = line : new (z.y4,z.y5)
   L.y2y3            = line : new (z.y2,z.y3)
\end{tkzelements}
\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawPolygons(a,b,c y1,y2,y3,y4,y5,y6)
   \tkzDrawPoints(x1,x2,x3,x4,x5,x6,L)
   \tkzDrawPoints(a,b,c,o,O,y1,y2,y3,y4,y5,y6)
   \tkzLabelPoints[below right](a,b,c,o,O,y1,y2,y3,y4,y5,y6)
   \tkzLabelPoints[below left](x1,x2,x3,x4,x5,x6)
   \tkzLabelPoints[above](L)
   \tkzDrawCircles(L,x o,p O,a)
   \tkzDrawSegments(L,O x1,x4 x2,x5 x3,x6)
\end{tikzpicture}
```

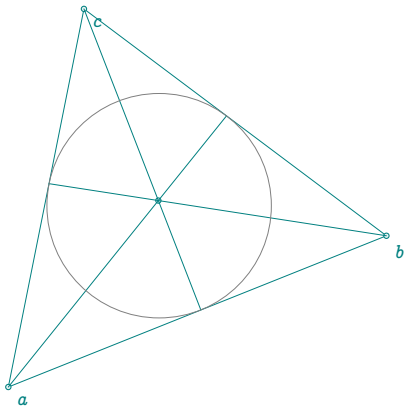


22.56 Inversion

```
 \begin{tkzelements}
   z.A         = point: new (-1,0)
   z.B         = point: new (2,2)
   z.C         = point: new (2,4)
   z.E         = point: new (1,6)
   C.AC        = circle:    new (z.A,z.C)
   L.Tt1,L.Tt2 = C.AC: tangent_from (z.E)
   z.t1        = L.Tt1.pb
   z.t2        = L.Tt2.pb
   L.AE        = line: new (z.A,z.E)
   z.H         = L.AE : projection (z.t1)
   z.Bp,
   z.Ep,
   z.Cp        = C.AC: inversion ( z.B, z.E, z.C )
\end{tkzelements}

\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawPoints(A,B,C)
   \tkzDrawCircles(A,C A,B)
   \tkzDrawLines(A,B' E,t1 E,t2 t1,t2 A,E)
   \tkzDrawPoints(A,B,C,E,t1,t2,H,B',E')
   \tkzLabelPoints(A,B,C,E,t1,t2,B',E')
   \tkzLabelPoints[above](C')
\end{tikzpicture}
```



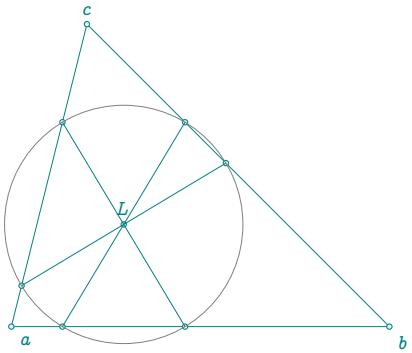
22.57 Gergonne point




