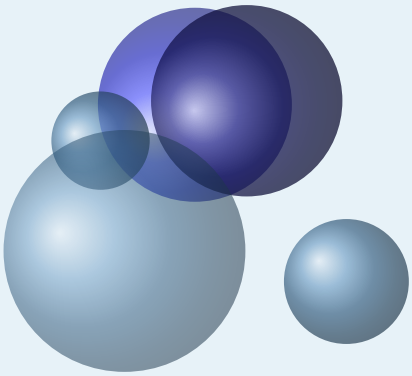
tkz-elements 3.00c

AlterMundus

tkz-elements 3.00c

Euclidean Geometry



Alain Matthes

November 23, 2024 Documentation V.3.00c

http://altermundus.fr

tkz-elements

Alain Matthes

AlterMundus

☞ This document compiles some notes about **tkz-elements**, the initial version of a Lua library designed to perform all the necessary calculations for defining objects in Euclidean geometry figures. Your document must be compiled using LuaLaTeX.
With tkz-elements, definitions and calculations are exclusively conducted using Lua.
The primary programming approach offered is oriented towards object programming, utilizing object classes such as point, line, triangle, circle, and ellipse. Currently, after the calculations are completed, tkz-euclide or TikZ is used for drawing purposes.
I discovered Lua and object-oriented programming while developing this package, so it's highly likely that I've made a few mistakes. If you'd like to contribute to the development of this package or provide advice on how to proceed, please contact me via email.

☞ Acknowledgements : I received much valuable advices, remarks, corrections from Nicolas Kisselhoff, David Carlisle, Roberto Giacomelli and Qrrbrbirlbel.
Special thanks to Wolfgang Büchel for his invaluable contribution in correcting the examples.

☞ I would also like to extend my gratitude to Eric Weisstein, creator of MathWorld.

☞ You can find some examples on my site and a french documentation: altermundus.fr.

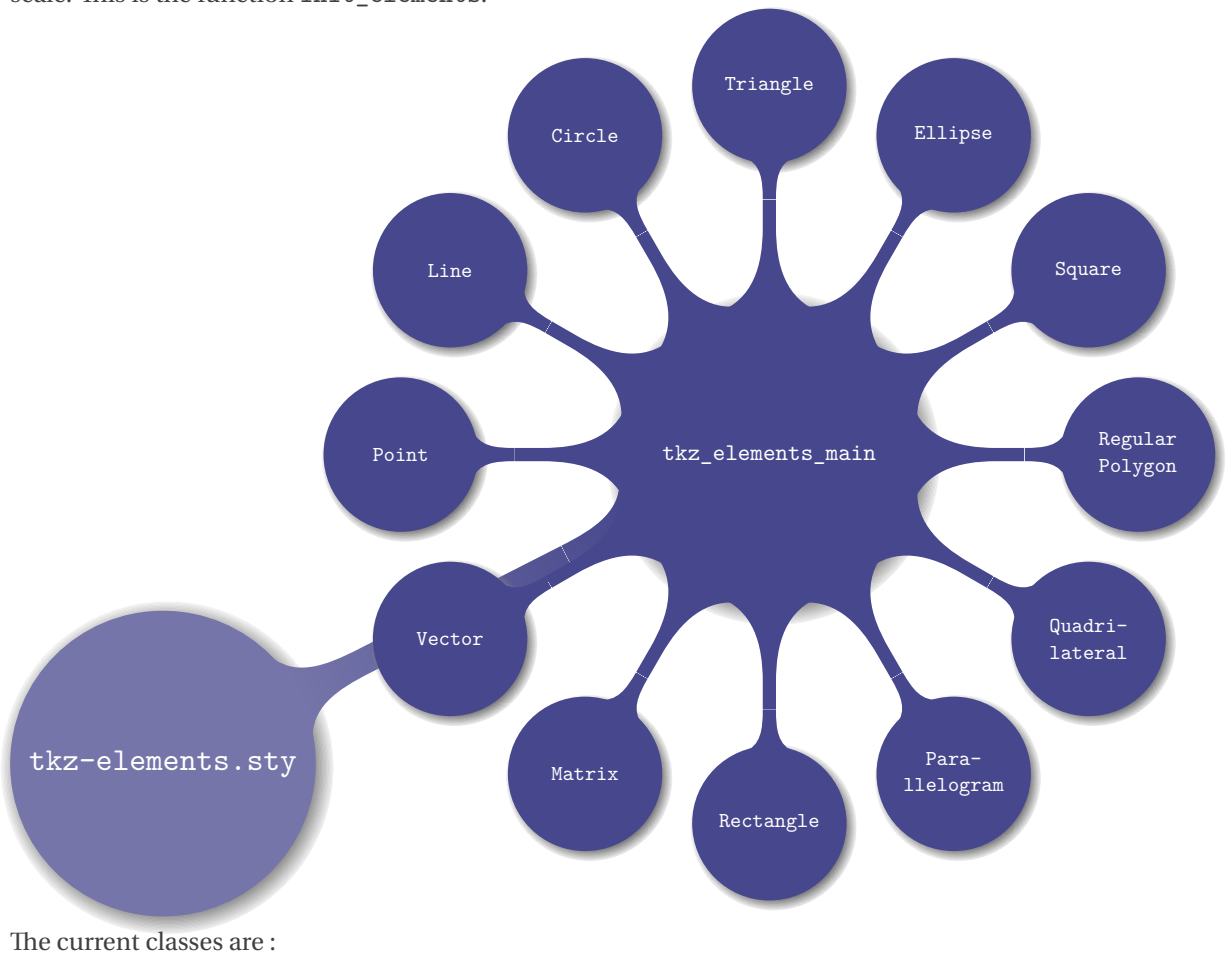
Contents

1 News

The documentation you are reading corresponds to the latest version 3.0 of tkz-elements. This version introduces an important new feature: the code Lua part of the code can now be processed using the \directlua primitive of LuaLaTeX. See the examples given in the Transfers section.
This introduces a slight complication whatever the method used to execute the Lua code. If you want to use the **tkzelements** environment, then you need to load the luacode package. If you prefer to use the \directlua primitive, you'll need to delete and reset the tables and scale with the init_elements function.
Some complex examples require the use of the \directlua primitive.

2 Structure

After loading the package, the scale is initialized to 1, and then all values in various tables are cleared.
The package defines two macros \tkzGetNodes and \tkzUseLua.
Additionally, the package loads the file tkz_elements_main.lua. This file initializes all the tables that will be used by the modules in which the classes are defined. In this file, a function is defined to reset all tables and the scale. This is the function init_elements.



The current classes are :
point (z) ; line (L) ; circle (C) ; triangle (T) ; ellipse (E) ; quadrilateral (Q) ; square (S) ; rectangle (R) ; parallelogram (P) ; regular_polygon (RP); vector (V) and matrix (M).
If name is name of a class, you can find its definition in the file tkz_elements_name.lua.

3 Why tkz-elements?

3.1 Calculation accuracy

3.1.1 Calculation accuracy in TikZ

With Ti*k*Z, the expression `veclen(x,y)` calculates the expression $\sqrt{x^2+y^2}$. This calculation is achieved through a polynomial approximation, drawing inspiration from the ideas of `Rouben Rostamian`.

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

☞ $\sqrt{65^2+72^2} \approx 96.9884$.

3.1.2 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, though insignificant when it comes to the placement of an object on a page by a hundredth of a point, becomes problematic for the results of mathematical demonstrations. Moreover, these inaccuracies can accumulate and lead to erroneous constructions.

To address this lack of precision, I initially introduced the `fp`, followed by the package `xfp`. More recently, with the emergence of LuaLaTeX, I incorporated a **Lua** option aimed at performing calculations with **Lua**.
This was the primary motivation behind creating the package, with the secondary goal being the introduction of object-oriented programming (OOP) and simplifying programming with Lua. The concept of OOP persuaded me to explore its various possibilities further.
At that time, I had received some Lua programming examples from `Nicolas Kisselhoff`, but I struggled to understand the code initially, so I dedicated time to studying Lua patiently. Eventually, I was able to develop **tkz-elements**, incorporating many of his ideas that I adapted for the package.

3.1.3 Using objects

Subsequently, I came across an article by `Roberto Giacomelli`[1] on object-oriented programming using **Lua** and Ti*k*Z tools. This served as my second source of inspiration. Not only did this approach enable programming to be executed step-by-step, but the introduction of objects facilitated a direct link between the code and geometry. As a result, the code became more readable, explicit, and better structured.

3.1.4 Example: Apollonius circle

> Problem: The objective is to identify an inner tangent circle to the three exinscribed circles of a triangle.

For additional details, refer to MathWorld for more details.

1 Grafica ad oggetti con LuaTEX

This example served as my reference for testing the tkz-euclide package. Initially, with my first methods and the tools available to me, the results lacked precision. However, with tkz-elements, I now have access to more powerful and precise tools that are also easier to use.
The fundamental principles of figure construction with **tkz-euclide** remain intact: definitions, calculations, tracings, labels, as well as the step-by-step programming, mirroring the process of construction with a ruler and compass.
This version utilizes the simplest construction method made possible by Lua.

```
\directlua{
  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)
}
```

The creation of an object encapsulates its attributes (its characteristics) and methods (i.e. the actions that are specific to it). Subsequently, it is assigned a reference (a name) which is linked to the object using a table. This table functions as an associative array that links the reference, called a `key`, to a `value`, in this case, the object. Further elaboration on these notions will be provided later.
For instance, let `T` be a table associating the object `triangle` with the key `ABC`. `T.ABC` is also a table, and its elements are accessed using keys that correspond to attributes of the triangle. These attributes have been defined within 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 ()
```

In this context, `excentral` is a method associated with the `T.ABC` object. It defines the triangle formed by the centers of the exinscribed circles.
Of particular importance are two lines of code. The first one below demonstrates that the exceptional precision provided by Lua allows for the definition of a radius through a complex calculation. The radius of the radical circle is determined by $\sqrt{\Pi(S,C(Ja,Ea))}$ (square root of the power of point S with respect to the exinscribed circle with center `Ja` passing through `Ea`).

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

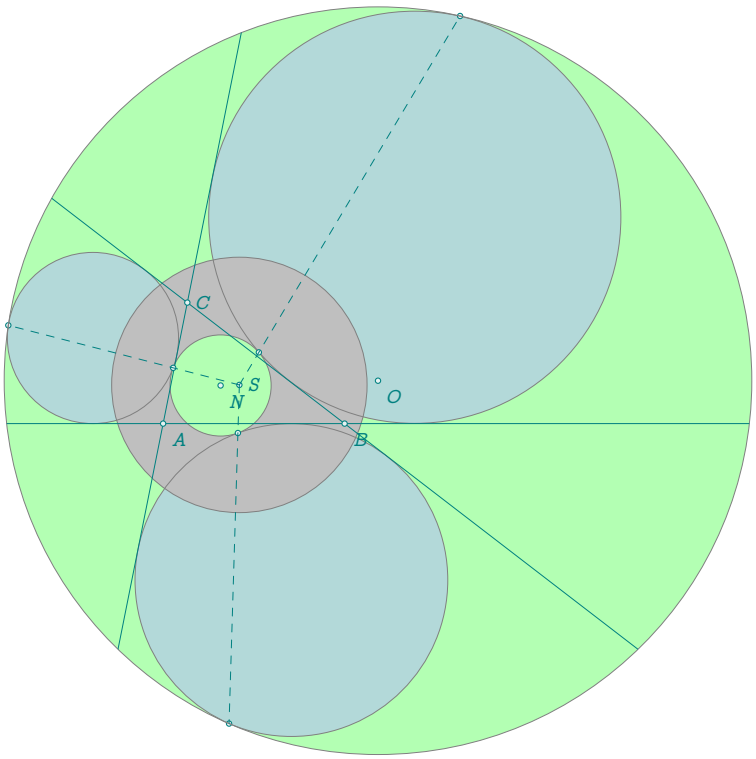
```
C.ortho  = circle: radius (z.S,math.sqrt(C.JaEa: power(z.S)))
```

Lastly, it's worth noting that the inversion of the Euler circle with respect to the radical circle yields the Apollonius circle[3]. This transformation requires an object as a 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 accomplish this, nodes need to be created. This is the role of the macro . Refer to 7.1.1
The subsequent section exclusively deals with drawings, and is managed 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)
   \tkzDrawCircles(Ja,Ea Jb,Eb Jc,Ec S,a O,xa N,Ea)
   \tkzClipCircle(O,xa)
   \tkzDrawLines[add=3 and 3](A,B A,C B,C)
   \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.

4 Presentation

4.1 With Lua

The primary function of tkz-elements is to calculate dimensions and define points, which is achieved using Lua. You can view tkz-elements as a kernel that is utilized either by tkz-euclide or by TikZ. The lua code can be implemented immediately using the **\directlua** primitive, or it can take place within a **tkzelements** environment which is based on **luacode**. In the latter case, you need to load the luacode package. In the first case, if you create a complex document, you'll be able to reset the tables and scale with the init_elements function.

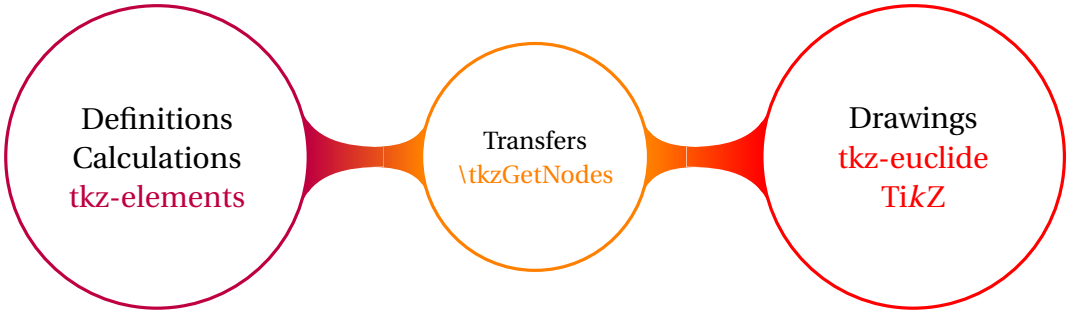
The key points are:

- The source file must be ☞ UTF8 encoded.
- Compilation is done with ☞ LuaLaTeX.
- You need to load TikZ or tkz-euclide and tkz-elements.
- Definitions and calculations are performed in an (orthonormal) Cartesian coordinate system, using Lua with the macro **\directlua** or within the tkzelements environment.

On the right, you can see the minimum template.
The code is divided into two parts, represented by lua code, argument to the primitive \directlua and the environment **tikzpicture**. In the first part, you place your Lua code, while in the second, you use tkz-euclide commands.

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

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

 ...code...
}

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

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

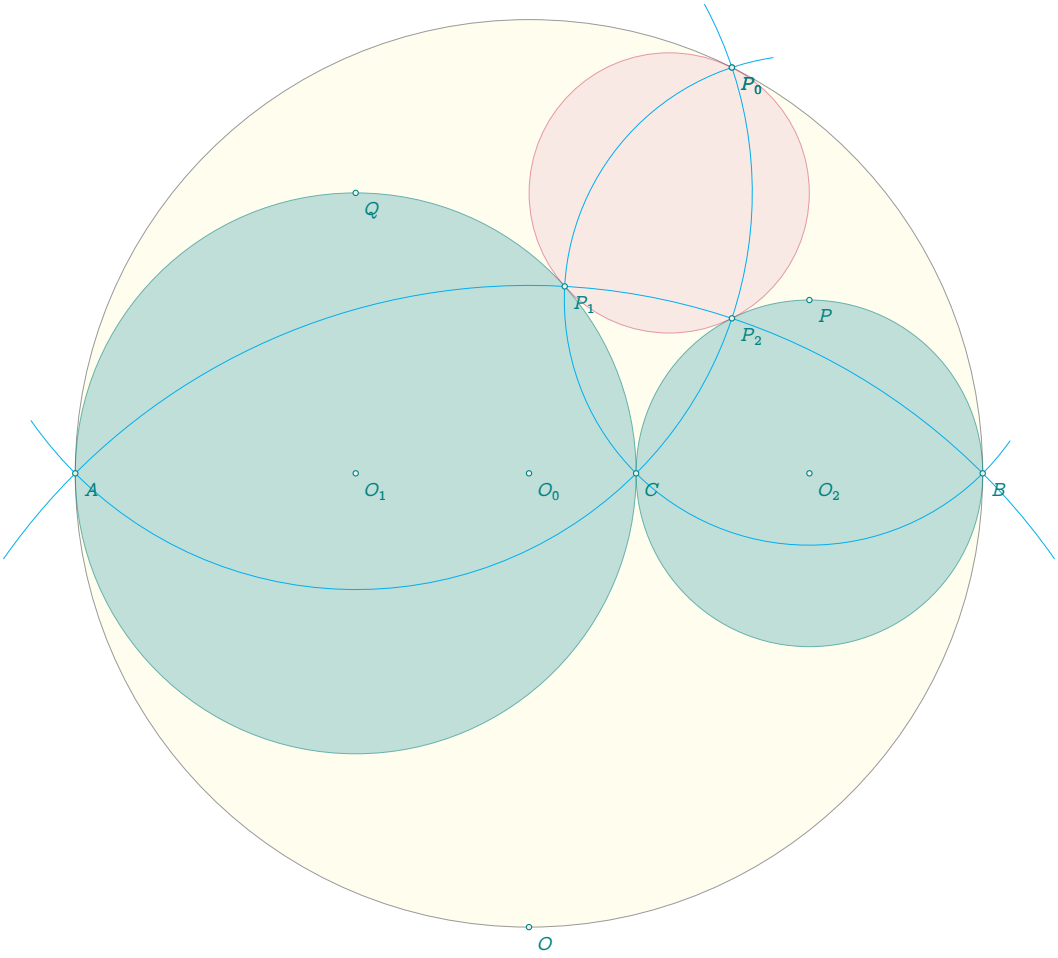
4.2 The main process



After obtaining all the necessary points for the drawing, they must be transformed into **nodes** so that TikZ or tkz-euclide can render the figure. This is accomplished using the macro **\tkzGetNodes**. This macro iterates through all the elements of the table z using the key (which is essentially the name of the point) and retrieves the associated values, namely the coordinates of the point (node).

4.3 Complete example: Pappus circle

4.3.1 The figure



4.3.2 The code

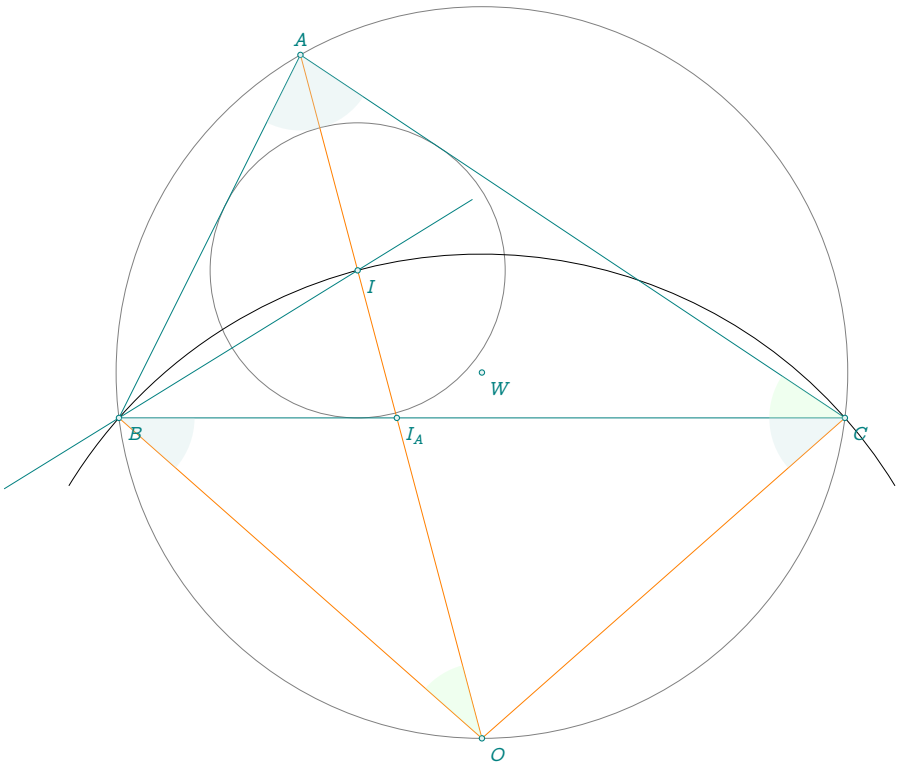
```
% !TEX TS-program = lualatex
\documentclass{article}
\usepackage{tkz-euclide}
\usepackage{tkz-elements}
\begin{document}

\directlua{
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                 %  "north" 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"
}

\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}
```

4.4 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}
\directlua{
   z.A      = point: new (2 , 4)
   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
}
```



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}
```

5 Writing Convention

5.1 Miscellaneous

- Numerical variable: the writing conventions for real numbers are the same as for Lua.
- Complex numbers: Similar to 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))`.(Refer 23.3)
- Boolean: you can write `bool = true` or `bool = false` then with Lua you can use the code :

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

and 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
```

5.2 Assigning a Name to a Point

At present, the only obligation is to store the points in the table `z` [4] if you intend to use them in Ti*k*Z or `tkz-euclide`. f a point will not be used, you can designate it as you wish while adhering to Lua conventions.
Points in the lua code must follow a convention in the form `z.name`, where `name` represents the name of the corresponding **node**.
As for the conventions for designating `name` you must adhere to Lua conventions in particular cases.

1. The use of prime can be 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. Alternatively, for a more explicit code, suppose you want to designate a point as "euler". You could, for example, 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"]` = …

5.3 Assigning a Name to Other Objects

You have the flexibility to assign names to objects other than points. However, it's advisable to adhere to certain conventions to enhance code readability. For my examples, I've chosen the following conventions: first of all, I store the objects in tables: `L` for lines and segments, `C` for circles, `T` for triangles, `E` for ellipses.

- For lines, I use the names of the two points they pass through. For example, if a line passes through points A and B, I name the line `L.AB`.
- Circles are stored in table named `C`. For example, I name `C.AB` the circle of center A passing through B. Other names like C.euler or C.external are also acceptable.
- Triangles are stored in table named `T`. For example, I name `T.ABC` the triangle whose vertices are A, B and C. However, names like `T.feuerbach` are also acceptable.
- Ellipses are stored in table named `E`. For ellipses, I name `E.ABC` the ellipse with center A through vertex B and covertex C.

Adhering to these conventions can help improve the readability of the code.

5.4 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).
  ```

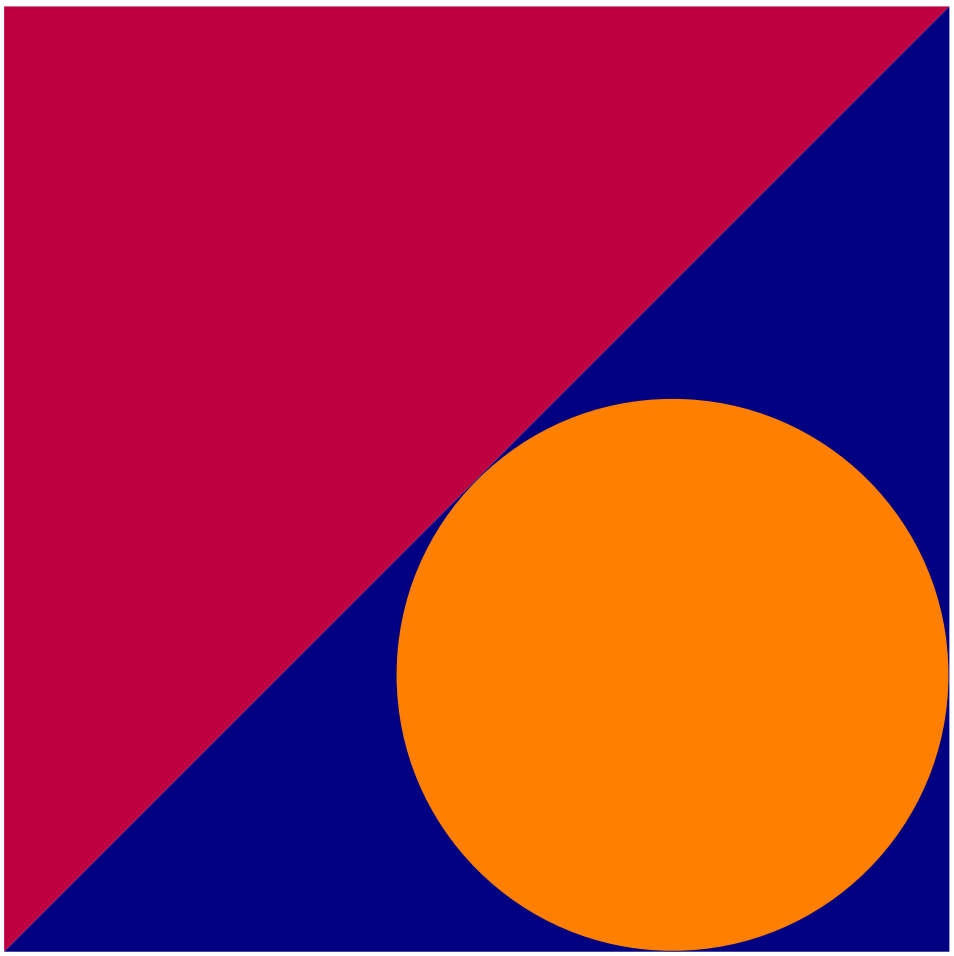
6 Work organization

Here's a sample organization.
The line `% !TEX TS-program = lualatex` ensures that you compile with LuaLaTeX. The `standalone` class is useful, as all you need to do here is create a figure.
You can load **`tkz-euclide`** in three different ways. The simplest is `\usepackage[mini]{tkz-euclide}` and you have full access to the package. You also have the option to use the `lua` option. This will allow you, if you want to perform calculations outside of **`tkz-elements`**, to obtain them using `lua`. Finally, the recommended method is to use the `mini` option. This allows you to load only the modules necessary for drawing. You can still optionally draw using Ti*k*Z.
The package `ifthen` is useful if you need to use some Boolean.
While it's possible to leave the Lua code in the macro `directlua`, externalizing this code has its advantages.
The first advantage is that, if you use a good editor, you have a better presentation of the code. Styles differ between Lua and LaTeX, making 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 incorrectly comment the code. For Lua code, you comment lines with `--` (double minus sign), whereas for LaTeX, you comment with `%`.
A third advantage is that the code can be reused.

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

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

\begin{document}
\directlua{
 scale = 1.25
 dofile ("sangaku.lua")
}

\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)
```



6.1 Scale problem

If necessary, it's better to perform scaling in the Lua section. This approach tends to be more accurate. However, there is a caveat to be aware of. I've made it a point to avoid using numerical values in my codes whenever possible. Generally, 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 account for scaling, use the function `value (v)` .

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

6.2 Code presentation

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

7 Transfers

7.1 From Lua to tkz-euclide or TikZ

In this section, we'll explore how to transfer points, booleans, and numerical values.

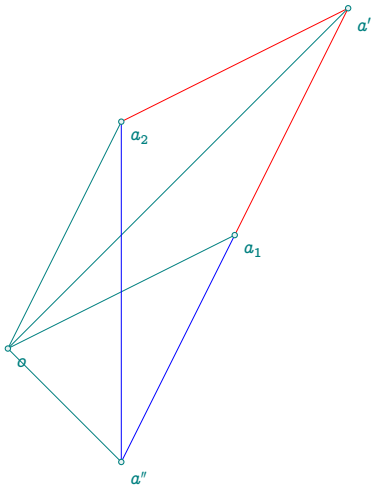
7.1.1 Points transfer

The necessary definitions and calculations are performed with the primitive **\directlua** or inside the environment **tkzelements**. Then, we execute the macro which transforms the affixes of the table z into **Nodes**. Finally, we proceed with the drawing.
At present, the drawing program is either Ti*k*Z or tkz-euclide. However, you have the option to use another package for plotting. To do so, you'll need to create a macro similar to **\tkzGetNodes**. Of course, this package must be capable of storing points like 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 23 for an explanation of the previous code.
Point names can contain the underscore _ and the macro **\tkzGetNodes** allows to obtain names of nodes containing **prime** or **double prime**. (Refer to the next example)

```
\directlua{
   init_elements ()
   scale = 1.5
   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
}
\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}
```



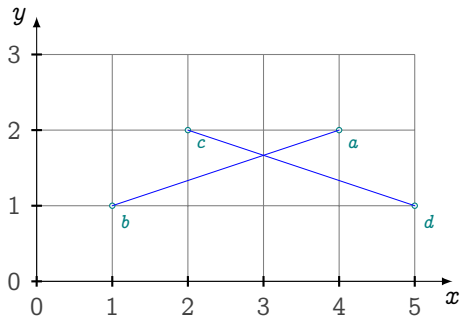
7.1.2 Other transfers

Sometimes it's useful to transfer angle, length measurements or boolean. For this purpose, I have created the macro (refer to 21.6) `tkzUseLua(value)`

```
\def\tkzUseLua#1{\directlua{tex.print(tostring(#1))}}
```

The intersection of the two lines lies at a point whose affix is: 3+1.67i

```
\directlua{
   init_elements ()
   z.b = point:  new (1,1)
   z.a = point:  new (4,2)
   z.c = point:  new (2,2)
   z.d = point:  new (5,1)
   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)
}
The intersection of the two lines lies at
    a point whose affix is:\tkzUseLua{x}
\begin{tikzpicture}
  \tkzGetNodes
 \tkzInit[xmin =0,ymin=0,xmax=5,ymax=3]
  \tkzGrid\tkzAxeX\tkzAxeY
   \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}
```

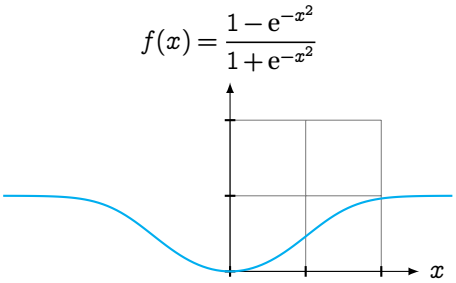


7.1.3 Example 1

In this example, it's necessary to transfer the function to the Lua part, then retrieve the curve point coordinates from TEX.
The main tools used are a table and its methods (`insert`,`concat`) and the `load` function.

```
\makeatletter\let\percentchar\@percentchar\makeatother
\directlua{
  function list (f,min,max,nb)
    local tbl = {}
    for x = min, max, (max - min) / nb do
        table.insert (tbl, ('(\percentchar f,\percentchar f)'):format (x, f (x)))
    end
   return table.concat (tbl)
  end
}
\def\plotcoords#1#2#3#4{%
\directlua{%
  f = load ((([[
```

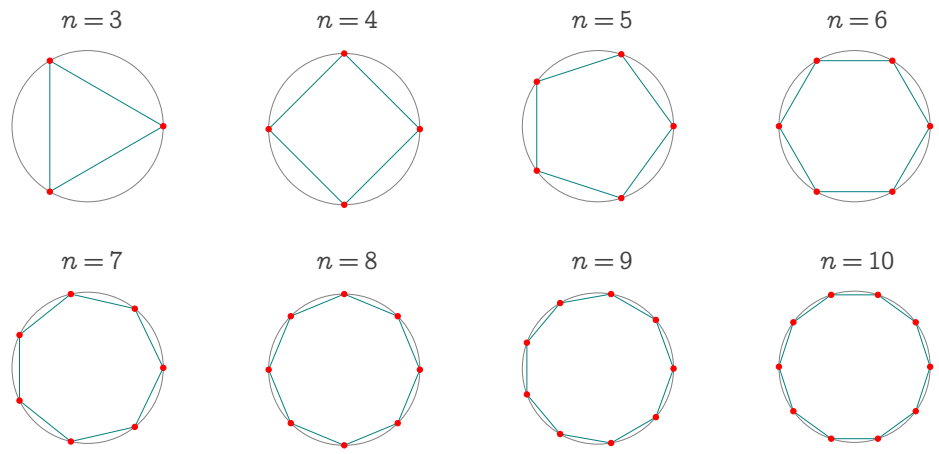
```
        return function (x)
            return (\percentchar s)
        end
    ]]):format ([[#1]]), nil, 't', math) ()
tex.print(list(f,#2,#3,#4))}
}
\begin{tikzpicture}
\tkzInit[xmin=1,xmax=3,ymin=0,ymax=2]
\tkzGrid
\tkzDrawX[right=3pt,label={$x$}]
\tkzDrawY[above=3pt,label={$f(x) = \dfrac{1-\mathrm{e}^{-x^2}}{1+\mathrm{e}^{-x^2}}$}]
\draw[cyan,thick] plot coordinates {\plotcoords{(1-exp(-x^2))/(exp(-x^2)+1)}{-3}{3}{100}};
\end{tikzpicture}
```



7.1.4 Example 2

This consists in passing a number (the number of sides) from TEX to Lua. This is made easier by using the `\directlua` primitive. This example is based on a answer from egreg [egreg–tex.stackexchange.com]

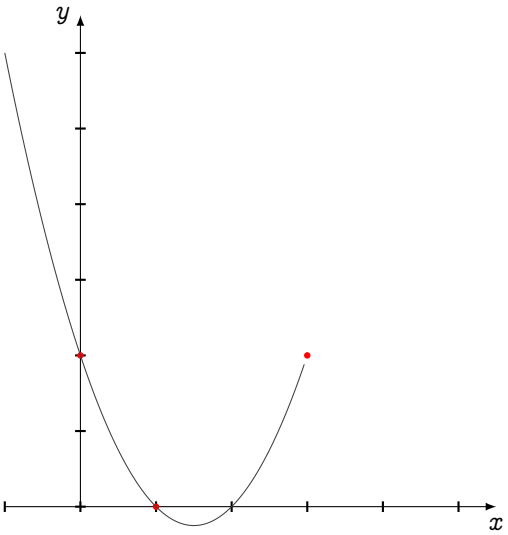
```
\directlua{
  z.I      = point:    new (0,0)
  z.A      = point:    new (2,0)
}
\def\drawPolygon#1{
\directlua{
  RP.six   = regular_polygon : new (z.I,z.A,#1)
  RP.six : name ("P_")
  }
\begin{tikzpicture}[scale=.5]
 \def\nb{\tkzUseLua{RP.six.nb}}
 \tkzGetNodes
 \tkzDrawCircles(I,A)
 \tkzDrawPolygon(P_1,P_...,P_\nb)
 \tkzDrawPoints[red](P_1,P_...,P_\nb)
\end{tikzpicture}
}
\foreach [count=\i] \n in {3, 4, ..., 10} {
  \makebox[0.2\textwidth]{%
    \begin{tabular}[t]{@{}c@{}}
      $n=\n$ \\[1ex]
      \drawPolygon{\n}
    \end{tabular}%
  }\ifnum\i=4 \\[2ex]\fi
}
```



7.1.5 Example 3

This time, the transfer will be carried out using an external file. The following example is based on this one, but using a table.

```
\directlua{
   z.a   = point: new (1,0)
   z.b   = point: new (3,2)
   z.c   = point: new (0,2)
   A,B,C =  parabola (z.a,z.b,z.c)

 function f(t0, t1, n)
  local out=assert(io.open("tmp.table","w"))
  local y
  for t = t0,t1,(t1-t0)/n  do
   y = A*t^2+B*t +C
    out:write(t, " ", y, " i\string\n")
  end
  out:close()
 end
 }
\begin{tikzpicture}
   \tkzGetNodes
   \tkzInit[xmin=-1,xmax=5,ymin=0,ymax=6]
   \tkzDrawX\tkzDrawY
   \tkzDrawPoints[red,size=2](a,b,c)
   \directlua{f(-1,3,100)}%
   \draw[domain=-1:3] plot[smooth] file {tmp.table};
\end{tikzpicture}
```



7.1.6 Example 4

The result is identical to the previous one.

```
\directlua{
   z.a   = point: new (1,0)
   z.b   = point: new (3,2)
   z.c   = point: new (0,2)
   A,B,C =  parabola (z.a,z.b,z.c)

 function f(t0, t1, n)
```

```
 local tbl = {}
 for t = t0,t1,(t1-t0)/n  do
     y = A*t^2+B*t +C
     table.insert (tbl, "("..t..","..y..")")
  end
  return table.concat (tbl)
end
}
\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawX\tkzDrawY
   \tkzDrawPoints[red,size=2pt](a,b,c)
   \draw[domain=-2:3,smooth] plot coordinates {\directlua{tex.print(f(-2,3,100))}};
\end{tikzpicture}
```

7.1.7 Example 5

```
\makeatletter\let\percentchar\@percentchar\makeatother
\directlua{
function cellx (start,step,n)
return start+step*(n-1)
end
}
\def\calcval#1#2{%
\directlua{
  f = load ((([[
        return function (x)
            return (\percentchar s)
        end
    ]]):format ([[#1]]), nil, 't', math) ()
x = #2
tex.print(string.format("\percentchar.2f",f(x)))}
}
\def\fvalues(#1,#2,#3,#4) {%
\def\firstline{$x$}
    \foreach \i in {1,2,...,#4}{%
      \xdef\firstline{\firstline &  \tkzUseLua{cellx(#2,#3,\i)}}}
\def\secondline{$f(x)=#1$}
     \foreach \i in {1,2,...,#4}{%
      \xdef\secondline{\secondline &
     \calcval{#1}{\tkzUseLua{cellx(#2,#3,\i)}}}}
\begin{tabular}{l*{#4}c}
  \toprule
  \firstline  \\
  \secondline \\
  \bottomrule
  \end{tabular}
}
\fvalues(x^2-3*x+1,-2,.25,8)
\vspace{12pt}
```

x	-2.0	-1.75	-1.5	-1.25	-1.0	-0.75	-0.5	-0.25
$f(x)=x^2-3*x+1$	11.00	9.31	7.75	6.31	5.00	3.81	2.75	1.81

8 Class and object

8.1 Class

Object-oriented programming (OOP) is a programming model based 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 serves as a blueprint for creating objects, providing initial values for attributes and implementations of methods[5] cthat are common to all objects of a certain kind.

8.2 Object

An Object is an instance of a class. Each object contains attributes and methods. Attributes are information or object characteristics of the object stored in the data table (called fields), while methods define the object's 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**.
These objects 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 themselves, such as `T.ABC.bc` which creates the **line** object, representing a straight line passing through the last two points defining the triangle.

8.2.1 Attributes

Attributes are accessed using the classic method, so `T.pc` retrieves the third point of the triangle and `C.OH.center` retrieves the center of the circle. Additionally, I've added a `get_points` function that returns the points of an object. This function 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)` retrieves the center and a point of the circle.

8.2.2 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.

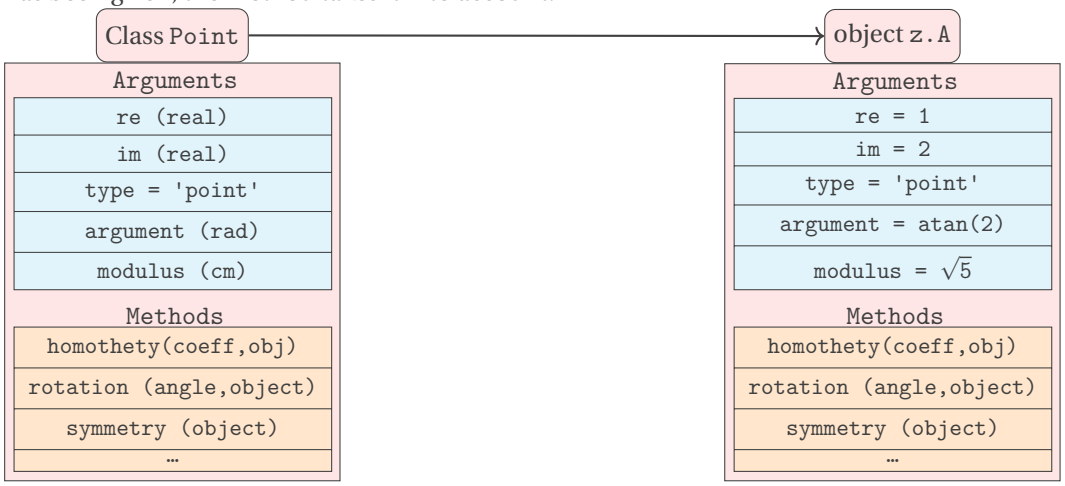
```
\directlua{
      init_elements ()
      z.A = point (1,0)
      z.B = z.A : north (2)
      z.C = z.A : rotation (math.pi/3,z.B)
      tex.print(tostring(z.C))
}
```

The coordinates of C are: -0.73205080756888 and 1.0

5 action which an object is able to perform.

9 Class point

The foundation of the entire framework is the `point` class. This class is hybrid in the sense that it deals with both points in a plane and complex numbers. The principle is as follows: the plane is equipped with an orthonormal basis, which allows us to determine the position of a point using its abscissa and ordinate coordinate. Similarly, any complex number can be viewed simply 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, along with 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.



9.1 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 referenced as a node in Ti*k*Z 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 denoted as `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.

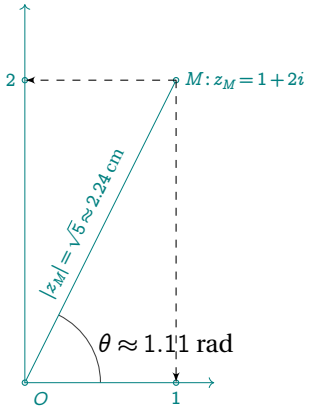
Attributes	Application	Example
re	`z.A.re = 1`	[8.2.2]
im	`z.A.im = 2`	[8.2.2]
type	`z.A.type = 'point'`	
argument	`z.A.argument` $\approx$ `0.78539816339745`	[9.1.1]
modulus	`z.A.modulus` $\approx$ `2.2360...` $=\sqrt{5}$	[9.1.1]

9.1.1 Example: point attributes

```
\directlua{
   init_elements ()
   z.M = point: new (1,2)}

\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}
```

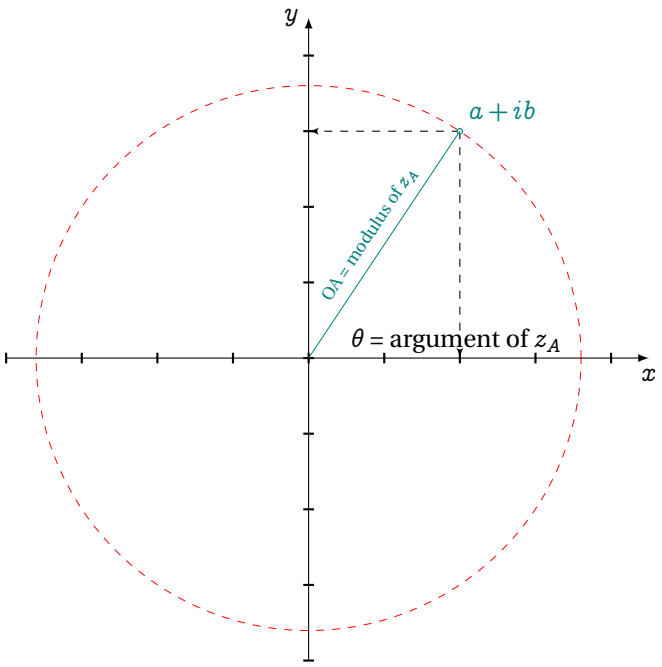
Attributes of `z.M`

- `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



9.1.2 Argand diagram

```
\directlua{
init_elements ()
   z.A = point : new ( 2 , 3 )
   z.O = point : new ( 0 , 0 )
   z.I = point : new ( 1 , 0 )
}
\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}
```



9.2 Methods of the class point

The methods described in the following table are standard and can be found 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**. Refer to section (23.3) for the metamethods.

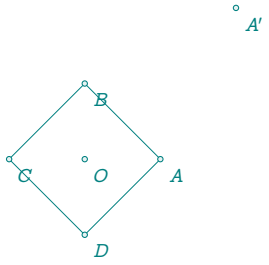
Table 2: Functions & Methods of the class point.

Functions	**Application**	
new(r,r)	z.A = point : new(1,2)	[9.2.3]
polar (d,an)	z.A = point : polar(1,math.pi/3)	[24.11]
polar_deg (d,an)	an in deg	polar coordinates an deg
Methods	**Application**	
Points		
north(r)	r distance to the point (1 if empty)	[11.2.19 ; 8.2.2]
south(r)		
east(r)		
west(r)		
normalize()	z.b = z.a: normalize ()	[9.2.3]
get_points (obj)	retrieves points from the object	[9.2.7; 10.2.20]
orthogonal (d)	z.B=z.A:orthogonal(d)	$\overrightarrow{OB} \perp \overrightarrow{OA}$ and $OB = d$. [9.2.4]
at ()	z.X = z.B : at (z.A)	$\overrightarrow{OB} = \overrightarrow{AX}$ and $OB = d$. [9.2.5]
Transformations		
symmetry(obj)	obj : point, line, etc.	[9.2.8]
rotation(an , obj)	point, line, etc.	[9.2.7]
homothety(r,obj)	z.c = z.a : homothety (2,z.b)	[24.44]
Misc.		
print()	displays the affix of the point	[9.2.8]

9.2.1 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 (Refer to the following example). If d is absent then it is considered equal to 1.