```
\begin{tkzelements}
z.a  = point: new(1,0)
z.b  = point: new(6,2)
z.c  = point: new(2,5)
T    = triangle : new (z.a,z.b,z.c)
z.g  = T : gergonne_point ()
z.i  = T.incenter
z.ta,z.tb,z.tc = get_points (T :  intouch ())
\end{tkzelements}
\begin{tikzpicture}
\tkzGetNodes
\tkzDrawPolygons(a,b,c)
\tkzDrawPoints(a,b,c,g)
\tkzLabelPoints(a,b,c)
\tkzDrawSegments (a,ta b,tb c,tc)
\tkzDrawCircle(i,ta)
\end{tikzpicture}
```

22.58 Antiparallel through Lemoine point

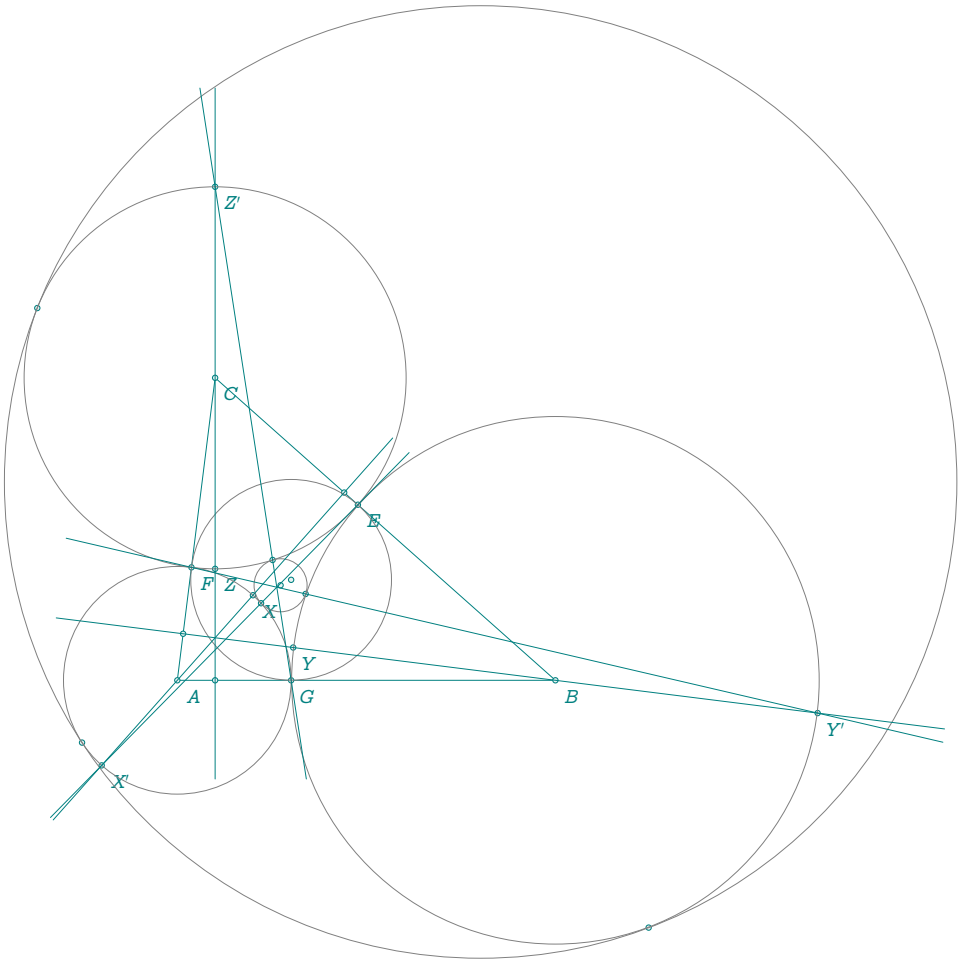
```
\begin{tkzelements}
   z.a         = point:  new (0,0)
   z.b         = point:  new (5,0)
   z.c         = point:  new (1,4)
   T           = triangle: new (z.a,z.b,z.c)
   z.L         = T : lemoine_point ()
   L.anti      = T : antiparallel (z.L,0)
   z.x_0,z.x_1 = get_points (L.anti)
   L.anti      = T : antiparallel (z.L,1)
   z.y_0,z.y_1 = get_points (L.anti)
   L.anti      = T : antiparallel (z.L,2)
   z.z_0,z.z_1 = get_points (L.anti)
\end{tkzelements}

\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawPolygons(a,b,c)
   \tkzDrawPoints(a,b,c,L,x_0,x_1,y_0,y_1,z_0,z_1)
   \tkzLabelPoints(a,b)
   \tkzLabelPoints[above](L,c)
   \tkzDrawSegments(x_0,x_1 y_0,y_1 z_0,z_1)
   \tkzDrawCircle(L,x_0)
\end{tikzpicture}
```



22.59 Soddy circle without function

```
\begin{tkzelements}
z.A = point : new ( 0   , 0  )
z.B = point : new ( 5   , 0  )
z.C = point : new ( 0.5 ,  4  )
T.ABC = triangle : new ( z.A,z.B,z.C )
z.I = T.ABC.incenter
z.E,z.F,z.G = T.ABC : projection (z.I)
C.ins = circle : new (z.I,z.E)
T.orthic = T.ABC : orthic ()
z.Ha,z.Hb,z.Hc = get_points (T.orthic)
C.CF = circle : new ( z.C , z.F )
C.AG = circle : new ( z.A , z.G )
C.BE = circle : new ( z.B , z.E )
L.Ah = line : new ( z.A , z.Ha )
L.Bh = line : new ( z.B , z.Hb )
L.Ch = line : new ( z.C , z.Hc )
z.X,z.Xp = intersection (L.Ah,C.AG)
z.Y,z.Yp = intersection (L.Bh,C.BE)
z.Z,z.Zp = intersection (L.Ch,C.CF)
L.XpE = line    : new (z.Xp,z.E)
L.YpF = line    : new (z.Yp,z.F)
L.ZpG = line    : new (z.Zp,z.G)
z.S = intersection (L.XpE,L.YpF)
z.Xi = intersection(L.XpE,C.AG)
z.Yi = intersection(L.YpF,C.BE)
_,z.Zi = intersection(L.ZpG,C.CF)
```

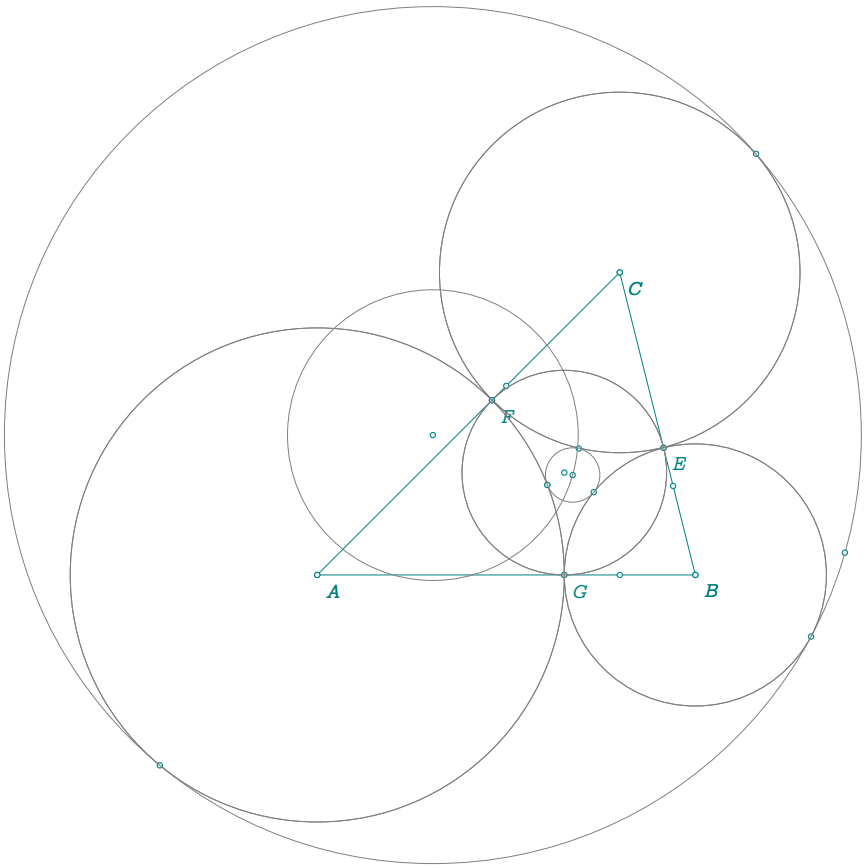
```
z.S = triangle : new (z.Xi,z.Yi,z.Zi).circumcenter
C.soddy_int = circle : new (z.S,z.Xi)
C.soddy_ext = C.ins : inversion (C.soddy_int)
z.w = C.soddy_ext.center
z.s = C.soddy_ext.through
z.Xip,z.Yip,z.Zip = C.ins : inversion (z.Xi,z.Yi,z.Zi)
\end{tkzelements}

\begin{tikzpicture}
\tkzGetNodes
\tkzDrawPolygon(A,B,C)
\tkzDrawPoints(A,B,C,E,F,G,Ha,Hb,Hc,X,Y,Z,X',Y',Z',Xi,Yi,Zi,I)
\tkzDrawPoints(Xi',Yi',Zi',S)
\tkzLabelPoints(A,B,C,E,F,G,X,Y,Z,X',Y',Z')
\tkzDrawCircles(A,G B,E C,F I,E S,Xi w,s)
\tkzDrawLines(X',Ha Y',Hb Z',Hc)
\tkzDrawLines(X',E Y',F Z',G)
\end{tikzpicture}
```



22.60 Soddy circle with function

```
\begin{tkzelements}
z.A = point : new ( 0  , 0  )
z.B = point : new ( 5  , 0  )
```

```
z.C = point : new (4 ,  4  )
T.ABC = triangle : new ( z.A,z.B,z.C )
z.I = T.ABC.incenter
z.E,z.F,z.G = T.ABC : projection (z.I)
T.orthic = T.ABC : orthic ()
z.Ha,z.Hb,z.Hc = get_points (T.orthic)
C.ins = circle : new (z.I,z.E)
z.s,z.xi,z.yi,z.zi = T.ABC : soddy_center ()
C.soddy_int = circle : new (z.s,z.xi)
C.soddy_ext = C.ins : inversion (C.soddy_int)
z.w = C.soddy_ext.center
z.t = C.soddy_ext.through
z.Xip,z.Yip,z.Zip = C.ins : inversion (z.xi,z.yi,z.zi)
  \end{tkzelements}