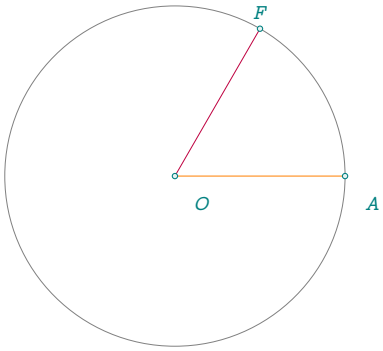
```
\directlua{
   init_elements ()
   z.O   = point : new ( 0, 0 )
   z.A   = z.O : east ()
   z.Ap  = z.O : east (2) : north (2)
   z.B   = z.O : north ()
   z.C   = z.O : west ()
   z.D   = z.O : south ()
}
\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawPolygon(A,B,C,D)
   \tkzDrawPoints(A,B,C,D,O,A')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



9.2.2 Method polar

This involves defining a point using its modulus and argument.

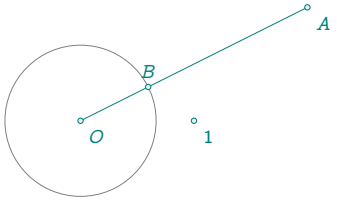
```
\directlua{
   init_elements ()
   z.O     = point:   new  (0, 0)
   z.A     = point:   new  (3, 0)
   z.F     = point:   polar (3, math.pi/3)
}
\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawCircle(O,A)
   \tkzDrawSegments[new](O,A)
   \tkzDrawSegments[purple](O,F)
   \tkzDrawPoints(A,O,F)
   \tkzLabelPoints[below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



9.2.3 Method normalize ()

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

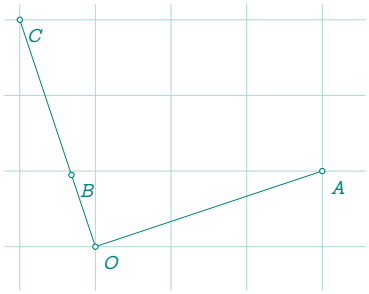
```
\directlua{
init_elements ()
   scale = 1.5
   z.O = point : new (0,0)
   z.A = point : new (1,2)
   z.B = z.A : normalize ()
   z.I = point : new (1,0)
}
\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawSegment(O,A)
   \tkzDrawCircle(O,B)
   \tkzDrawPoints(O,A,B,I)
   \tkzLabelPoints(O,A,B)
   \tkzLabelPoint[below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



9.2.4 Method orthogonal (d)

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 $OB = d$.

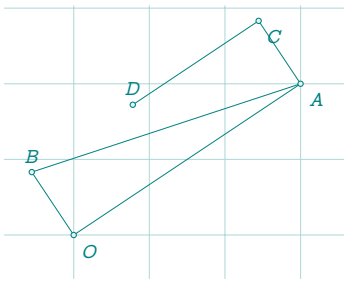
```
\directlua{
init_elements ()
  z.A = point : new (  3 , 1  )
  z.B = z.A : orthogonal (1)
  z.O = point : new ( 0,0  )
  z.C = z.A : orthogonal ()
}
\begin{tikzpicture}[gridded]
  \tkzGetNodes
  \tkzDrawSegments(O,A O,C)
  \tkzDrawPoints(O,A,B,C)
  \tkzLabelPoints[below right](O,A,B,C)
\end{tikzpicture}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



9.2.5 Method at

This method is complementary to the previous one, so you may not wish to have $\overrightarrow{OB} \perp \overrightarrow{OA}$ but $\overrightarrow{AB} \perp \overrightarrow{OA}$.

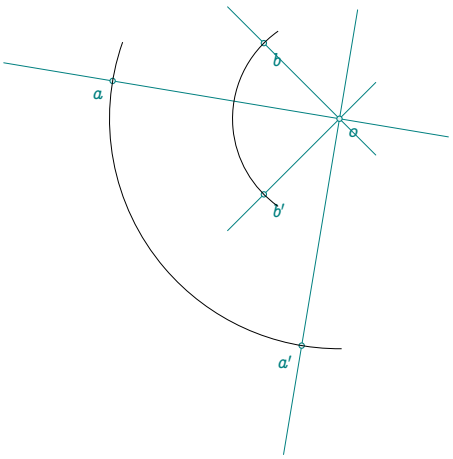
```
\directlua{%
init_elements ()
z.O = point : new ( 0,0  )
z.A = point : new (  3 , 2  )
z.B = z.A : orthogonal (1)
z.C = z.A+z.B
z.D =(z.C-z.A):orthogonal(2) : at (z.C)
}
  \begin{tikzpicture}[gridded]
  \tkzGetNodes
  \tkzLabelPoints[below right](O,A,C)
  \tkzLabelPoints[above](B,D)
  \tkzDrawSegments(O,A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



9.2.6 Method rotation first example

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

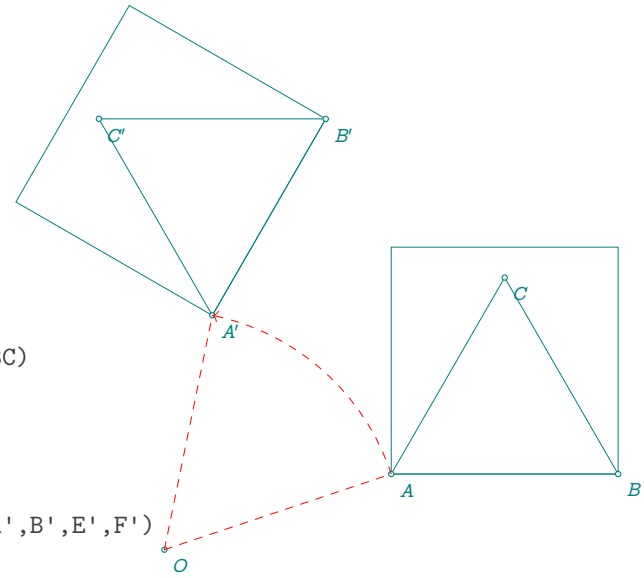
```
\directlua{%
init_elements ()
  z.a       = point:  new(0, -1)
  z.b       = point:  new(4, 0)
  z.o       = point:  new(6, -2)
  z.ap,z.bp = z.o : rotation (math.pi/2,z.a,z.b)
}
  \begin{tikzpicture}
  \tkzGetNodes
  \tkzDrawLines(o,a o,a' o,b o,b')
  \tkzDrawPoints(a,a',b,b',o)
  \tkzLabelPoints(b,b',o)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



9.2.7 Method rotation second example

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

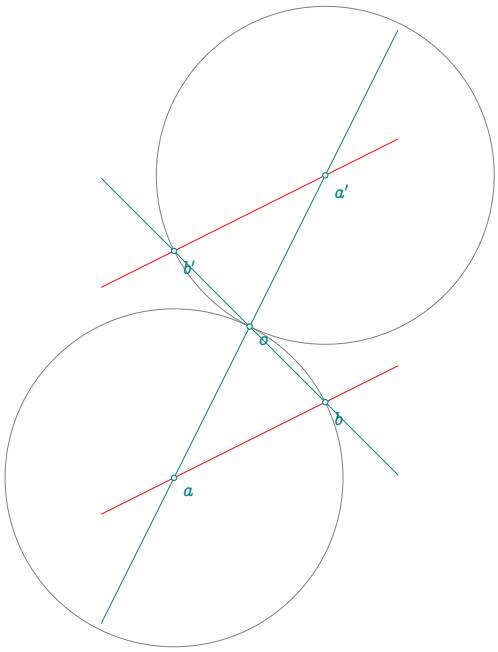
```
\directlua{%
init_elements ()
  scale = .75
   z.O   = point : new ( -1 , -1 )
   z.A   = point : new ( 2 , 0 )
   z.B   = point : new ( 5 , 0 )
   L.AB  = line : new (z.A,z.B)
   T.ABC = L.AB : equilateral ()
   S.fig = L.AB : square ()
   _,_,z.E,z.F = get_points (   S.fig   )
   S.new = z.O : 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)
}
\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}
```



9.2.8 Method symmetry

Example of the symmetry of an object

```
\directlua{%
init_elements ()
  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))
}
\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}
```



10 Class line

10.1 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 represents both 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.

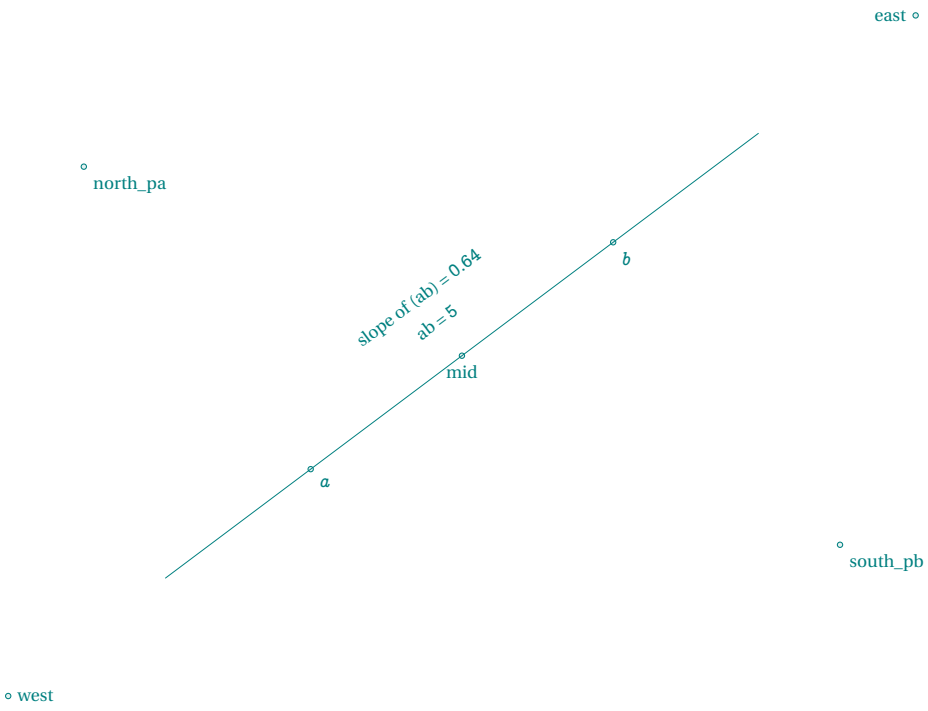
Creation
```
L.AB = line : new ( z.A , z.B )
```

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	[10.1.1]
length	`l = L.AB.length`	[21.6 ; 10.1.1]
north_pa		[10.1.1]
north_pb		
south_pa		
south_pb		[10.1.1]
east		
west		
vec	`V.AB = L.AB.vec`	defines $\overrightarrow{AB}$ [19]

10.1.1 Example: attributes of class line

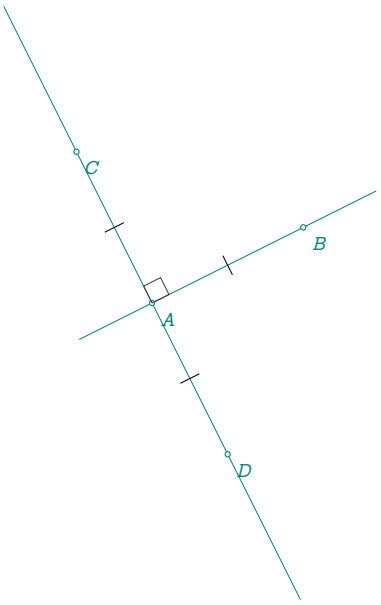



```
\directlua{%
init_elements ()
  scale  = .5
   z.a   = point: new (1, 1)
   z.b   = point: new (5, 4)
   L.ab  = line : new (z.a,z.b)
   z.m   = L.ab.mid
   z.w   = L.ab.west
   z.e   = L.ab.east
   z.r   = L.ab.north_pa
   z.s   = L.ab.south_pb
   sl    = L.ab.slope
   len   = L.ab.length
}

\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}
```

10.1.2 Method new and line attributes

The notation can be `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.

```
\directlua{%
init_elements ()
   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
}
\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}
```



10.2 Methods of the class line

Here's the list of methods for the **line** object. The results can be real numbers, 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)	Create line (AB) ; [24.4]
Points		
gold_ratio ()	z.C=L.AB : gold_ratio()	[24.26 ; 4.3 ; 24.12]
normalize ()	z.C=L.AB : normalize()	AC =1 and $C \in (AB)$ [10.2.9]
normalize_inv ()	z.C=L.AB : normalize_inv()	CB=1 and $C \in (AB)$
barycenter (r,r)	z.C=L.AB : barycenter (1,2)	[10.2.10]
point (r)	z.C=L.AB : point (2)	$\overrightarrow{AC} = 2\overrightarrow{AB}$ [24.19 ; 10.2.7]
midpoint ()	z.M=L.AB : midpoint ()	better is z.M = L.AB.mid
harmonic_int (pt)	z.D=L.AB : harmonic_int (z.C)	[24.12]
harmonic_ext (pt)	z.D=L.AB : harmonic_ext (z.C)	[24.12]
harmonic_both (r)	z.C,z.D=L.AB : harmonic_both(φ)	[21.2]
_east(d)	z.M=L.AB : _east(2)	BM = 2 A, B, M aligned
_west(d)	z.M=L.AB : _east(2)	BM = 2 A, B, M aligned
_north_pa(d)	z.M=L.AB: _north_pa(2)	AM=2; $AM \perp AB$; $\overrightarrow{AB}, \overrightarrow{AM}$ counterclockwise.
...		[10.2.11; 13.1.1]
_south_pa(d)	z.M=L.AB: _south_pa(2)	AM=2; $AM \perp AB$; $\overrightarrow{AB}, \overrightarrow{AM}$ clockwise
_north_pb(d)	z.M=L.AB: _north_pb(2)	BM=2; $BM \perp BA$; $\overrightarrow{BA}, \overrightarrow{BM}$ clockwise
_south_pb(d)	z.M=L.AB: _south_pb(2)	BM=2; $BM \perp BA$; $\overrightarrow{AB}, \overrightarrow{AM}$ counterclockwise
report(d,pt)	z.M=L.AB: report(2,z.N)	MN=2; $AB \parallel MN$; [ex. 10.2.1]
colinear_at(pt,k)	z.D=L.AB: colinear_at(z.C,2)	CD=2AB; $AB \parallel CD$; [ex. 10.2.8]
Lines		
ll_from (pt)	L.CD=L.AB: ll_from(z.C)	$(CD) \parallel (AB)$; [10.2.11]
ortho_from (pt)	L.CD=L.AB: ortho_from(z.C)	$(CD) \perp (AB)$; [10.2.12]
mediator ()	L.uv=L.AB: mediator()	perpendicular bisector of (A, B) [a]; [10.2.13]
Triangles		
equilateral (<swap>)	T.ABC=L.AB:equilateral()	$(\overrightarrow{AB}, \overrightarrow{AC}) > 0$ or < 0 with swap [b]; [9.2.7]
isosceles (an<,swap>)	T.ABC=L.AB:isosceles(math.pi/6)	[10.2.3]
two_angles (an,an)	T.ABC=L.AB:two_angles(an,an)	note [c] [10.2.2]
school ()	30°,60°, 90°	
sss (r,r)	$AC = r$ $BC = r$	[10.2.4]
sas (r,an)	$AC = r$ $\widehat{BAC} = an$	[10.2.4]
ssa (r,an)	$AC = r$ $\widehat{ABC} = an$	[10.2.4]
Squares		
square ()	S.AB=L.AB : square ()	create a square S.AB.[d]; [9.2.7]

a You can use perpendicular_bisector intead of **mediator**.
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
d _,_,z.C,z.D = get_points(S.AB)

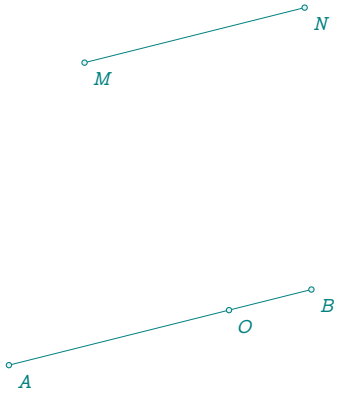
Table 5: Methods of the class line.(part 2)

Methods	Comments	
Sacred triangles		
gold (<swap>)	T.ABC=L.AB:gold()	right in B and $AC = \varphi \times AB$; [6]
euclide (<swap>)	T.ABC=L.AB:euclide()	$AB = AC$; $(\overrightarrow{AB}, \overrightarrow{AC}) = \pi/5$; [6]
golden (<swap>)	T.ABC=L.AB:golden()	$(\overrightarrow{AB}, \overrightarrow{AC}) = 2 \times \pi/5$; [6]
divine ()		[6]
egyptian ()		[6]
cheops ()		[6]
Circles		
circle ()	C.AB = L.AB : circle ()	center pa through pb
apollonius (r)	C.apo = L.AB : apollonius (2)	Set of points tq. MA/MB = 2; [10.2.20]
Transformations		
reflection (obj)	new obj = L.AB : reflection (obj	[10.2.18]
translation (obj)	new obj = L.AB : translation (obj)	[10.2.17]
projection (obj)	z.H = L.AB : projection (z.C)	$CH \perp (AB)$ and $H \in (AB)$; [10.2.15; 10.2.16]
...		[10.2.15; 10.2.16]
Miscellaneous		
distance (pt)	d = L.Ab : distance (z.C)	[10.2.19]
in_out (pt)	b = L.AB: in_out(z.C)	b=true if $C \in (AB)$; [23.6.1]
slope ()	a = L.AB : slope()	better is L.AB.slope; [10.1.1]
in_out_segment (pt)	b = L.AB : in_out_segment(z.C)	b=true if $C \in [AB]$

10.2.1 Method report

report (d,pt) If the point is absent, the transfer is made from the first point that defines the line.

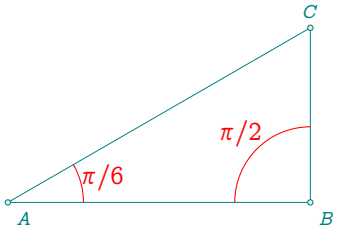
```
  \directlua{%
init_elements ()
  z.A  = point : new (1,-1)
  z.B  = point : new (5,0)
  L.AB = line : new ( z.A , z.B )
  z.M  = point : new (2,3)
  z.N  = L.AB :  report (3,z.M)
  z.O  = L.AB :  report (3)
  }
  \begin{tikzpicture}
  \tkzGetNodes
  \tkzDrawSegments(A,B M,N)
  \tkzDrawPoints(A,B,M,N,O)
  \tkzLabelPoints(A,B,M,N,O)
  \end{tikzpicture}
```



10.2.2 Method two_angles

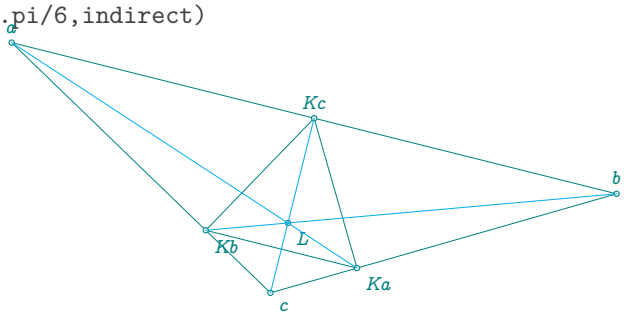
The angles are on either side of the given segment

```
\directlua{%
init_elements ()
   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
}
\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawPolygons(A,B,C)
   \tkzDrawPoints(A,B,C)
   \tkzLabelPoints(A,B)
   \tkzLabelPoints[above](C)
   \tkzMarkAngle[red](C,B,A)
   \tkzMarkAngle[red](B,A,C)
   \tkzLabelAngle[red,pos=1.3](C,B,A){$\pi/2$}
   \tkzLabelAngle[red,pos=1.3](B,A,C){$\pi/6$}
\end{tikzpicture}
```



10.2.3 Method isosceles

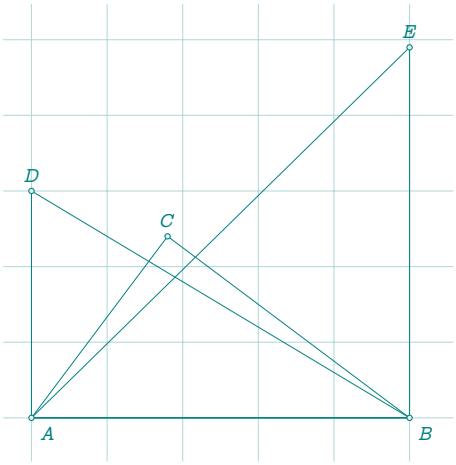
```
\directlua{%
init_elements ()
   scale          = 2
   z.a            = point :   new (1,2)
   z.b            = point :   new (5,1)
   L.ab           = line :    new (z.a,z.b)
   T.abc          = L.ab :    isosceles (math.pi/6,indirect)
   z.c            = T.abc.pc
   z.L            = T.abc : lemoine_point ()
   T.SY           = T.abc : symmedian ()
   z.Ka,z.Kb,z.Kc = get_points (T.SY)
   L.Kb          = T.abc : symmedian_line (1)
  _,z.Kb        = get_points(L.Kb)
}
\begin{tikzpicture}
\tkzGetNodes
\tkzDrawPolygons(a,b,c Ka,Kb,Kc)
\tkzDrawPoints(a,b,c,L,Ka,Kb,Kc)
\tkzLabelPoints(c,L,Ka,Kb)
\tkzLabelPoints[above](a,b,Kc)
\tkzDrawSegments[cyan](a,Ka b,Kb c,Kc)
\end{tikzpicture}
```



10.2.4 Methods sss, sas, ssa

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.

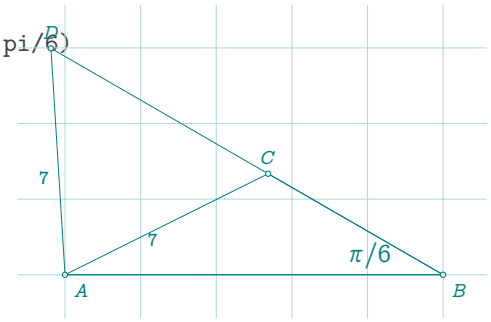
```
\directlua{%
init_elements ()
  z.A = point : new ( 0 , 0 )
  z.B = point : new ( 5 , 0 )
  L.AB = line : new ( z.A , z.B )
  T.ABC =  L.AB : sss (value(3),value(4))
  T.ABD =  L.AB : sas (value(3),math.pi/2)
  T.ABE =  L.AB : ssa (value(7),math.pi/2)
  z.C = T.ABC.pc
  z.D = T.ABD.pc
  z.E = T.ABE.pc
}
\hspace{\fill}
\begin{tikzpicture}[gridded]
  \tkzGetNodes
  \tkzDrawPolygons(A,B,C A,B,D A,B,E)
  \tkzDrawPoints(A,B,C,D,E)
  \tkzLabelPoints(A,B)
  \tkzLabelPoints[above](C,D,E)
\end{tikzpicture}
```



10.2.5 Triangle with side between side and angle

In some cases, two solutions are possible.

```
\directlua{%
init_elements ()
   scale = 1
   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
}
\begin{tikzpicture}[gridded]
   \tkzGetNodes
   \tkzDrawPolygons(A,B,C A,B,D)
   \tkzDrawPoints(A,B,C,D)
   \tkzLabelPoints(A,B)
   \tkzLabelPoints[above](C,D)
   \tkzLabelAngle[teal](C,B,A){$\pi/6$}
   \tkzLabelSegment[below left](A,C){$7$}
   \tkzLabelSegment[below left](A,D){$7$}
\end{tikzpicture}
```



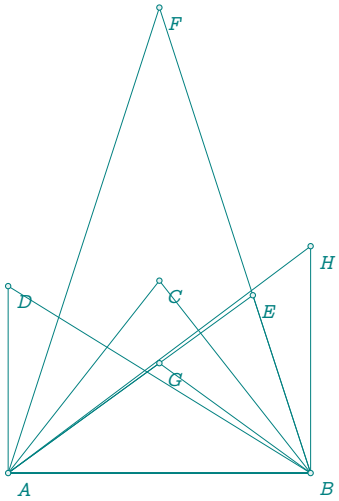
10.2.6 About sacred triangles

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 = $\sqrt{\varphi}$

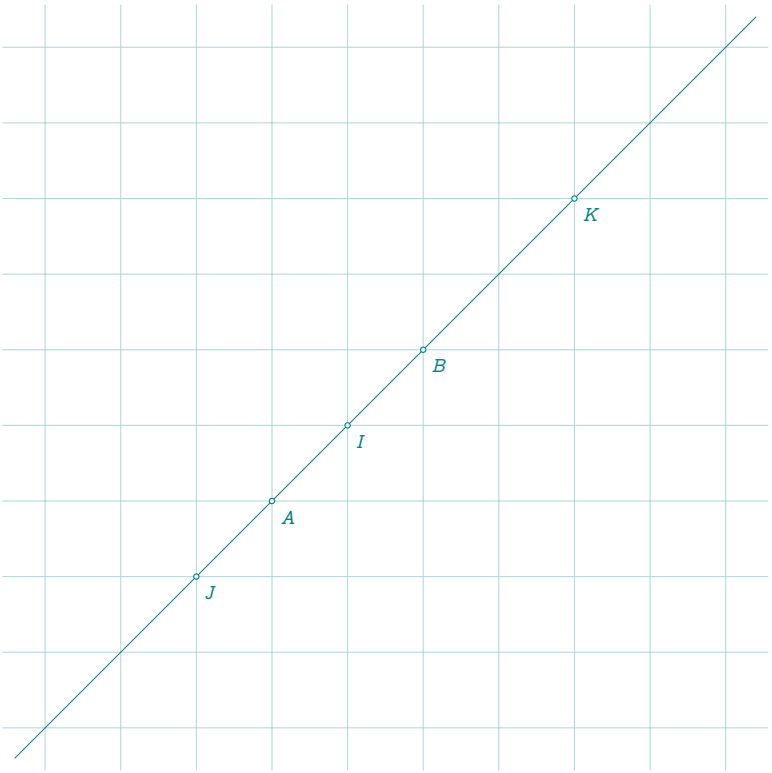
```
\directlua{%
init_elements ()
   z.A = point : new ( 0 , 0 )
   z.B = point : new ( 4 , 0 )
   L.AB = line : new ( z.A , z.B )
   T.ABC = L.AB : cheops ()
   z.C = T.ABC.pc
   T.ABD = L.AB : gold ()
   z.D = T.ABD.pc
   T.ABE = L.AB : euclide ()
   z.E = T.ABE.pc
   T.ABF = L.AB : golden ()
   z.F = T.ABF.pc
   T.ABG = L.AB : divine ()
   z.G = T.ABG.pc
   T.ABH = L.AB : pythagoras ()
   z.H = T.ABH.pc
}
\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}
```



10.2.7 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.

```
\directlua{%
init_elements ()
   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)
}
\begin{tikzpicture}[gridded]
\tkzGetNodes
   \tkzDrawLine(J,K)
   \tkzDrawPoints(A,B,I,J,K)
   \tkzLabelPoints(A,B,I,J,K)
 \end{tikzpicture}
```



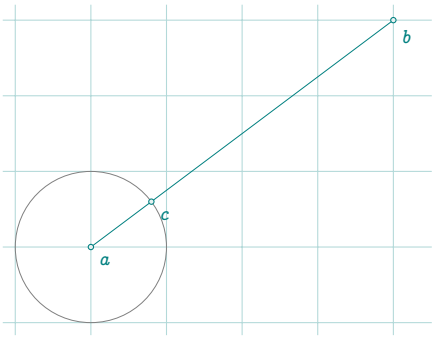
10.2.8 Method colinear_at

If the coefficient is missing then it defaults to 1 and in the following example we obtain: $CE = AB$ and $(AB) \parallel (CE)$. For point D: $CD = .5AB$ and $(AB) \parallel (CD)$.

```
\directlua{%
init_elements ()
  z.A      = point: new (0 , 0)
  z.B      = point: new (4 , 0)
  z.C      = point: new (1 , 3)
  L.AB     = line : new (z.A,z.B)
  z.D      = L.AB : colinear_at (z.C,.5)
  z.E      = L.AB : colinear_at (z.C)
}
\begin{tikzpicture}
  \tkzGetNodes
  \tkzDrawSegments(A,B C,E)
  \tkzDrawPoints(A,B,C,D,E)
  \tkzLabelPoints(A,B,C,D,E)
\end{tikzpicture}
```

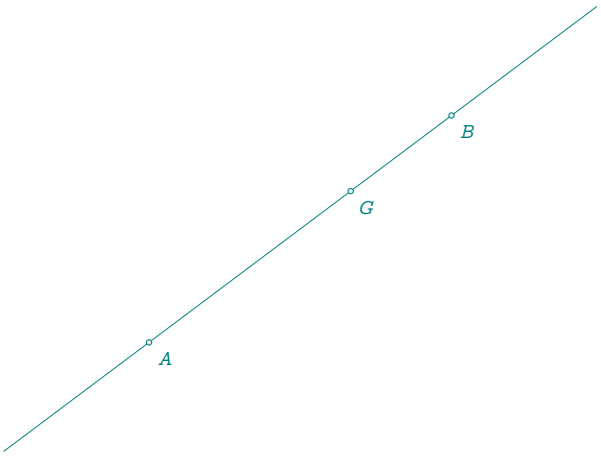
10.2.9 Method normalize

```
\directlua{%
init_elements ()
   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 ()
}

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



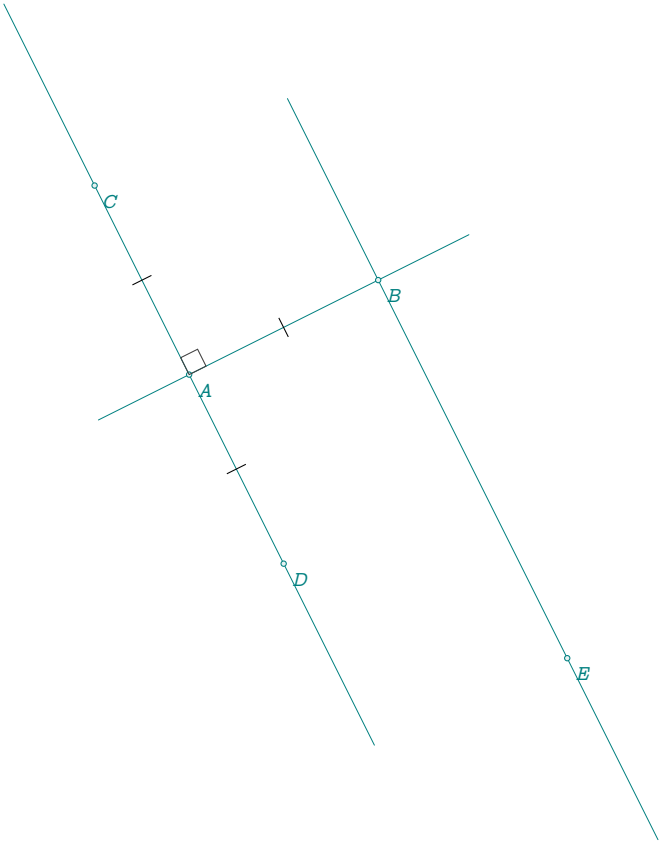
10.2.10 Method barycenter

```
\directlua{%
init_elements ()
   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)
}
\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawLine(A,B)
   \tkzDrawPoints(A,B,G)
   \tkzLabelPoints(A,B,G)
\end{tikzpicture}
```



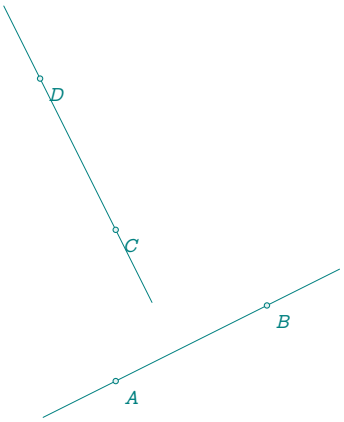
10.2.11 Method ll_from

```
\directlua{%
init_elements ()
   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
}
\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}
```



10.2.12 Method ortho_from

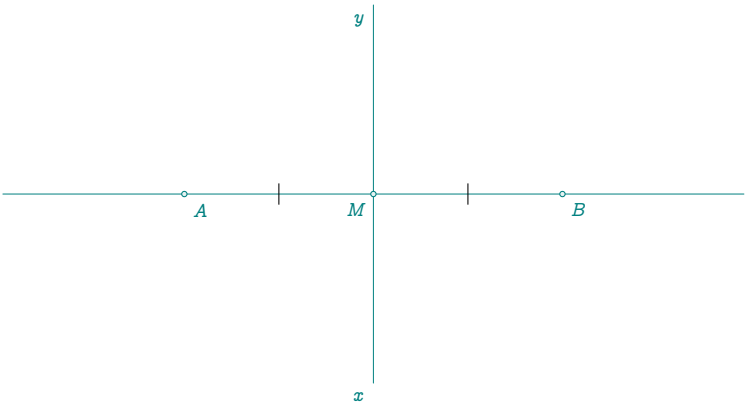
```
\directlua{%
init_elements ()
   z.A   = point : new (1,1)
   z.B   = point : new (3,2)
   L.AB  = line :  new (z.A,z.B)
   z.C   = point : new (1,3)
   L.CD  = L.AB :  ortho_from(z.C)
   z.D   = L.CD.pb
}
\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawLines(A,B C,D)
   \tkzDrawPoints(A,...,D)
   \tkzLabelPoints(A,...,D)
\end{tikzpicture}
```



10.2.13 Method mediator

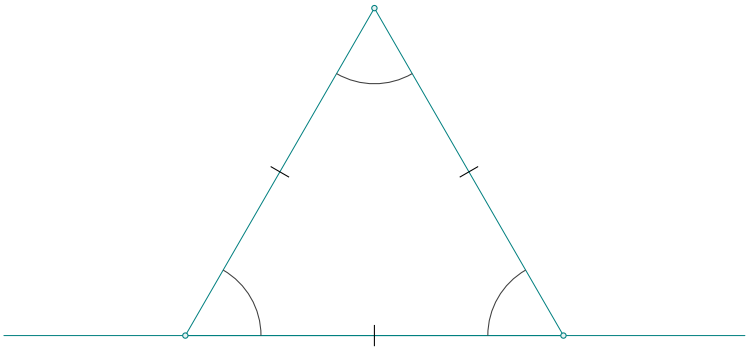
In Mathworld, the mediator is the plane through the midpoint of a line segment and perpendicular to that segment, also called a mediating plane. The term ”mediator” was introduced by J. Neuberg (Altshiller-Court 1979, p. 298). Here, I have adopted the French term and the mediator or the perpendicular bisector of a line segment, is a line segment perpendicular to the segment and passing through the midpoint of this segment.

```
\directlua{%
init_elements ()
  z.A    = point: new(0,0)
  z.B    = point: new(5,0)
  L.AB   = line: new (z.A,z.B)
  L.med  = L.AB : mediator ()
  z.M    = L.AB.mid
  z.x,z.y= get_points(L.med)
}
\begin{tikzpicture}
\tkzGetNodes
\tkzDrawLine(A,B)
\tkzDrawSegments(x,y)
\tkzDrawPoints(A,B,M)
\tkzLabelPoints(A,B)
\tkzLabelPoints[below left](x,y,M)
\tkzMarkSegments(A,M M,B)
\end{tikzpicture}
```



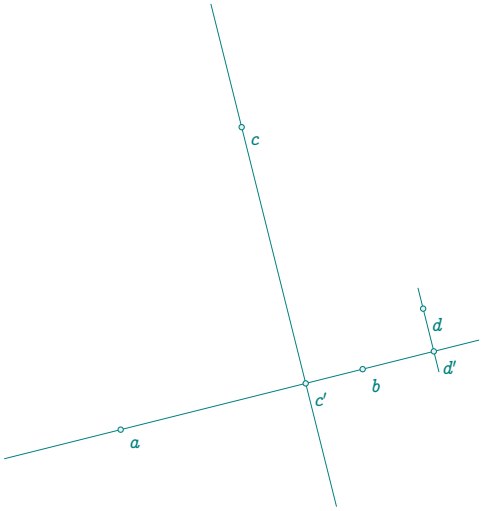
10.2.14 Method equilateral

```
\directlua{%
init_elements ()
  z.A    = point: new(0,0)
  z.B    = point: new(5,0)
  L.AB   = line: new (z.A,z.B)
  T.ABC  = L.AB : equilateral ()
  z.C    = T.ABC.pc
}
\begin{tikzpicture}
\tkzGetNodes
\tkzDrawLine(A,B)
\tkzDrawPolygon(A,B,C)
\tkzMarkSegments(A,B B,C C,A)
\tkzDrawPoints(A,B,C)
\tkzMarkAngles(B,A,C C,B,A A,C,B)
\end{tikzpicture}
```



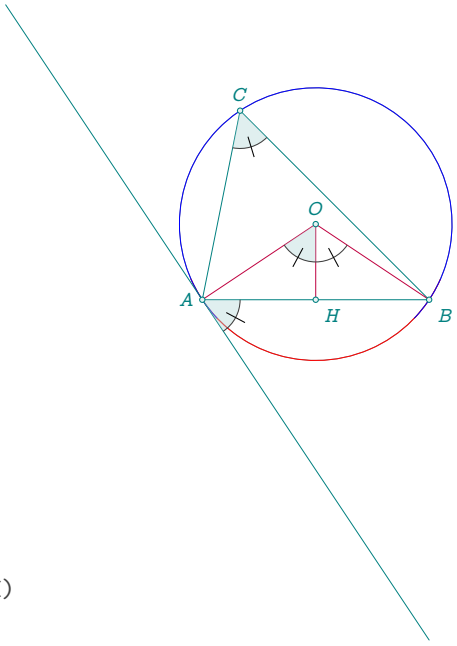
10.2.15 Method projection

```
\directlua{%
init_elements ()
   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)
}
 \begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawLines(a,b c,c' d,d')
   \tkzDrawPoints(a,...,d,c',d')
   \tkzLabelPoints(a,...,d,c',d')
 \end{tikzpicture}
```



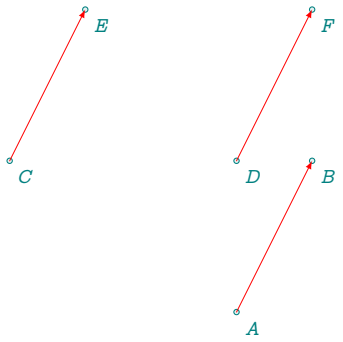
10.2.16 Example: combination of methods

```
\directlua{%
init_elements ()
   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)
}
\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(B,H)
   \tkzLabelPoints[above](O,C)
   \tkzLabelPoints[left](A)
\end{tikzpicture}
```



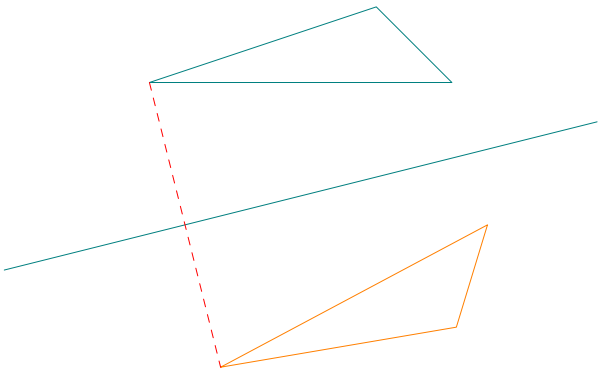
10.2.17 Method translation

```
\directlua{%
init_elements ()
   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)
}
\begin{tikzpicture}
\tkzGetNodes
\tkzDrawPoints(A,...,F)
\tkzLabelPoints(A,...,F)
\tkzDrawSegments[->,red,> =latex](C,E D,F A,B)
\end{tikzpicture}
```



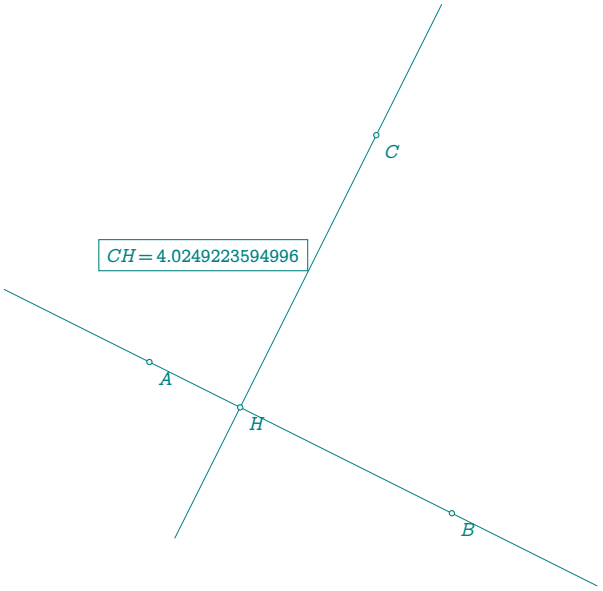
10.2.18 Method reflection of an object

```
\directlua{%
init_elements ()
   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)
}
\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawLine(A,B)
   \tkzDrawPolygon(E,F,G)
   \tkzDrawPolygon[new](E',F',G')
   \tkzDrawSegment[red,dashed](E,E')
\end{tikzpicture}
```



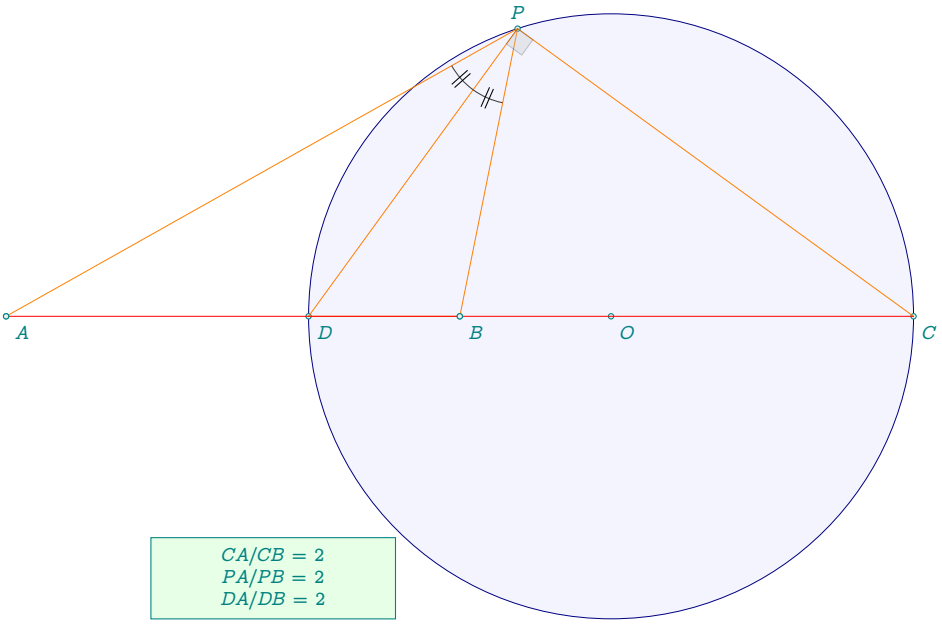
10.2.19 Method distance

```
\directlua{%
init_elements ()
   z.A    = point : new (0 , 0)
   z.B    = point : new (4 , -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)
}
\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}
```



10.2.20 Method apollonius (Apollonius circle MA/MB = k)

```
\directlua{%
init_elements ()
   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)
}
\begin{tikzpicture}
   \tkzGetNodes
   \tkzFillCircle[blue!20,opacity=.2](O,C)
   \tkzDrawCircle[blue!50!black](O,C)
   \tkzDrawPoints(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$}
   \tkzLabelPoints[below right](A,B,O,C,D)
   \tkzLabelPoints[above](P)
   \tkzMarkRightAngle[opacity=.3,fill=lightgray](D,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

11 Class circle

11.1 Attributes of a circle

This class is defined by two points: the center and a point through which the circle passes

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

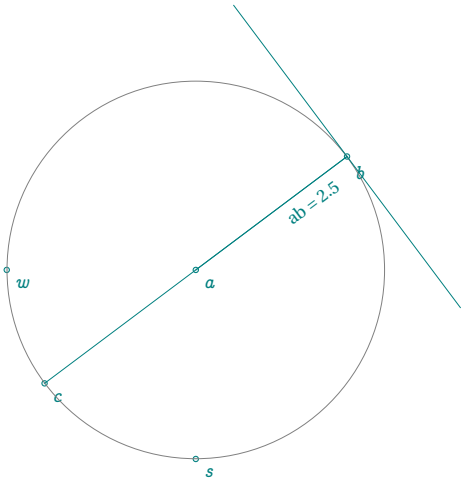
Table 7: Circle attributes.