\begin{tikzpicture}
\tkzGetNodes
\tkzDrawPolygon(A,B,C)
\tkzDrawCircles(A,G B,E C,F I,E s,xi w,t)
\tkzDrawPoints(A,B,C,E,F,G,s,w,xi,t)
\tkzLabelPoints(A,B,C)
\tkzDrawPoints(A,B,C,E,F,G,Ha,Hb,Hc,xi,yi,zi,I)
\tkzDrawPoints(Xi',Yi',Zi')
\tkzLabelPoints(A,B,C,E,F,G)
\tkzDrawCircles(A,G B,E C,F I,E w,s)
\end{tikzpicture}
```

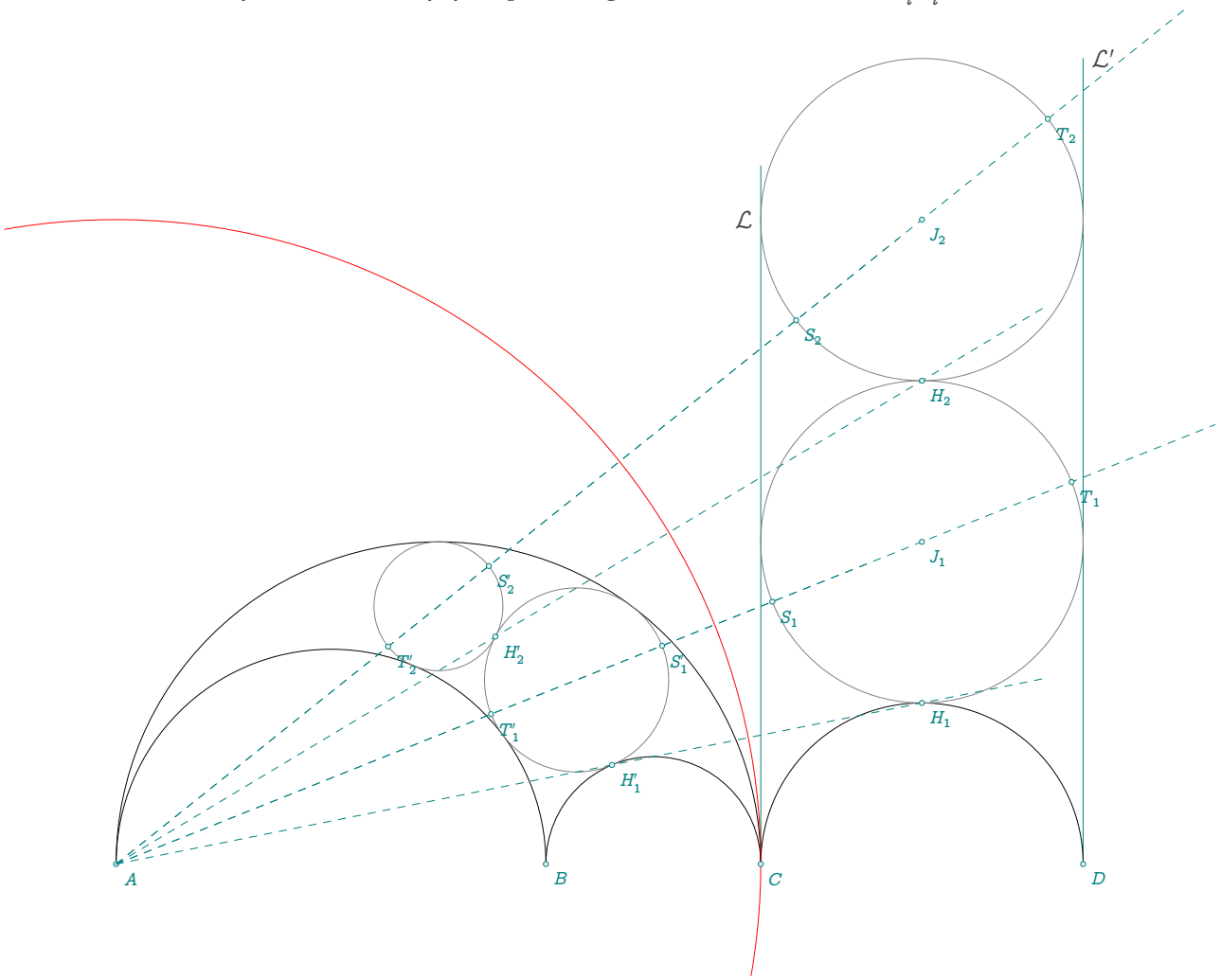


22.60.1 Pappus chain

Soit le point D appartenant à la droite (AC) tel que

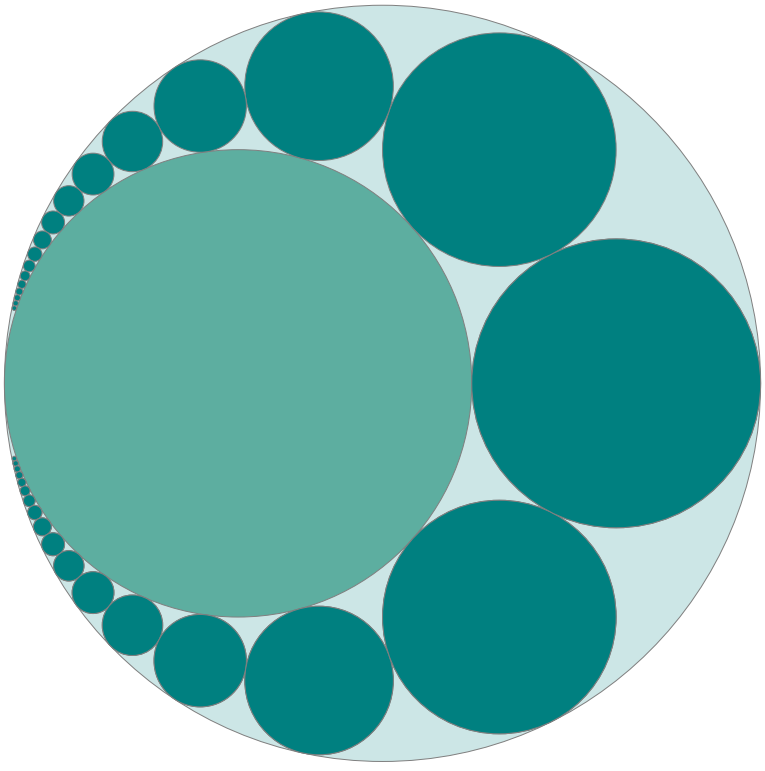
$$DB \cdot DA = AC^2$$

alors B est l'image de D dans l'inversion de centre A et puissance AC^2. Les demi-cercles de diamètre $[AB]$ et$[AC]$ passent par le pôle A. Ils ont pour images les demi-droites $\mathcal{L}'$ et $\mathcal{L}$.
Les cercles de centre J_i et de diamètre S_iT_i ont pour images les cercles de diamètre $S'_iT'_i$.




```
\begin{tkzelements}
   xC,nc     = 10,16
   xB        = xC/tkzphi
   xD        = (xC*xC)/xB
   xJ        = (xC+xD)/2
   r         = xD-xJ
   z.A       = point : new ( 0 , 0 )
   z.B       = point : new ( xB , 0)
   z.C       = point : new ( xC , 0)
   L.AC      = line : new (z.A,z.C)
   z.i       = L.AC.mid
   L.AB      = line:new (z.A,z.B)
   z.j       = L.AB.mid
   z.D       = point : new ( xD , 0)
```

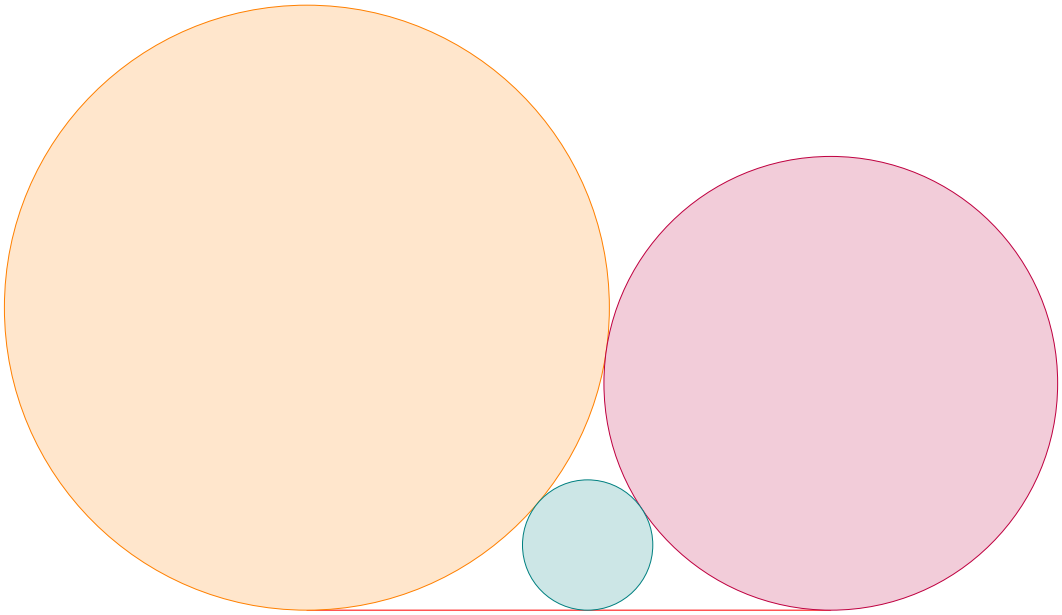
```
   C.AC     = circle: new (z.A,z.C)
   for i    = -nc,nc do
      z["J"..i]   = point: new (xJ,2*r*i)
      z["H"..i]   = point: new (xJ,2*r*i-r)
      z["J"..i.."p"], z["H"..i.."p"]   = C.AC : inversion (z["J"..i],z["H"..i])
      L.AJ        = line : new (z.A,z["J"..i])
      C.JH        = circle: new ( z["J"..i] ,  z["H"..i])
      z["S"..i], z["T"..i]             = intersection (L.AJ,C.JH)
      z["S"..i.."p"], z["T"..i.."p"]   = C.AC : inversion (z["S"..i],z["T"..i])
      L.SpTp      = line:new (  z["S"..i.."p"], z["T"..i.."p"])
      z["I"..i]   = L.SpTp.mid
    end
\end{tkzelements}

\def\nc{\tkzUseLua{nc}}

\begin{tikzpicture}[ultra thin]
   \tkzGetNodes
   \tkzDrawCircle[fill=teal!20](i,C)
   \tkzDrawCircle[fill=PineGreen!60](j,B)
   \foreach \i in {-\nc,...,0,...,\nc} {
   \tkzDrawCircle[fill=teal]({I\i},{S\i'})
  }
\end{tikzpicture}
```



22.61 Three Circles