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	[11.1.1]
ct	L = C.AB.ct [11.1.1]	

11.1.1 Example: circle attributes

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

```
\directlua{%
init_elements ()
   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))
}
\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}
```



11.2 Methods of the class circle

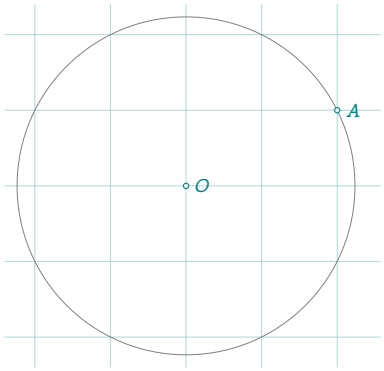
Table 8: Circle methods.

Methods	Comments	
new(O,A)	C.OA = circle : new (z.O,z.A)	center O through A; [11.2.1]
radius(O,r)	C.OA = circle : radius (z.O,2)	center O radius =2 cm; [11.2.2]
diameter(A,B)	C.OA = circle :diameter(z.A,z.B)	diameter $[AB]$; [11.2.3]
Points		
antipode (pt)	z.C = C.OA: antipode (z.B)	$[BC]$ = diameter; [11.2.4]
midarc (pt,pt)	z.D = C.AB: midarc (z.B,z.C)	D is the midarc of $\overset{\frown}{BC}$; [11.2.5]
point (r)	z.E = C.AB: point (0.25)	r between 0 and 1; [11.2.6]
random_pt(lower, upper)		
inversion (obj)	z.Bp = C.AC: inversion (z.B)	[11.2.7]
internal_similitude (C)	z.I= C.one: internal_similitude(C.two)	[11.2.8]
external_similitude (C)	z.J= C.one: external_similitude(C.two)	[11.2.9]
radical_center (C1<,C2>)	or only (C1)	[11.2.10]
Lines		
radical_axis (C)	[11.2.11 ; 24.3]	
tangent_at (pt)	z.P=C.OA:tangent_at(z.M)	[11.2.12]
tangent_from (pt)	z.M,z.N=C.OA: tangent_from (z.P)	[11.2.12]
common_tangent (C)	z.a,z.b = C.AC: common_tangent (C.EF)	[11.2.13 ; 24.7]
Circles		
orthogonal_from (pt)	C=C.OA:orthogonal_from (z.P)	[11.2.14 ;24.4 ; 24.38]
orthogonal_through(pta,ptb)	C=C.OA:orthogonal_through (z.z1,z.z2)	[11.2.15]
midcircle (C)	C.inv = C.OA: midcircle (C.EF)	[11.2.16]
radical_circle (C1<,C2>)	or only (C1)	[11.2.17]
Miscellaneous		
power (pt)	r = C.OA: power (z.M)	[11.2.18 ; 11.2.19 ; 24.33]
in_out (pt)	C.OA : in_out (z.M)	[11.2.20]
in_out_disk (pt)	C.OA : in_out_disk (z.M)	[11.2.20]
draw ()	for further use	
circles_position (C1)	result = string	[11.2.21]

11.2.1 Method new

A circle is defined by its centre and a point through which it passes.

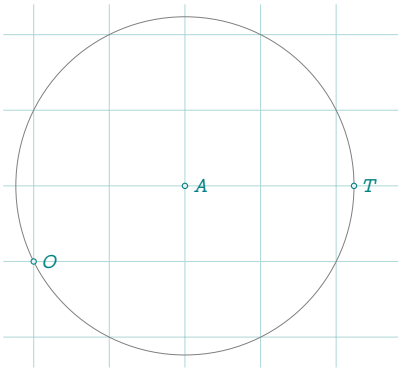
```
\directlua{%
init_elements ()
z.O     = point:    new (0,0)
z.A     = point:    new (2,1)
C       = circle:   new (z.O , z.A)
}
\begin{tikzpicture}[gridded]
\tkzGetNodes
\tkzDrawCircles(O,A)
\tkzDrawPoints(A,O)
\tkzLabelPoints[right](A,O)
\end{tikzpicture}
```



11.2.2 Method radius

We define a circle with its centre and radius.

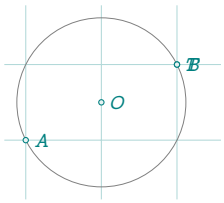
```
\directlua{%
init_elements ()
z.O     = point:    new (0,0)
z.A     = point:    new (2,1)
C       = circle:   radius (z.A , math.sqrt(5))
z.T     = C.through
}
\begin{tikzpicture}[gridded]
\tkzGetNodes
\tkzDrawCircles(A,T)
\tkzDrawPoints(A,O,T)
\tkzLabelPoints[right](A,O,T)
\end{tikzpicture}
```



11.2.3 Method diameter

A circle is defined by two points at the ends of one of its diameters.

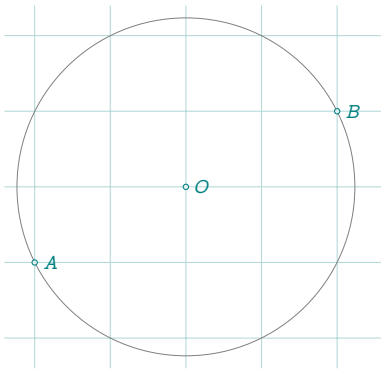
```
\directlua{%
init_elements ()
z.A     = point:    new (0,0)
z.B     = point:    new (2,1)
C       = circle:   diameter (z.A , z.B)
z.O     = C.center
z.T     = C.through
}
\begin{tikzpicture}[gridded]
\tkzGetNodes
\tkzDrawCircles(O,T)
\tkzDrawPoints(A,B,O,T)
\tkzLabelPoints[right](A,B,O,T)
\end{tikzpicture}
```



11.2.4 Method antipode

This method is used to define a point that is diametrically opposed to a point on a given circle.

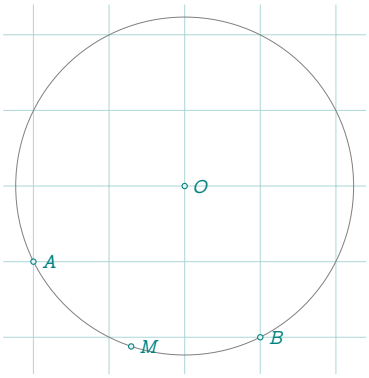
```
\directlua{%
init_elements ()
z.A     = point:    new (0,0)
z.O     = point:    new (2,1)
C       = circle:   new (z.O , z.A)
z.B     = C : antipode (z.A)
}
\begin{tikzpicture}[gridded]
\tkzGetNodes
\tkzDrawCircles(O,A)
\tkzDrawPoints(A,B,O)
\tkzLabelPoints[right](A,B,O)
\end{tikzpicture}
```



11.2.5 Method midarc

The definition given in [Weisstein, Eric W. "Mid-Arc Points." From MathWorld–A Wolfram Web Resource.] is as follows: The mid-arc points of a triangle as defined by Johnson (1929) are the points on the circumcircle of the triangle which lie half-way along each of the three arcs determined by the vertices. These points arise in the definition of the Fuhrmann circle and Fuhrmann triangle, and lie on the extensions of the perpendicular bisectors of the triangle sides drawn from the circumcenter.
The definition I use here is more general: the defined point is simply the point that divides an arc into two arcs of the same length.

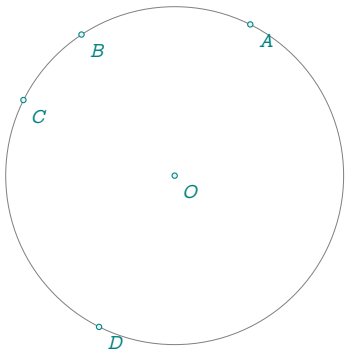
```
\directlua{%
init_elements ()
z.A     = point:    new (0,0)
z.O     = point:    new (2,1)
C       = circle:   new (z.O , z.A)
z.B     = C : point (0.25)
z.M     = C : midarc (z.A,z.B)
}
\begin{tikzpicture}[gridded]
\tkzGetNodes
\tkzDrawCircles(O,A)
\tkzDrawPoints(A,B,O,M)
\tkzLabelPoints[right](A,B,O,M)
\end{tikzpicture}
```



11.2.6 Method point (r)

Let C be a circle with centre O and passing through A such that `z.A = C.through`. This method defines a point M on the circle from A such that the ratio of the length of $\overset{\frown}{AM}$ to the circumference of the circle is equal to r.
In the next example, $r = \dfrac{1}{6}$ corresponds to $\dfrac{\pi/3}{2\pi}$, so the angle $\widehat{AOE}$ has the measure $\pi/3$.
If $r = .5$ the defined point is diametrically opposed to A, the angle $\widehat{AOD}$ has the measure π.

```
 \directlua{%
init_elements ()
   z.O  = point:  new (0,0)
   z.A  = point:  new (1,2)
   C.OA = circle:  new (z.O,z.A)
   z.B = C.OA: point (1/6)
   z.C = C.OA: point (0.25)
   z.D = C.OA: point (0.5)
}
\begin{tikzpicture}
\tkzGetNodes
\tkzDrawCircle(O,A)
\tkzDrawPoints(A,...,D,O)
\tkzLabelPoints(A,...,D,O)
\end{tikzpicture}
```

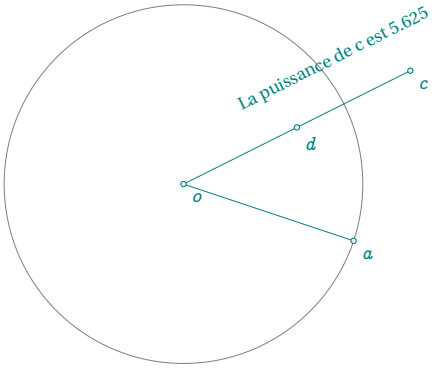


11.2.7 Method inversion (obj): point, line and circle

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.

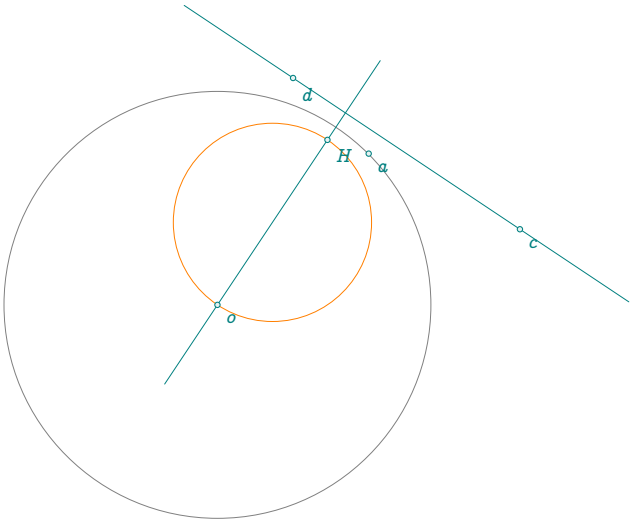
Inversion: point The `inversion` method can be used on a point, a group of points, a line or a circle. Depending on the type of object, the function determines the correct algorithm to use.

```
\directlua{%
init_elements ()
   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)
}
\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}
```



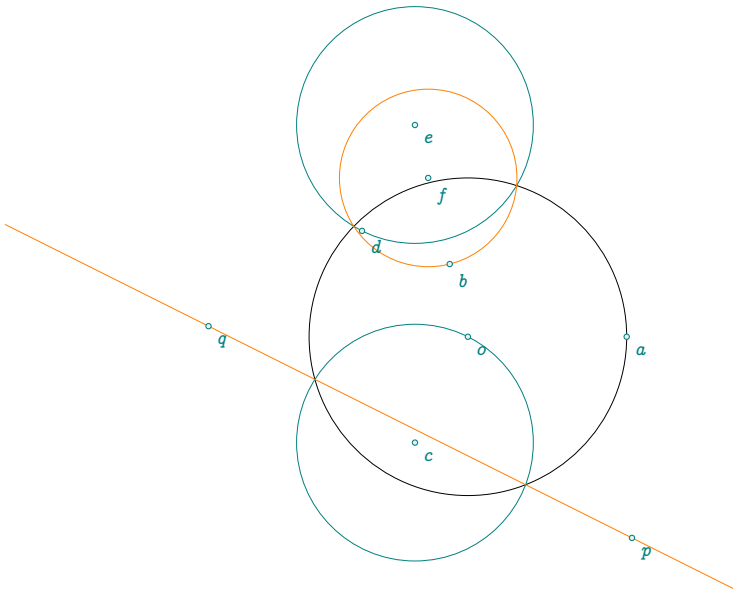
Inversion: line The result is either a straight line or a circle.

```
\directlua{%
init_elements ()
   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)
}
\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}
```



Inversion: circle The result is either a straight line or a circle.

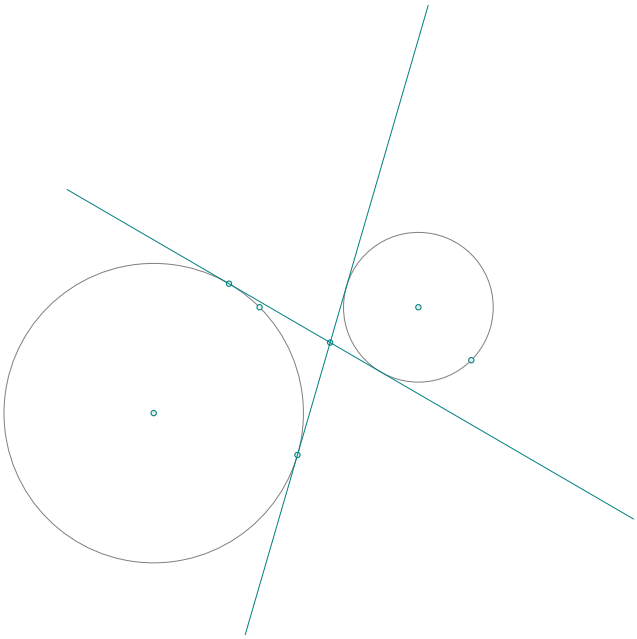
```
\directlua{%
init_elements ()
scale = .7
z.o,z.a  = point:  new (-1,3),point:  new (2,3)
z.c      = point:  new (-2,1)
z.e,z.d  = point:  new (-2,7),point: new (-3,5)
C.oa     = circle: new (z.o,z.a)
C.ed     = circle: new (z.e,z.d)
C.co     = circle: new (z.c,z.o)
obj      = C.oa: inversion (C.co)
   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"
}
\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}
```



11.2.8 Method internal_similitude

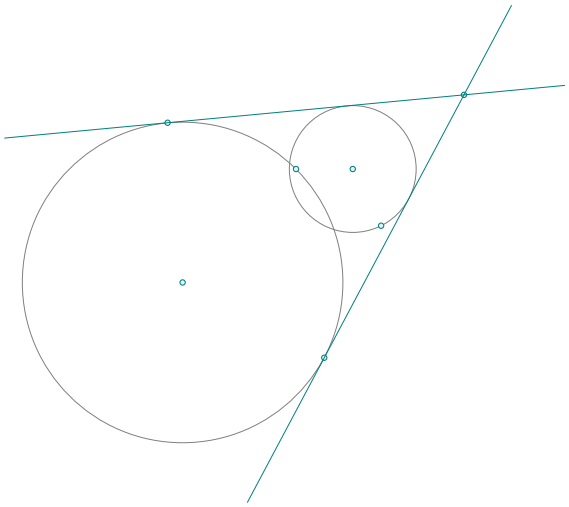
Circles are geometrically similar to one another and mirror symmetric. Hence, a pair of circles has both types of homothetic centers, internal and external, unless the centers are equal or the radii are equal; these exceptional cases are treated after general position. These two homothetic centers lie on the line joining the centers of the two given circles, which is called the line of centers. Circles with radius zero can also be included (see exceptional cases), and negative radius can also be used, switching external and internal. [Wikipedia]

```
\directlua{%
init_elements ()
  scale = 0.7
z.A  = point : new ( 0  , 0  )
z.a  = point : new ( 2 ,  2 )
z.B  = point : new ( 5  , 2  )
z.b  = point : new ( 6  , 1  )
C.Aa = circle : new (z.A,z.a)
C.Bb = circle : new (z.B,z.b)
z.I  = C.Aa : internal_similitude (C.Bb)
L.TA1,L.TA2 = C.Aa : tangent_from (z.I)
z.A1 = L.TA1.pb
z.A2 = L.TA2.pb
}
\begin{tikzpicture}
\tkzGetNodes
\tkzDrawCircles(A,a B,b)
\tkzDrawPoints(A,a,B,b,I,A1,A2)
\tkzDrawLines[add = 1 and 2](A1,I A2,I)
\end{tikzpicture}
```



11.2.9 Method external_similitude

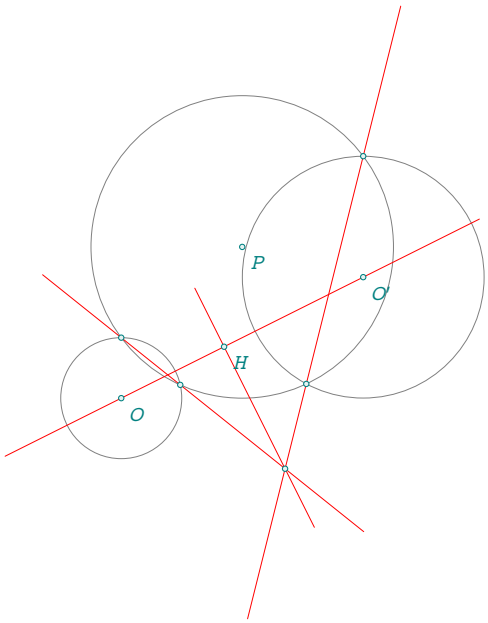
```
\directlua{%
init_elements ()
z.A  = point : new ( 0  , 0  )
z.a  = point : new ( 2 ,  2 )
z.B  = point : new ( 3  , 2  )
z.b  = point : new ( 3.5  , 1  )
C.Aa = circle : new (z.A,z.a)
C.Bb = circle : new (z.B,z.b)
z.I  = C.Aa : external_similitude (C.Bb)
L.TA1,L.TA2 = C.Aa : tangent_from (z.I)
z.A1 = L.TA1.pb
z.A2 = L.TA2.pb
}
\begin{tikzpicture}
\tkzGetNodes
\tkzDrawCircles(A,a B,b)
\tkzDrawPoints(A,a,B,b,I,A1,A2)
\tkzDrawLines[add = .25 and .1](A1,I A2,I)
\end{tikzpicture}
```



11.2.10 Method radical_center (C1,C2)

The radical lines of three circles are concurrent in a point known as the radical center (also called the power center). This theorem was originally demonstrated by Monge (Dörrie 1965, p. 153). [Weisstein, Eric W. "Radical Center." From MathWorld–A Wolfram Web Resource.]
Here I have also named `radical_center` the point of intersection of the radical axis of two circles with the centre axis. See the following example for how to obtain point *H*.

```
\directlua{%
init_elements ()
   scale    = .8
   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)
   z.X      = C.Ox : radical_center(C.Pz,C.Opy)
   %   L.OOp    = line : new (z.O,z.Op)
   %   z.H      = L.OOp : projection (z.X)
   z.H = C.Ox : radical_center(C.Opy)
}
\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}
```



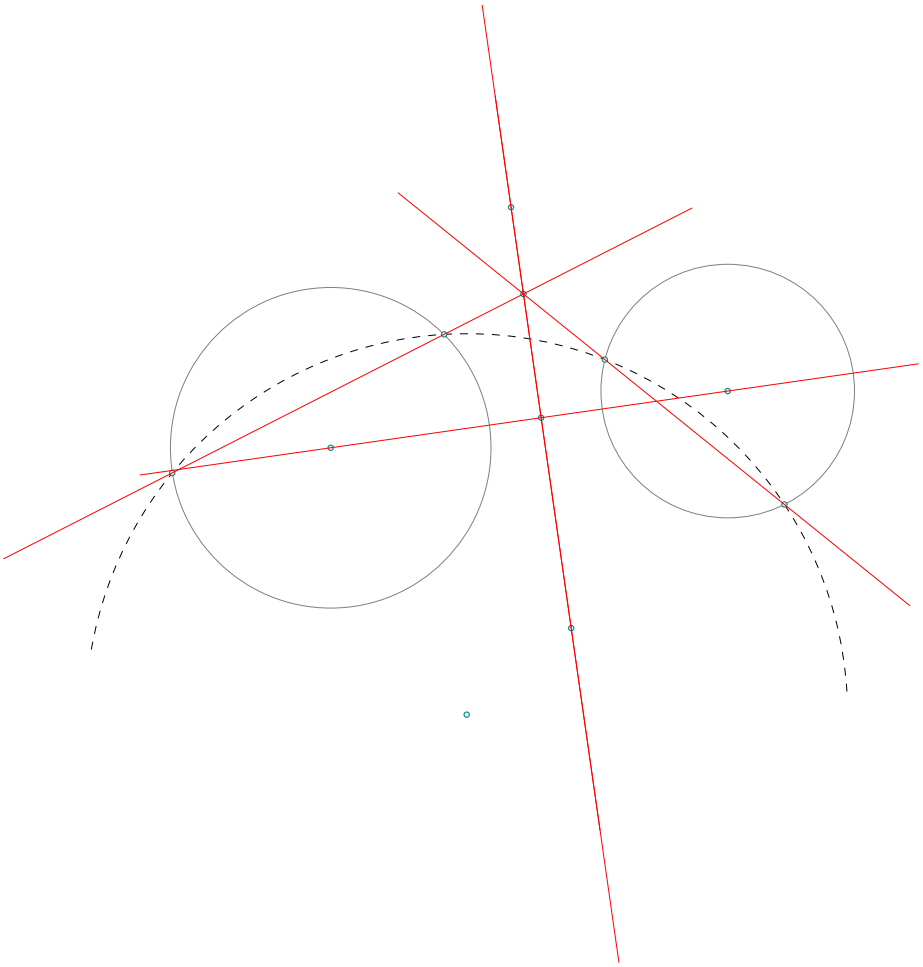
11.2.11 Method radical_axis(C)

The radical line, also called the radical axis, is the locus of points of equal circle power with respect to two non-concentric circles. By the chordal theorem, it is perpendicular to the line of centers (Dörrie 1965). [Weisstein, Eric W. "Radical Line." From MathWorld–A Wolfram Web Resource.]

Radical axis v1

```
\directlua{%
init_elements ()
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)
```

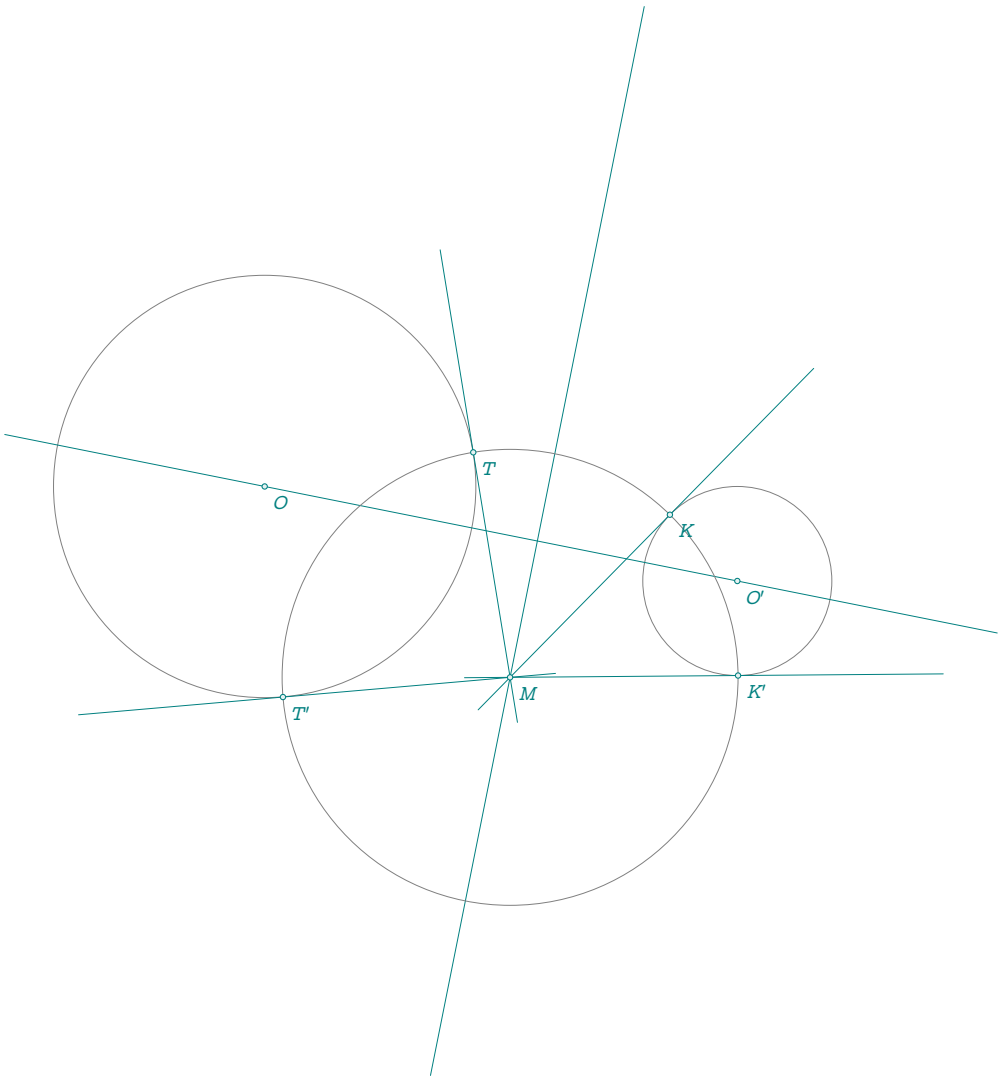
```
_,z.Bp   = intersection (C.OAp,C.YAp)
L.AB     = line : new (z.A,z.B)
L.ApBp   = line : new (z.Ap,z.Bp)
z.M      = intersection (L.AB,L.ApBp)
z.H      = L.XY : projection (z.M)
}
\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawCircles(X,B Y,A')
   \tkzDrawArc[dashed,delta=30](O,A')(A)
   \tkzDrawPoints(A,B,A',B',M,H,X,Y,O,E,F)
   \tkzDrawLines[red](A,M A',M X,Y E,F)
   \tkzDrawLines[red,add=1 and 3](M,H)
\end{tikzpicture}
```



Radical axis v2

```
\directlua{%
init_elements ()
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)
```

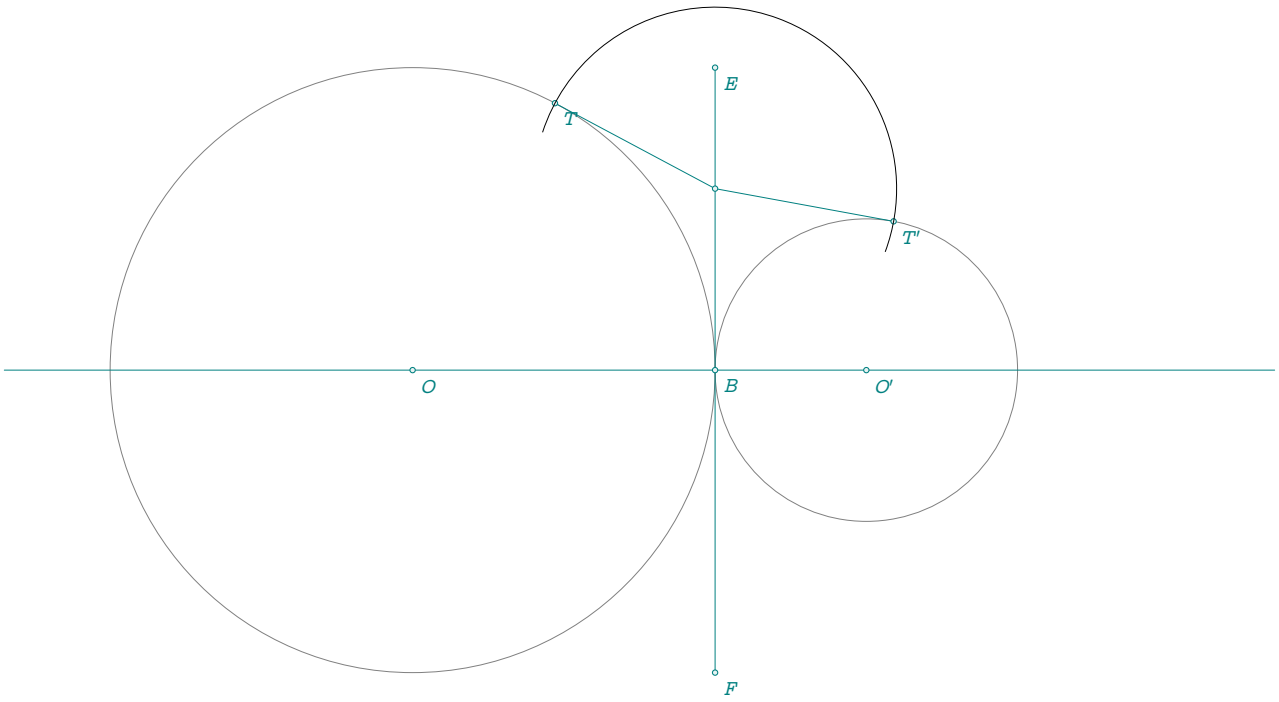
```
C.OB        = circle :    new (z.O,z.B)
C.OpD       = circle :    new (z.Op,z.D)
L.EF        = C.OB : radical_axis (C.OpD)
z.E,z.F     = get_points (L.EF)
z.M         = L.EF : point (.75)
L.MT,L.MTp  = C.OB : tangent_from (z.M)
_,z.T       = get_points (L.MT)
_,z.Tp      = get_points (L.MTp)
L.MK,L.MKp  = C.OpD : tangent_from (z.M)
_,z.K       = get_points (L.MK)
_,z.Kp      = get_points (L.MKp)
}
\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawCircles(O,B O',D)
   \tkzDrawLine(E,F)
   \tkzDrawLine[add=.25 and .25](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}
```



Radical axis v3

```
   \directlua{%
init_elements ()
   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)
}
\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawCircles(O,B O',B)
```

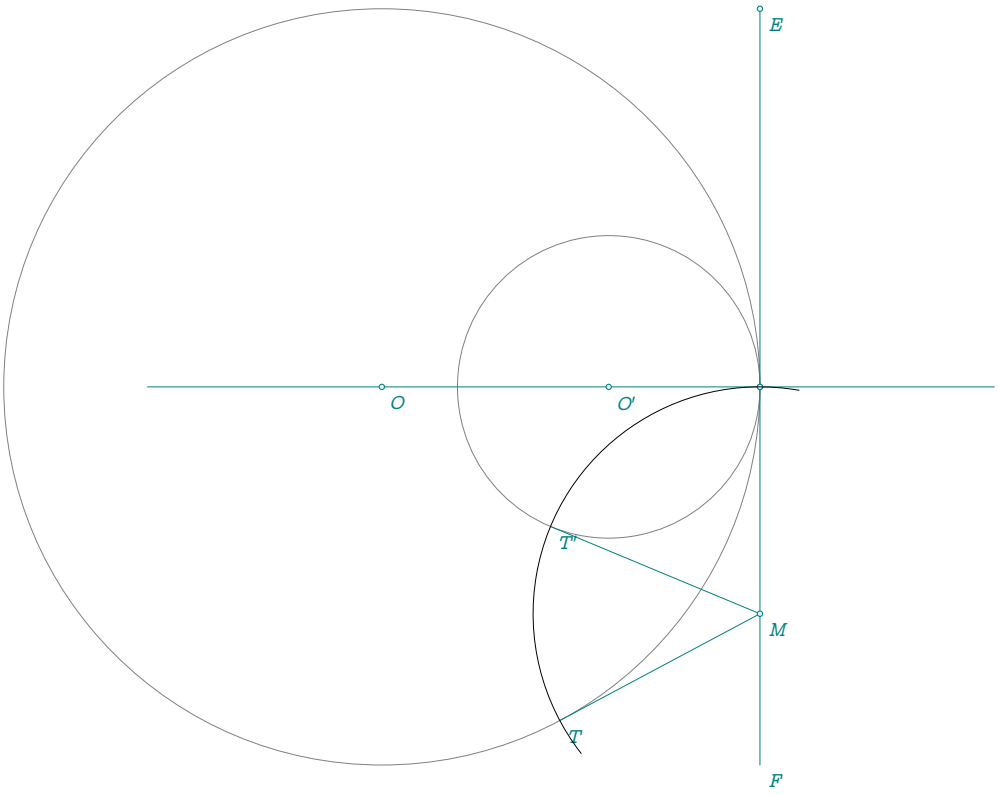
```
   \tkzDrawSegments(M,T M,T')
   \tkzDrawSegments(E,F)
   \tkzDrawLine[add=.5 and .5](O,O')
   \tkzDrawPoints(O,B,O',E,F,M,T,T')
   \tkzLabelPoints(O,O',B,E,F,T,T')
   \tkzDrawArc(M,T')(T)
\end{tikzpicture}
```



Radical axis v4

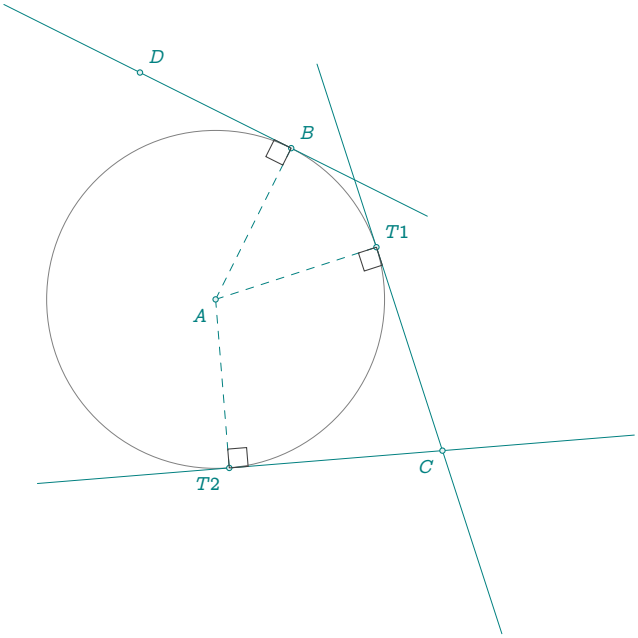
```
\directlua{%
init_elements ()
   z.O     = point : new (0,0)
   z.B     = point : new (5,0)
   z.Op    = point : new (3,0)
   C.OB    = circle :    new (z.O,z.B)
   C.OpB   = circle :    new (z.Op,z.B)
   L.EF    = C.OB : radical_axis (C.OpB)
   z.E,z.F = get_points(L.EF)
   z.H     = L.EF.mid
   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)
}
\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}
```



11.2.12 Methods tangent_at (P) and tangent_from (P)

```
 \directlua{%
init_elements ()
   z.A  = point:  new (0,0)
   z.B  = point:  new (1,2)
   C.AB = circle: new (z.A,z.B)
   z.C  = point:  new (3,-2)
   L.T  = C.AB : tangent_at (z.B)
   z.D  = L.T.pb
   L.T1,L.T2 = C.AB : tangent_from (z.C)
   z.T1 = L.T1.pb
   z.T2 = L.T2.pb
}
\begin{tikzpicture}
\tkzGetNodes
\tkzDrawCircle(A,B)
\tkzDrawLines[add =.5 and .5](B,D C,T1 C,T2)
\tkzDrawSegments[dashed](A,B A,T1 A,T2)
\tkzDrawPoints(A,...,D,T1,T2)
\tkzLabelPoints[below left](A,T2,C)
\tkzLabelPoints[above right](B,T1,D)
\tkzMarkRightAngles(A,B,D A,T1,C A,T2,C)
\end{tikzpicture}
```

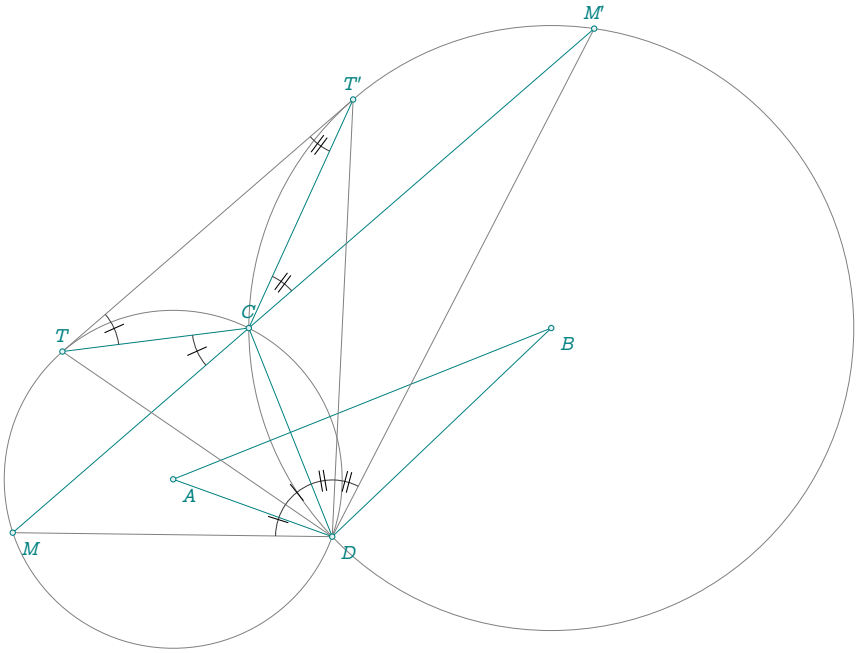


11.2.13 Common tangent: Angle of two intersecting circles

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.

```
\directlua{%
init_elements ()
   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)
}
\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}
```

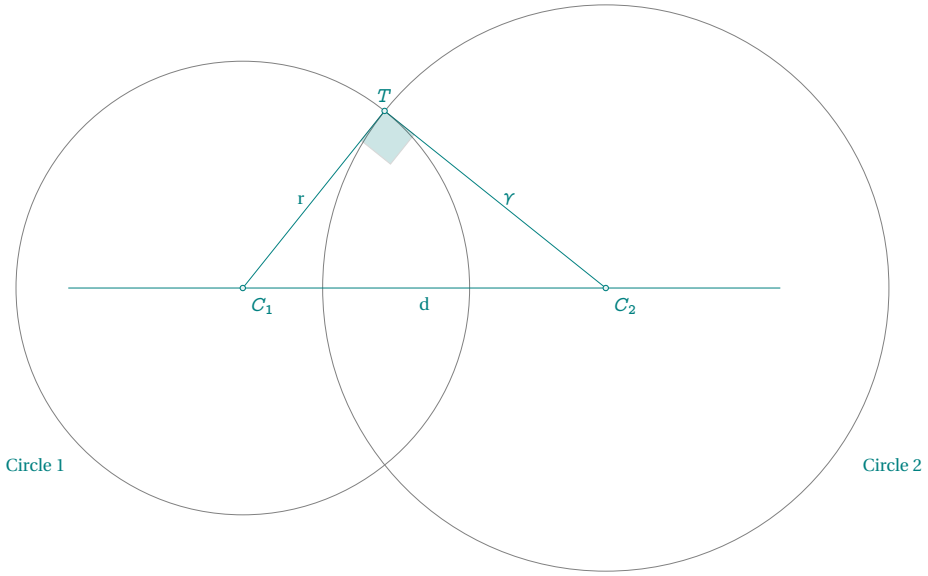


11.2.14 Method orthogonal_from (pt)

In geometry, two circles are said to be orthogonal if their respective tangent lines at the points of intersection are perpendicular (meet at a right angle). [wikipedia]
This method determines a circle with a given centre, orthogonal to a circle that is also given.

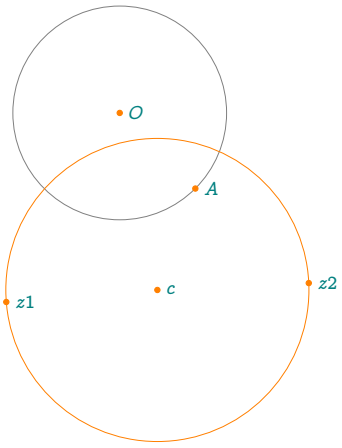
```
\directlua{%
init_elements ()
   scale    = .6
   z.C_1    = point: new (0,0)
   z.C_2    = point: new (8,0)
   z.A      = point: new (5,0)
   C        = circle: new (z.C_1,z.A)
   z.S,z.T  = get_points (C: orthogonal_from (z.C_2))
}
\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawCircles(C_1,T C_2,T)
   \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}
```



11.2.15 Method orthogonal_through

```
\directlua{%
init_elements ()
   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
}
\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}
```



11.2.16 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;

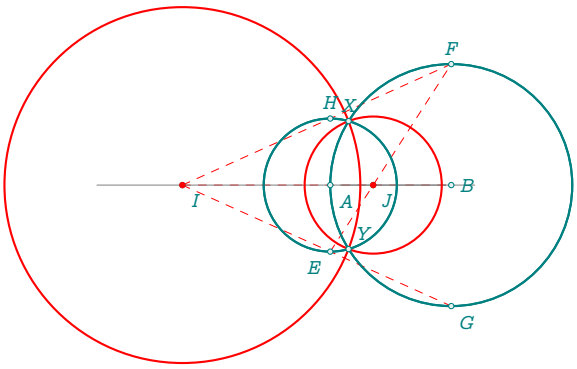
(iv) If the circles intersect in a single point, the unique midcircle has center at the external similitude center or at internal similitude center.

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.

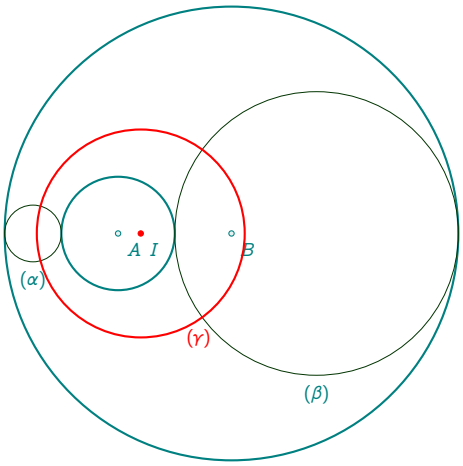
Consider two intersecting circles $(\mathcal{A})$ and $(\mathcal{B})$. We can obtain the centers of similarity of these two circles by constructing EH and FG two diameters parallel of the circles $(\mathcal{A})$ and $(\mathcal{B})$. The line (GE) intercepts the line (AB) in J and the line (EF) intercepts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

```
\directlua{%
init_elements ()
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)
}
```



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

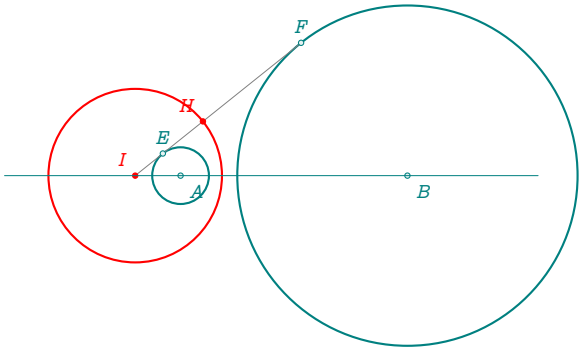
```
\directlua{%
init_elements ()
      scale =.75
   z.A = point : new ( 3 , 0 )
   z.B = point : new ( 5 , 0 )
   z.O = point : new ( 2 , 0 )
   z.P = point : new ( 1 , 0 )
   L.AB = line :  new (z.A,z.B)
   C.AO = circle : new (z.A,z.O)
   C.BP = circle : new (z.B,z.P)
   z.R,z.S = intersection (L.AB,C.BP)
   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)
}
```



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 $IH^2 = IE \times IF$.