```
\begin{tkzelements}
function threecircles(c1,r1,c2,r2,c3,h1,h3,h2)
```

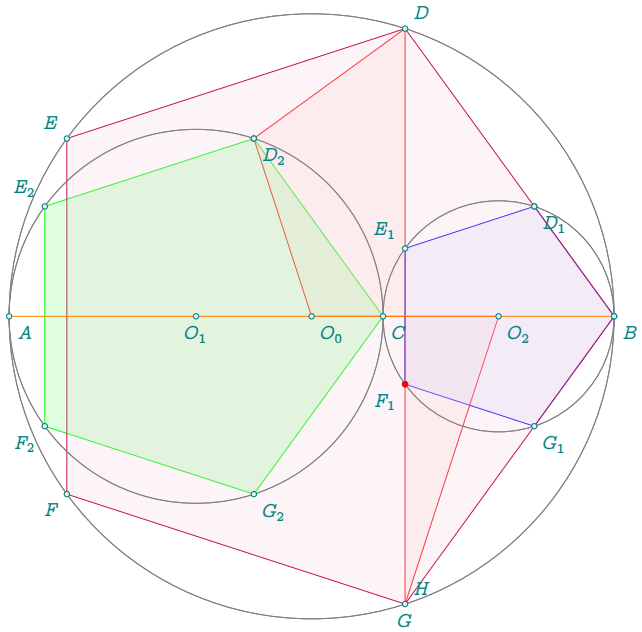
```
   local xk = math.sqrt (r1*r2)
   local cx = (2*r1*math.sqrt(r2))/(math.sqrt(r1)+math.sqrt(r2))
   local cy = (r1*r2)/(math.sqrt(r1)+math.sqrt(r2))^2
   z[c2] = point : new ( 2*xk , r2 )
   z[h2] = point : new (2*xk,0)
   z[c1] = point : new (0,r1)
   z[h1] = point : new (0,0)
   L.h1h2 = line: new(z[h1],z[h2])
   z[c3] = point : new (cx,cy)
   z[h3] = L.h1h2: projection (z[c3])
end
   threecircles("A",4,"B",3,"C","E","G","F")
\end{tkzelements}

\begin{tikzpicture}
\tkzGetNodes
\tkzDrawSegment[color = red](E,F)
\tkzDrawCircle[orange,fill=orange!20](A,E)
\tkzDrawCircle[purple,fill=purple!20](B,F)
\tkzDrawCircle[teal,fill=teal!20](C,G)
\end{tikzpicture}
```



22.62 pentagons in a golden arbelos

```
\begin{tkzelements}
z.A      = point: new (0 , 0)
z.B      = point: new (10 , 0)
L.AB     = line:  new ( z.A, z.B)
z.C      = L.AB : gold_ratio ()
L.AC     = line:  new ( z.A, z.C)
L.CB     = line:  new ( z.C, z.B)
z.O_0    = L.AB.mid
z.O_1    = L.AC.mid
z.O_2    = L.CB.mid
```

```
C.OQB    = circle: new ( z.O_Q, z.B)
C.O1C    = circle: new ( z.O_1, z.C)
C.O2B    = circle: new ( z.O_2, z.B)
z.M_Q    = C.O1C : external_similitude (C.O2B)
L.OQC    = line:new(z.O_Q,z.C)
T.golden = L.OQC : golden ()
z.L      = T.golden.pc
L.OQL    = line:new(z.O_Q,z.L)
z.D      = intersection (L.OQL,C.OQB)
L.DB     = line:new(z.D,z.B)
z.Z      = intersection (L.DB,C.O2B)
L.DA     = line:new(z.D,z.A)
z.I      = intersection (L.DA,C.O1C)
L.O2Z    = line:new(z.O_2,z.Z)
z.H      = intersection (L.O2Z,C.OQB)
C.BD     = circle:new (z.B,z.D)
C.DB     = circle:new (z.D,z.B)
_,z.G    = intersection (C.BD,C.OQB)
z.E      = intersection (C.DB,C.OQB)
C.GB     = circle:new (z.G,z.B)
_,z.F    = intersection (C.GB,C.OQB)
k        = 1/tkzphi^2
kk       = tkzphi
z.D_1,z.E_1,z.F_1,z.G_1 = z.B :   homothety (k, z.D,z.E,z.F,z.G)
z.D_2,z.E_2,z.F_2,z.G_2 = z.M_Q : homothety (kk,z.D_1,z.E_1,z.F_1,z.G_1)
\end{tkzelements}

\begin{tikzpicture}[scale=.8]
\tkzGetNodes
\tkzDrawPolygon[red](O_2,O_Q,I,D,H)
\tkzDrawPolygon[blue](B,D_1,E_1,F_1,G_1)
\tkzDrawPolygon[green](C,D_2,E_2,F_2,G_2)
\tkzDrawPolygon[purple](B,D,E,F,G)
\tkzDrawCircles(O_Q,B O_1,C O_2,B)
\tkzFillPolygon[fill=red!20,opacity=.20](O_2,O_Q,I,D,H)
\tkzFillPolygon[fill=blue!20,opacity=.20](B,D_1,E_1,F_1,G_1)
\tkzFillPolygon[fill=green!60,opacity=.20](C,D_2,E_2,F_2,G_2)
\tkzFillPolygon[fill=purple!20,opacity=.20](B,D,E,F,G)
\tkzDrawCircles(O_Q,B O_1,C O_2,B)
\tkzDrawSegments[new](A,B)
\tkzDrawPoints(A,B,C,O_Q,O_1,O_2,Z,I,H,B,D,E,F)
\tkzDrawPoints(D_1,E_1,F_1,G_1)
\tkzDrawPoints(D_2,E_2,F_2,G_2)
\tkzDrawPoints[red](F_1)
\tkzLabelPoints(A,B,C,O_Q,O_2)
\tkzLabelPoints[below](O_1,G)
\tkzLabelPoints[above right](D,H)
\tkzLabelPoints[above left](E,E_1,E_2)
\tkzLabelPoints[below left](F,F_1,F_2)
\tkzLabelPoints(D_1,G_1)
\tkzLabelPoints(D_2,G_2)
\end{tikzpicture}
\vspace{\fill}
```


D
E
D_2
E_2
E_1
D_1
A
O_1
O_0
C
O_2
B
F_1
F_2
G_1
F
G_2
H
G

Index

23 Cheat_sheet

r denotes a real number, d a positive real number, n an integer, an an angle, b a boolean, s a character string, pt a point, v variable, L a straight line, C a circle, T a triangle, E an ellipse, V a vector,Q a quadrilateral, P a parallelogram, R a rectangle, S a square, RP a regular polygon, O an object (pt, L,C,T), . . a list of points or an object, < > optional argument.

point

Attributes table(1)

re	-> r
im	-> r
type	-> s
argument	-> r
modulus	-> d

Methods table(2)

+ - * /	-> pt
..	-> r
^	-> r
conj	-> pt
abs	-> r
mod	-> d
norm	-> d
arg	-> d
get	-> r,r
sqrt	-> pt
new	-> pt
polar	-> pt
polar_deg	-> pt
north(d)	-> pt
south(d)	-> pt
east(d)	-> pt
west(d)	-> pt
normalize(pt)	-> pt
symmetry (...)	-> O
rotation (an , ...)	-> O
homothety (r , ...)	-> O
orthogonal(d)	-> pt
at()	-> pt

line

Attributes table(3)

pa,pb	-> pt
type	-> s
mid	-> pt
north_pa	-> pt
north_pb	-> pt
south_pa	-> pt
south_pb	-> pt
east	-> pt
west	-> pt
slope	-> r
length	-> d
vec	-> V

Methods table(6)

new (pt,pt)	-> d
distance (pt)	-> d
slope ()	-> r
in_out (pt)	-> b
in_out_segment (pt)	-> b
barycenter (r,r)	-> pt
point (t)	-> pt
midpoint ()	-> pt
harmonic_int (pt)	-> pt
harmonic_ext (pt)	-> pt
harmonic_both (d)	-> pt
gold_ratio()	-> pt
normalize ()	-> pt
normalize_inv ()	-> pt
_north_pa (d)	-> pt
_north_pb (d)	-> pt
_south_pa (d)	-> pt
_south_pb (d)	-> pt
_east (d)	-> pt
_west (d)	-> pt
translation (...)	-> O
projection (...)	-> O
reflection (...)	-> O
ll_from (pt)	-> L
ortho_from (pt)	-> L
mediator ()	-> L
circle ()	-> C
circle_swap ()	-> C
diameter ()	-> C
apollonius (r)	-> C
equilateral (<swap>)	-> T
isosceles (an,<swap>)	-> T
school ()	-> T
two_angles (an,an)	-> T
half ()	-> T
sss (r,r,r)	-> T
sas (r,an)	-> T
ssa (r,an)	-> T
gold (<swap>)	-> T
euclide (<swap>)	-> T
golden (<swap>)	-> T
divine ()	-> T
cheops ()	-> T
pythagoras ()	-> T
sublime ()	-> T
egyptian ()	-> T
square (<swap>)	-> T
report (r,pt)	-> T

triangle

Attributes table(9)

pa,pb,pc	-> pt
circumcenter	-> pt
centroid	-> pt
incenter	-> pt
eulercenter	-> pt
orthocenter	-> pt
spiekercenter	-> pt
type	-> s
a	-> d
b	-> d
c	-> d
ab	-> L
bc	-> L
ca	-> L
alpha	-> r
beta	-> r
gamma	-> r

Methods table(10)

new (pt,pt,pt)	-> pt
trilinear (r,r,r)	-> pt
barycentric (r,r,r)	-> pt
bevan_point ()	-> pt
mittenpunkt_point ()	-> pt
gergonne_point ()	-> pt
nagel_point ()	-> pt
feuerbach_point ()	-> pt
lemoine_point()	-> pt
symmedian_point()	-> pt
spieker_center()	-> pt
barycenter (r,r,r)	-> pt
base (u,v)	-> pt
euler_points ()	-> pt
nine_points ()	-> pt
point (t)	-> pt
soddy_center ()	-> pt
euler_line ()	-> L
symmedian_line (n)	-> L
altitude (n)	-> L
bisector (n)	-> L
bisector_ext(n)	-> L
antiparallel(pt,n)	-> L
euler_circle ()	-> C
circum_circle()	-> C
in_circle ()	-> C
ex_circle (n)	-> C
first_lemoine_circle()	-> C
second_lemoine_circle()	-> C
spieker_circle()	-> C
soddy_circle ()	-> C
orthic()	-> T
medial()	-> T
incentral()	-> T
excentral()	-> T
intouch()	-> T
contact()	-> T
extouch()	-> T
feuerbach()	-> T
anti ()	-> T

```
tangential ()              -> T
cevian (pt)                -> T
symmedian ()               -> T
euler ()                   -> T
projection (pt)     -> pt,pt,pt
parallelogram ()           -> pt
area ()                    -> d
barycentric_coordinates(pt)
-> r,r,r
in_out (pt)                -> pt
check_equilateral ()       -> b
```

circle

Attributes table(7)