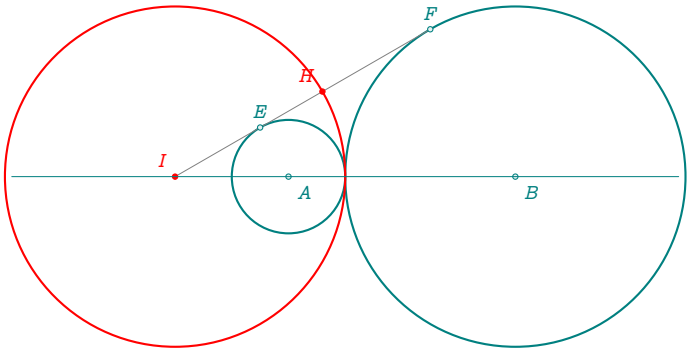
```
\directlua{%
init_elements ()
scale=.75
local a,b,c,d
z.A = point : new ( 0 , 0 )
z.B = point : new ( 4 , 0 )
z.a = point :  new ( .5 , 0)
z.b = point :  new ( 1 , 0)
C.Aa = circle  :  new (z.A,z.a)
C.Bb = circle  :  new (z.B,z.b)
L.AB = line :  new (z.A,z.B)
z.E = C.Aa.north
z.F = C.Bb.north
L.EF = 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
}
```



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

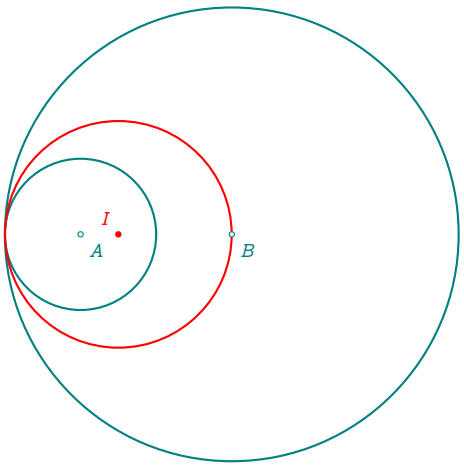
- $(\mathcal{B})$ being external and angent to $(\mathcal{A})$. The construction is identical to the previous one.

```
\directlua{%
init_elements ()
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
}
```



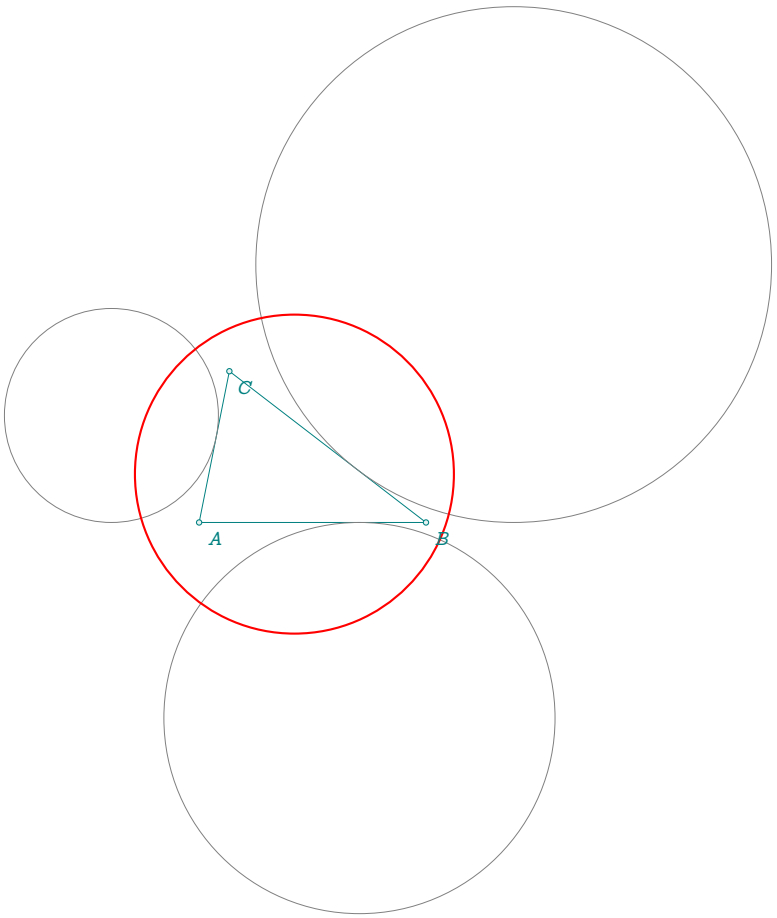
- When one of the given circles is inside and tangent to the other, the construction is easy.

```
\directlua{%
init_elements ()
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)
}
```



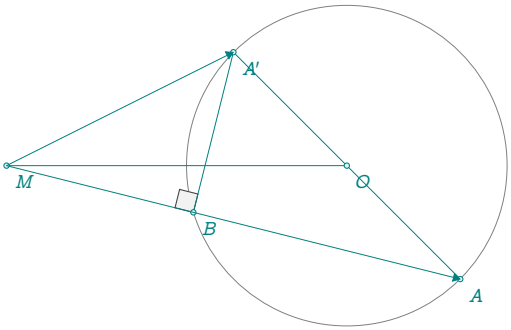
11.2.17 Radical circle

```
\directlua{%
init_elements ()
   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 )
   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)
}
\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}
```



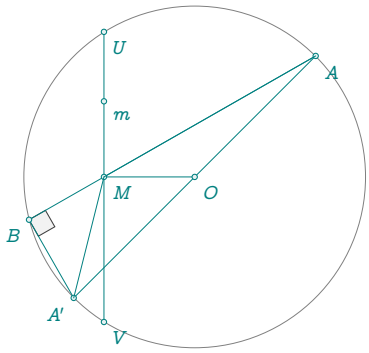
11.2.18 Method power(C) Power v1

```
\directlua{%
init_elements ()
   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)
}
\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}
```



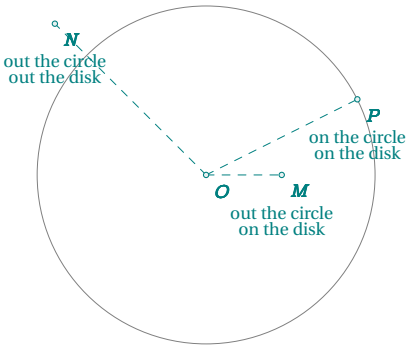
11.2.19 Method power(C) Power v2

```
\directlua{%
init_elements ()
   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)
}
\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}
```



11.2.20 Method in_out for circle and disk

```
  \directlua{%
init_elements ()
  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)
  }
```



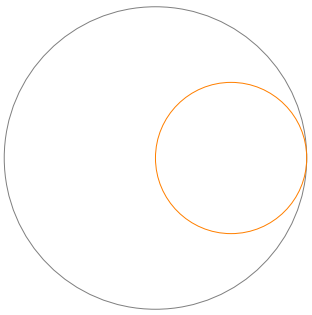
```
\def\tkzPosPoint#1#2#3#4{%
\tkzLabelPoints(O,M,N,P)
   \ifthenelse{\equal{\tkzUseLua{#1}}{true}}{
   \tkzLabelPoint[below=#4pt,font=\scriptsize](#2){on  the #3}}{%
   \tkzLabelPoint[below=#4pt,font=\scriptsize](#2){out  the #3}}}
\begin{tikzpicture}
\tkzGetNodes
\tkzDrawSegments[dashed](O,M O,N O,P)
\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}
```

11.2.21 Method circles_position

This function returns a string indicating the position of the circle in relation to another. Useful for creating a function. Cases are:

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

```
\directlua{%
init_elements ()
   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
}

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



12 Class triangle

12.1 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 )`

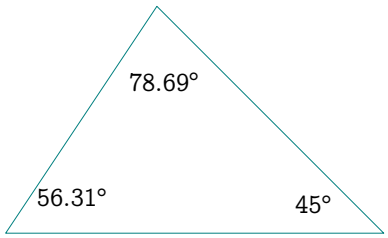
(Refer to examples: 24.6; 24.8; 24.15). Multiple attributes are then created.

Table 9: Triangle attributes.

Attributes	Application
pa	T.ABC.pa
pb	T.ABC.pb
pc	T.ABC.pc
type	'triangle'
circumcenter	T.ABC.circumcenter; [12.2.1]
centroid	T.ABC.centroid
incenter	T.ABC.incenter
orthocenter	T.ABC.orthocenter
eulercenter	T.ABC.eulercenter
spiekercenter	T.ABC.spiekercenter; [3.1.4]
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

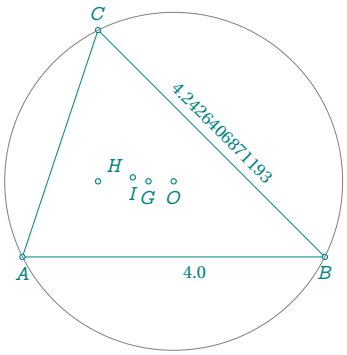
12.2 Triangle attributes: angles

```
\directlua{%
init_elements ()
  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)
}
\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}
```



12.2.1 Example: triangle attributes

```
\directlua{%
init_elements ()
   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
}
\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawPolygon(A,B,C)
   \tkzDrawPoints(A,B,C,O,G,I,H)
   \tkzLabelPoints[below](A,B,O,G,I)
   \tkzLabelPoints[above right](H,C)
   \tkzDrawCircles(O,A)
   \tkzLabelSegment[sloped](A,B){\tkzUseLua{c}}
   \tkzLabelSegment[sloped,above](B,C){\tkzUseLua{a}}
\end{tikzpicture}
```



12.3 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)	[12.2] [a]
Points		
lemoine_point ()	T.ABC : lemoine_point () intersection of the symmedians	[10.2.3]
symmedian_point ()	Lemoine point or the Grebe point	[12.3.26]
lemoine_point ()	symmedian point or the Grebe point	[12.3.26]
bevan_point ()	Circumcenter of the excentral triangle	[12.3.20]
mittenpunkt_point ()	Symmedian point of the excentral triangle	[12.3.3]
gergonne_point ()	Intersection of the three cevians that lead to the contact points	[12.3.1]
nagel_point ()	Intersection of the three cevians that lead to the extouch points	[12.3.2]
feuerbach_point ()	The point at which the incircle and euler circle are tangent.	[12.3.21]
spieker_center ()	Incenter of the medial triangle	[24.33]
barycentric (ka,kb,kc)	T.ABC: barycentric (2,1,1) barycenter of ({A,2},{B,1},{C,1})	Remark [b]
base (u,v)	z.D = T.ABC: base(1,1) -> ABDC is a parallelogram	[12.3.7]
trilinear (u,v,w)	z.D = T.ABC: trilinear(1,1,1) -> ABDC parallelogram	[12.3.5]
projection (p)	Projection of a point on the sides	[24.22; 12.3.4]
euler_points ()	Euler points of euler circle	[12.3.8]
nine_points ()	9 Points of the euler circle	[12.3.9]
parallelogram ()	z.D = T.ABC : parallelogram () -> ABCD parallelogram	[24.17]
Lines		
altitude (n)	L.AHa = T.ABC : altitude () n empty or 0 line from A [c]	[12.3.10]
bisector (n)	L.Bb = T.ABC : bisector (1) n = 1 line from B [d]	[12.3.11]
bisector_ext(n)	n=2 line from the third vertex.	[12.3.31]
symmedian_line (n)	Cevian with respect to Lemoine point.	[12.3.26 ; 10.2.3]
euler_line ()	the line through N ,G, H and O if the triangle is not equilateral [e]	[24.31]
antiparallel(pt,n)	n=0 antiparallel through pt to (BC), n=1 to (AC) etc.	[24.53]
Circles		
euler_circle ()	C.NP = T.ABC : euler_circle () -> N euler point [f]	[12.3.12]
circum_circle ()	C.OA = T.ABC : circum () Triangle's circumscribed circle	[12.3.13]
in_circle ()	Inscribed circle of the triangle	[12.3.14]
ex_circle (n)	Circle tangent to the three sides of the triangle ; n =1 swap ; n=2 2 swap	[12.3.15]
first_lemoine_circle ()	The center is the midpoint between Lemoine point and the circumcenter.[g]	[24.51]
second_lemoine_circle ()		24.53]
spieker_circle ()	The incircle of the medial triangle	[12.3.16]
bevan_circle ()	Circumscribed circle of a excentral triangle	[12.3.20]
cevian_circle ()	Circumscribed circle of a Cevian triangle	[12.3.17]
symmedial_circle ()	Circumscribed circle of a symmedial triangle	[12.3.26]
pedal_circle ()	Circumscribed circle of the podar triangle	[12.3.18]
conway_circle ()	Circumscribed circle of Conway points	[12.3.19]

a T or T.name with what you want for name, is possible.
b The function barycenter is used to obtain the barycentre for any number of points
c z.Ha = L.AHa.pb recovers the common point of the opposite side and altitude. The method orthic is usefull. If you don't need to use the triangle object several times, you can obtain a bisector or a altitude with the function altitude (z.A,z.B,z.C) ; [28]
d _,z.b = get_points(L.Bb) recovers the common point of the opposite side and bisector. If you don't need to use the triangle object several times, you can obtain a bisector with the function bisector (z.A,z.B,z.C) [28]
e N center of nine points circle, G centroid, H orthocenter , O circum center
f 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.
g 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.

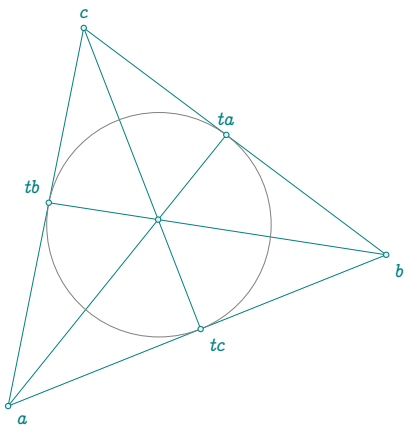
Methods	Comments
Triangles	
orthic ()	T = T.ABC : orthic () triangle joining the feet of the altitudes ; [12.3.10]
medial ()	T = T.ABC : medial () triangle with vertices at the midpoints; [12.3.23 ; 24.25 ; 12.3.26]
incentral ()	Cevian triangle of the triangle with respect to its incenter. [12.3.24]
excentral ()	Triangle with vertices corresponding to the excenters. [12.3.21]
extouch ()	Triangle formed by the points of tangency with the excircles. [24.15]
intouch ()	Contact triangle formed by the points of tangency of the incircle [12.3.1]
contact ()	contact = intouch ; [12.3.1]
tangential ()	Triangle formed by the lines tangent to the circumcircle at the vertices; [12.3.25]
feuerbach ()	Triangle formed by the points of tangency of the euler circle with the excircles; [12.3.21]
anti ()	Anticomplementary Triangle The given triangle is its medial triangle.[a] ; [12.3.27]
cevian (pt)	Triangle formed with the endpoints of the three cevians with respect to pt; [12.3.17]
pedal (pt)	Triangle formed by projections onto the sides of pt [12.3.18]
symmedial ()	Triangle formed with the intersection points of the symmedians ; [12.3.26]
euler ()	Triangle formed with the euler points ; [12.3.8]
similar ()	Triangle formed with straight lines parallel to the sides [12.3.22]
Ellipses	
steiner_inellipse ()	[ex. 12.3.30]
steiner_circumellipse ()	[ex. 12.3.30]
euler_ellipse ()	[ex. (12.3.29]
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

a You can use **similar** instead of **anti**.

12.3.1 Gergonne point

In this example, some usefull methods are applied like intouch or contact. The points of contact of the inscribed circle (incircle) with the triangle in question are obtained.

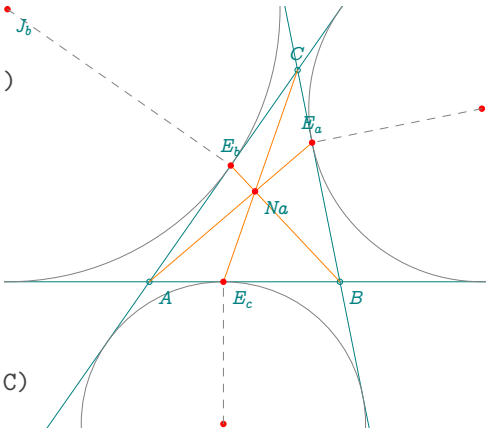
```
\directlua{%
init_elements ()
z.a  = point: new(1,0)
z.b  = point: new(6,2)
z.c  = point: new(2,5)
T    = triangle : new (z.a,z.b,z.c)
z.g  = T : gergonne_point ()
z.i  = T.incenter
z.ta,z.tb,z.tc = get_points (T :  intouch ())
}
  \begin{tikzpicture}
  \tkzGetNodes
  \tkzDrawPolygons(a,b,c)
  \tkzDrawSegments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



12.3.2 Method Nagel_point

Let E_a be the point at which the J_a-excircle meets the side (BC) of a triangle ABC, and define E_b and E_c similarly. Then the lines A,E_a, B,E_b and C,E_c concur in the Nagel point Na.

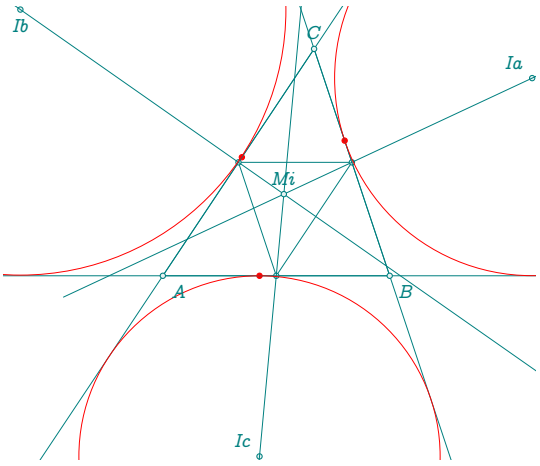
```
\directlua{%
init_elements ()
  scale         = .7
  z.A           = point :   new (0,0)
  z.B           = point :   new (3.6,0)
  z.C           = point :   new (2.8,4)
  T.ABC         = triangle: new (z.A,z.B,z.C)
  z.Na          = T.ABC : nagel_point ()
  z.J_a,z.J_b,
  z.J_c         = get_points (T.ABC : excentral ())
  z.E_a,z.E_b,
  z.E_c         = get_points (T.ABC : extouch ())
}
\begin{tikzpicture}
  \tkzGetNodes
  \tkzDrawPoints(A,B,C)
  \tkzDrawPoints[red,size=2](J_a,J_b,J_c)
  \tkzClipBB
  \tkzDrawLines[add=1.75 and 1.75,teal](A,B A,C B,C)
  \tkzDrawCircles(J_a,E_a J_b,E_b J_c,E_c)
  \tkzDrawSegments[dashed,gray](J_a,E_a J_b,E_b J_c,E_c)
  \tkzDrawSegments[orange](A,E_a B,E_b C,E_c)
  \tkzDrawPoints[red,size=2](Na,E_a,E_b,E_c)
  \tkzLabelPoints(A,B,Na)
  \tkzLabelPoints(E_c,J_a,J_b,J_c)
  \tkzLabelPoints[above](E_a,E_b,C)
\end{tikzpicture}
```



12.3.3 Method mittenpunkt

The Mittenpunkt is the symmedian point of the excentral triangle. The mittenpunkt (also called the middles-point) of a triangle is the symmedian point of the excentral triangle, i.e., the point of concurrence of the lines from the excenters through the corresponding triangle side midpoints. [Weisstein, Eric W. "Mittenpunkt." From MathWorld–A Wolfram Web Resource.]

```
\directlua{%
init_elements ()
  scale = 1
  z.A   = point : new ( 0   , 0  )
  z.B   = point : new ( 6   , 0  )
  z.C   = point : new ( 4   , 6  )
  T     = triangle : new (z.A,z.B,z.C)
  z.Ma,
  z.Mb,
  z.Mc  = get_points (T : medial ())
  z.Ia,z.Ib,z.Ic = get_points(T : excentral ())
  z.Mi  = T : mittenpunkt_point ()
  T.int = T : extouch ()
  z.Ta,z.Tb,z.Tc = get_points(T.int)
}
```

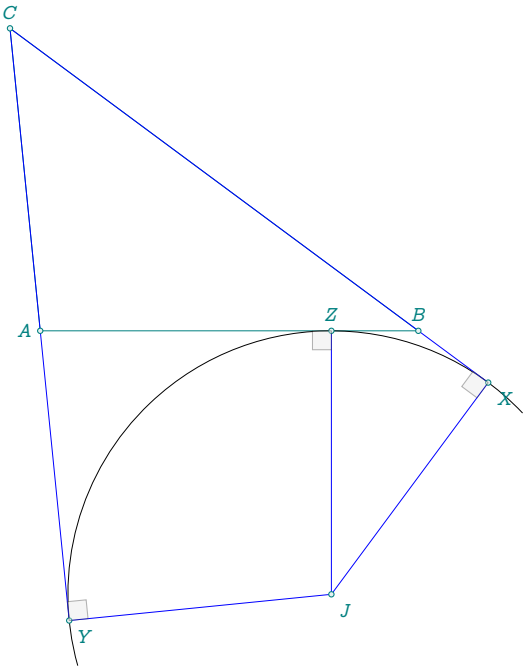


```
\begin{tikzpicture}[scale=.5]
  \tkzGetNodes
  \tkzDrawPolygons[](A,B,C Ma,Mb,Mc)
  \tkzDrawPoints(Ma,Mb,Mc,Ia,Ib,Ic)
  \tkzDrawPoints[red](Ta,Tb,Tc)
  \tkzLabelPoints[below](Ib)
  \tkzLabelPoints[above left](Ia,Ic)
  \tkzClipBB
  \tkzDrawLines[add=0 and 1](Ia,Ma Ib,Mb Ic,Mc)
  \tkzDrawLines[add=1 and 1](A,B A,C B,C)
  \tkzDrawCircles[red](Ia,Ta Ib,Tb Ic,Tc)
  \tkzDrawPoints(B,C,A,Mi)
  \tkzLabelPoints(B,A)
  \tkzLabelPoints[above](C,Mi)
\end{tikzpicture}
```

12.3.4 Method projection

This involves obtaining the projections of a point onto the sides of a triangle. In the following example, we are going to find the projections of a centre of an exinscribed circle.

```
\directlua{%
init_elements ()
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,_  = get_points(T.ABC: ex_circle (2))
z.X ,
z.Y,
z.Z    = T.ABC : projection (z.J)
}
```

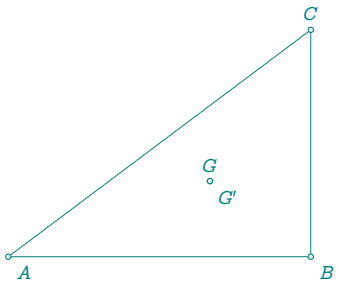



```
\begin{center}
  \begin{tikzpicture}
  \tkzGetNodes
  \tkzDrawPolygon(A,B,C)
  \tkzDrawArc(J,X)(Y)
  \tkzDrawSegments[blue](J,X J,Y J,Z C,Y C,X)
  \tkzDrawPoints(A,B,C,J,X,Y,Z)
  \tkzLabelPoints(J,X,Y)
  \tkzLabelPoints[above](C,B,Z)
  \tkzLabelPoints[left](A)
  \tkzMarkRightAngles[fill=gray!20,opacity=.4](A,Z,J A,Y,J J,X,B)
  \end{tikzpicture}
\end{center}
```

12.3.5 Method trilinear

Given a reference triangle ABC, the trilinear coordinates of a point P with respect to ABC are an ordered triple of numbers, each of which is proportional to the directed distance from P to one of the side lines. Trilinear coordinates are denoted alpha:beta:gamma or (alpha,beta,gamma) and also are known as homogeneous coordinates or "trilinears." Trilinear coordinates were introduced by Plücker in 1835. [Weisstein, Eric W. "Trilinear Coordinates." From MathWorld–A Wolfram Web Resource.]

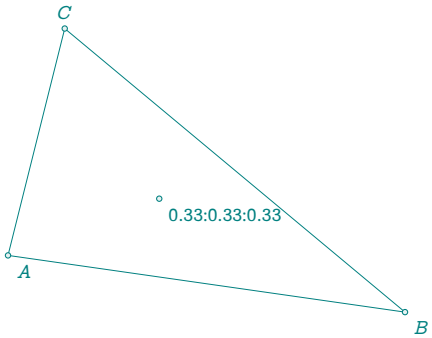
```
\directlua{%
init_elements ()
z.A = point : new ( 0  , 0  )
z.B = point : new ( 4  , 0  )
z.C = point : new ( 4 , 3 )
T.ABC = triangle : new ( z.A , z.B , z.C )
a = T.ABC.a
b = T.ABC.b
c = T.ABC.c
z.Gp = T.ABC : trilinear (b*c,a*c,a*b)
z.G = T.ABC : barycentric (1,1,1)
}
\begin{tikzpicture}
\tkzGetNodes
\tkzDrawPolygon(A,B,C)
\tkzDrawPoints(A,B,C,G',G)
\tkzLabelPoints(A,B,G')
\tkzLabelPoints[above](C,G)
\end{tikzpicture}
```



12.3.6 Method barycentric_coordinates

This method produces a triplet of coordinates which are the barycentric coordinates of a point as a function of the three points of a given triangle.

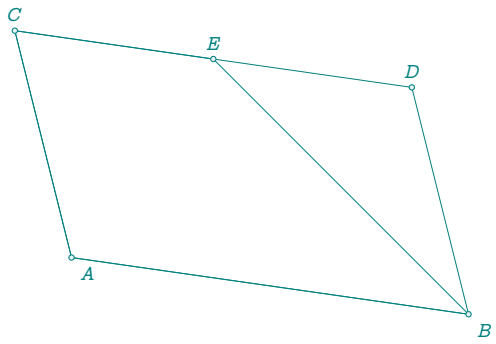
```
\directlua{%
init_elements ()
  z.A       = point: new (1,1)
  z.B       = point: new (8,0)
  z.C       = point: new (2,5)
  T         = triangle: new(z.A,z.B,z.C)
  z.G       = T.centroid
  ca,cb,cc  = T : barycentric_coordinates  (z.G)
}
\begin{tikzpicture}
  \tkzGetNodes
  \tkzDrawPolygons(A,B,C)
  \tkzDrawPoints(A,B,C,G)
  \tkzLabelPoints(A,B,C,G)
  \tkzLabelPoint(G){\pmpn{\tkzUseLua{ca}}:\pmpn{\tkzUseLua{cb}}:\pmpn{\tkzUseLua{cc}}}
\end{tikzpicture}
```



12.3.7 Method base

In the next example, the point D is defined by $\overrightarrow{AD} = 1 \cdot \overrightarrow{AB} + 1 \cdot \overrightarrow{AC}$.

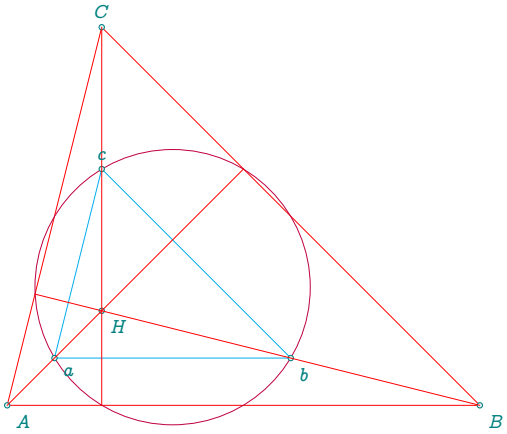
```
\directlua{%
init_elements ()
    scale         = .75
    z.A           = point: new (1,1)
    z.B           = point: new (8,0)
    z.C           = point: new (0,5)
    z.X           = point: new (2,2)
    T             = triangle: new(z.A,z.B,z.C)
    z.D           = T : base (1,1)
    z.E           = T : base (.5,1)
}
\begin{tikzpicture}
     \tkzGetNodes
     \tkzDrawPolygons(A,B,D,C A,B,E,C)
     \tkzDrawPoints(A,B,C,D,E)
     \tkzLabelPoints(A,B)
     \tkzLabelPoints[above](C,D,E)
\end{tikzpicture}
```



12.3.8 Method euler_points

The points a, b and c are the Euler points. They are the midpoints of the segments AH, BH and CH.

```
  \directlua{%
init_elements ()
    scale     = 1.25
    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.N       = T.eulercenter
    z.a,
    z.b,
    z.c       = get_points (T : euler ())
    z.H       = T.orthocenter
    T.orthic  = T: orthic()
    z.Ha,
    z.Hb,
    z.Hc      = get_points (T.orthic)
  }
  \begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawPolygons[red](A,B,C)
   \tkzDrawPolygons[cyan](a,b,c)
   \tkzDrawCircle[purple](N,a)
   \tkzDrawPoints(a,b,B,C,A,c,H)
   \tkzDrawSegments[red](C,Hc B,Hb A,Ha)
   \tkzLabelPoints(A,B,a,b,H)
   \tkzLabelPoints[above](c,C)
  \end{tikzpicture}
```

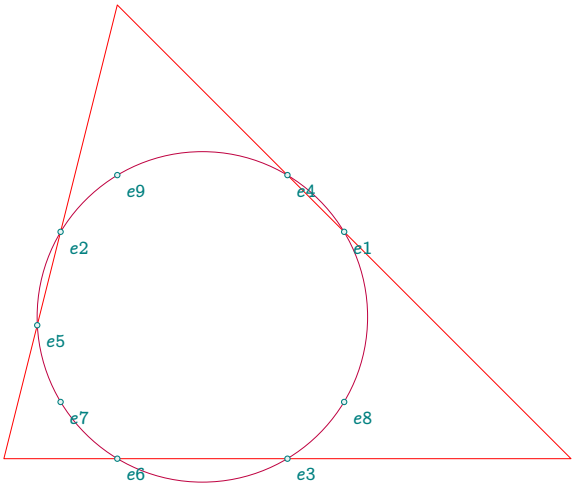


12.3.9 Method nine_points

This method gives the nine main points belonging to the Euler circle: in order, first the midpoints of the sides of the triangle, then the feet of the altitudes and finally the three Euler points. Refer to the last example. In the next

example, we look for the centre of gravity in two different ways: the first uses the `trilinear` method, the second the `barycentric` method.

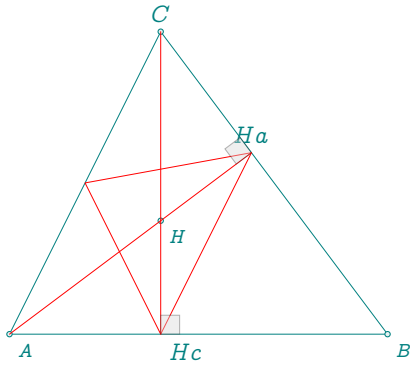
```
\directlua{%
init_elements ()
  scale     = 1.5
  z.A       = point: new (0,0)
  z.B       = point: new (5,0)
  z.C       = point: new (1,4)
    T       = triangle: new(z.A,z.B,z.C)
  z.N       = T.eulercenter
  z.e1,
  z.e2,
  z.e3,
  z.e4,
  z.e5,
  z.e6,
  z.e7,
  z.e8,
  z.e9      = T : nine_points ()
}
\begin{tikzpicture}
 \tkzGetNodes
 \tkzDrawPolygons[red](A,B,C)
 \tkzDrawCircle[purple](N,e1)
 \tkzDrawPoints(e1,e2,e3,e4,e5,e6,e7,e8,e9)
  \tkzLabelPoints(e1,e2,e3,e4,e5,e6,e7,e8,e9)
\end{tikzpicture}
```



12.3.10 Method altitude

There are several methods to obtain one or more altitudes of a triangle. One possible method is the `orthic` method. This method allows for defining the `orthic` triangle whose vertices are the feet of the altitudes from each vertex. If only one altitude is needed, one can use the `altitude(n)` method. The numeric value n can be 0, 1, or 2. By default, if it is absent, it is considered to be 0. Considering the triangle ABC, $n = 0$ means no cyclic permutation of the vertices, and the altitude will be from the first point, here A. If $n = 1$, the point trio BCA is considered, and the altitude will be from B. For $n = 2$, the altitude will be from C.

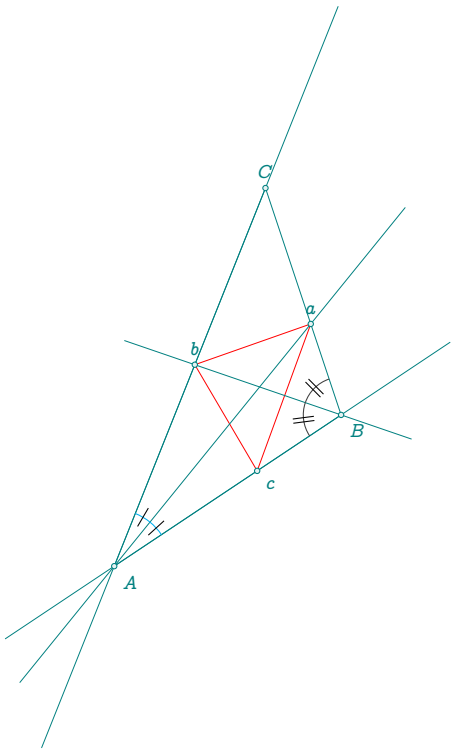
```
\directlua{%
init_elements ()
  z.A       = point: new (0,0)
  z.B       = point: new (5,0)
  z.C       = point: new (2,4)
    T       = triangle: new(z.A,z.B,z.C)
  z.H       = T.orthocenter
  L.HA  = T : altitude ()
  L.HC = T :altitude (2)
  z.Hc = L.HC.pb
  z.Ha = L.HA.pb
  z.a,z.b,z.c = get_points (T : orthic ())
}
\begin{tikzpicture}
 \tkzGetNodes
 \tkzDrawPolygon(A,B,C)
 \tkzDrawPolygon[red](a,b,c)
 \tkzDrawPoints(A,B,C,H)
 \tkzDrawSegments[red](C,Hc A,Ha)
 \tkzLabelPoints(A,B,H)
 \tkzLabelPoints[font=\small](Hc)
 \tkzLabelPoints[font=\small,above](Ha,C)
 \tkzMarkRightAngles[fill = gray!30,opacity=.4](B,Hc,C A,Ha,C)
 \end{tikzpicture}
```



12.3.11 Method bisector

There are several methods to obtain one or more bisectors of a triangle. One possible method is the `incentral` method. This method allows for defining the `incentral` triangle whose vertices are the feet of the bisectors from each vertex. If only one bisector is needed, one can use the `bisector(n)` method. The numeric value n can be 0, 1, or 2. By default, if it is absent, it is considered to be 0. Considering the triangle ABC, $n = 0$ means no cyclic permutation of the vertices, and the bisector will be from the first point, here A. If $n = 1$, the point trio BCA is considered, and the bisector will be from B. For $n = 2$, the bisector will be from C.

```
\directlua{%
init_elements ()
   z.A    = point : new (0 , 0)
   z.B    = point : new (3 , 2)
   z.C    = point : new (2 , 5)
   T.ABC  = triangle : new ( z.A , z.B , z.C )
   L.AE   = T.ABC : bisector ()
   z.E    = L.AE.pb
   z.F    = T.ABC : bisector (1).pb
   z.a,z.b,z.c = get_points (T.ABC : incentral ())
}
\begin{tikzpicture}
 \tkzGetNodes
 \tkzDrawPolygon(A,B,C)
 \tkzDrawPolygon[red](a,b,c)
 \tkzDrawLines(A,B A,C A,E B,F)
 \tkzDrawPoints(A,B,C,a,b,c)
 \tkzLabelPoints(A,B,c)
 \tkzLabelPoints[above](C,b,a)
 \tkzMarkAngles[mark=|](B,A,a a,A,C)
 \tkzMarkAngles[mark=||](C,B,b b,B,A)
\end{tikzpicture}
```

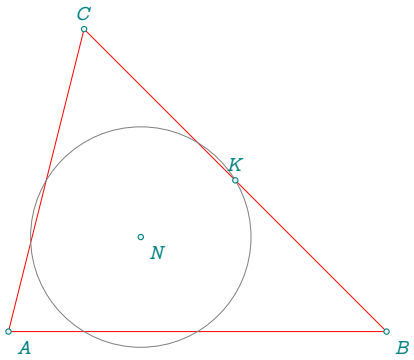


12.3.12 Method euler_circle

The nine-point circle, also called Euler's circle or the Feuerbach circle, is the circle that passes through the perpendicular feet $H_A, H_B, and H_C$ dropped from the vertices of any reference triangle DeltaABC on the sides opposite them. Euler showed in 1765 that it also passes through the midpointsM_A, M_B, M_C of the sides of DeltaABC. By Feuerbach's theorem, the nine-point circle also passes through the midpoints $E_A, E_B, and E_C$ of the segments that join the vertices and the orthocenter H. These points are commonly referred to as the Euler points. Weisstein, Eric W. "Nine-Point Circle." From MathWorld–A Wolfram Web Resource.

There are several ways of obtaining the Euler circle. The first would be to use an attribute of the triangle to determine the centre. This centre is defined by `z.N = T.eulercenter`. Next, the circle passes through the midpoint of one of the sides. IF this circle is useful later on, it is best to define it using the `euler_circle` method.

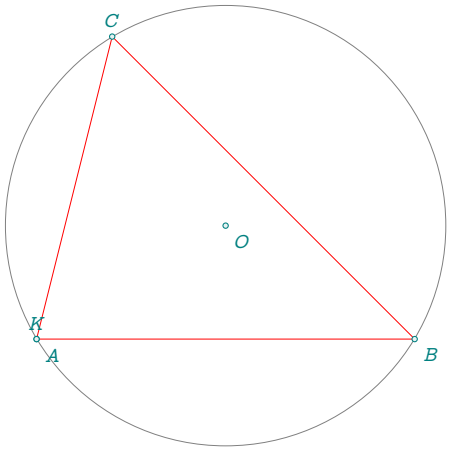
```
\directlua{%
init_elements ()
  z.A       = point: new (0,0)
  z.B       = point: new (5,0)
  z.C       = point: new (1,4)
  T         = triangle: new(z.A,z.B,z.C)
  C.euler   = T : euler_circle ()
  z.N,z.K   = get_points (C.euler)
}
\begin{tikzpicture}
 \tkzGetNodes
 \tkzDrawPolygons[red](A,B,C)
 \tkzDrawCircle(N,K)
 \tkzDrawPoints(A,B,C,N,K)
 \tkzLabelPoints(A,B,N)
 \tkzLabelPoints[above](C,K)
\end{tikzpicture}
```



12.3.13 Method circum_circle

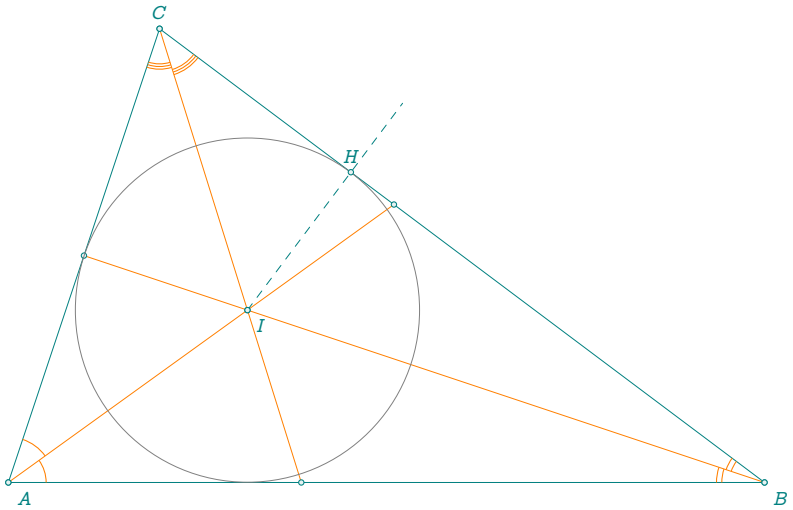
To obtain the circumscribed circle, simply use the `T.circumcenter` attribute, but if it is necessary to determine the circle then the method is `circum_circle`.

```
\directlua{%
init_elements ()
  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)
  C.circum  = T : circum_circle ()
  z.O,z.K   = get_points (C.circum)
}
\begin{tikzpicture}
 \tkzGetNodes
 \tkzDrawPolygons[red](A,B,C)
 \tkzDrawCircle(O,K)
 \tkzDrawPoints(A,B,C,O,K)
 \tkzLabelPoints(A,B,O)
 \tkzLabelPoints[above](C,K)
\end{tikzpicture}
```



12.3.14 Method in_circle

An incircle is an inscribed circle of a polygon, i.e., a circle that is tangent to each of the polygon's sides. The center I of the incircle is called the incenter, and the radius r of the circle is called the inradius.
The incenter is the point of concurrence of the triangle's angle bisectors. In addition, the points $M_A, M_B, and M_C$ of intersection of the incircle with the sides of ABC are the polygon vertices of the pedal triangle [12.3.18] taking the incenter as the pedal point (c.f. tangential triangle [12.3.23]). This triangle is called the contact triangle.
[Weisstein, Eric W. "Incircle." From MathWorld–A Wolfram Web Resource.]



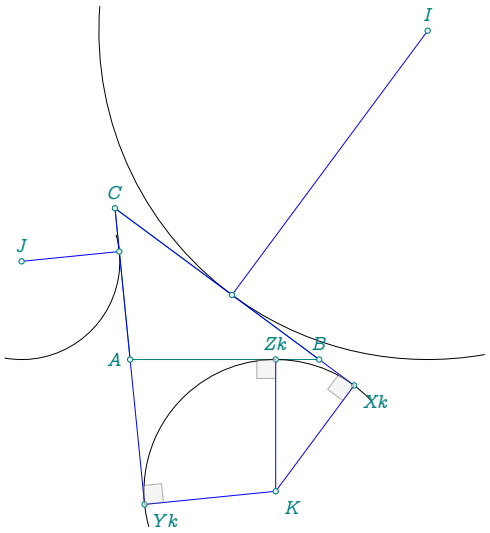
```
\directlua{%
init_elements ()
  z.A    = point : new (0 , 0)
  z.B    = point : new (5 , 0)
  z.C    = point : new (1 , 3)
  T.ABC  = triangle : new(z.A,z.B,z.C)
  z.E    = T.ABC : bisector ().pb
  z.F    = T.ABC : bisector (1).pb
  z.G    = T.ABC : bisector (2).pb
  C.IH     = T.ABC : in_circle ()
  z.I,z.H  = get_points (C.IH)
}
```

```
\begin{tikzpicture}%
  [ new/.style ={color = orange },
    one/.style = { new,/tkzmkangle/size=.5 },
    two/.style = { new,/tkzmkangle/size=.6 },
    l/.style   = { /tkzmkangle/arc=l },
    ll/.style  = { /tkzmkangle/arc=ll },
    lll/.style = { /tkzmkangle/arc=lll }]
\tkzGetNodes
\tkzDrawPolygon(A,B,C)
\tkzDrawSegments[new](A,E B,F C,G)
\tkzDrawSegments[dashed,add=0 and .5](I,H)
\tkzDrawPoints(A,B,C,E,F,G,I)
\tkzDrawCircle(I,H)
\tkzDrawPoints(I,A,B,C,H)
\begin{scope}[one]
  \tkzMarkAngles[l](B,A,E)
  \tkzMarkAngles[ll](C,B,F)
  \tkzMarkAngles[lll](A,C,G)
\end{scope}
\begin{scope}[two]
  \tkzMarkAngles[l](E,A,C)
  \tkzMarkAngles[ll](F,B,A)
  \tkzMarkAngles[lll](G,C,B)
\end{scope}
\tkzLabelPoints(A,B,I)
\tkzLabelPoints[above](C,H)
\end{tikzpicture}
```

12.3.15 Method ex_circle

Given a triangle, extend two sides in the direction opposite their common vertex. The circle tangent to these two lines and to the other side of the triangle is called an excircle, or sometimes an escribed circle. The center of the excircle is called the excenter and lies on the external angle bisector of the opposite angle.

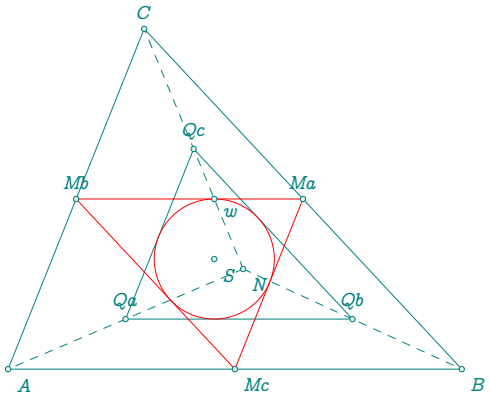
```
\directlua{%
init_elements ()
z.A    = point: new (0 , 0)
z.B    = point: new (5 , 0)
z.C    = point: new (-.4 , 4)
T.ABC  = triangle: new (z.A,z.B,z.C)
z.I,_  = get_points(T.ABC: ex_circle ())
z.J,_  = get_points(T.ABC: ex_circle (1))
z.K,_  = get_points(T.ABC: ex_circle (2))
z.Xk ,
z.Yk,
z.Zk    = T.ABC : projection (z.K)
z.Xi ,
z.Yi,
z.Zi    = T.ABC : projection (z.I)
z.Xj ,
z.Yj,
z.Zj    = T.ABC : projection (z.J)
}
```



```
\begin{tikzpicture}
\tkzGetNodes
\tkzDrawPolygon(A,B,C)
\tkzDrawArc(K,Xk)(Yk)
\tkzDrawArc(I,Yi)(Zi)
\tkzDrawArc(J,Zj)(Yj)
\tkzDrawSegments[blue](K,Xk K,Yk K,Zk C,Yk C,Xk I,Xi J,Yj)
\tkzDrawPoints(A,B,C,I,J,K,Xk,Yk,Zk,Xi,Yj)
\tkzLabelPoints(K,Xk,Yk)
\tkzLabelPoints[above](C,B,Zk,I,J)
\tkzLabelPoints[left](A)
\tkzMarkRightAngles[fill=gray!20,opacity=.4](A,Zk,K A,Yk,K K,Xk,B)
\end{tikzpicture}
```

12.3.16 Method spieker_circle

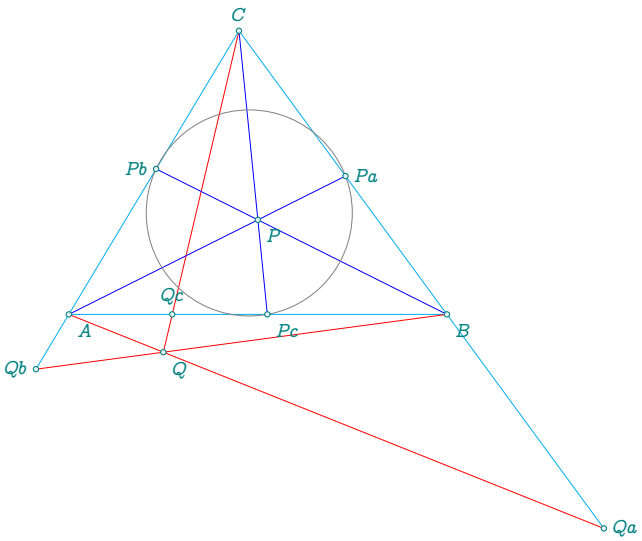
In geometry, the incircle of the medial triangle of a triangle is the Spieker circle. Its center is the Spieker center.




```
\directlua{%
init_elements ()
   z.A      = point:  new (1,1)
   z.B      = point:  new (5,1)
   z.C      = point:  new (2.2,4)
   T        = triangle: new (z.A,z.B,z.C)
   C.first_lemoine = T:spieker_circle()
   z.S,z.w  = get_points( C.first_lemoine )
   z.Ma,z.Mb,z.Mc = get_points(T : medial ())
   z.N           = T : nagel_point ()
   z.Qa = midpoint(z.A,z.N)
   z.Qb = midpoint(z.B,z.N)
   z.Qc = midpoint(z.C,z.N)
}
\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawPolygons(A,B,C Qa,Qb,Qc)
   \tkzDrawPolygons[red](Ma,Mb,Mc)
   \tkzDrawCircles[red](S,w)
   \tkzDrawSegments[dashed](N,A N,B N,C)
   \tkzDrawPoints(A,B,C,S,w,Ma,Mb,Mc,Qa,Qb,Qc,N)
   \tkzLabelPoints(A,B,S,w,Mc,N)
   \tkzLabelPoints[above](C,Ma,Mb,Qa,Qb,Qc)
\end{tikzpicture}
```

12.3.17 Methods cevian and cevian_circle

A Cevian is a line segment which joins a vertex of a triangle with a point on the opposite side (or its extension). The condition for three general Cevians from the three vertices of a triangle to concur is known as Ceva's theorem. Picking a Cevian point P in the interior of a triangle ABC and drawing Cevians from each vertex through P to the opposite side produces a set of three intersecting Cevians APa, BPb, and CPc with respect to that point. The triangle $PaPbPc$ is known as the Cevian triangle of ABC with respect to P, and the circumcircle of $PaPbPc$ is similarly known as the Cevian circle. [Weisstein, Eric W. "Cevian Triangle." From MathWorld–A Wolfram Web Resource.]

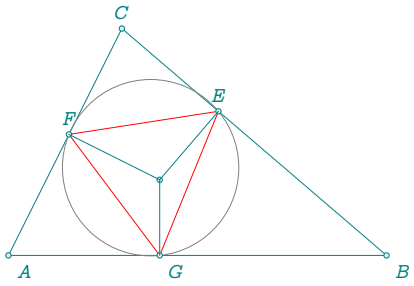


```
\directlua{%
init_elements ()
 z.A            = point: new (0,0)
 z.B            = point: new (4,0)
 z.C            = point: new (1.8,3)
 T.ABC          = triangle: new(z.A,z.B,z.C)
 z.Q            = point : new (1,-0.4)
 z.P            = point : new (2,1)
 T.cevian       = T.ABC : cevian (z.Q)
 z.Qa,z.Qb,z.Qc = get_points (T.cevian)
 T.cevian       = T.ABC : cevian (z.P)
 z.Pa,z.Pb,z.Pc = get_points (T.cevian)
 C.cev          = T.ABC : cevian_circle (z.P)
 z.w            = C.cev.center
}
\begin{tikzpicture}
\tkzGetNodes
\tkzDrawPolygons[cyan](A,B,C)
\tkzDrawSegments[cyan](A,Qb B,Qa)
\tkzDrawSegments[red](A,Qa B,Qb C,Q)
\tkzDrawSegments[blue](A,Pa B,Pb C,Pc)
\tkzDrawCircles(w,Pa)
\tkzDrawPoints(A,B,C,Qa,Qb,Qc,P,Q,Pa,Pb,Pc)
\tkzLabelPoints(A,B,P,Q,Pc)
\tkzLabelPoints[above](C,Qc)
\tkzLabelPoints[left](Qb,Pb)
\tkzLabelPoints[right](Qa,Pa)
\end{tikzpicture}
```

12.3.18 Methods pedal and pedal_circle

Given a point P, the pedal triangle of P is the triangle whose polygon vertices are the feet of the perpendiculars from P to the side lines.

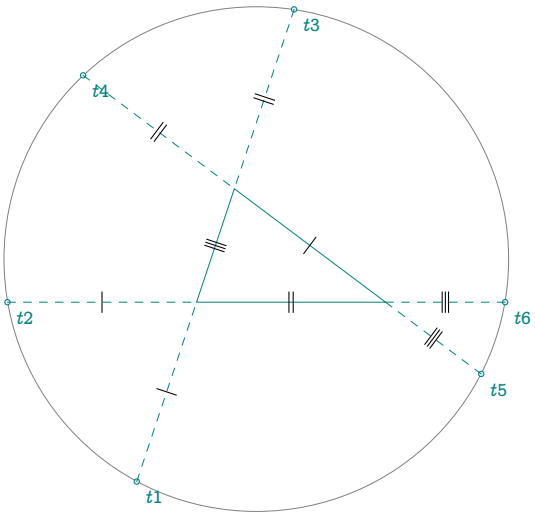
```
  \directlua{%
init_elements ()
    z.A    = point: new(0,0)
    z.B    = point: new(5,0)
    z.C    = point: new(1.5,3)
    z.O    = point: new (2,1)
    T.ABC  = triangle: new (z.A,z.B,z.C)
    T.pedal = T.ABC : pedal (z.O)
    z.E,z.F,z.G = get_points(T.pedal)
    C.pedal = T.ABC : pedal_circle (z.O)
    z.w = C.pedal.center
    z.T = C.pedal.through
  }
  \begin{tikzpicture}
  \tkzGetNodes
  \tkzDrawPolygon(A,B,C)
  \tkzDrawPolygon[red](E,F,G)
  \tkzDrawCircle(w,T)
  \tkzDrawPoints(A,B,C,E,F,G,O)
  \tkzLabelPoints(A,B,G)
  \tkzLabelPoints[above](C,E,F)
  \tkzDrawSegments(O,E O,F O,G)
  \end{tikzpicture}
```



12.3.19 Methods conway_points and conway_circle

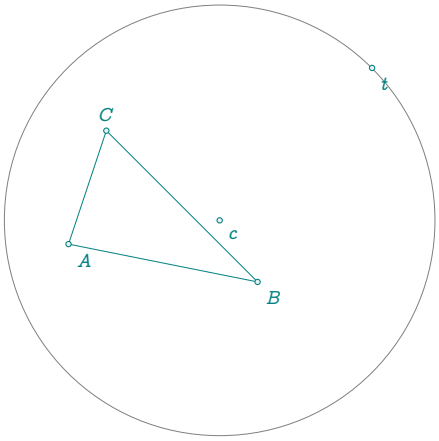
In plane geometry, Conway's circle theorem states that when the sides meeting at each vertex of a triangle are extended by the length of the opposite side, the six endpoints of the three resulting line segments lie on a circle whose centre is the centre of incidence of the triangle.

```
  \directlua{%
init_elements ()
    z.A      = point:new (0,0)
    z.C      = point:new (5,0)
    z.B      = point:new (1,3)
    T.ABC    = triangle : new (z.A,z.B,z.C)
    C.conway = T.ABC : conway_circle ()
    z.w,z.t  = get_points(C.conway)
    z.t1,z.t2,z.t3,z.t4,
    z.t5,z.t6= T.ABC : conway_points ()
  }
  \hspace*{5cm}
  \begin{tikzpicture}
     \tkzGetNodes
     \tkzDrawPolygon(A,B,C)
     \tkzDrawCircles(w,t)
     \tkzDrawPoints(t1,t2,t3,t4,t5,t6)
     \tkzLabelPoints(t1,t2,t3,t4,t5,t6)
     \tkzDrawSegments[dashed](t1,A t2,A t3,B)
     \tkzDrawSegments[dashed](t4,B t5,C t6,C)
     \tkzMarkSegments(B,C t1,A t2,A)
     \tkzMarkSegments[mark=||](A,C t3,B t4,B)
     \tkzMarkSegments[mark=|||](A,B t5,C t6,C)
  \end{tikzpicture}
```



12.3.20 Methods bevan_circle and bevan_point

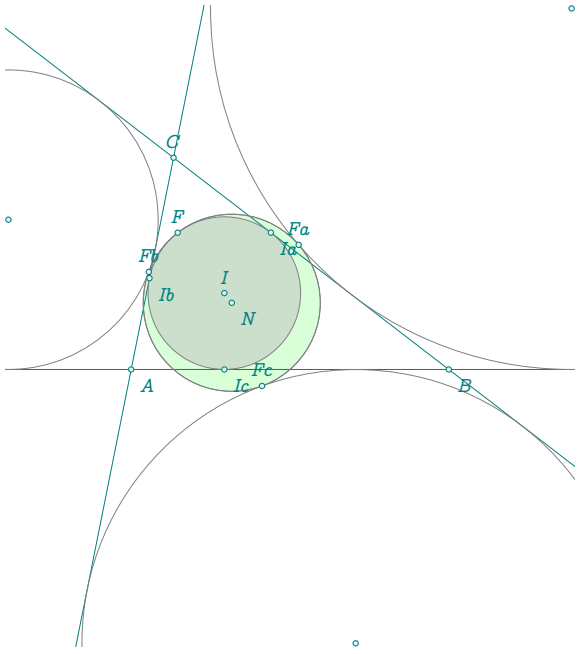
```
\directlua{%
init_elements ()
  scale = .5
    z.A           = point: new (1,1)
    z.B           = point: new (6,0)
    z.C           = point: new (2,4)
    T             = triangle: new(z.A,z.B,z.C)
    C.bevan       = T : bevan_circle ()
    z.c,z.t       = get_points (C.bevan)
   % or z.c      = T : bevan_point ()
}
\begin{tikzpicture}
    \tkzGetNodes
    \tkzDrawPolygons(A,B,C)
    \tkzDrawCircle(c,t)
    \tkzDrawPoints(A,B,C,c,t)
    \tkzLabelPoints(A,B,c,t)
    \tkzLabelPoints[above](C)
\end{tikzpicture}
```



12.3.21 Method feuerbach and method feuerbach_point

The Feuerbach triangle is the triangle formed by the three points of tangency of the nine-point circle with the excircles. (The fact that the excircles touch the nine-point circle is known as Feuerbach's theorem.) Refer to Weisstein, Eric W. "Feuerbach Triangle." From MathWorld–A Wolfram Web Resource..
The exinscribed circles of a triangle are tangent to the circle of the nine points of a triangle at points which form the Feuerbach triangle ($FaFbFc$). The inscribed circle and the circle of nine points are tangent at a point called the Feurerbach point F.

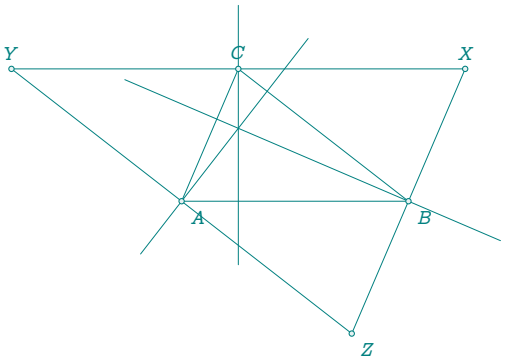
```
\directlua{%
init_elements ()
   scale             = .8
   z.A               = point: new (0,0)
   z.B               = point: new (6,0)
   z.C               = point: new (0.8,4)
   T.ABC             = triangle : new ( z.A,z.B,z.C )
   z.N               = T.ABC.eulercenter
   z.Fa,z.Fb,z.Fc    = get_points ( T.ABC : feuerbach () )
   z.F               =  T.ABC : feuerbach_point ()
   z.Ja,z.Jb,z.Jc    = get_points ( T.ABC : excentral () )
   z.I = T.ABC.incenter
   z.Ia,z.Ib,z.Ic = get_points (T.ABC :  intouch ())
}
\begin{tikzpicture}
\tkzGetNodes
\tkzDrawPoints(Ja,Jb,Jc)
\tkzClipBB
\tkzFillCircles[green!30,,opacity=.5](N,Fa)
\tkzFillCircles[lightgray,,opacity=.5](I,F)
\tkzDrawLines[add=3 and 3](A,B A,C B,C)
\tkzDrawCircles(Ja,Fa Jb,Fb Jc,Fc N,Fa N,F I,F)
\tkzDrawPoints(A,B,C,F,Fa,Fb,Fc,N,I,Ia,Ib,Ic)
\tkzLabelPoints(N,A,B,Ia,Ib,Ic)
\tkzLabelPoints[above](Fa,Fb,Fc,F,I,C)
\end{tikzpicture}
```



12.3.22 Method similar

The `similar` method creates a new triangle whose sides are parallel to the sides of the original triangle and pass through its vertices.

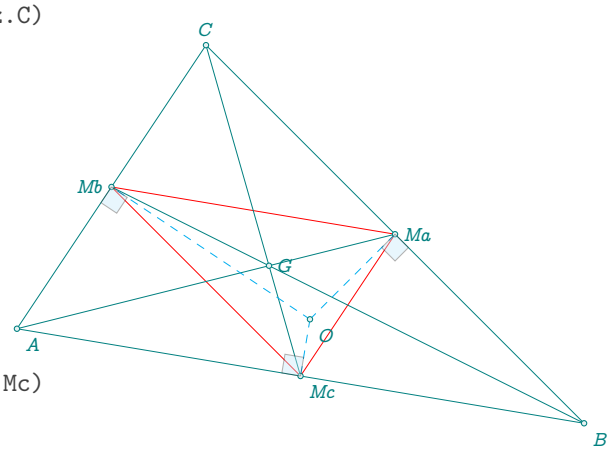
```
  \directlua{%
init_elements ()
   scale =.5
   z.A            = point: new (0 , 0)
   z.B            = point: new (6 , 0)
   z.C            = point: new (1.5 , 3.5)
   T.ABC          = triangle: new (z.A,z.B,z.C)
   z.X,z.Y,z.Z    = get_points ( T.ABC : similar ())
   z.H_a,z.H_b,
   z.H_c          = get_points (T.ABC : orthic ())
  }
  \begin{tikzpicture}
     \tkzGetNodes
     \tkzDrawPolygons(A,B,C  X,Y,Z)
     \tkzDrawLines(A,H_a B,H_b C,H_c)
     \tkzDrawPoints(A,B,C,X,Y,Z)
     \tkzLabelPoints(A,B,Z)
     \tkzLabelPoints[above](X,Y,C)
   \end{tikzpicture}
```



12.3.23 Method medial

The triangle $MaMbMc$ formed by joining the midpoints of the sides of a triangle ABC. The medial triangle is sometimes also called the auxiliary triangle (Dixon 1991). [Weisstein, Eric W. "Medial Triangle." From MathWorld–A Wolfram Web Resource.]

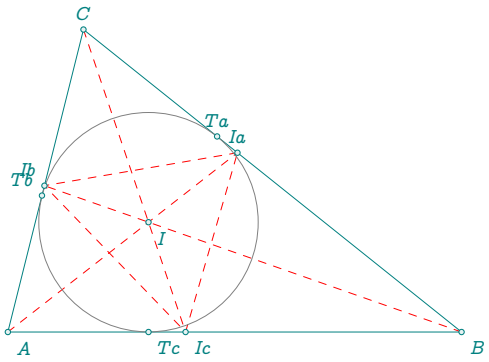
```
  \directlua{%
init_elements ()
    scale           = 1.25
      z.A           = point: new (0,1)
      z.B           = point: new (6,0)
      z.C           = point: new (2,4)
      T             = triangle: new(z.A,z.B,z.C)
      T.med         = T : medial ()
      z.Ma,z.Mb,z.Mc= get_points (T.med)
      z.G           = T.centroid
      z.O           = T.circumcenter
  }
  \begin{tikzpicture}
  \tkzGetNodes
  \tkzDrawPolygons(A,B,C)
  \tkzDrawPolygons[red](Ma,Mb,Mc)
  \tkzDrawSegments(A,Ma B,Mb C,Mc)
   \tkzDrawSegments[dashed,cyan](O,Ma O,Mb O,Mc)
  \tkzDrawPoints(A,B,C,Ma,Mb,Mc,O,G)
  \tkzLabelPoints(A,B,Mc,O)
  \tkzLabelPoints[above](C)
  \tkzLabelPoints[left](Mb)
  \tkzLabelPoints[right](Ma,G)
  \tkzMarkRightAngles[fill=cyan!20,
                opacity=.4](O,Ma,B O,Mb,A O,Mc,A)
  \end{tikzpicture}
```



12.3.24 Method incentral

The incentral triangle $IaIbIc$ is the Cevian triangle of a triangle ABC with respect to its incenter I. It is therefore also the triangle whose vertices are determined by the intersections of the reference triangle's angle bisectors with the respective opposite sides. [Weisstein, Eric W. "Incentral Triangle." From MathWorld–A Wolfram Web Resource.]

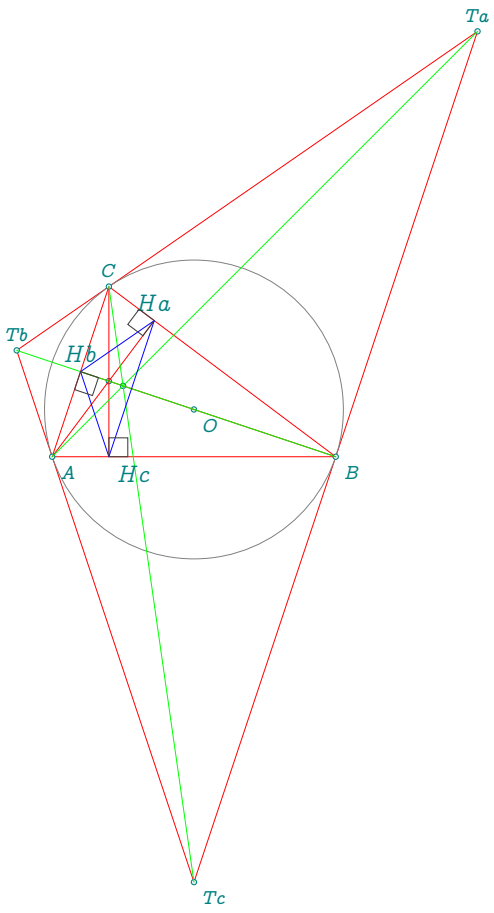
```
\directlua{%
init_elements ()
 z.A        = point: new (0 , 0)
 z.B        = point: new (6 , 0)
 z.C        = point: new (1 , 4)
 T.ABC      = triangle: new (z.A,z.B,z.C)
 z.I        = T.ABC.incenter
 z.Ia,z.Ib,
 z.Ic       = get_points (T.ABC : incentral ())
 z.Ta,z.Tb,
 z.Tc       = get_points (T.ABC :  intouch ())
}
\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawPolygon(A,B,C)
   \tkzDrawPolygon[dashed,red](Ia,Ib,Ic)
   \tkzDrawSegments[dashed,red](A,Ia B,Ib C,Ic)
   \tkzDrawCircle(I,Ta)
   \tkzDrawPoints(A,B,C,Ia,Ib,Ic,I,Ta,Tb,Tc)
   \tkzLabelPoints(A,B,Ic,I,Tc)
   \tkzLabelPoints[above](Ia,Ta,C)
   \tkzLabelPoints[above left](Ib,Tb)
\end{tikzpicture}
```



12.3.25 Method tangential

The tangential triangle is the triangle $TaTbTc$ formed by the lines tangent to the circumcircle of a given triangle DeltaABC at its vertices. It is therefore antipedal triangle of ABC with respect to the circumcenter O. It is also anticevian triangle of ABC with the symmedian point K as the anticevian point (Kimberling 1998, p. 156). Furthermore, the symmedian point K of ABC is the Gergonne point of $TaTbTc$.
The sides of an orthic triangle are parallel to the tangents to the circumcircle at the vertices (Johnson 1929, p. 172). This is equivalent to the statement that each line from a triangle's circumcenter to a vertex is always perpendicular to the corresponding side of the orthic triangle (Honsberger 1995, p. 22), and to the fact that the orthic and tangential triangles are homothetic. [Weisstein, Eric W. "Tangential Triangle." From MathWorld–A Wolfram Web Resource.]

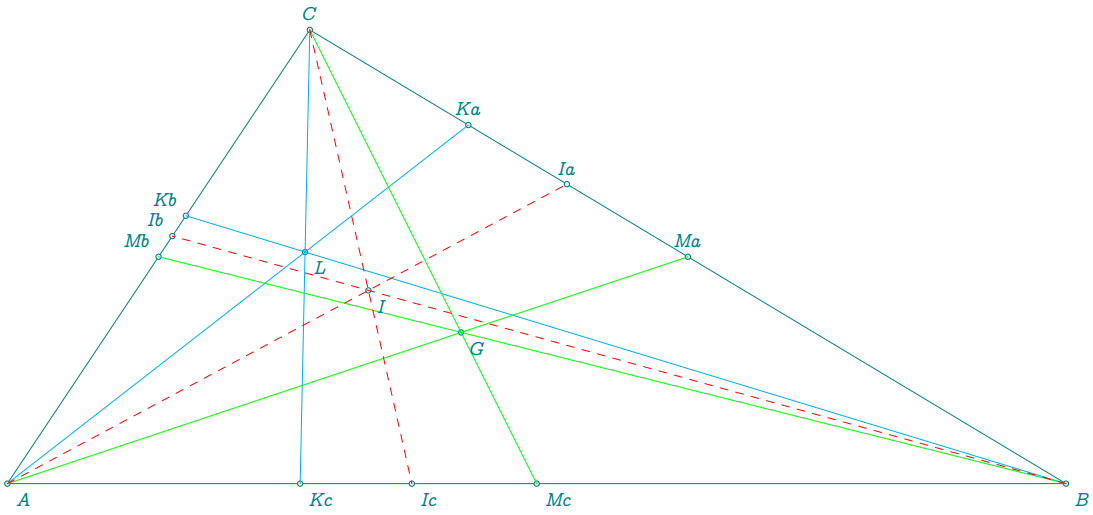
```
\directlua{%
init_elements ()
  scale    = .75
  z.A       = point: new (0,0)
  z.B       = point: new (5,0)
  z.C       = point: new (1,3)
    T       = triangle: new(z.A,z.B,z.C)
  z.H       = T.orthocenter
  z.O       = T.circumcenter
  z.L       = T : symmedian_point ()
  T.orthic  = T: orthic()
  z.Ha,
  z.Hb,
  z.Hc      = get_points (T.orthic)
  z.Ta,
  z.Tb,
  z.Tc       = get_points (T : tangential ())
}
\begin{tikzpicture}
 \tkzGetNodes
 \tkzDrawPolygons[red](A,B,C Ta,Tb,Tc)
 \tkzDrawCircle(O,A)
 \tkzDrawPoints(A,B,C,O,H,Ta,Tb,Tc,L)
 \tkzDrawSegments[red](C,Hc B,Hb A,Ha)
 \tkzDrawSegments[green](C,Tc B,Tb A,Ta)
 \tkzDrawPolygon[blue](Ha,Hb,Hc)
 \tkzLabelPoints(A,B,O,Tc)
 \tkzLabelPoints[above](C,Tb,Ta)
 \tkzLabelPoints[font=\small](Hc)
 \tkzLabelPoints[font=\small,above](Ha,Hb)
 \tkzMarkRightAngles(A,Ha,C B,Hb,A C,Hc,B)
\end{tikzpicture}
```



12.3.26 Method symmedial

The symmedial triangle $LaLbLc$ is the triangle whose vertices are the intersection points of the symmedians with the reference triangle ABC.
The symmedial circle is the circumcircle of the symmedial triangle.
The following example groups several concepts around the symmedian. As a reminder, a symmedian of a triangle is the reflection of the median with respect to the angle bisector.
The points of contact of the symmedians with the sides of the triangle are obtained using the `symmedian` method. The intersection of the symmedians is the point known as the `Lemoine` or `Symmedian` point. You can use the triangle methods `lemoine_point` or `symmedian_point`. If you only need one of the lines, you can use the method `symmedian_line(n)`. $n = 0$ corresponds to the line coming from the first vertex of the triangle, $n = 1$ to the second, and so on.
In the next example, L is the `Lemoine point` or the `Symmedian point`. $LaLbLc$ is the symmedian triangle.[Weisstein, Eric W. "Symmedian Point." From MathWorld–A Wolfram Web Resource.]

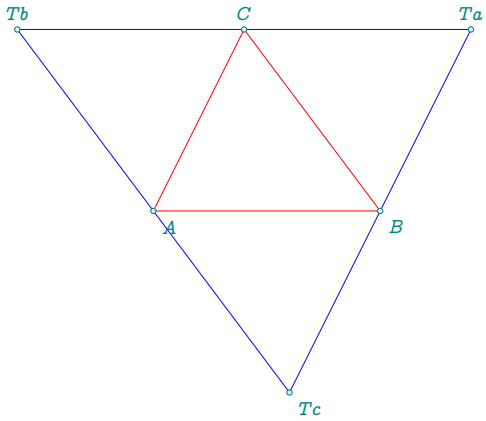
```
\directlua{%
init_elements ()
   scale          = 2
   z.A            = point :   new (0,0)
   z.B            = point :   new (7,0)
   z.C            = point :   new (2,3)
   T.ABC          = triangle : new (z.A,z.B,z.C)
   z.L            = T.ABC : lemoine_point ()
   T.SY           = T.ABC : symmedian ()
   T.med          = T.ABC : medial ()
   z.Ka,z.Kb,z.Kc = get_points (T.SY)
   z.Ma,z.Mb,z.Mc = get_points (T.med)
   L.Kb           = T.ABC : symmedian_line (1)
  _,z.Kb          = get_points(L.Kb)
  z.G             = T.ABC.centroid
  z.Ia,z.Ib,z.Ic  = get_points ( T.ABC : incentral ())
  %          z.T = T.ABC : trilinear (0,1,1)
  z.I             = T.ABC.incenter
}
\begin{tikzpicture}
\tkzGetNodes
\tkzDrawPolygons(A,B,C)
\tkzDrawPoints(A,B,C,L,Ka,Kb,Kc,G,Ma,Mb,Mc,Ia,Ib,Ic,I)
\tkzDrawSegments[cyan](A,Ka B,Kb C,Kc)
\tkzDrawSegments[green](A,Ma B,Mb C,Mc)
\tkzDrawSegments[dashed,red](A,Ia B,Ib C,Ic)
\tkzLabelPoints[above](C,Ka,Ia,Ma)
\tkzLabelPoints[above left](Kb,Ib,Mb)
\tkzLabelPoints(A,B,L,Kc,I,Ic,Mc,G)
\end{tikzpicture}
```



12.3.27 Method anti

The anticomplementary triangle is the triangle $TaTbTc$ which has a given triangle ABC as its medial triangle. It is therefore the anticevian triangle with respect to the triangle centroid G (Kimberling 1998, p. 156). [Weisstein, Eric W. "Anticomplementary Triangle." From MathWorld–A Wolfram Web Resource.]

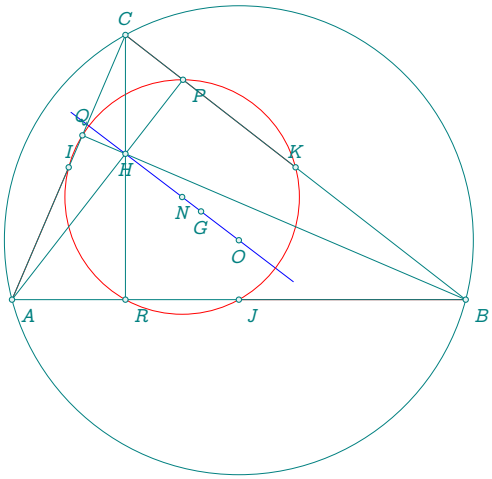
```
\directlua{%
init_elements ()
  scale     = .6
  z.A       = point: new (0,0)
  z.B       = point: new (5,0)
  z.C       = point: new (2,4)
  T         = triangle: new(z.A,z.B,z.C)
  T.similar = T: anti()
  z.Ta,
  z.Tb,
  z.Tc      = get_points (T.similar)
}
\begin{tikzpicture}
 \tkzGetNodes
 \tkzDrawPolygons[red](A,B,C)
 \tkzDrawPolygon[blue](Ta,Tb,Tc)
 \tkzDrawPoints(A,B,C,Ta,Tb,Tc)
 \tkzLabelPoints(A,B,Tc)
 \tkzLabelPoints[above](Ta,Tb,C)
\end{tikzpicture}
```



12.3.28 Euler line

The line on which the orthocenter H, triangle centroid G, circumcenter O, nine-point center N, and a number of other important triangle centers lie. [Weisstein, Eric W. "Euler Line." From MathWorld–A Wolfram Web Resource.]

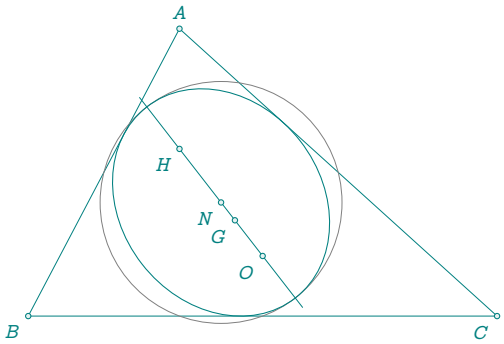
```
\directlua{%
init_elements ()
   z.A           = point: new (0 , 0)
   z.B           = point: new (6 , 0)
   z.C           = point: new (1.5 , 3.5)
   T.ABC         = triangle: new (z.A,z.B,z.C)
   z.O           = T.ABC.circumcenter
   z.G           = T.ABC.centroid
   z.N           = T.ABC.eulercenter
   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 ())
}
\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}
```



12.3.29 Euler ellipse

The Euler ellipse is a conic, tangent to the three sides of a triangle, with the orthocentre and the centre of the circumscribed circle as foci. Example of obtaining the Euler circle as well as the Euler ellipse.

```
\directlua{%
init_elements ()
z.A     = point: new (2,3.8)
z.B     = point: new (0 ,0)
z.C     = point: new (6.2 ,0)
L.AB    = line : new ( z.A , z.B )
T.ABC   = triangle: new (z.A,z.B,z.C)
z.K     = midpoint (z.B,z.C)
E.euler = T.ABC : euler_ellipse ()
z.N     = T.ABC.eulercenter
C.euler = circle : new (z.N,z.K)
ang     = math.deg(E.euler.slope)
z.O     = T.ABC.circumcenter
z.G     = T.ABC.centroid
z.H     = T.ABC.orthocenter
}
```

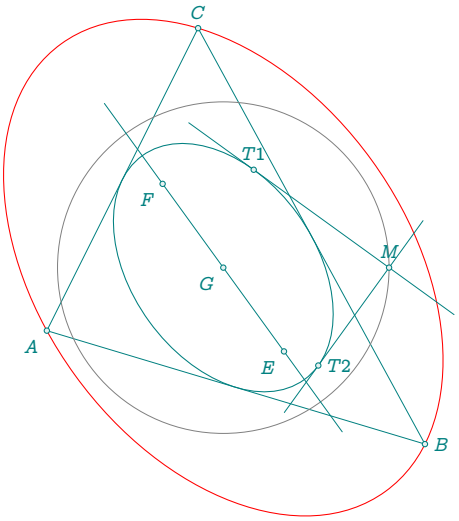



```
\begin{tikzpicture}
\tkzGetNodes
\tkzDrawPolygon(A,B,C)
\tkzDrawCircle(N,K)
\tkzDrawEllipse[teal](N,\tkzUseLua{E.euler.Rx},
      \tkzUseLua{E.euler.Ry},\tkzUseLua{ang})
\tkzDrawLine(O,H)
\tkzDrawPoints(A,B,C,N,O,H,G)
\tkzLabelPoints[below left](B,C,N,O,H,G)
\tkzLabelPoints[above](A)
\end{tikzpicture}
```

12.3.30 Steiner inellipse and circumellipse

In this example, the inner and outer Steiner ellipses, referred to as the "inellipse" and "circumellipse" [Weisstein, Eric W. "Steiner Inellipse." From MathWorld–A Wolfram Web Resource. and Weisstein, Eric W. "Steiner Circumellipse." From MathWorld–A Wolfram Web Resource.], respectively, along with the orthoptic circle, are depicted.. The triangle must be acutangle.

```
\directlua{%
init_elements ()
  scale    = .5
  z.A      = point: new (1 , 4)
  z.B      = point: new (11 , 1)
  z.C      = point: new (5 , 12)
  T.ABC    = triangle: new(z.A,z.B,z.C)
  E        = T.ABC: steiner_inellipse ()
  z.G      = E.center
  ang      = math.deg(E.slope)
  z.F      = E.Fa
  z.E      = E.Fb
  C        = E: orthoptic_circle ()
  z.w      = C.center
  z.o      = C.through
  EE       = T.ABC : steiner_circumellipse ()
  z.M      = C : point (0)
  L.T1,L.T2= E : tangent_from (z.M)
  z.T1     = L.T1.pb
  z.T2     = L.T2.pb
}
```



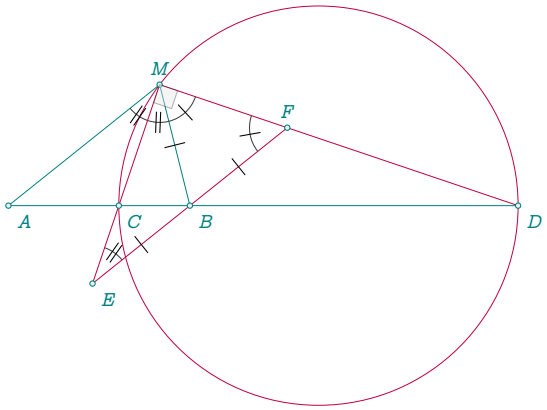

```
\begin{tikzpicture}
\tkzGetNodes
\tkzDrawPolygon(A,B,C)
\tkzDrawCircles(w,o)
\tkzDrawEllipse[teal](G,\tkzUseLua{E.Rx},
  \tkzUseLua{E.Ry},\tkzUseLua{ang})
\tkzDrawEllipse[red](G,\tkzUseLua{EE.Rx},
  \tkzUseLua{EE.Ry},\tkzUseLua{ang})
\tkzDrawLines(F,E M,T1 M,T2) %
\tkzDrawPoints(A,B,C,F,E,G,M,T1,T2)
\tkzLabelPoints[above](C,M,T1)
\tkzLabelPoints[right](T2,B)
\tkzLabelPoints[below left](A,F,E,G)
\end{tikzpicture}
```

12.3.31 Harmonic division and bisector

Let four points A, C, B and D, in this order, lying on the straight line (d) and M un point pris hors de (d). Then, if two of the following three propositions are true, then the third is also true:

1. The division (A,B;C,D) is harmonic. ($CA/CB = DA/DB$)
2. (MC) is the internal angle bisector of $\widehat{AMB}$.
3. $(MD) \perp (MC)$.

```
\directlua{%
init_elements ()
   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)
}
```




```
\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}
```

13 Class ellipse

13.1 Attributes of an ellipse

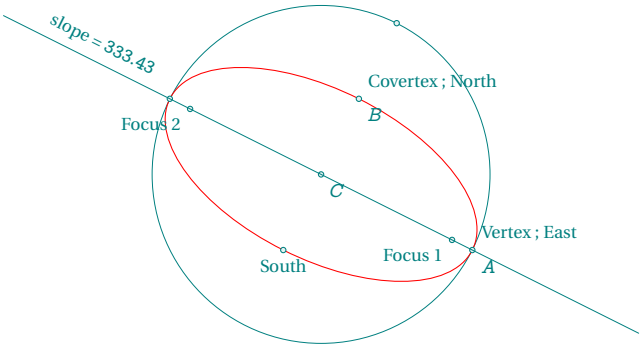
The first attributes are the three points that define the ellipse: : the `center` , the `vertex` and the`covertex`. The first method to define an ellipse is to give its center, then the point named **`vertex`** which defines the major axis and finally the point named **`covertex`** which defines the 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 13.1.1
`north`	
`west`	
`east`	

13.1.1 Atributes of an ellipse: example

```
\directlua{%
init_elements ()
   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
}
```




```
\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 = .25 and .25](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}
```

13.2 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 macro: **\tkzUseLua** defined in tkz-elements. Refer to 7.1.2 or 21.6 or next examples.

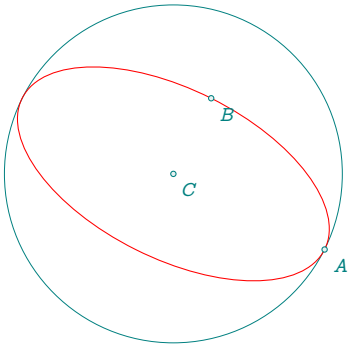
Table 12: Ellipse methods.

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)	[ex. 10.2.7]
tangent_from (pt)	[ex. 10.2.7]
point (t)	vertex = point (0) covertex = point (0.25) [ex. 10.2.7]
orthoptic_circle ()	[ex. 12.3.30]

13.2.1 Method new

The main method for creating a new ellipse is new. The arguments are three: center, vertex and covertex For attributes [13].

```
\directlua{%
init_elements ()
   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)
}
\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}
```

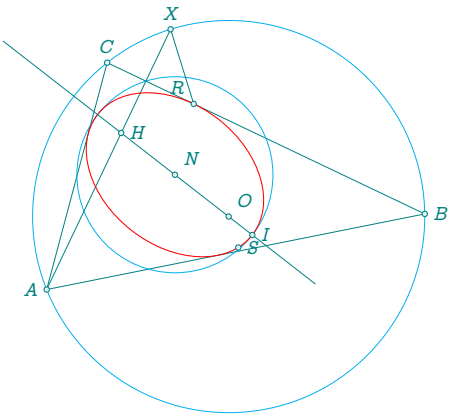


The macro **\tkzUseLua (variable)** is used to transfer values to Ti*k*Z or tkz-euclide.

13.2.2 Method foci

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

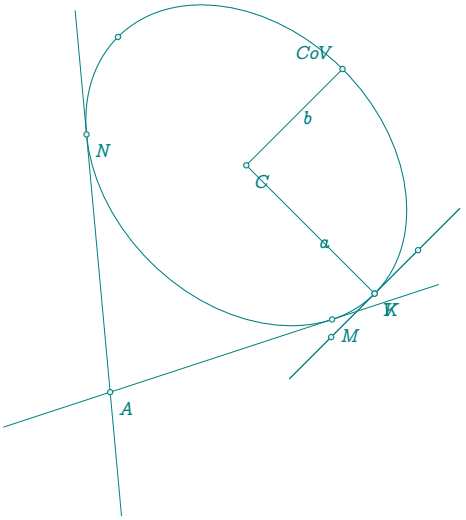
```
\directlua{%
init_elements ()
   z.A      = point: new (0 , 0)
   z.B      = point: new (5 , 1)
   L.AB     = line : new (z.A,z.B)
   z.C      = point: new (.8 , 3)
   T.ABC    = triangle: new (z.A,z.B,z.C)
   z.N      = T.ABC.eulercenter
   z.H      = T.ABC.orthocenter
   z.O      = T.ABC.circumcenter
   _,_,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)
}

\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}
```



13.2.3 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)
With `lua`*, the radian is used as unit for angles.*

```
\directlua{%
init_elements ()
   scale       = .6
   z.C         = point: new (2 , 3)
   z.A         = point: new (-1 , -2)
   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)
}
```




```
\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}
```

14 Class Quadrilateral

14.1 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.

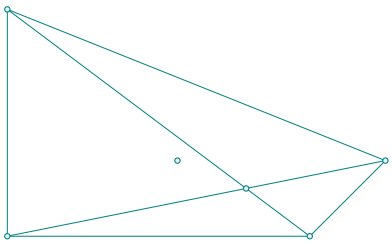
Creation `Q.new = rectangle : new (z.A,z.B,z.C,z.D)`

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.

14.1.1 Quadrilateral attributes

```
\directlua{%
init_elements ()
z.A      = point : new ( 0 , 0 )
z.B      = point : new ( 4 , 0 )
z.C      = point : new ( 5 , 1 )
z.D      = point : new ( 0 , 3 )
Q.ABCD   = quadrilateral : new ( z.A , z.B , z.C , z.D )
z.I      = Q.ABCD.i
z.G      = Q.ABCD.g
}

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



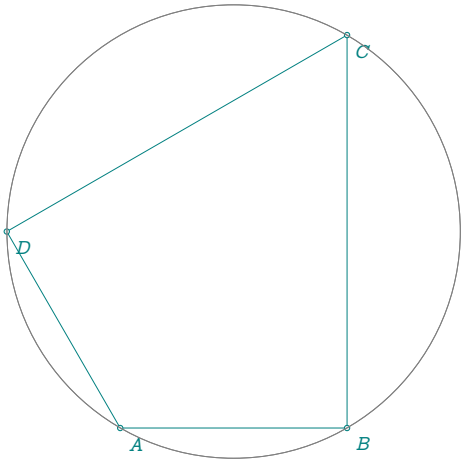
14.2 Quadrilateral methods

Table 14: Quadrilateral methods.

Methods	Comments
iscyclic ()	inscribed ? (Refer to next example)

14.2.1 Inscribed quadrilateral

```
\directlua{%
init_elements ()
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
}

\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}
```



15 Class square

15.1 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."

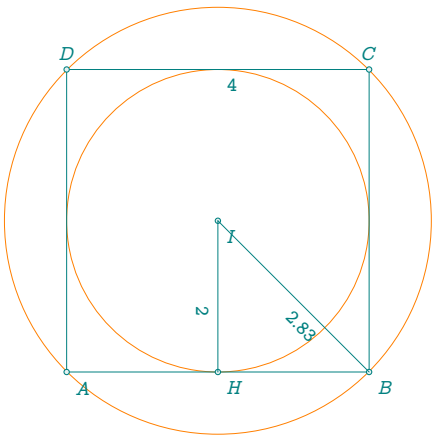
Creation `S.AB = square : new (z.A,z.B,z.C,z.D)`

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.

15.1.1 Example: square attributes

```
\directlua{%
init_elements ()
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
}
\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}
```



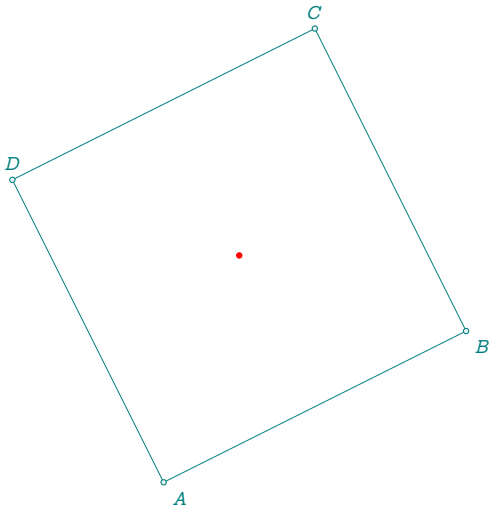
15.2 Square methods

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)

15.2.1 Square with side method

```
\directlua{%
init_elements ()
   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
}

\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 Class rectangle

16.1 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.

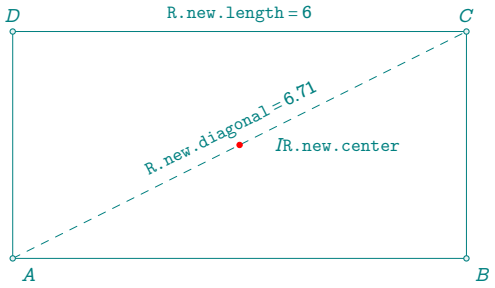
Creation `R.ABCD = rectangle : new (z.A,z.B,z.C,z.D)`

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.