```
center                     -> pt
through                    -> pt
north                      -> pt
south                      -> pt
east                       -> pt
west                       -> pt
opp                        -> pt
type                       -> s
radius                     -> d
ct                         -> L
```

Methods table(8)

```
new (pt,pt)                -> C
radius (pt, r)             -> C
diameter (pt,pt)           -> C
in_out (pt)                -> b
in_out_disk (pt)           -> b
circles_position (C)       -> s
power (pt)                 -> r
antipode (pt)              -> pt
midarc (pt,pt)             -> pt
point (r)                  -> pt
random_pt (lower, upper)   -> pt
internal_similitude (C)    -> pt
external_similitude (C)    -> pt
radical_center(C,<C>)      -> pt
tangent_at (pt)            -> L
radical_axis (C)           -> L
radical_circle(C,<C>)      -> C
orthogonal_from (pt)       -> C
orthogonal_through(pt,pt)  -> C
midcircle(C)               -> C
external_tangent(C)        -> L,L
internal_tangent(C)        -> L,L
common_tangent(C)          -> L,L
tangent_from (pt)          -> L,L
inversion (...)            -> O
```

ellipse

Attributes table(12)

```
center                     -> pt
vertex                     -> pt
covertex                   -> pt
Fa                         -> pt
Fb                         -> pt
north                      -> pt
south                      -> pt
east                       -> pt
west                       -> pt
Rx                         -> d
Ry                         -> d
slope                      -> r
type                       -> s
```

Methods table(12)

```
new (pt,pt,pt)             -> E
foci (pt,pt,pt)            -> E
radii (pt,r,r,an)          -> E
in_out (pt)                -> b
tangent_at (pt)            -> L
tangent_from (pt)          -> L
point (r)                  -> pt
```

square

Attributes table(15)

```
pa,pb,pc,pd                -> pt
type                       -> s
side                       -> d
center                     -> pt
exradius                   -> d
inradius                   -> d
diagonal                   -> d
proj                       -> pt
ab bc cd da                -> L
ac bd                      -> L
```

Methods table(16)

```
new (pt,pt,pt,pt)          -> S
rotation (pt,pt)           -> S
side (pt,pt,<swap>)        -> S
```

rectangle

Attributes table(17)

```
pa,pb,pc,pd                -> pt
type                       -> s
center                     -> pt
exradius                   -> d
length                     -> r
width                      -> r
diagonal                   -> d
ab bc cd da                -> L
ac bd                      -> L
```

Methods table(18)

```
new (pt,pt,pt,pt)          -> R
angle (pt,pt,an)           -> R
gold  (pt,pt,<swap>)       -> R
diagonal  (pt,pt,<swap>)   -> R
side  (pt,pt,r,<swap>)     -> R
get_lengths ()             ->r,r
```

quadrilateral

Attributes table(13)

```
pa,pb,pc,pd                -> pt
ab bc cd da                -> L
ac bd                      -> L
type                       -> s
i                          -> pt
g                          -> pt
a b c d                    -> r
```

Methods table(14)

```
new (pt,pt,pt,pt)          -> Q
iscyclic ()                -> b
```

parallelogram

Attributes table(19)

```
pa,pb,pc,pd                -> pt
ab bc cd da                -> L
ac bd                      -> L
type                       -> s
center                     -> pt
```

Methods table(20)

```
new (pt,pt,pt,pt)          ->
fourth (pt,pt,pt)          ->
```

Regular_polygon

Attributes table(21)

```
center                     -> pt
through                    -> pt
circle                     -> C
type                       -> s
side                       -> d
exradius                   -> d
inradius                   -> d
proj                       -> pt
nb                         -> i
angle                      -> an
```

Methods table(22)

```
new (pt,pt,n)              -> PR
incircle ()                -> C
name (s)                   -> ?
```

vector

Attributes table(23)

```
pa,pb                      -> pt
type                       -> s
norm                       -> d
slope                      -> r
```

Methods table(24)

```
new (pt,pt)                -> V
+ - *                      -> pt
normalize (V)              -> V
orthogonal (d)             -> V
scale (r)                  -> V
at (pt)                    -> V
```

Misc.

Attributes table(25)

```
scale (default =1)         -> r
tkzphi                     -> r
tkzinvphi                  -> r
tkzsqrtphi                 -> r
```

```
tkz_epsilon (default=1e-8)-> r
length                    -> d
islinear(pt,pt,pt)        -> b
isortho(pt,pt,pt)         -> b
\tkzUseLua{v}             -> ?
value{r}                  -> r
real                      -> r
```

```
angle_normalize (an)      -> an
barycenter (...)          -> pt
bisector (pt,pt,pt)       -> L
bisector_ext (pt,pt,pt)   -> L
altitude (pt,pt,pt)       -> L
midpoint (pt,pt)          -> pt
equilateral (pt,pt)       -> T
```

```
format_number(r,n)        -> r
```

Macros

```
\tkzDN[n]{r}              -> r
\tkzDrawLuaEllipse((pt,pt,pt))
```