16.1.1 Example

```
\directlua{%
init_elements ()
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
}

\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}
```



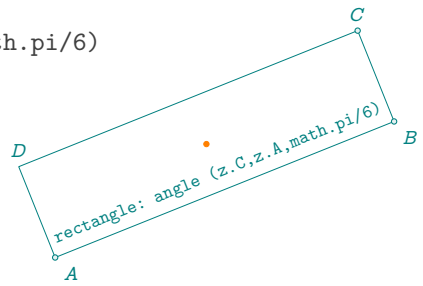
16.2 Rectangle methods

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

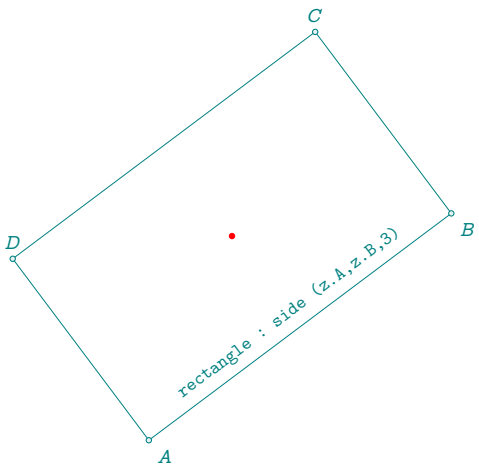
16.2.1 Angle method

```
\directlua{%
init_elements ()
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
}

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



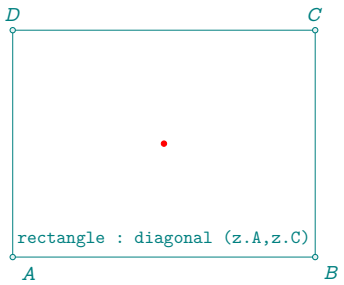
16.2.2 Side method

```
\directlua{%
init_elements ()
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
}
\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}
```



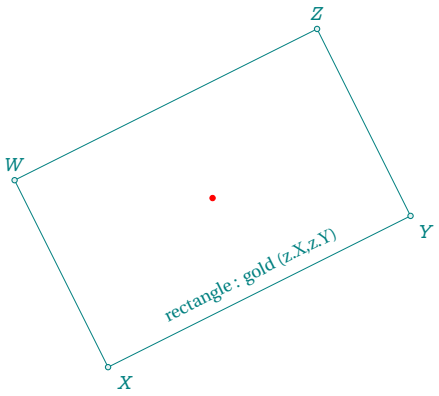
16.2.3 Diagonal method

```
\directlua{%
init_elements ()
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
}

\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}
```



16.2.4 Gold method

```
\directlua{%
init_elements ()
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
}

\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}
```



17 Class parallelogram

17.1 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.

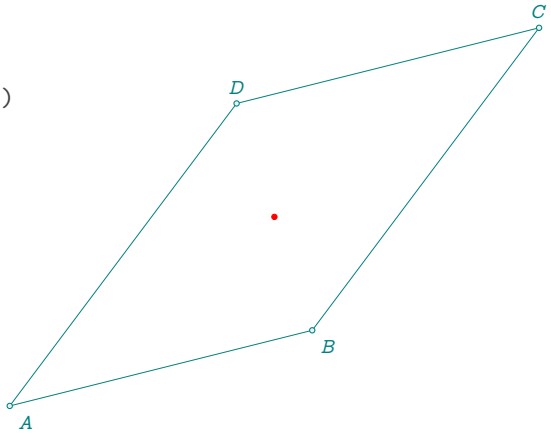
Creation `P.new = parallelogram : new (z.A,z.B,z.C,z.D)`

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.

17.1.1 Example: attributes

```
\directlua{%
init_elements ()
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
}
\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}
```



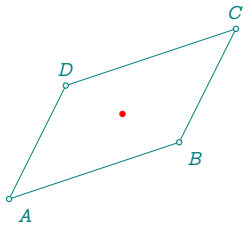
17.2 Parallelogram methods

Table 20: Parallelogram methods.

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

17.2.1 parallelogram with fourth method

```
\directlua{%
init_elements ()
   scale = .75
z.A      = point : new ( 0 , 0 )
z.B      = point : new ( 3 , 1 )
z.C      = point : new ( 4 , 3 )
P.four   = parallelogram : fourth (z.A,z.B,z.C)
z.D      = P.four.pd
z.I      = P.four.center
}
\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}
```



18 Class regular polygon

18.1 regular_polygon attributes

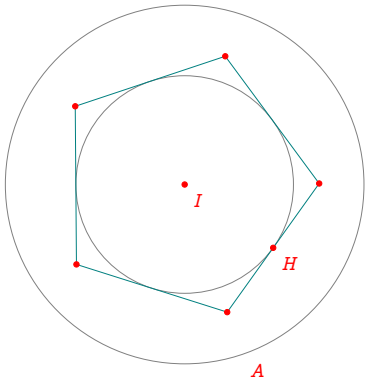
Creation `RP.IA = regular_polygon : new (z.I,z.A,6)`

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

18.1.1 Pentagon

```
\directlua{%
init_elements ()
scale = .75
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
}
\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}
```



18.2 regular_polygon methods

Table 22: regular_polygon 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)	[18.1.1]

19 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.head % we recover the final point (head)
```

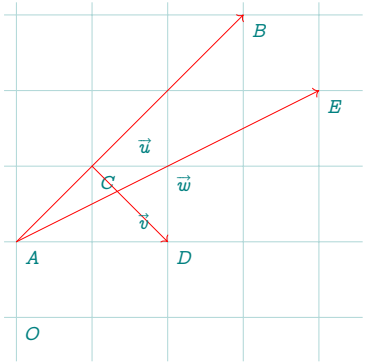
19.1 Attributes of a vector

Table 23: Vector attributes.

Attributes	Application	Example
`tail`	`V.AB.t = z.A`	[8.2.2]
`head`	`V.AB.head = z.B`	[8.2.2]
`type`	`V.AB.type = 'vector'`	
`slope`	`V.AB.slope`	[19.1.1]
`length`	`V.AB.norm`	[19.1.1]
`mtx`	`V.AB.mtx`	The result is a column matrix `{{V.AB.t},{V.AB.h}}`

19.1.1 Example vector attributes

```
\directlua{%
init_elements ()
  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.head
}
\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°

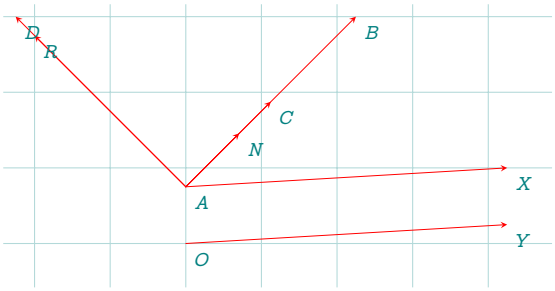
19.2 Methods of the class vector

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)`	$\overrightarrow{DB} = \overrightarrow{AC}$

19.2.1 Example of methods

```
\directlua{%
init_elements ()
scale = .75
  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.head
  V.AD  = V.AB : orthogonal ()
  z.D   = V.AD.head
  V.AN  = V.AB : normalize ()
  z.N   = V.AN.head
  V.AR  = V.AB : orthogonal(2*math.sqrt(2))
  z.R   = V.AR.head
  V.AX  = 2*V.AC - V.AR
  z.X   = V.AX.head
  V.OY  = V.AX : at (z.O)
  z.Y   = V.OY.head
}
\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}
```



20 Class matrix

The matrix class is currently experimental, and its attribute and method names have not yet been finalized, indicating that this class is still evolving. Certain connections have been made with other classes, such as the point class. Additionally, a new attribute, mtx, has been included, associating a column matrix with the point, where the elements correspond to the point's coordinates in the original base. Similarly, an attribute has been added to the vector class, where mtx represents a column matrix consisting of the two affixes that compose the vector.
This matrix class has been created to avoid the need for an external library, and has been adapted to plane transformations. It allows you to use complex numbers.
☞ To display matrices, you'll need to load the amsmath package.
☞ While some methods are valid for any matrix size, the majority are reserved for square matrices of order 2 and 3.

20.1 Matrix creation

– The first method is: [20.5.1]

```
M = matrix: new ({{a,b},{c,d}})
or M = matrix: new {{a,b},{c,d}}
a, b, c, et d being real or complex numbers.
```

$$M = \begin{bmatrix} 2.40 & 1.80 \\ 4 & 5.17 \end{bmatrix}$$

– It is also possible to obtain a square matrix with: [20.5.7]

```
M = matrix : square (2,a,b,c,d)
```

– In the case of a column vector: [20.5.2]

```
V = matrix : vector (1,2,3)
```

$$V = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

– Homogeneous transformation matrix [20.5.4]

The objective is to generate a matrix with homogeneous coordinates capable of transforming a coordinate system through rotation, translation, and scaling. To achieve this, it is necessary to define both the rotation angle, the coordinates of the new origin ans the scaling factors.

```
H = matrix : htm (math.pi/3,1,2,2,1)
```

$$H = \begin{bmatrix} 1 & -0.87 & 1 \\ 0.87 & 0.50 & 2 \\ 0 & 0 & 1 \end{bmatrix}$$

20.2 Display a matrix: method print

This method (Refer to 20.5.8) is necessary to control the results, so here are a few explanations on how to use it. It can be used on real or complex matrices, square or not. A few options allow you to format the results. You need to load the amsmath package to use the "print" method. Without this package, it is possible to display the contents of the matrix without formatting with print_array (M)

```
  \directlua{%
init_elements ()
  M = matrix : new {{1,-1},{2,0}}
  M : print ()
  }
```

$$\begin{bmatrix} 1 & -1 \\ 2 & 0 \end{bmatrix}$$

20.3 Attibutes of a matrix

Table 25: Matrix attributes.

Attributes	Application	
set	M.set = {{a,b},{c,d}}	table of tables
rows	M.rows	number of rows
cols	M.cols	number of columns
type	M.type = "matrix"	the type of object
det	M.det	determinant of a square matrix or nil

20.3.1 Attribute set

A simple array such as {{1,2},{2,-1}} is often considered a "matrix". In "tkz-elements", we'll consider M defined by matrix : new ({{1,1},{0,2}}) as a matrix and M.set as an array (M.set = {{1,1},{0,2}}).
You can access a particular element of the matrix, for example: M.set[2][1] gives 0.
\tkzUseLua{M.set[2][1]} is the expression that displays 2.
The number of rows is accessed with M.rows and the number of columns with M.cols, here's an example:

```
\directlua{%
init_elements ()
M = matrix : new ({{1,2,3},{4,5,6}})
M : print ()
tex.print("Rows:  "..M.rows)
tex.print("Cols:  "..M.cols)
}
```

$$\begin{bmatrix}1 & 2 & 3\\ 4 & 5 & 6\end{bmatrix}$$ Rows: 2 Cols: 3

20.3.2 Determinant with real numbers

The matrix must be square. This library was created for matrices of dimension 2 or 3, but it is possible to work with larger sizes. det is an attribute of the "matrix" object, but the determinant can also be obtained with the function determinant(M).

```
  \directlua{%
init_elements ()
  M = matrix : square (3,1,1,0,2,-1,-2,1,-1,2)
  M : print ()
  tex.print ('\\\\')
  tex.print ("Its determinant is:  " .. M.det)
  }
```

$$\begin{bmatrix}1 & 1 & 0\\ 2 & -1 & -2\\ 1 & -1 & 2\end{bmatrix}$$
Its determinant is: -10.0

20.3.3 Determinant with complex numbers

```
\directlua{%
init_elements ()
 a = point :new (1,-2)
 b = point :new (0,1)
 c = point :new (1,1)
 d = point :new (1,-1)
 A = matrix : new ({{a, b}, {c,d}})
 tex.print(tostring(A.det))
}
```

-4.00i

20.4 Metamethods for the matrices

Conditions on matrices must be valid for certain operations to be possible.

Table 26: Matrix metamethods.

Metamethods	Application	
__add(M1,M2)	M1 + M2	
__sub(M1,M2)	M1 - M2	
__unm(M	- M	
__mul(M1,M2)	M1 * M2	
__pow(M,n)	M ^ n	n integer > or < 0 or 'T'
__tostroing(M,n)	tex.print(tostring(M))	displays the matrix
__eq(M1,M2)	true or false	

20.4.1 Addition and subtraction of matrices

To simplify the entries, I've used a few functions to simplify the displays.

```
\directlua{%
init_elements ()
  A = matrix : new ({{1,2},{2,-1}})
  B = matrix : new ({{-1,0},{1,3}})
  S = A + B
  D = A - B
  dsp(A,'A')
  nl() nl()
  dsp(B,'B')
  nl() nl()
  dsp(S,'S') sym(" = ") dsp(A) sym(' + ') dsp(B)
  nl() nl()
  dsp(D,'D') sym(" = ") dsp(A) sym(' - ') dsp(B)
}
```

$$A = \begin{bmatrix} 1 & 2 \\ 2 & -1 \end{bmatrix}$$

$$B = \begin{bmatrix} -1 & 0 \\ 1 & 3 \end{bmatrix}$$

$$S = \begin{bmatrix} 0 & 2 \\ 3 & 2 \end{bmatrix} = \begin{bmatrix} 1 & 2 \\ 2 & -1 \end{bmatrix} + \begin{bmatrix} -1 & 0 \\ 1 & 3 \end{bmatrix}$$

$$D = \begin{bmatrix} 2 & 2 \\ 1 & -4 \end{bmatrix} = \begin{bmatrix} 1 & 2 \\ 2 & -1 \end{bmatrix} - \begin{bmatrix} -1 & 0 \\ 1 & 3 \end{bmatrix}$$

20.4.2 Multiplication and power of matrices

To simplify the entries, I've used a few functions. You can find their definitions in the sources section of this documentation.

```
  \directlua{%
init_elements ()
    A = matrix : new ({{1,2},{2,-1}})
    B = matrix : new ({{-1,0},{1,3}})
    P = A * B
    I = A^-1
    C = A^3
    K = 2 * A
    T = A^'T'
  }
```

$$P = \begin{bmatrix} 1 & 6 \\ -3 & -3 \end{bmatrix} = \begin{bmatrix} 1 & 2 \\ 2 & -1 \end{bmatrix} * \begin{bmatrix} -1 & 0 \\ 1 & 3 \end{bmatrix}$$

$$A^{-1} = \begin{bmatrix} 0.20 & 0.40 \\ 0.40 & -0.20 \end{bmatrix}$$

$$K = \begin{bmatrix} 2 & 4 \\ 4 & -2 \end{bmatrix}$$

$$A^T = \begin{bmatrix} 1 & 2 \\ 2 & -1 \end{bmatrix}$$

20.4.3 Metamethod eq

Test whether two matrices are equal or identical.

20.5 Methods of the class matrix

Table 27: Matrix methods.

Functions	Comments	
new(...)	M = matrix : new ({{1,2},{2,-1}})	
square()	M = matrix : square (2,1,2,2,-1)	
vector()	M = matrix : vector (2,1)	
htm()	M = matrix : htm (2,1,2,2,-1)	
Methods	**Comments**	
print(s,n)	M : print ()	s='matrix' or ... default 'bmatrix'
htm_apply(...)	M : htm_apply (...)	
get()	M : get (i,j)	i = raws , j = cols Refer to 20.5.10
inverse()	M : inverse ()	
adjugate()	M : adjugate ()	
transpose()	M : transpose ()	
is_diagonal()	true or false	result :boolean
is_orthogonal()	true or false	
homogenization()	M : homogenization ()	

20.5.1 Function new

This is the main method for creating a matrix. Here's an example of a 2x3 matrix with complex coefficients:

```
    \directlua{%
init_elements ()
    a = point : new (1,0)
    b = point : new (1,1)
    c = point : new (-1,1)
    d = point : new (0,1)
    e = point : new (1,-1)
    f = point : new (0,-1)
    M = matrix : new ({{a,b,c},{d,e,f}})
    M : print ()
    }
```

$$\begin{bmatrix} 1 & 1+i & -1+i \\ i & 1-i & i \end{bmatrix}$$

20.5.2 Function vector

The special case of a column matrix, frequently used to represent a vector, can be treated as follows:

```
    \directlua{%
init_elements ()
    M = matrix : vector (1,2,3)
    M : print ()
    }
```

$$\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

20.5.3 Method homogenization

homogenization of vector: the aim is to be able to use a homogeneous transformation matrix
Let's take a point A such that `z.A = point : new (2,-1)`. In order to apply a `htm` matrix, we need to perform a few operations on this point. The first is to determine the vector (matrix) associated with the point. This is straightforward, since there's a point attribute called `mtx` which gives this vector:

```
z.A = point : new (2,-1)
V = z.A.mtx : homogenization ()
```

which gives:

```
  \directlua{%
init_elements ()
    pi  = math.pi
    M   = matrix : htm (pi/4 , 3 , 1)
    z.A = point : new (2,-1)
    V   = z.A.mtx : homogenization ()
    z.A.mtx : print ()
    tex.print ('then after homogenization: ')
    V : print ()
  }
```

$\begin{bmatrix} 2 \\ -1 \end{bmatrix}$ then after homogenization: $\begin{bmatrix} 2 \\ -1 \\ 1 \end{bmatrix}$

20.5.4 Function htm: homogeneous transformation matrix

There are several ways of using this transformation. First, we need to create a matrix that can associate a rotation with a translation.
The main method is to create the matrix:

```
pi = math.pi
M = matrix : htm (pi/4 , 3 , 1)
```

A 3x3 matrix is created which combines a $\pi/4$ rotation and a $\vec{t} = (3,1)$ translation.
$\begin{bmatrix} 0.71 & -0.71 & 3 \\ 0.71 & 0.71 & 1 \\ 0 & 0 & 1 \end{bmatrix}$
Now we can apply the matrix M. Let A be the point defined here: 20.5.3. By homogenization, we obtain the column matrix V.

```
W = A * V
```

$$\begin{bmatrix} 0.71 & -0.71 & 3 \\ 0.71 & 0.71 & 1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ -1 \\ 1 \end{bmatrix} = \begin{bmatrix} 5.12 \\ 1.71 \\ 1 \end{bmatrix}$$

All that remains is to extract the coordinates of the new point.

20.5.5 Method get_htm_point

In the previous section, we obtained the W matrix. Now we need to obtain the point it defines.
The method `get_htm_point` extracts a point from a vector obtained after applying a `htm` matrix.

```
\directlua{%
init_elements ()
  W : print ()
  z.P = get_htm_point(W)
  tex.print("The affix of $P$ is: ")
  tex.print(display(z.P))
}
```

$\begin{bmatrix} 5.12 \\ 1.71 \\ 1 \end{bmatrix}$ The affix of P is: 5.12+1.71i

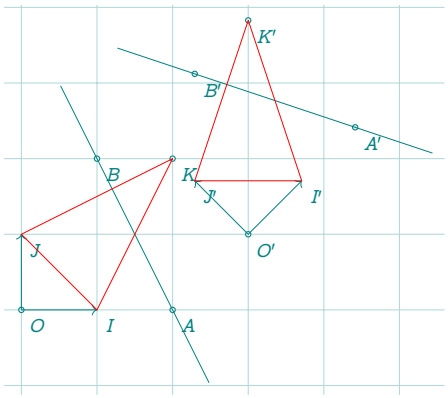
20.5.6 Method htm_apply

The above operations can be simplified by using the `htm_apply` method directly at point A.

```
z.Ap = M: htm_apply (z.A)
```

Then the method `htm_apply` transforms a point, a list of points or an object.

```
\directlua{%
init_elements ()
  pi       = math.pi
  M        = matrix : htm (pi/4 , 3 , 1 )
  z.O      = point : new (0,0)
  V.ori    = z.O.mtx : homogenization ()
  z.I      =  point : new (1,0)
  z.J      =  point : new (0,1)
  z.A      = point: new (2,0)
  z.B      = point: new (1,2)
  L.AB     = line : new (z.A,z.B)
  z.Op,z.Ip,z.Jp = M : htm_apply (z.O,z.I,z.J)
  L.ApBp   = M : htm_apply (L.AB)
  z.Ap     = L.ApBp.pa
  z.Bp     = L.ApBp.pb
  z.K      = point : new (2,2)
  T       = triangle : new ( z.I , z.J , z.K )
  Tp      =  M : htm_apply (T)
  z.Kp    = Tp.pc
}
```



New cartesian coordinates system:

```
\directlua{%
init_elements ()
  pi = math.pi
  tp = tex.print
  nl = '\\\\'
  a = point(1,0)
  b = point(0,1)
  R = matrix : htm (pi/5,2,1)
  R : print () tp(nl)
  v = matrix : vector (1,2)
  v : print ()
  v.h = v : homogenization ()
  v.h : print () tp(nl)
  V =   R * v.h
  V : print ()
  z.N = get_htm_point(V)
  tex.print(display(z.N))
}
```

$$\begin{bmatrix} 0.81 & -0.59 & 2 \\ 0.59 & 0.81 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 \\ 2 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} 1.63 \\ 3.21 \\ 1 \end{bmatrix} \quad 1.63+3.21i$$

20.5.7 Function square

We have already seen this method in the presentation of matrices. We first need to give the order of the matrix, then the coefficients, row by row.

```
\directlua{%
init_elements ()
M = matrix : square (2,2,3,-5,4)
M : print ()
}
```

$$\begin{bmatrix} 2 & 3 \\ -5 & 4 \end{bmatrix}$$

20.5.8 Method print

With the `amsmath` package loaded, this method can be used. By default, the `bmatrix` environment is selected, although you can choose from `matrix`, `pmatrix`, `Bmatrix`, "vmatrix", "Vmatrix". Another option lets you set the

number of digits after the decimal point. The "tkz_dc" global variable is used to set the number of decimal places. Here's an example:

```
\directlua{%
init_elements ()
    M = matrix : new ({{math.sqrt(2),math.sqrt(3)},{math.sqrt(4),math.sqrt(5)}})
    M : print ('pmatrix')
}
```

$$\begin{pmatrix} 1.414 & 1.732 \\ 2 & 2.236 \end{pmatrix}$$

You can also display the matrix as a simple array using the `print_array (M)` function. refer to the next example. In the case of a square matrix, it is possible to transmit a list of values whose first element is the order of the matrix.

```
    \directlua{%
init_elements ()
    M = matrix : square (2,1,0,0,2)
    M : print ()
    }
```

$$\begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}$$

20.5.9 Display a table or array: function print_array

We'll need to display results, so let's look at the different ways of displaying them, and distinguish the differences between arrays and matrices.
Below, *A* is an array. It can be displayed as a simple array or as a matrix, but we can't use the attributes and `A : print ()` is not possible because *A* is not an object of the class `matrix`. If you want to display an array like a matrix you can use the function `print_matrix` (refer to the next example).

```
\directlua{%
init_elements ()
  A = {{1,2},{1,-1}}
  tex.print ('A = ') print_array (A)
  tex.print (' or ')
  print_matrix (A)
  M = matrix : new ({{1,1},{0,2}})
  tex.print ('\\\\')
  tex.print ('M = ')  M : print ()
}
```

A = {{ 1 , 2 },{ 1 , -1 }}or $\begin{bmatrix} 1 & 2 \\ 1 & -1 \end{bmatrix}$

$$M = \begin{bmatrix} 1 & 1 \\ 0 & 2 \end{bmatrix}$$

20.5.10 Get an element of a matrix: method get

```
  \directlua{%
init_elements ()
  M = matrix : new {{1,2},{2,-1}}
  S = M: get(1,1) + M: get(2,2)
  tex.print(S)
  }
```

0

20.5.11 Inverse matrix: : method inverse

```
\directlua{%
init_elements ()
 A = matrix : new ({{1,2},{2,-1}})
 tex.print("Inverse of $A = $")
 B =  A : inverse ()
 B : print ()
}
```

Inverse of $A = \begin{bmatrix} 0.429 & 0.286 \\ 0.286 & -0.143 \end{bmatrix}$

20.5.12 Inverse matrix with power syntax

```
\directlua{%
init_elements ()
  M = matrix : new ({{1,0,1},{1,2, 1},{0,-1,2}})
  tex.print("$M = $")  print_matrix (M)
  tex.print('\\\\')
  tex.print("Inverse of $M = M^{-1} = $")
  print_matrix (M^-1)
}
```

$$M = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 2 & 1 \\ 0 & -1 & 2 \end{bmatrix}$$

Inverse of $M = M^{-1} =$

$$\begin{bmatrix} 1.250 & -0.250 & -0.500 \\ -0.500 & 0.500 & 0 \\ -0.250 & 0.250 & 0.500 \end{bmatrix}$$

20.5.13 Transpose matrix: method transpose

A transposed matrix can be accessed with `A: transpose ()` or with `A^{'T'}`.

```
    \directlua{%
init_elements ()
      A = matrix : new ({{1,2},{2,-1}})
      AT = A : transpose ()
      tex.print("$A^{'T'} = $")
       AT : print ()
    }
```

$$A^{'T'} = \begin{bmatrix} 1 & 2 \\ 2 & -1 \end{bmatrix}$$

Remark: `(A ^'T')^'T' = A`

20.5.14 Method method adjugate

```
\directlua{%
init_elements ()
  N =  matrix : new {{1, 0, 3},{2, 1, 0},{-1, 2, 0}}
  tex.print('N = ') print_matrix(N)
  tex.print('\\\\')
  N.a = N : adjugate ()
  N.i = N * N.a
  tex.print('adj(N) = ') N.a : print ()
  tex.print('\\\\')
  tex.print('N $\\times$ adj(N) = ') print_matrix(N.i)
  tex.print('\\\\')
  tex.print('det(N) = ') tex.print(N.det)
}
```

$$\mathrm{N} = \begin{bmatrix} 1 & 0 & 3 \\ 2 & 1 & 0 \\ -1 & 2 & 0 \end{bmatrix} \mathrm{adj(N)} = \begin{bmatrix} 0 & 6 & -3 \\ 0 & 3 & 6 \\ 5 & -2 & 1 \end{bmatrix}$$

$$\mathrm{N} \times \mathrm{adj(N)} = \begin{bmatrix} 15 & 0 & 0 \\ 0 & 15 & 0 \\ 0 & 0 & 15 \end{bmatrix}$$

$\det(\mathrm{N}) = 15.0$

20.5.15 Method method identity

Creating the identity matrix order 3

```
  \directlua{%
init_elements ()
  Id_3 = matrix : identity (3)
  Id_3 : print ()
  }
```

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

20.5.16 Diagonalization: method diagonalize

For the moment, this method only concerns matrices of order 2.

```
  \directlua{%
init_elements ()
  A = matrix : new  {{5,-3}, {6,-4}}
   tex.print('A = ') A : print ()
   D,P =  A : diagonalize ()
   tex.print('D = ')  D : print ()
   tex.print('P = ') P : print ()
   R = P^(-1)*A*P
   tex.print('\\\\')
   tex.print('Test: $D = P^{-1}AP = $ ')
    R : print ()
     tex.print('\\\\')
   tex.print('Verification: $P^{-1}P = $ ')
   T = P^(-1)*P
   T  : print ()
  }
```

$$A=\begin{bmatrix}5 & -3\\ 6 & -4\end{bmatrix} D=\begin{bmatrix}2 & 0\\ 0 & -1\end{bmatrix} P=\begin{bmatrix}1 & 1\\ 1 & 2\end{bmatrix}$$

Test: $D = P^{-1}AP = \begin{bmatrix}2 & 0\\ 0 & -1\end{bmatrix}$

Verification: $P^{-1}P = \begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}$

20.5.17 Method is_orthogonal

The method returns `true` if the matrix is orthogonal and `false` otherwise.

```
  \directlua{%
init_elements ()
  local  cos = math.cos
  local  sin = math.sin
  local  pi  = math.pi
  A = matrix : new ({{cos(pi/6),-sin(pi/6)}, {sin(pi/6),cos(pi/6)}})
  A : print ()
  bool = A : is_orthogonal ()
  tex.print('\\\\')
  if bool
  then
    tex.print("The matrix is orthogonal")
  else
  tex.print("The matrix is not orthogonal")
  end
  tex.print('\\\\')
  tex.print('Test: $A^T = A^{-1} ?$')
  print_matrix(transposeMatrix (A))
  tex.print('=')
  inv_matrix (A) : print ()
  }
```

$$\begin{bmatrix}0.866 & -0.500\\ 0.500 & 0.866\end{bmatrix}$$

The matrix is orthogonal

Test: $A^T = A^{-1}? \begin{bmatrix}0.866 & 0.500\\ -0.500 & 0.866\end{bmatrix} = \begin{bmatrix}0.866 & 0.500\\ -0.500 & 0.866\end{bmatrix}$

20.5.18 Method is_diagonal

The method returns `true` if the matrix is diagonal and `false` otherwise.

21 Math constants and functions

Table 28: 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) [24.1]
islinear (z1,z2,z3)	Are the points aligned? (z2-z1) || (z3-z1) ?
isortho (z1,z2,z3)	(z2-z1) $\perp$ (z3-z1) ? boolean
get_angle (z1,z2,z3)	the vertex is z1 [21.8]
bisector (z1,z2,z3)	L.Aa = bisector (z.A,z.B,z.C) from A [21.8]
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
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
solve_quadratic (a,b,c)	gives the solution of $ax^2 + bx + c = 0$ a,b,c real or complex [21.12.1]

21.1 Length of a segment

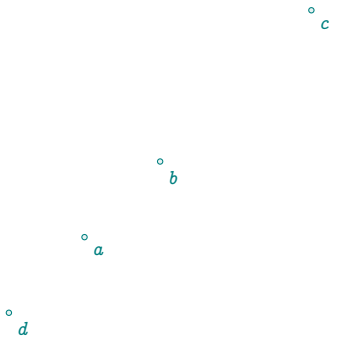
length(z.A,z.B) is a shortcut for point.abs(z.A-z.B). This avoids the need to use complexes.

21.2 Harmonic division with tkzphi

```
\directlua{%
init_elements ()
   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)
}
\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawLine[add= .2 and .2](a,n)
   \tkzDrawPoints(a,b,n,m)
   \tkzLabelPoints(a,b,n,m)
\end{tikzpicture}
```

21.3 Function islinear

```
\directlua{%
init_elements ()
   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
}
\begin{tikzpicture}
    \tkzGetNodes
    \tkzDrawPoints(a,...,d)
    \tkzLabelPoints(a,...,d)
\end{tikzpicture}
```



21.4 Function value

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

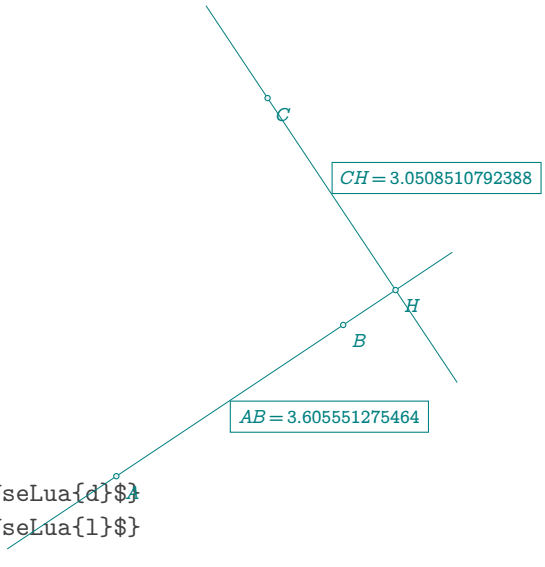
21.5 Function real

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

21.6 Transfer from lua to TEX

It's possible to transfer variable from Lua to TEX with the macro **\tkzUseLua**.

```
\directlua{%
init_elements ()
   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)
}
\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}
```

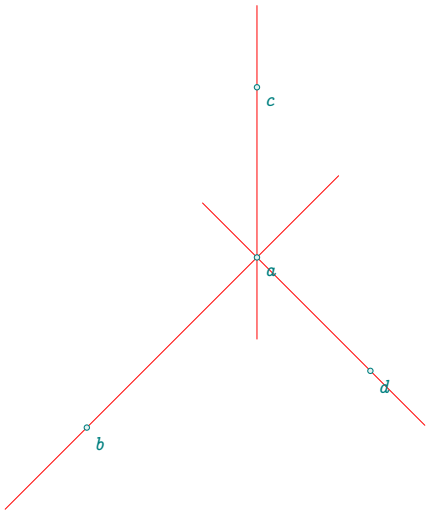


21.7 Normalized angles : Slope of lines (ab), (ac) and (ad)

```
\directlua{%
init_elements ()
   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))..'\\\\')
}
\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawLines[red](a,b a,c a,d)
   \tkzDrawPoints(a,b,c,d)
   \tkzLabelPoints(a,b,c,d)
\end{tikzpicture}
```

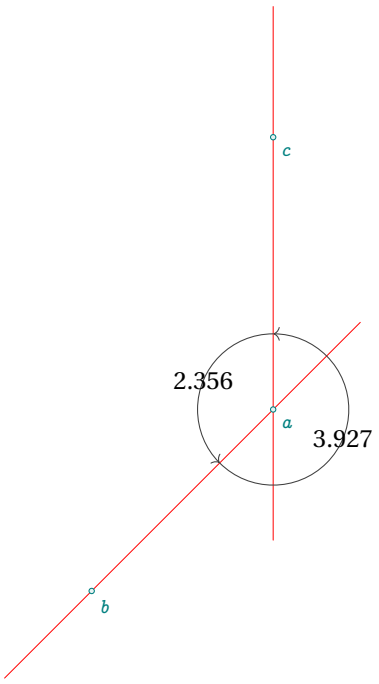
slope of (ab) : -2.3561944901923
slope normalized of (ab) : 3.9269908169872
slope of (ac) : 1.5707963267949
slope normalized of (ac) : 1.5707963267949
slope of (ad) : -0.78539816339745
slope normalized of (ad) : 5.4977871437821



21.8 Get angle

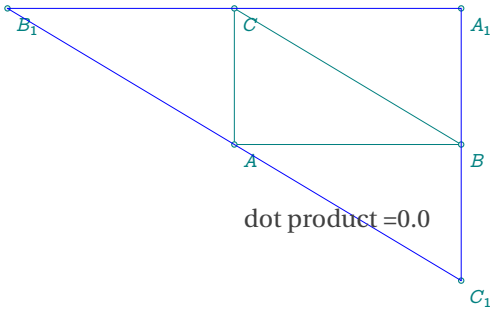
The function `get_angle (a,b,c)` gives the angle normalized of $(\overrightarrow{ab}, \overrightarrow{ac})$.

```
\directlua{%
init_elements ()
   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)
}

\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}
```



21.9 Dot or scalar product

```
\directlua{%
init_elements ()
   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)
}
\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.

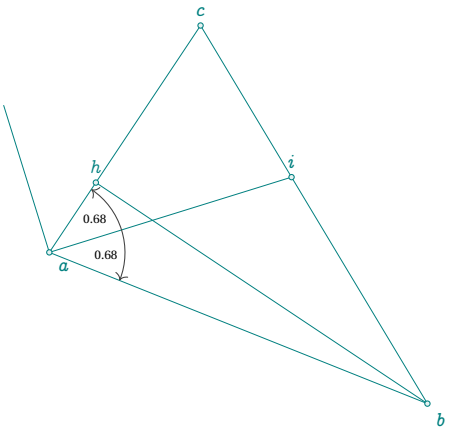
21.10 Alignment or orthogonality

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).

21.11 Bisector and altitude

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

```
\directlua{%
init_elements ()
   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
}

\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}
```



21.12 Other functions

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

21.12.1 Function solve_quadratic

This function solves the equation $ax^2+bx+c=0$ with real or complex numbers.

```
\directlua{%
init_elements ()
    tex.sprint('Solve : $x^2+1=0$ The solution set is  ')
    r1,r2 = solve_quadratic(1,0,1)
    tex.print('\\{'..tostring(r1)..' , '..tostring(r2)..'\\}')
    tex.print('\\\\')
    tex.sprint('Solve : $x^2+2x-3=0$ The solution set is  ')
    r1,r2 = solve_quadratic(1,2,-3)
    tex.print('\\{'..tostring(r1)..' , '..tostring(r2)..'\\}')
    tex.print('\\\\')
    a = point (0,1)
    b = point (1,1)
    c = point (-1,1)
    tex.sprint('Solve : $ix^2+(1+i)x+(-1+i)=0$ The solution set is  ')
    r1,r2 = solve_quadratic(a,b,c)
    tex.print('\\{'..tostring(r1)..' , '..tostring(r2)..'\\}')
}
```

Solve : $x^2+1=0$ The solution set is {i , -i}
Solve : $x^2+2x-3=0$ The solution set is {1.0 , -3.0}
Solve : $ix^2+(1+i)x+(-1+i)=0$ The solution set is {0.134-0.684i , -1.134+1.684i}

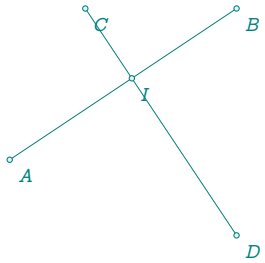
22 Intersections

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 _.

22.1 Line-line

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

```
\directlua{%
init_elements ()
   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
}

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

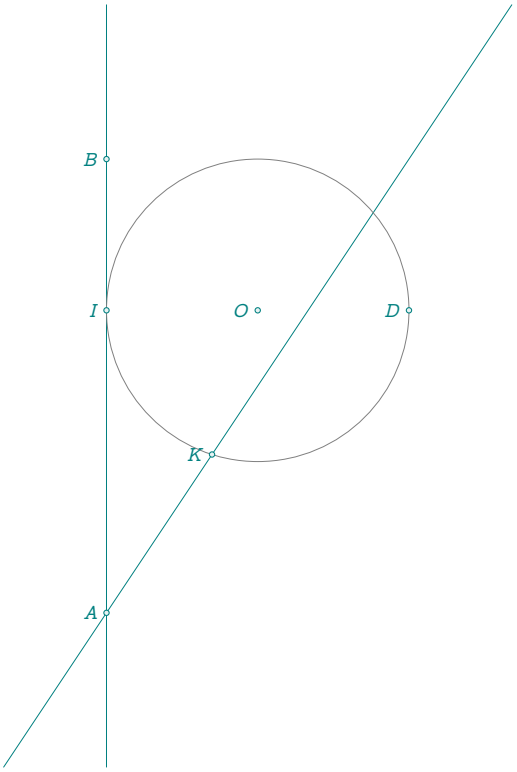


Other examples: 24.4, 24.5, 24.6

22.2 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).

```
\directlua{%
init_elements ()
   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)
}

\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}
```

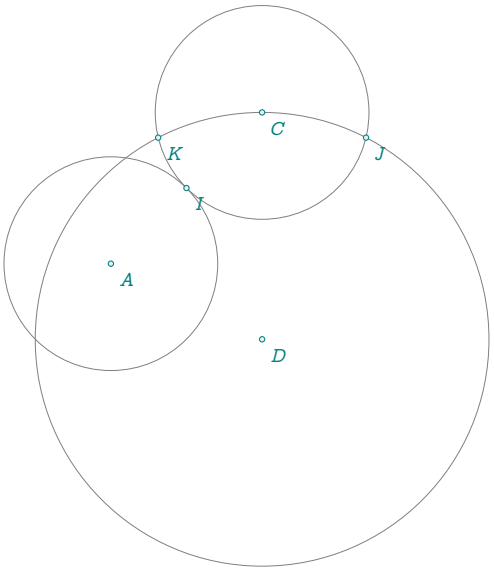


Other examples: 24.4

22.3 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).

```
\directlua{%
init_elements ()
   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)
}
\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}
```

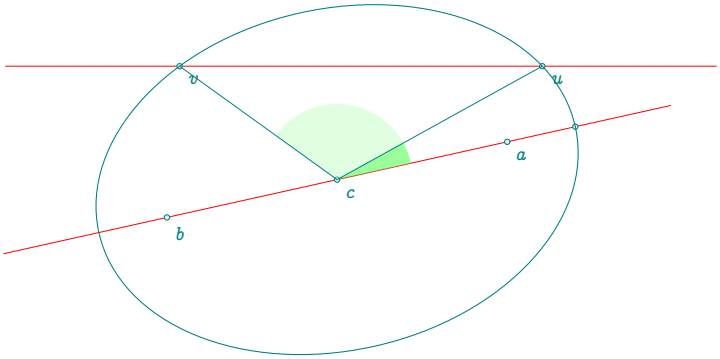


Other examples: 24.4, 4.3

22.4 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.

```
\directlua{%
init_elements ()
   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)
}
\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: 13.2.2, 24.31

23 In-depth study

23.1 The tables

23.1.1 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
```

23.1.2 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.

23.2 Transfers

We've seen (sous-section 7.1.1) 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.

23.3 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 29: 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 30: 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 21.9

- 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.

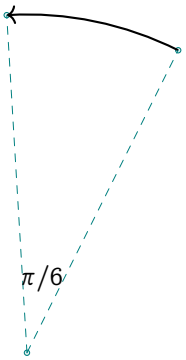
23.3.1 Example of complex use

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)).
```

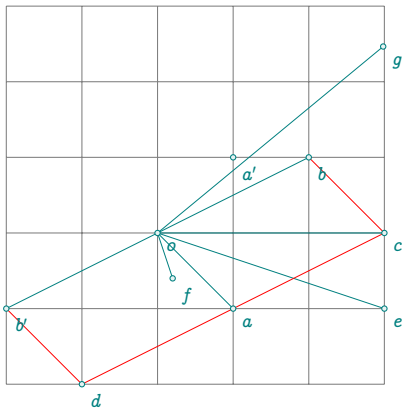
Then z.B = z.A * za describes a rotation of point A by an angle a.

```
      \directlua{%
init_elements ()
         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
      }
      \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}
```



23.3.2 Point operations (complex)

```
\directlua{%
init_elements ()
   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))
}

\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}
```



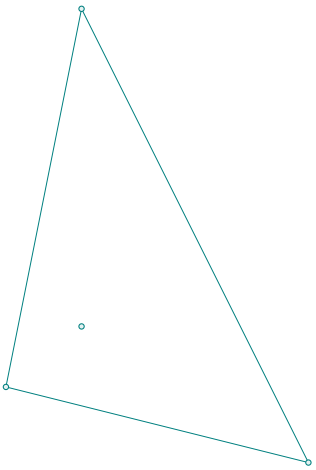
23.4 Barycenter

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
```

23.4.1 Using the barycentre

```
\directlua{%
init_elements ()
 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})
}

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



23.4.2 Incenter of a triangle

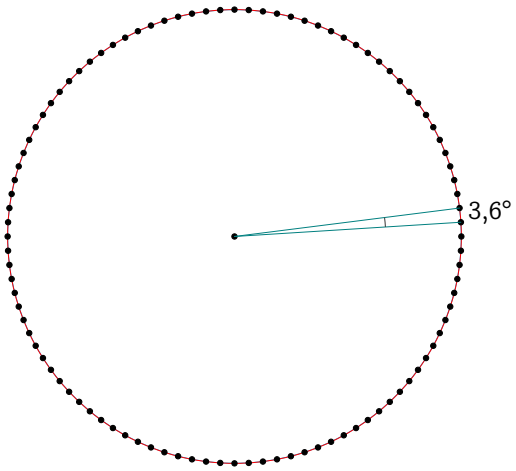
The calculation of the weights ka, kb and kc is precise, and the result obtained with the barycenter is excellent. Note the presence of the underscore _ for certain functions. These functions are internal (developer). Each external (user) function is associated with its internal counterpart.
Here's how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

23.5 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.

```
\directlua{%
init_elements ()
  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))
}
```



23.6 In_out method

This function can be used for the following objects

- line
- circle
- triangle
- ellipse

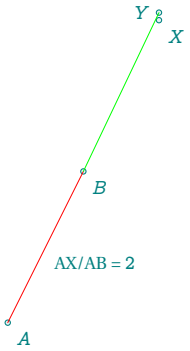
The disk object doesn't exist, so with `in\_out\_disk` it's possible to determine whether a point is in a disk.

23.6.1 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.

```
\directlua{%
init_elements ()
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)
}

\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}
```



23.7 Determinant and dot product

23.7.1 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
```

23.7.2 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
```

23.7.3 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
```

23.7.4 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.

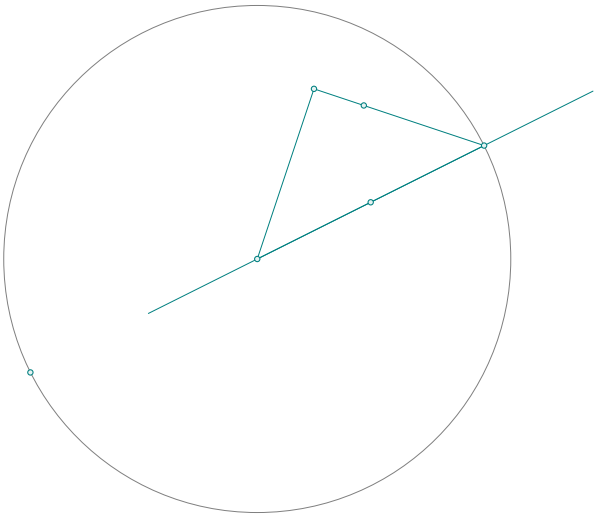
23.8 Point method

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.

```
\directlua{%
init_elements ()
   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)
}
\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawLine(A,B)
   \tkzDrawCircle(A,B)
   \tkzDrawPolygon(A,B,C)
   \tkzDrawPoints(A,B,C,I,J,K)
\end{tikzpicture}
```



23.9 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)`

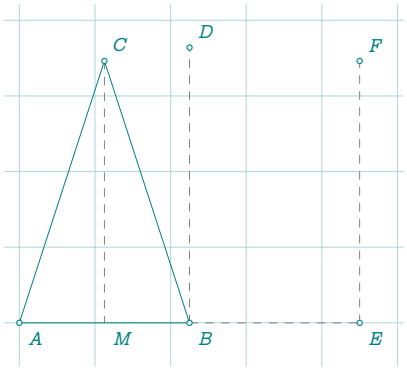
The difference is that programming is more direct with primary functions and a little more efficient, but loses visibility.

24 Examples

24.1 Length transfer

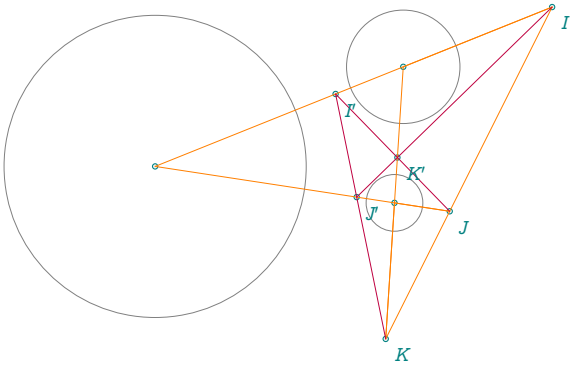
Use of `north and east` functions linked to points, to transfer lengths, Refer to (21.1)

```
\directlua{%
init_elements ()
  scale = .75
   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))
}
\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}
```



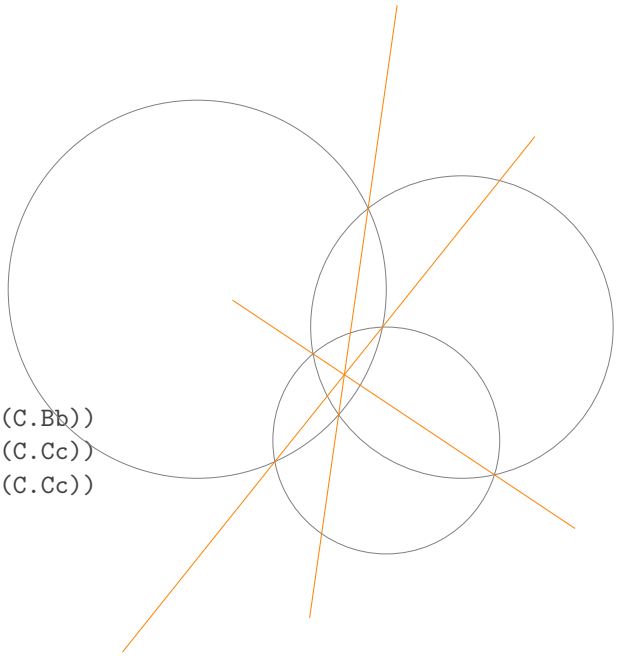
24.2 D'Alembert 1

```
\directlua{%
init_elements ()
   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)
}
\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}
```



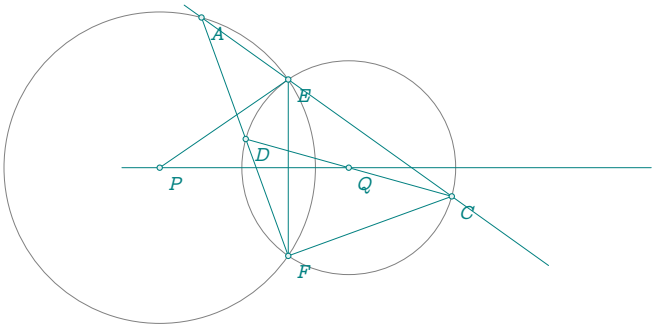
24.3 D'Alembert 2

```
\directlua{%
init_elements ()
   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))
}
\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawCircles(A,a B,b C,c)
   \tkzDrawLines[new](i,j k,l m,n)
\end{tikzpicture}
```



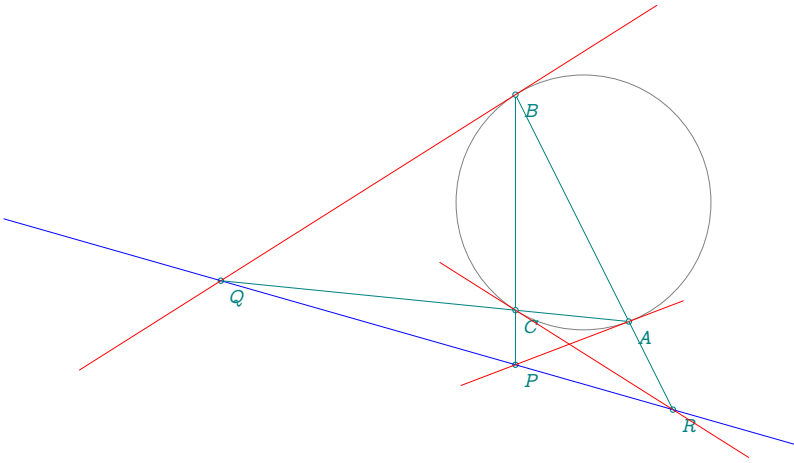
24.4 Altshiller

```
\directlua{%
init_elements ()
   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)
}
\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}
```



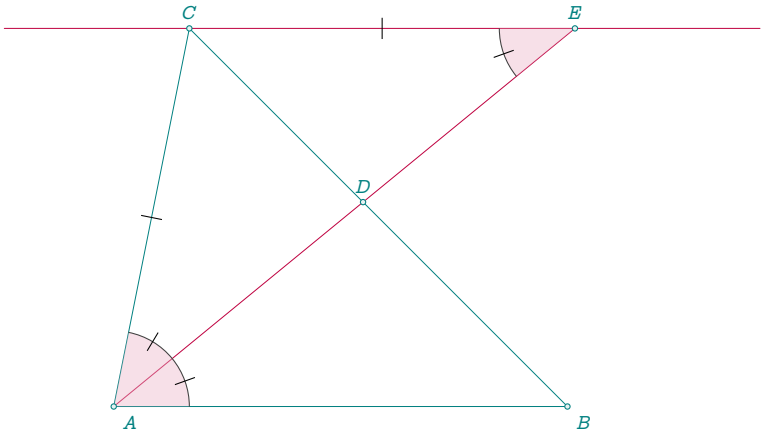
24.5 Lemoine

```
\directlua{%
init_elements ()
   scale = 1.25
   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)
}
\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}
```



24.6 Alternate

```
\directlua{%
init_elements ()
   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)
}
\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}
```

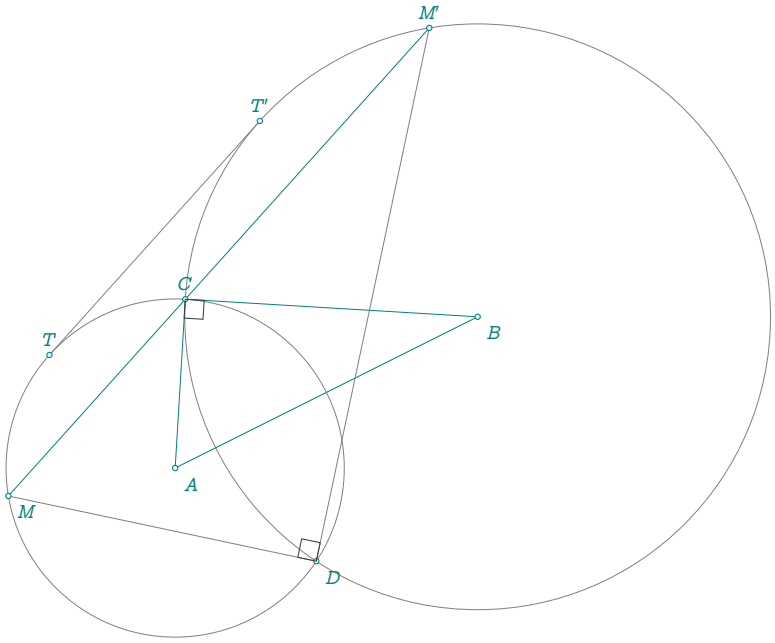


24.7 Method common tangent: orthogonality

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.

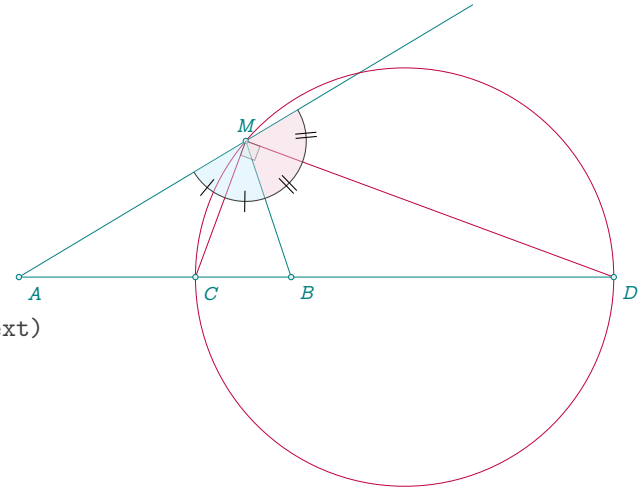
```
\directlua{%
init_elements ()
   z.A   = point : new ( 0  , 0  )
   z.B   = point : new ( 4  , 2  )
   L.AB = line : new ( z.A , z.B )
   z.a   = point : new ( 1 , 2 )
   C.Aa  = circle : new (z.A,z.a)
   C.BC = C.Aa : orthogonal_from (z.B)
```

```
   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)
}

\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawCircles(A,C B,C)
   \tkzDrawSegments(M,M' A,C B,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}
```



24.8 Apollonius circle

```
\directlua{%
init_elements ()
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)
}
```



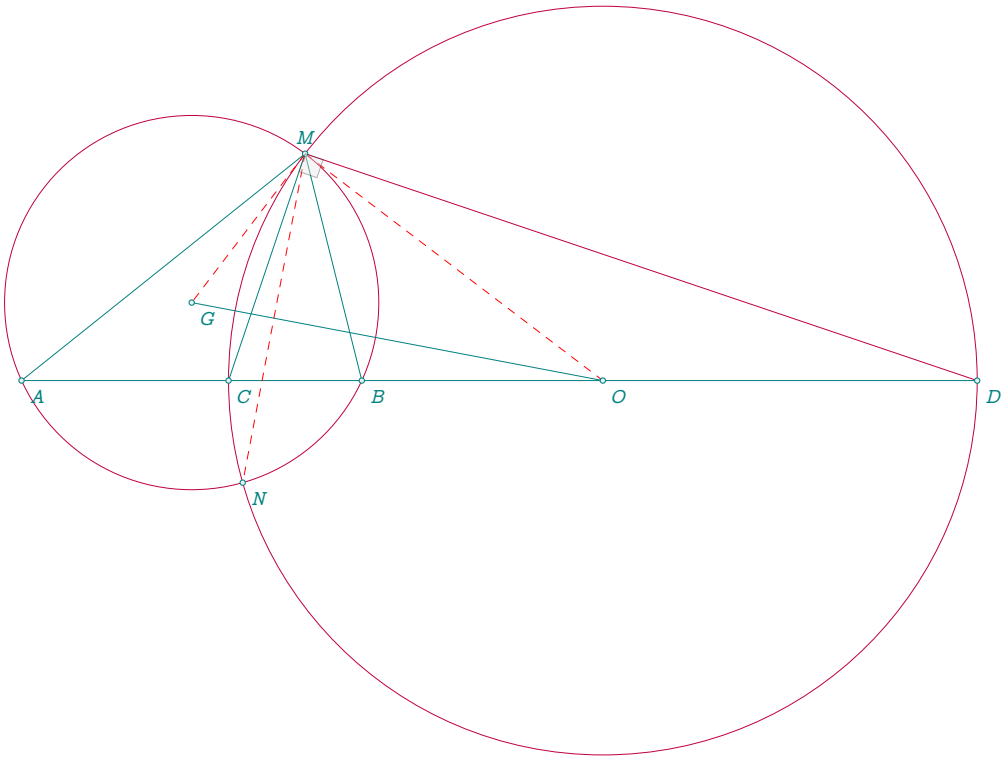

```
\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawSegment[add=0 and 1](A,M)
   \tkzDrawSegments[purple](M,C M,D)
   \tkzDrawCircle[purple](O,C)
   \tkzDrawSegments(A,B B,M D,B)
   \tkzDrawPoints(A,B,M,C,D)
   \tkzLabelPoints[below right](A,B,C,D)
   \tkzLabelPoints[above](M)
   \tkzFillAngles[opacity=.4,cyan!20](A,M,B)
   \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))`

24.9 Apollonius and circle circumscribed

```
\directlua{%
init_elements ()
   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
```

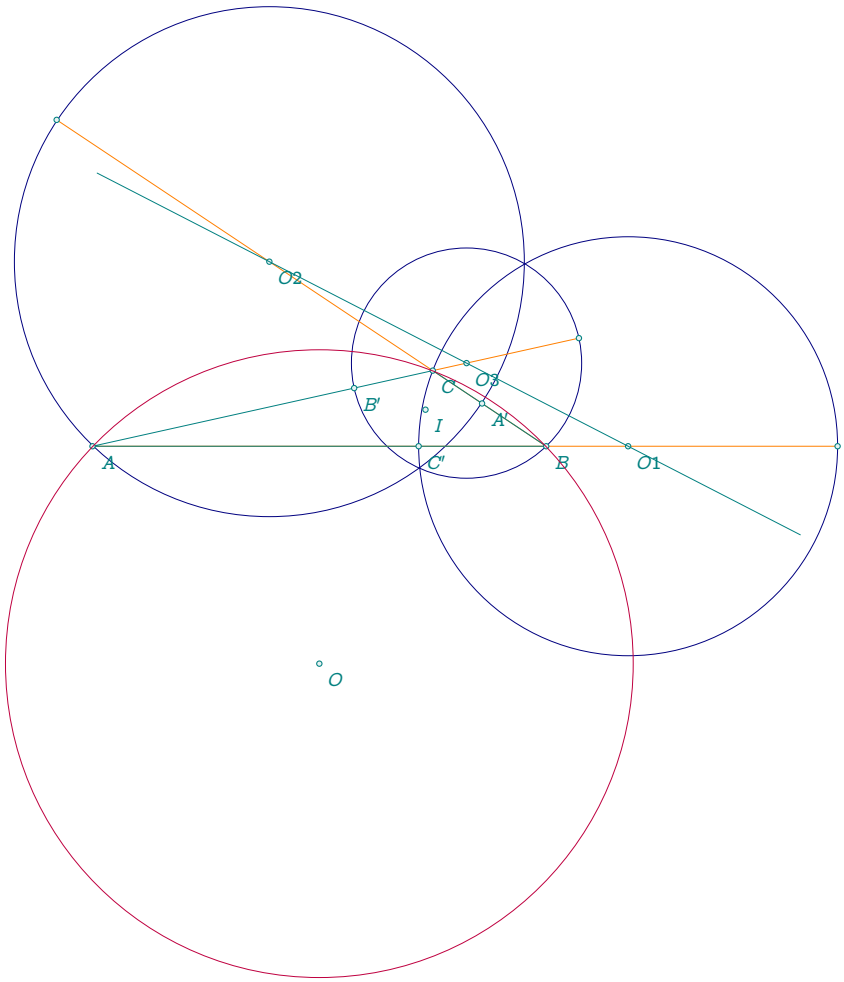
```
   z.G   = T.AMB.circumcenter
   C.GA  = circle: new (z.G,z.A)
   C.OC  = circle: new (z.O,z.C)
   _,z.N = intersection (C.GA , C.OC)
}
\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawPolygon(A,B,M)
   \tkzDrawCircles[purple](O,C G,A)
   \tkzDrawSegments[purple](M,D)
   \tkzDrawSegments(D,B O,G M,C)
   \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}
```



24.10 Apollonius circles in a triangle

```
\directlua{%
init_elements ()
   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
}

\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}
```

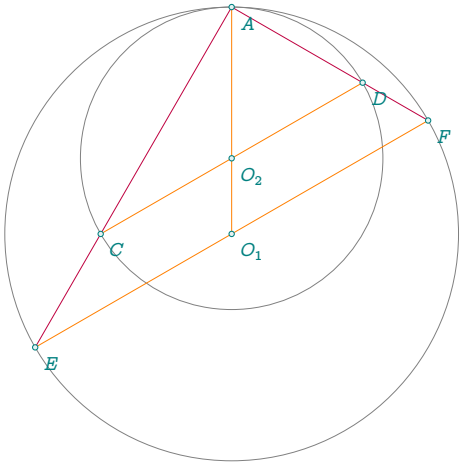


Same result using the function `T.ABC.ab : apollonius (k)`

```
   \directlua{%
init_elements ()
   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 )
   }
```

24.11 Archimedes

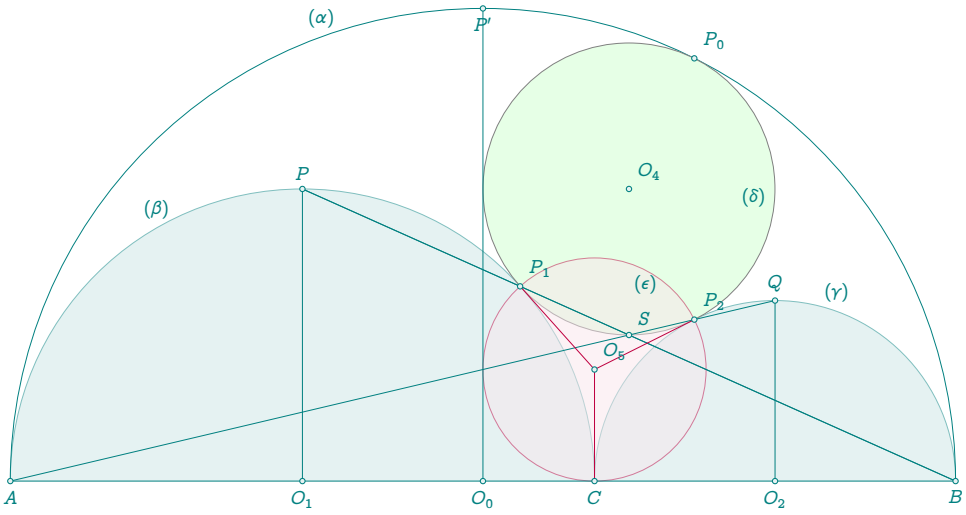
```
\directlua{%
init_elements ()
   z.O_1     = point:   new  (0, 0)
   z.O_2     = point:   new  (0, 1)
   z.A       = point:   new  (0, 3)
   z.F       = point:   polar (3, math.pi/6)
   L         = line:    new   (z.F,z.O_1)
   C         = circle:  new  (z.O_1,z.A)
   z.E       = intersection (L,C)
   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)
}
\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}
```



24.12 Bankoff circle

```
\directlua{%
init_elements ()
   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)
```

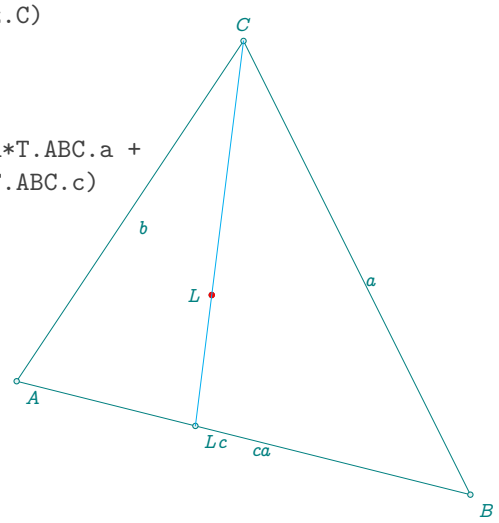
```
   z.S      = intersection (L.BP,L.AQ)
   L.PpOQ   = line : new (z.Pp,z.O_Q)
   L.PC     = line : new (z.P,z.C)
   z.Ap     = intersection (L.PpOQ,L.PC)
   L.CS     = line : new (z.C,z.S)
   C.M1A    = circle : new (z.M_1,z.A)
   C.M2B    = circle : new (z.M_2,z.B)
   z.P_Q    = intersection (L.CS,C.OQB)
   z.P_1    = intersection (C.M2B,C.O1C)
   z.P_2    = intersection (C.M1A,C.O2C)
   T.PQP1P2 = triangle : new (z.P_Q,z.P_1,z.P_2)
   z.O_4    = T.PQP1P2.circumcenter
   T.CP1P2  = triangle : new (z.C,z.P_1,z.P_2)
   z.O_5    = T.CP1P2.circumcenter
}

\begin{tikzpicture}
\tkzGetNodes
\tkzDrawSemiCircles[teal](O_Q,B)
\tkzDrawSemiCircles[teal,fill=teal!20,opacity=.5](O_1,C O_2,B)
\tkzDrawCircle[fill=green!10](O_4,P_Q)
\tkzDrawCircle[purple,fill=purple!10,opacity=.5](O_5,C)
\tkzDrawSegments(A,B O_Q,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_Q,P_2,P_1,O_Q,O_1,O_2,O_4,O_5,Q,P,P',S)
\tkzLabelPoints[below](A,B,C,O_Q,O_1,O_2,P')
\tkzLabelPoints[above](Q,P)
\tkzLabelPoints[above right](P_Q,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_Q,B)(110){$(\alpha)$}
  \tkzLabelCircle[left](O_4,P_2)(60){$(\delta)$}
  \tkzLabelCircle[left](O_5,C)(140){$(\epsilon)$}
\end{scope}
\end{tikzpicture}
```



24.13 Symmedian property

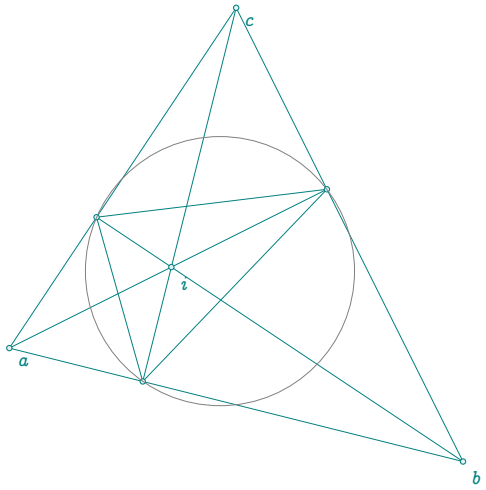
L is the symmedian point or lemoine point. $\dfrac{CL}{CLc} = \dfrac{a^2+b^2}{a^2+b^2+c^2}$

```
    \directlua{%
init_elements ()
       scale          = 1.5
       z.A            = point :   new (1,2)
       z.B            = point :   new (5,1)
       z.C            = point :   new (3,5)
       T.ABC          = triangle : new (z.A,z.B,z.C)
       T.SY           = T.ABC : symmedian ()
       z.La,z.Lb,z.Lc = get_points (T.SY)
       k              = (T.ABC.a*T.ABC.a +
                         T.ABC.b*T.ABC.b)/(T.ABC.a*T.ABC.a +
                         T.ABC.b*T.ABC.b+T.ABC.c*T.ABC.c)
       L.SY            = line : new (z.C,z.Lc)
       z.L             = L.SY : point (k)
    }
    \begin{tikzpicture}
    \tkzGetNodes
    \tkzDrawPolygons(A,B,C)
    \tkzDrawPoints(A,B,C,L,Lc)
    \tkzDrawPoints[red](L)
    \tkzDrawSegments[cyan](C,Lc)
    \tkzLabelPoints(A,B,Lc)
    \tkzLabelPoints[above](C)
    \tkzLabelPoints[left](L)
    \tkzLabelSegment(B,C){$a$}
    \tkzLabelSegment(A,C){$b$}
    \tkzLabelSegment(A,B){$ca$}
    \end{tikzpicture}
```



24.14 Example: Cevian with orthocenter

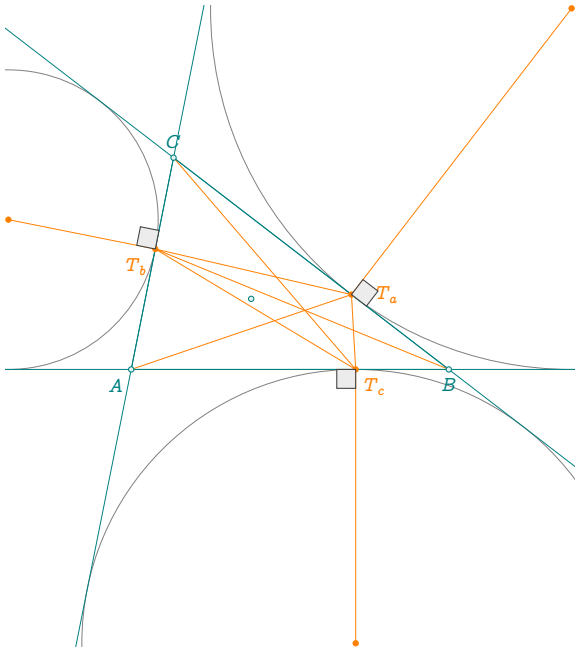
```
\directlua{%
init_elements ()
  scale = 1.5
  z.a = point:  new (1,2)
  z.b = point:  new (5,1)
  z.c = point:  new (3,5)
  T = triangle: new (z.a,z.b,z.c)
  z.i = T.orthocenter
  T.cevian = T : cevian (z.i)
  z.ta,z.tb,z.tc = get_points (T.cevian)
  C.cev = T : cevian_circle (z.i)
  z.w = C.cev.center
}
\begin{tikzpicture}
  \tkzGetNodes
  \tkzDrawPolygons(a,b,c ta,tb,tc)
  \tkzDrawSegments(a,ta b,tb c,tc)
  \tkzDrawPoints(a,b,c,i,ta,tb,tc)
  \tkzLabelPoints(a,b,c,i)
  \tkzDrawCircles(w,ta)
\end{tikzpicture}
```



24.15 Excircles

```
\directlua{%
init_elements ()
   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)
}
\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)
```

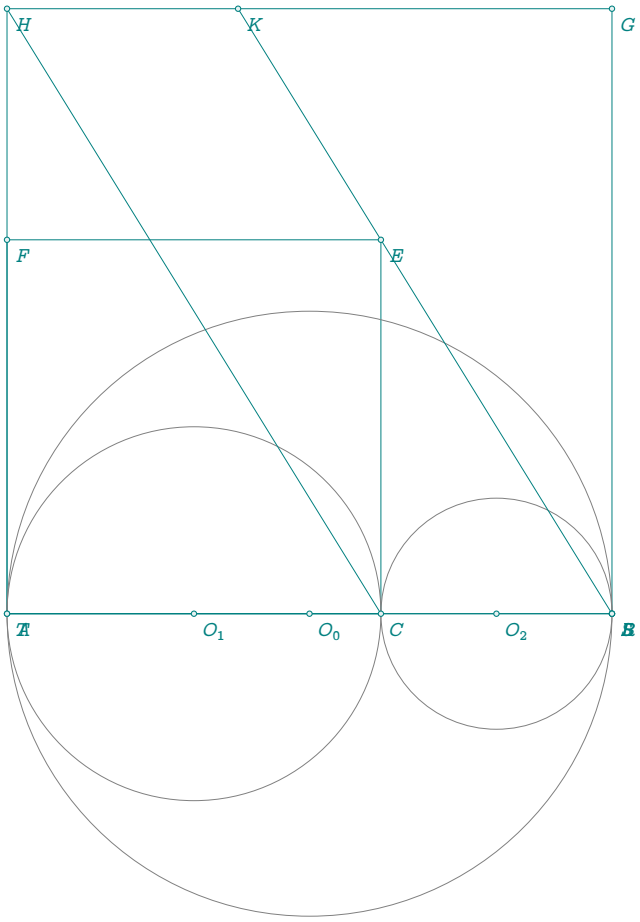
```
   \tkzLabelPoints[above](C)
   \tkzLabelPoints[new,below left](T_b)
   \tkzLabelPoints[new,below right](T_c)
   \tkzLabelPoints[new,right=6pt](T_a)
   \tkzMarkRightAngles[fill=gray!15](J_a,T_a,B J_b,T_b,C J_c,T_c,A)
\end{tikzpicture}
```



24.16 Divine ratio

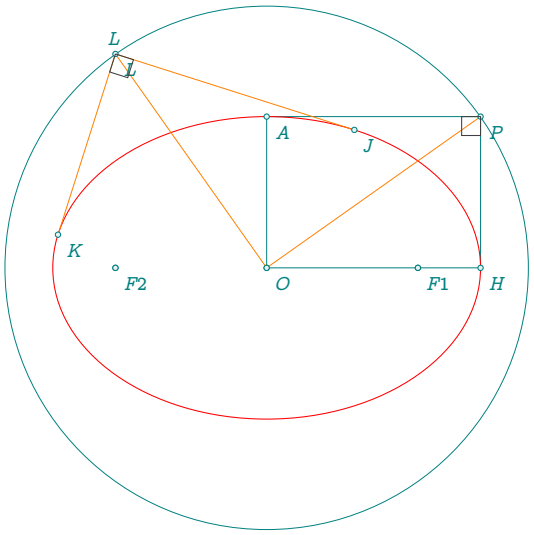
```
\directlua{%
init_elements ()
z.A          = point: new (0 , 0)
z.B          = point: new (8 , 0)
L.AB         = line: new (z.A,z.B)
z.C          = L.AB: gold_ratio ()
L.AC         = line: new (z.A,z.C)
z.O_1        = L.AC.mid
_,_,z.G,z.H = get_points(L.AB: square ())
_,_,z.E,z.F = get_points(L.AC: 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)
}
\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}
```



24.17 Director circle

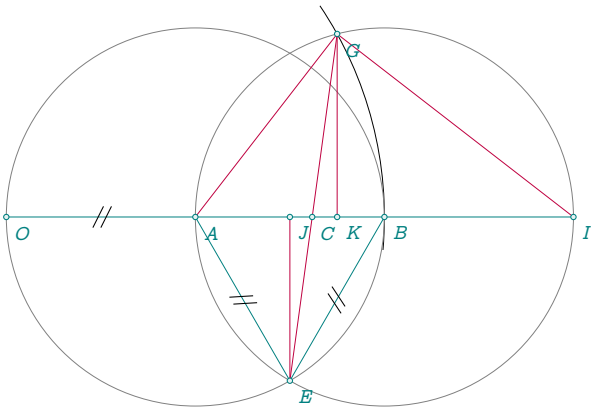
```
\directlua{%
init_elements ()
  scale    = .5
  z.O      = point: new (0 , 0)
  z.F1     = point: new (4 , 0)
  z.F2     = point: new (-4 , 0)
  z.H      = point: new (4*math.sqrt(2) , 0)
  E        = ellipse: foci (z.F2,z.F1,z.H)
  a,b      = E.Rx, E.Ry
  z.A      = E.covertex
  T        = triangle: 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
}
\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}
```



24.18 Gold division

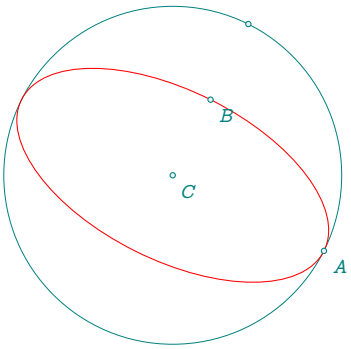
```
\directlua{%
init_elements ()
z.A          = point:  new (0,0)
z.B          = point:  new (2.5,0)
L.AB         = line:   new (z.A,z.B)
C.AB         = circle: new (z.A,z.B)
C.BA         = circle: new (z.B,z.A)
z.J          = L.AB: midpoint ()
L.JB         = line:new (z.J,z.B)
z.F,z.E      = intersection (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)
}
```

```
\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}
```



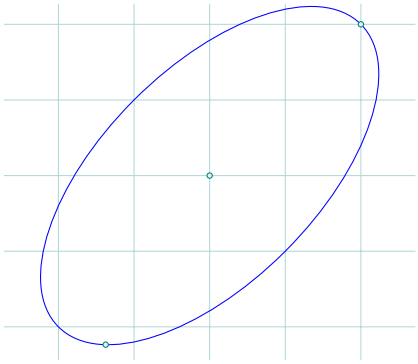
24.19 Ellipse

```
\directlua{%
init_elements ()
   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)
}
\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}
```



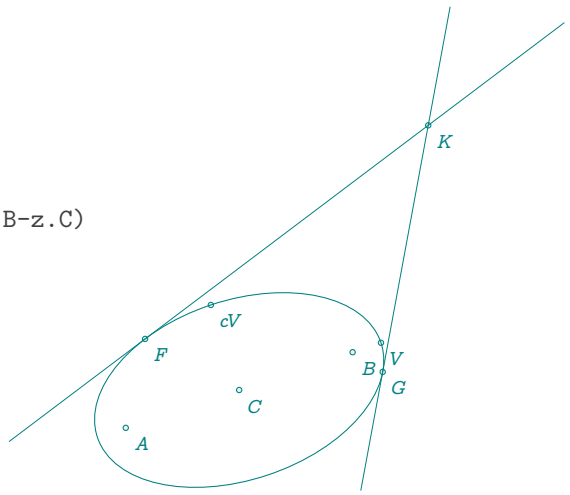
24.20 Ellipse with radii

```
\directlua{%
init_elements ()
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)
}
\begin{tikzpicture}[gridded]
\tkzGetNodes
\tkzDrawEllipse[blue](C,\tkzUseLua{a},
              \tkzUseLua{b},\tkzUseLua{ang})
\tkzDrawPoints(C,V,CoV)
\end{tikzpicture}
```



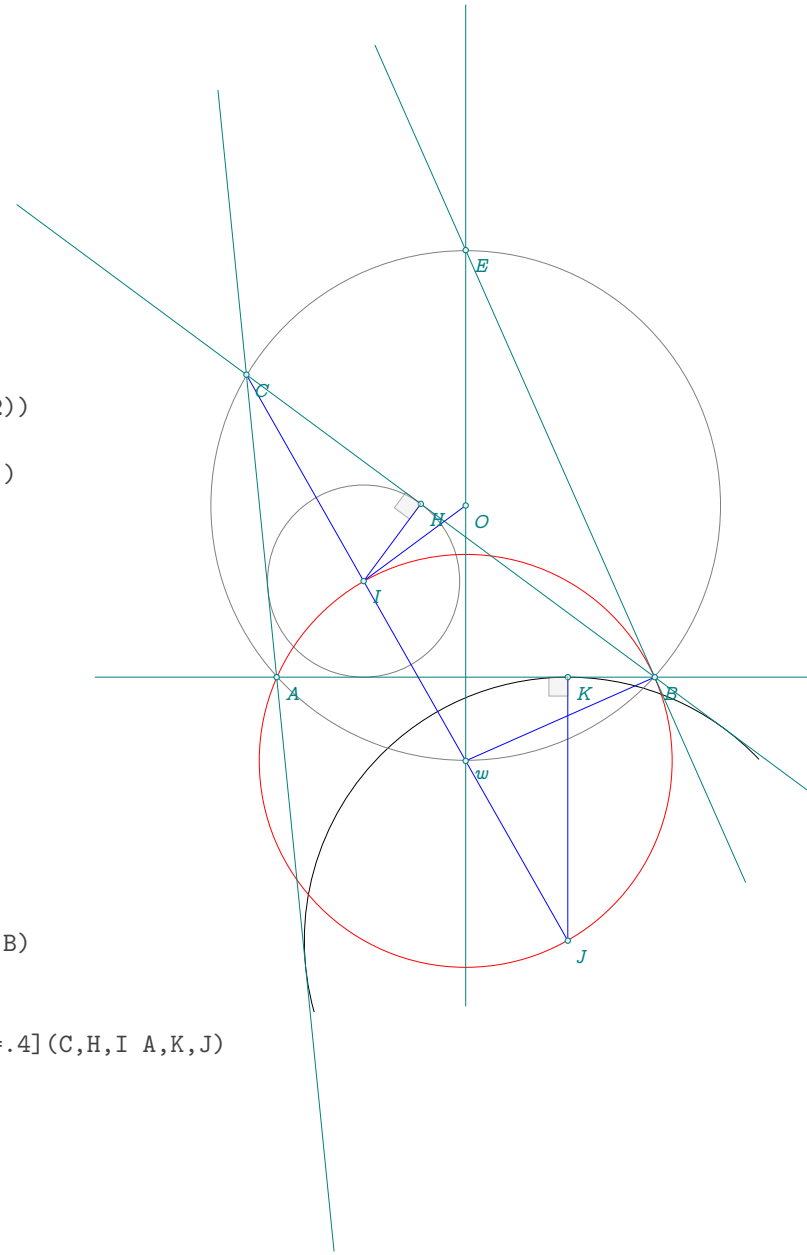
24.21 Ellipse_with_foci

```
\directlua{%
init_elements ()
   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
}
\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}
```



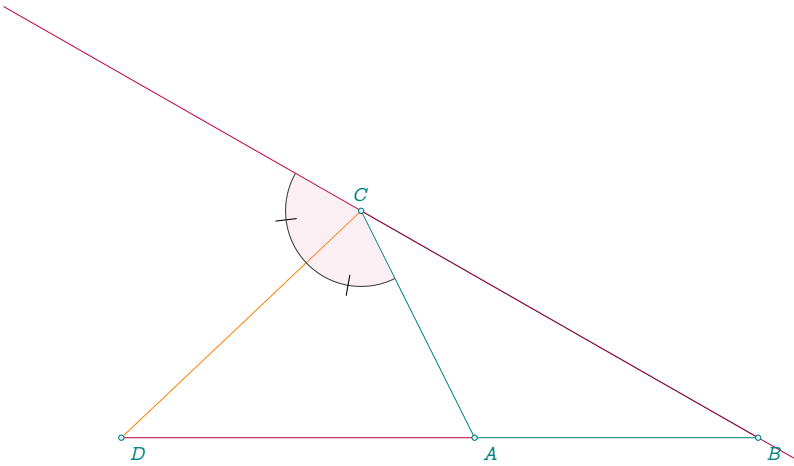
24.22 Euler relation

```
\directlua{%
init_elements ()
 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)
}
   \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}
```



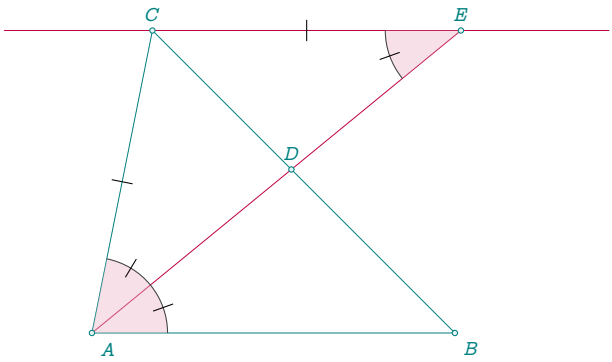
24.23 External angle

```
\directlua{%
init_elements ()
  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)
}
\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}
```



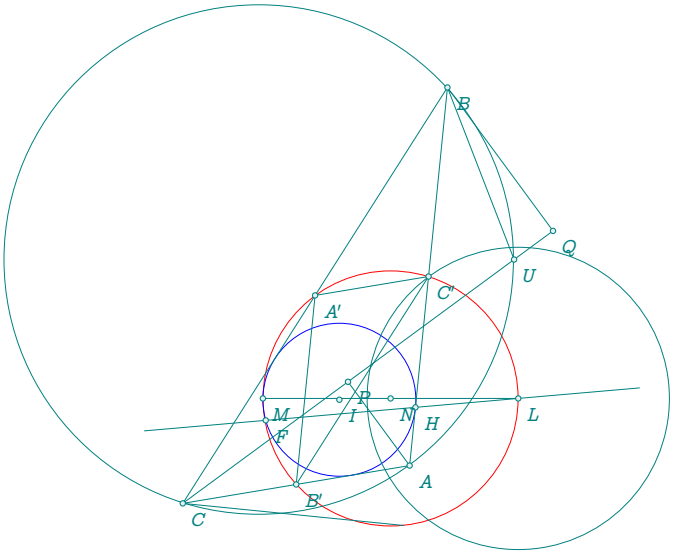
24.24 Internal angle

```
\directlua{%
init_elements ()
   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)
}
\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}
```



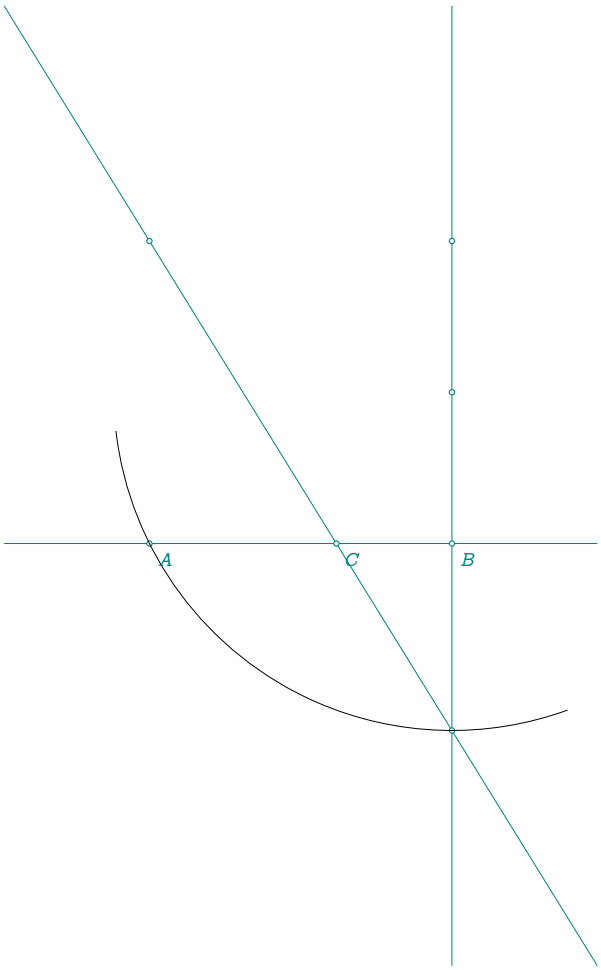
24.25 Feuerbach theorem

```
\directlua{%
init_elements ()
   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
}

\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}
```



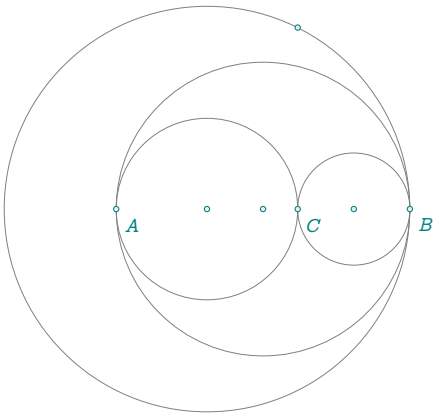
24.26 Gold ratio with segment

```
\directlua{%
init_elements ()
   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)
}
\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}
```



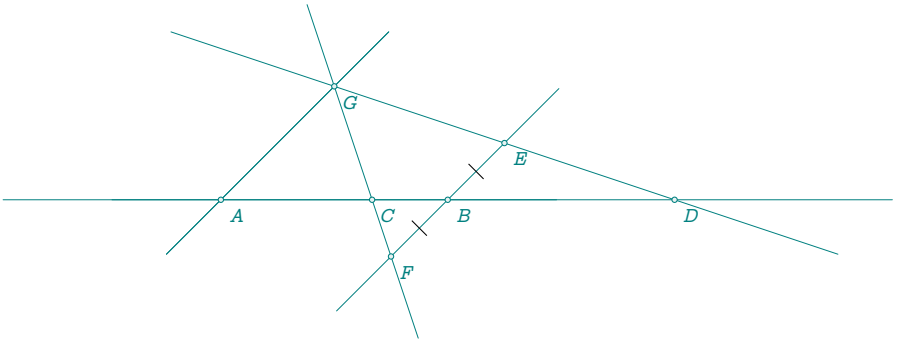
24.27 Gold Arbelos

```
\directlua{%
init_elements ()
   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
}
\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}
```



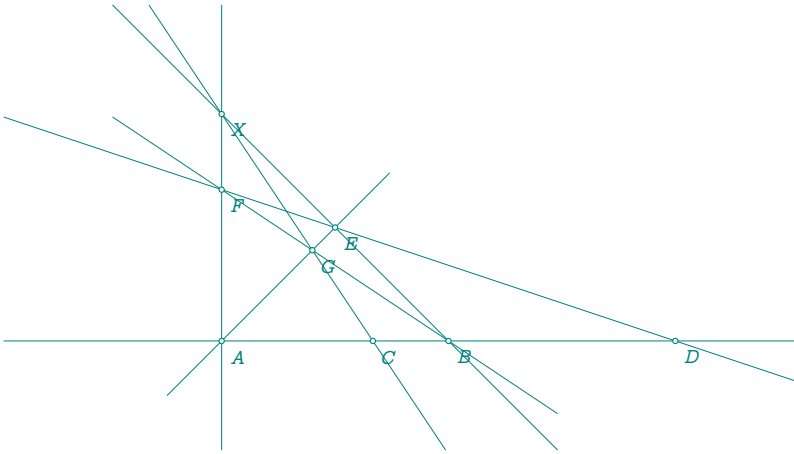
24.28 Harmonic division v1

```
\directlua{%
init_elements ()
 scale=.75
 z.A  = point: new (0 , 0)
 z.B  = point: new (4 , 0)
 z.G  = point: new (2,2)
 L.AG = line : new (z.A,z.G)
 L.AB = line : new (z.A,z.B)
 z.E = L.AG : colinear_at (z.B,.5)
 L.GE = line : new (z.G,z.E)
 z.D  = intersection (L.GE,L.AB)
 z.F = z.B : symmetry (z.E)
 L.GF = line :new (z.G,z.F)
 z.C  = intersection (L.GF,L.AB)
}
\begin{tikzpicture}
 \tkzGetNodes
 \tkzDrawLines(A,B A,G 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}
```



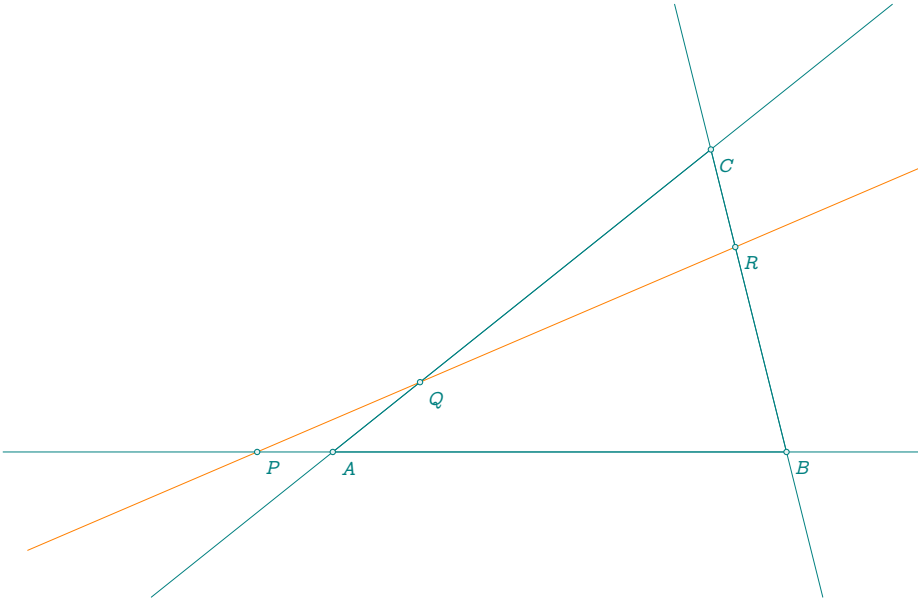
24.29 Harmonic division v2

```
\directlua{%
init_elements ()
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)
}
\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}
```



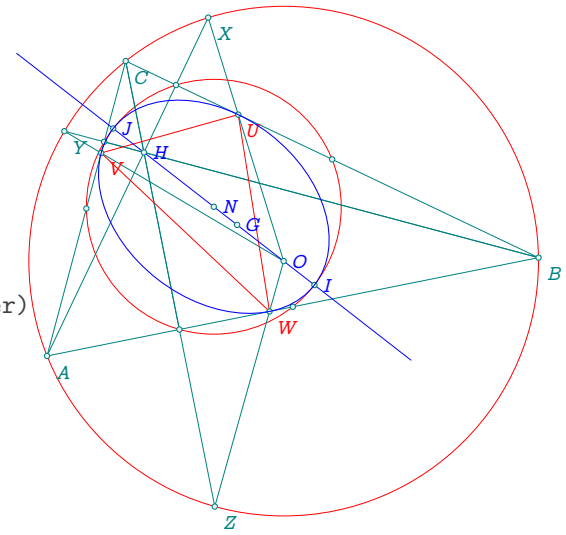
24.30 Menelaus

```
\directlua{%
init_elements ()
   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)
}
\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}
```



24.31 Euler ellipse

```
\directlua{%
init_elements ()
  scale     = 1.3
  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)
}
```

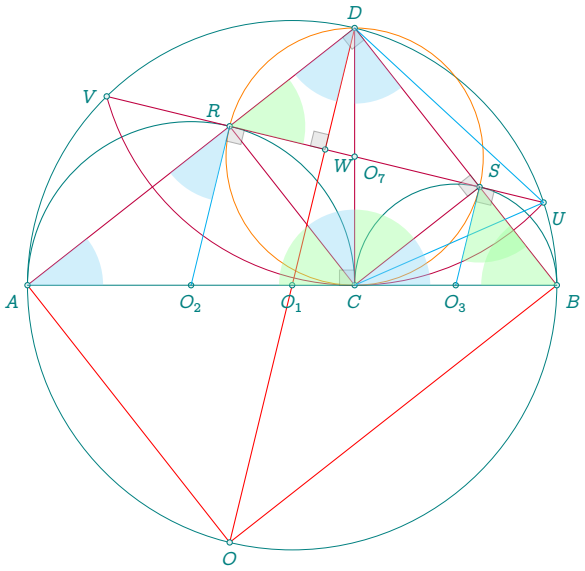



```
\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawPolygon(A,B,C)
   \tkzDrawCircles[red](N,Ma O,A)
   \tkzDrawSegments(A,X B,Y C,Z B,Hb C,Hc X,O Y,O Z,O)
   \tkzDrawPolygon[red](U,V,W)
   \tkzLabelPoints[red](U,V,W)
   \tkzLabelPoints(A,B,C,X,Y,Z)
   \tkzDrawLine[blue](I,J)
   \tkzLabelPoints[blue,right](O,N,G,H,I,J)
```

```
   \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}
```

24.32 Gold Arbelos properties

```
\directlua{%
init_elements ()
  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)
}
```



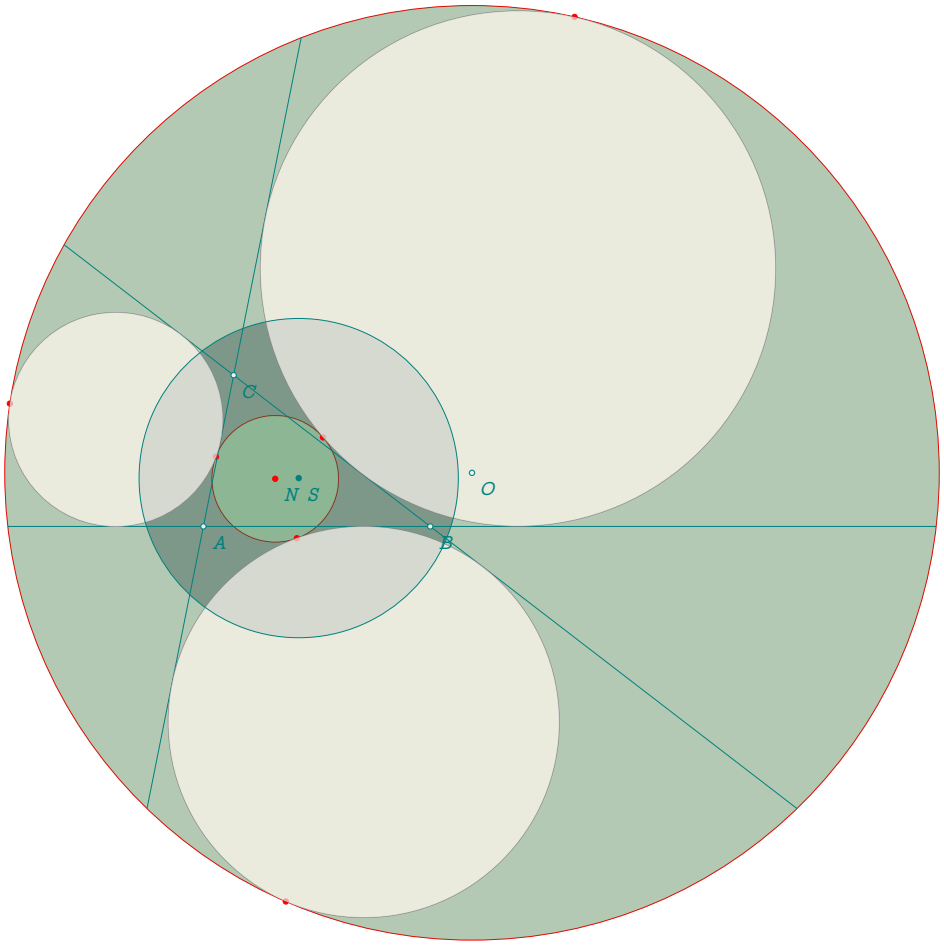

```
\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawCircles[teal](O_1,B)
   \tkzDrawSemiCircles[thin,teal](O_2,C O_3,B)
   \tkzDrawArc[purple,delta=0](D,V)(U)
   \tkzDrawCircle[new](O_7,C)
   \tkzDrawSegments[thin,purple](A,D D,B C,R C,S C,D U,V)
   \tkzDrawSegments[thin,red](O,D A,O O,B)
```

```
   \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}
```

24.33 Apollonius circle v1 with inversion

```
\directlua{%
init_elements ()
   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)
}
\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)
```

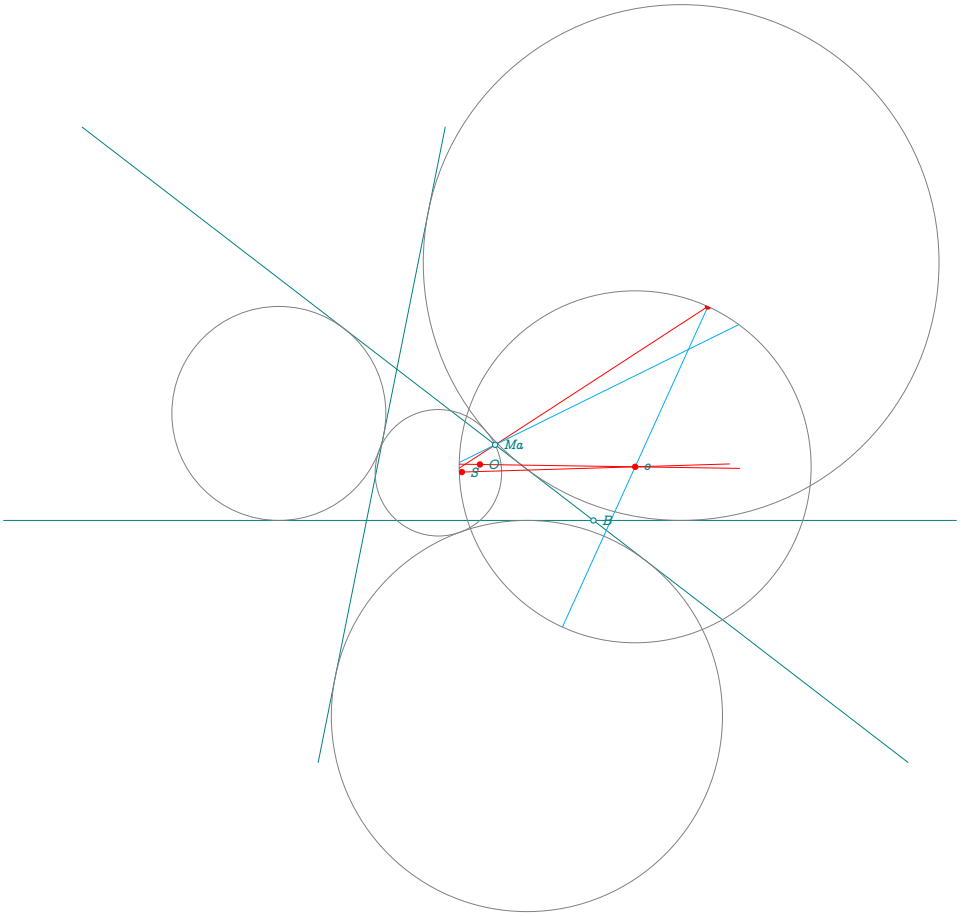
```
\tkzFillCircles[lightgray!30,opacity=.7](Ja,Ea Jb,Eb Jc,Ec)
\tkzDrawCircles[teal](S,a)
\tkzDrawPoints(A,B,C,O)
\tkzDrawPoints[teal](S)
\tkzLabelPoints(A,B,C,O,S,N)
\end{tikzpicture}
```



24.34 Apollonius circle v2

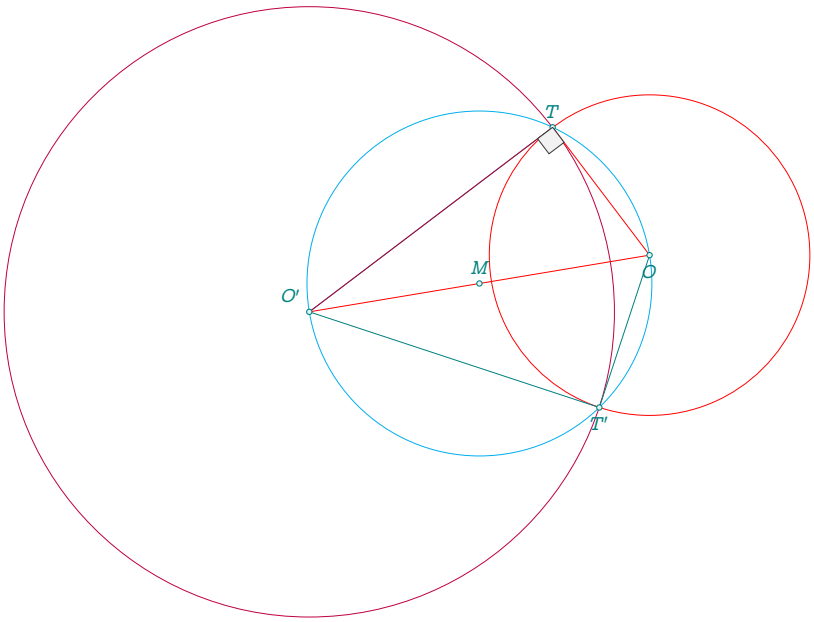
```
\directlua{%
init_elements ()
   scale       = .5
   z.A         = point: new (0,0)
   z.B         = point: new (6,0)
   z.C         = point: new (0.8,4)
   T.ABC       = triangle: new(z.A,z.B,z.C)
   z.O         = T.ABC.circumcenter
   z.H         = T.ABC.orthocenter
   z.G         = T.ABC.centroid
   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
}

\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}
```



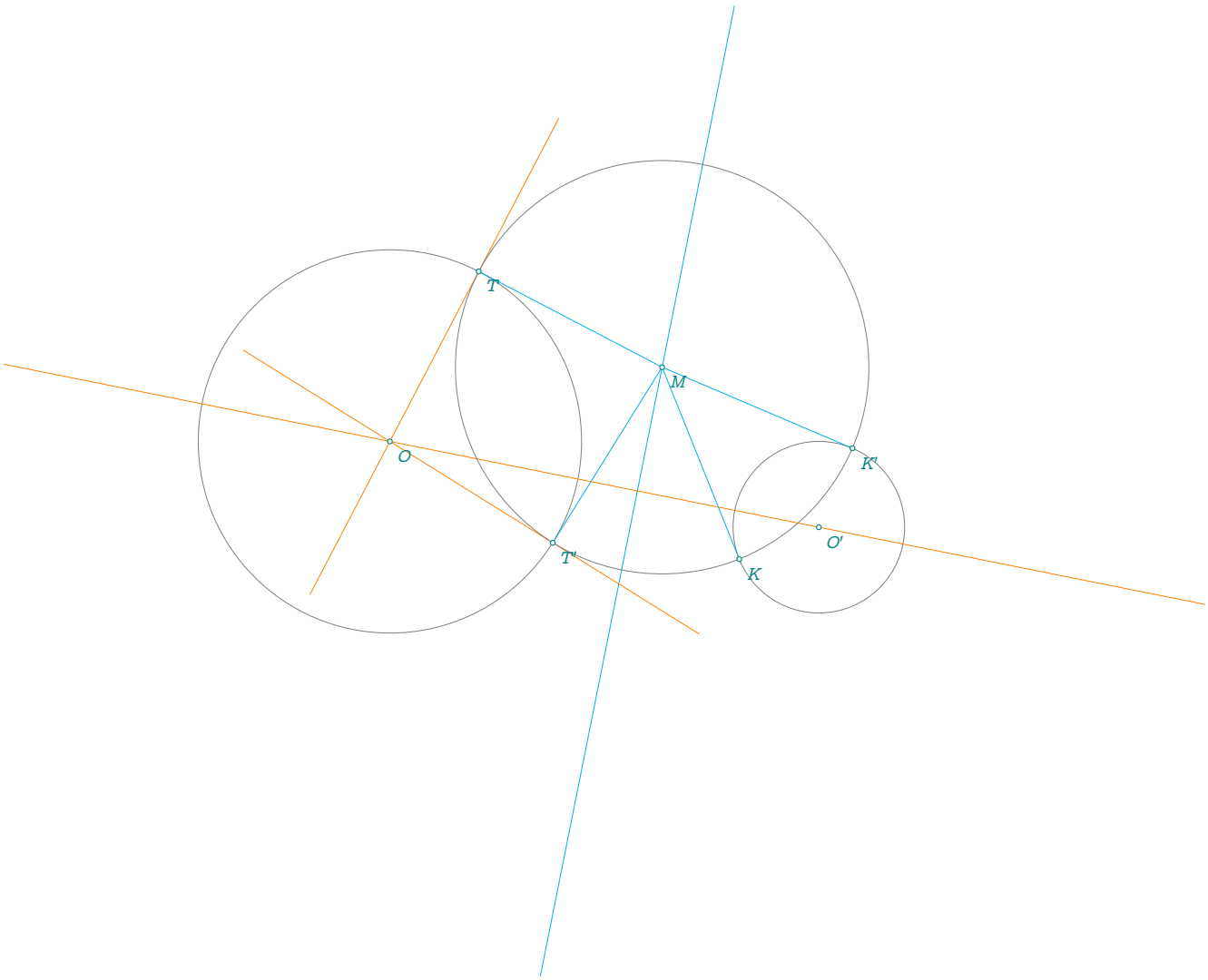
24.35 Orthogonal circles

```
\directlua{%
init_elements ()
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
}
\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}
```

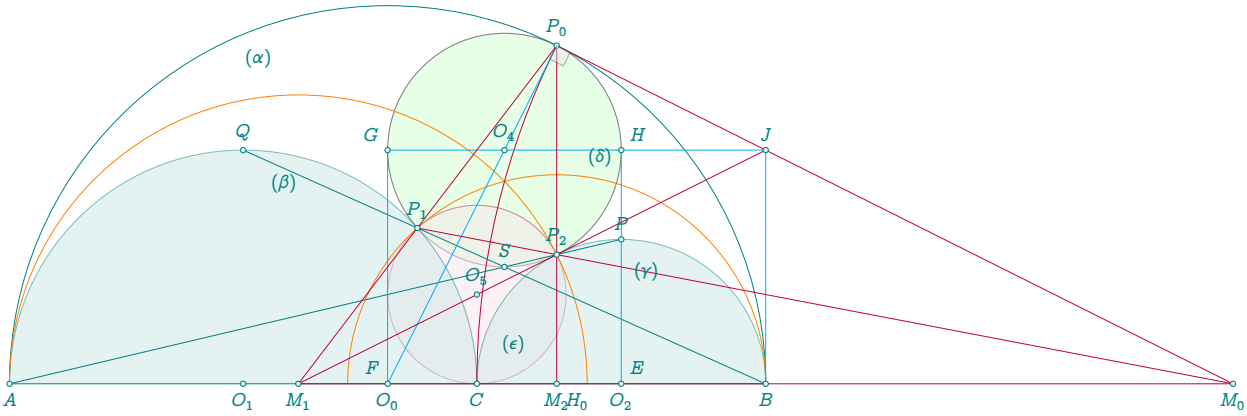


24.36 Orthogonal circle to two circles

```
\directlua{%
init_elements ()
   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
}
\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}
```



24.37 Midcircles



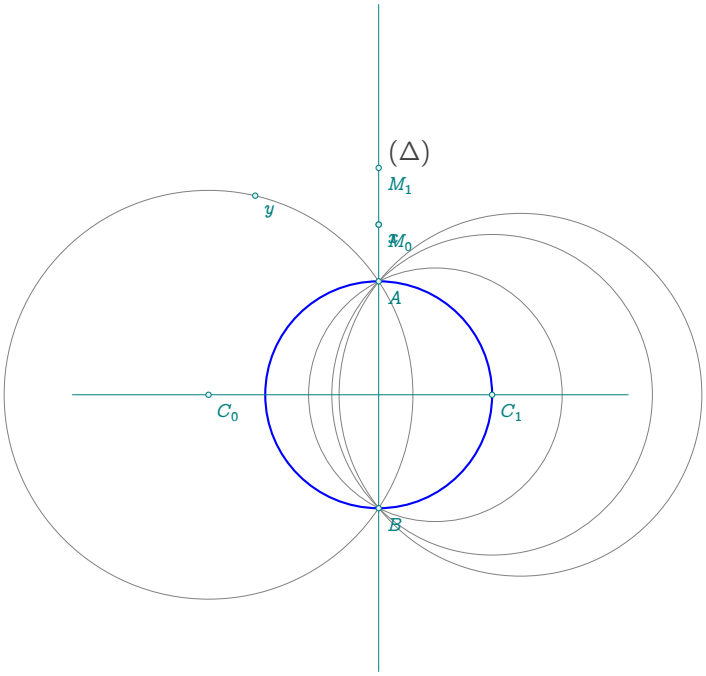

```
\directlua{%
init_elements ()
   z.A      = point: new (0 , 0)
   z.B      = point: new (10 , 0)
```

```
   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_Q    = 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.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.OQO1   = line : new (z.O_Q,z.O_1)
   L.OQO2   = line : new (z.O_Q,z.O_2)
   z.M_Q    = L.O1O2 : harmonic_ext (z.C)
   z.M_1    = L.OQO1 : harmonic_int (z.A)
   z.M_2    = L.OQO2 : 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_Q    = intersection (L.CS,C.OQB)
   z.P_1    = intersection (C.M2B,C.O1C)
   z.P_2    = intersection (C.M1A,C.O2C)
   T.PQ12   = triangle : new (z.P_Q,z.P_1,z.P_2)
   z.O_4    = T.PQ12.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.APQ    = line : new (z.A,z.P_Q)
   L.BPQ    = line : new (z.B,z.P_Q)
   C.O4PQ   = circle : new (z.O_4,z.P_Q)
   _,z.G    = intersection (L.APQ,C.O4PQ)
   z.H      = intersection (L.BPQ,C.O4PQ)
   z.Ap     = z.M_1: symmetry (z.A)
   z.H_4,z.F,z.E,z.H_Q = L.AB : projection (z.O_4,z.G,z.H,z.P_Q)
}

\begin{tikzpicture}
\tkzGetNodes
\tkzDrawCircle[thin,fill=green!10](O_4,P_Q)
\tkzDrawCircle[purple,fill=purple!10,opacity=.5](O_5,C)
\tkzDrawSemiCircles[teal](O_Q,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_Q,P_Q)(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}
```

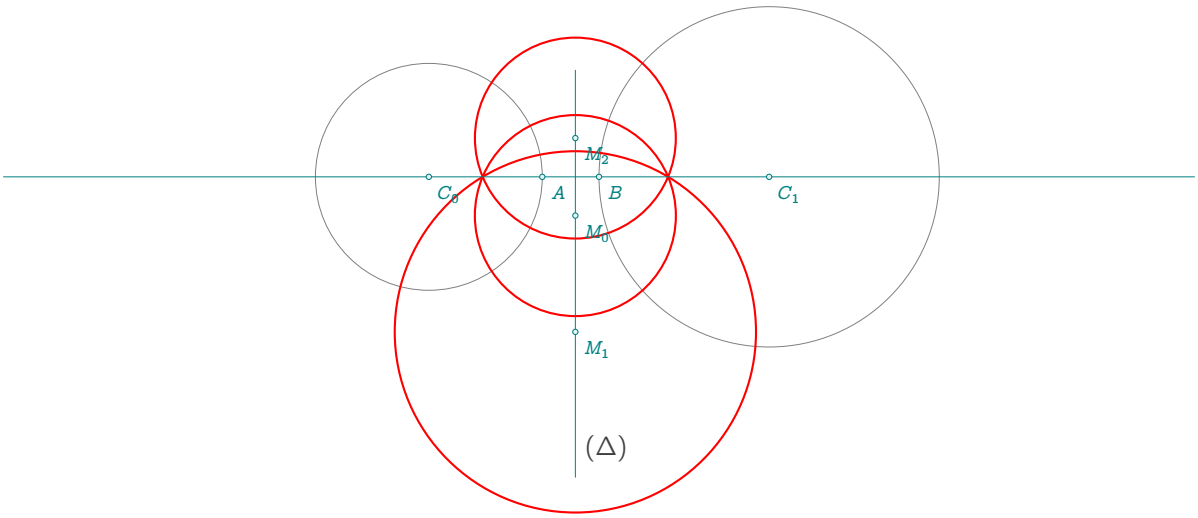
24.38 Pencil v1

```
\directlua{%
init_elements ()
   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
}
\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}
```



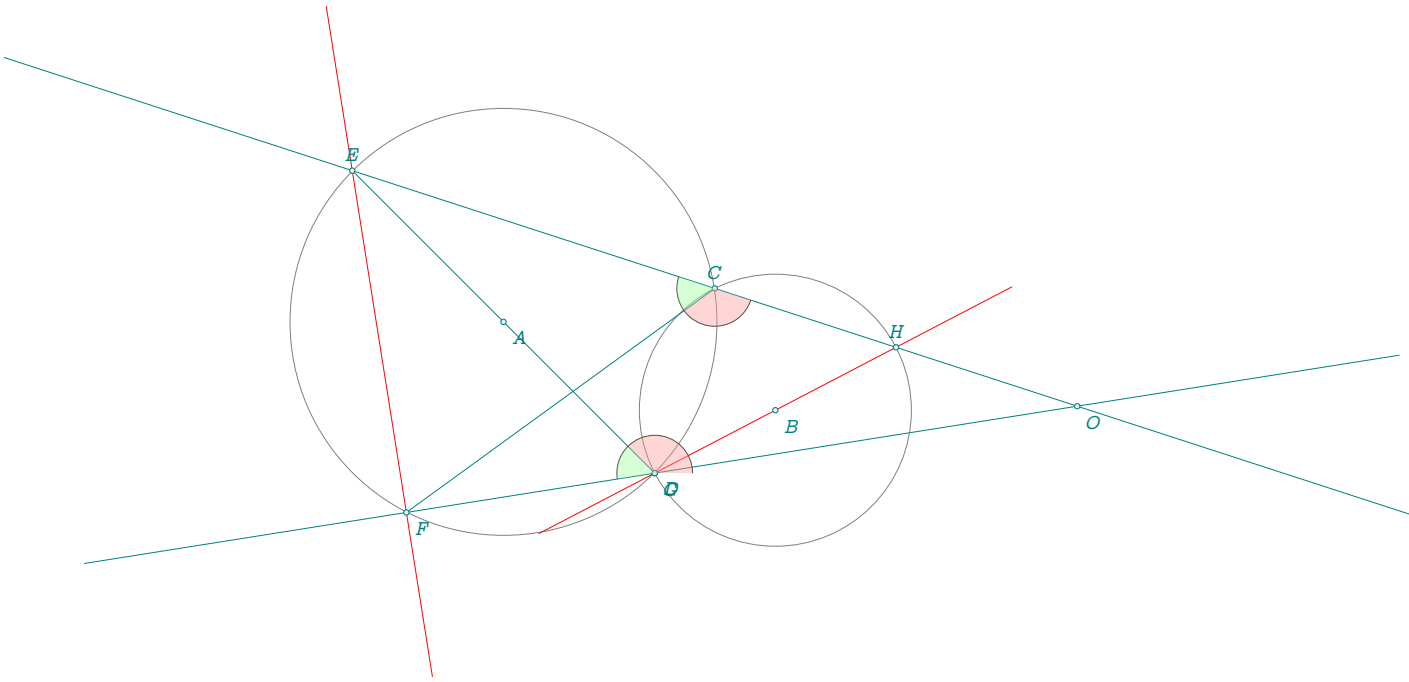
24.39 Pencil v2

```
\directlua{%
init_elements ()
   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
}
\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}
```



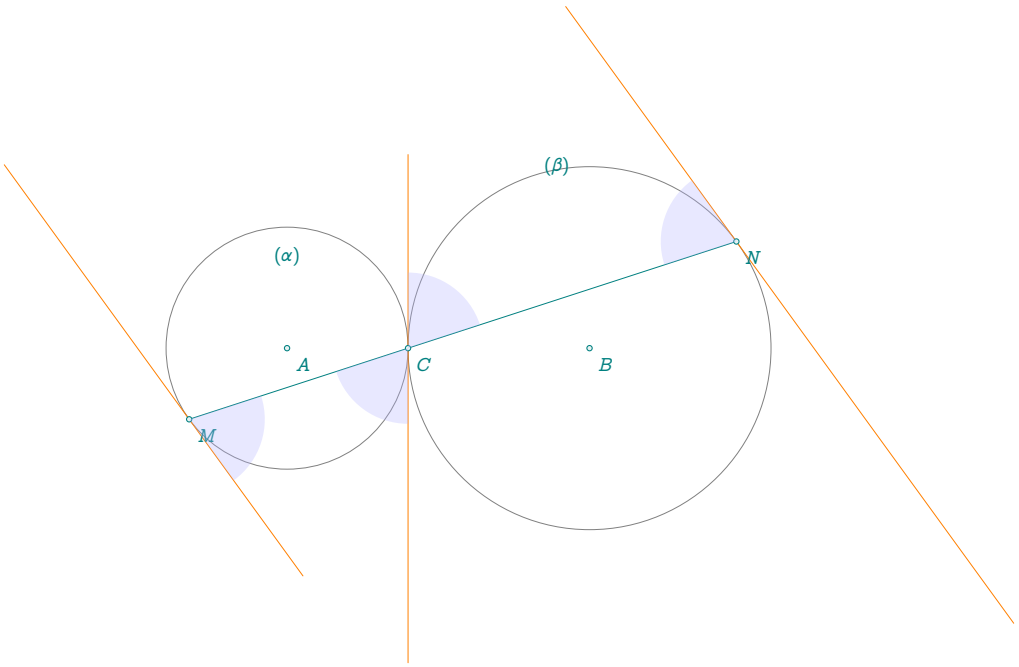
24.4Q Reim v1

```
\directlua{%
init_elements ()
   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)
}
\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}
```



24.41 Reim v2

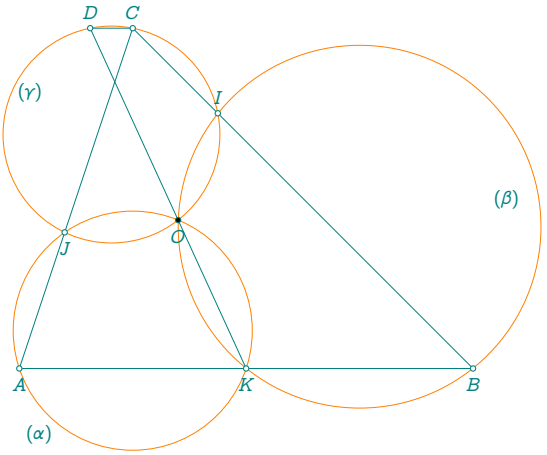
```
\directlua{%
init_elements ()
   scale    = .6
   z.A      = point: new (0,0)
   z.B      = point: new (10,0)
   z.C      = point: new (4,0)
   C.AC     = circle: new (z.A,z.C)
   z.c,z.cp = get_points (C.AC: tangent_at (z.C))
   z.M      = C.AC: point (0.6)
   L.MC     = line: new (z.M,z.C)
   C.BC     = 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))
}
\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}
```



24.42 Reim v3

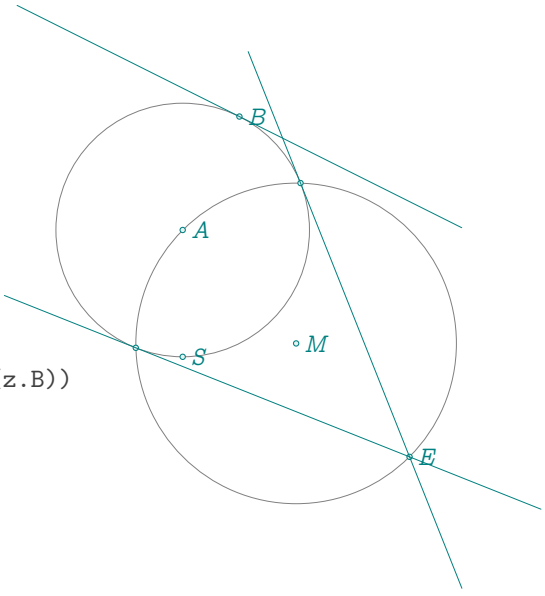
```
\directlua{%
init_elements ()
   z.A     = point: new (0,0)
   z.B     = point: new (8,0)
   z.C     = point: new (2,6)
   L.AB    = line : new (z.A,z.B)
   L.AC    = line : new (z.A,z.C)
   L.BC    = line : new (z.B,z.C)
   z.I     = L.BC : point (0.75)
   z.J     = L.AC : point (0.4)
   z.K     = 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)
}

\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}
```



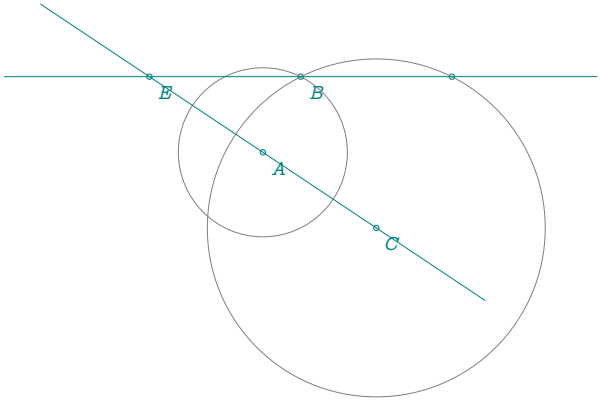
24.43 Tangent and circle

```
\directlua{%
init_elements ()
scale = .75
   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))
}
\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}
```



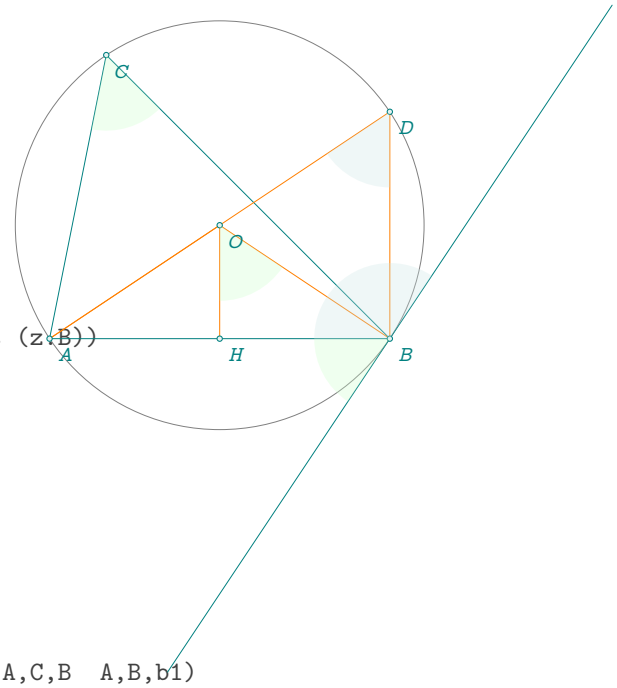
24.44 Homothety

```
\directlua{%
init_elements ()
scale = .5
   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)
}
\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}
```



24.45 Tangent and chord

```
\directlua{%
init_elements ()
   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)
}
\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}
```

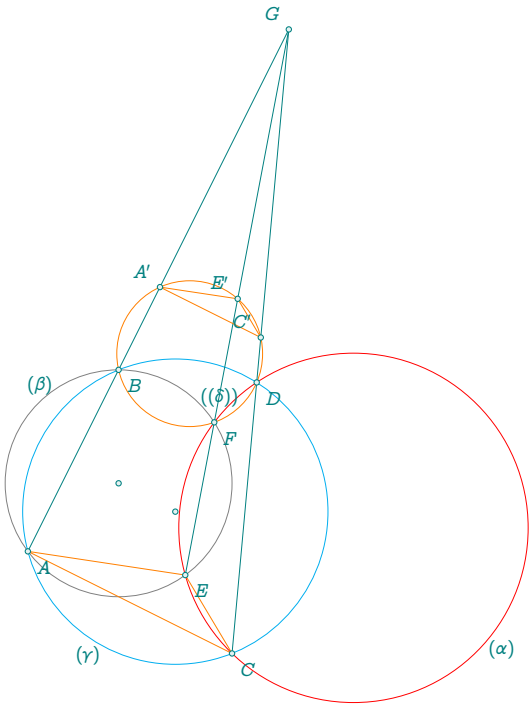


24.46 Three chords

```
\directlua{%
init_elements ()
z.O  = point: new (0 , 0)
z.B  = point: new (0 , 2)
z.P  = point: new (1 , -.5)
```

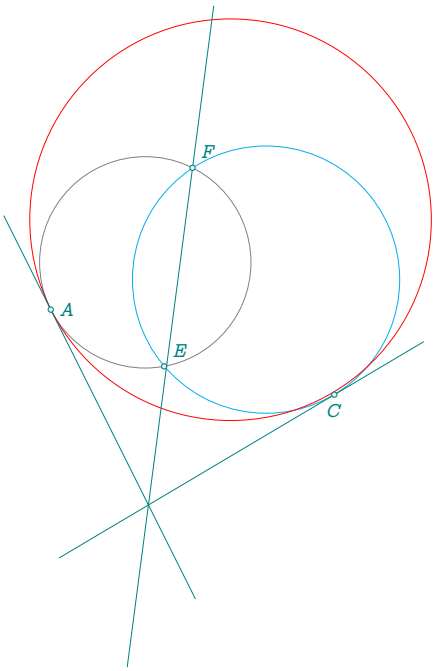
```
C.OB = circle : new (z.O,z.B)
C.PB = circle : new (z.P,z.B)
_,z.A    = intersection (C.OB,C.PB)
z.D  = C.PB: point(0.85)
z.C  = C.PB: point(0.5)
z.E  = C.OB: point(0.6)
L.AB = line : new (z.A,z.B)
L.CD = line : new (z.C,z.D)
z.G  = intersection (L.AB,L.CD)
L.GE = line : new (z.G,z.E)
z.F,_    = intersection (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)
}
```

```
\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}
```



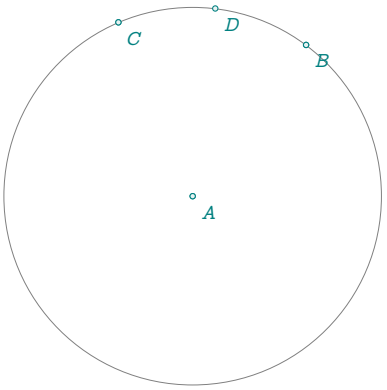
24.47 Three tangents

```
\directlua{%
init_elements ()
   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)
}
\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}
```



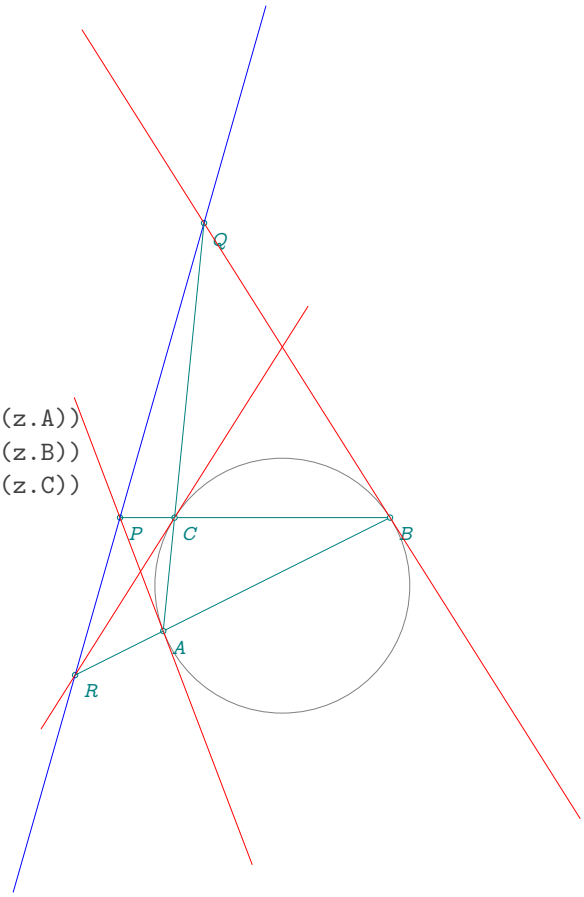
24.48 Midarc

```
\directlua{%
init_elements ()
   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)
}
\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawPoints(A,B,C)
   \tkzDrawCircles(A,B)
   \tkzDrawPoints(A,...,D)
   \tkzLabelPoints(A,...,D)
\end{tikzpicture}
```



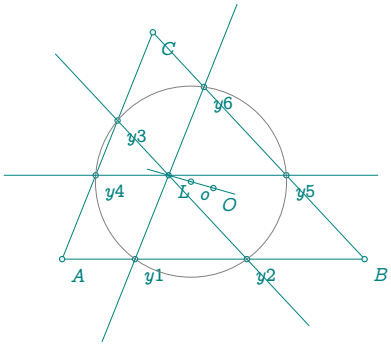
24.49 Lemoine Line without macro

```
\directlua{%
init_elements ()
   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)
}
\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}
```



24.50 First Lemoine circle

Draw lines through the symmedian point L and parallel to the sides of the triangle. The points where the parallel lines intersect the sides of the triangle then lie on a circle known as the first Lemoine circle. It has center at the Brocard midpoint, i.e., the midpoint of $[OL]$, where O is the circumcenter and K is the symmedian point [Weisstein, Eric W. "First Lemoine Circle." From MathWorld–A Wolfram Web Resource.]



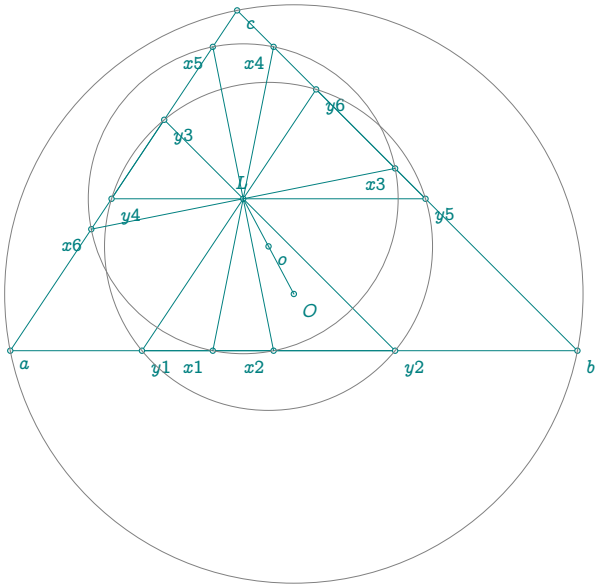
```
\directlua{%
init_elements ()
   z.A      = point:  new (1,1)
   z.B      = point:  new (5,1)
   z.C      = point:  new (2.2,4)
   T        = triangle: new (z.A,z.B,z.C)
   z.O = T.circumcenter
   C.first_lemoine = T:first_lemoine_circle()
  z.o,z.w  = get_points( C.first_lemoine )
   z.y1,z.y2         = intersection (T.ab,C.first_lemoine)
   z.y5,z.y6         = intersection (T.bc,C.first_lemoine)
   z.y3,z.y4         = intersection (T.ca,C.first_lemoine)
   z.L      = T : lemoine_point ()
}
\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawPolygons(A,B,C)
   \tkzDrawPoints(A,B,C,o,O,L,y1,y2,y3,y4,y5,y6)
   \tkzLabelPoints(A,B,C,o,O,L,y1,y2,y3,y4,y5,y6)
   \tkzDrawCircles(o,w)
   \tkzDrawLines(y1,y6 y5,y4 y2,y3 O,L)
\end{tikzpicture}
```

24.51 First and second Lemoine circles

Draw antiparallels through the symmedian point L. The points where these lines intersect the sides then lie on a circle, known as the cosine circle (or sometimes the second Lemoine circle). Refer to [24.53]
[Weisstein, Eric W. "Cosine Circle." From MathWorld–A Wolfram Web Resource.]

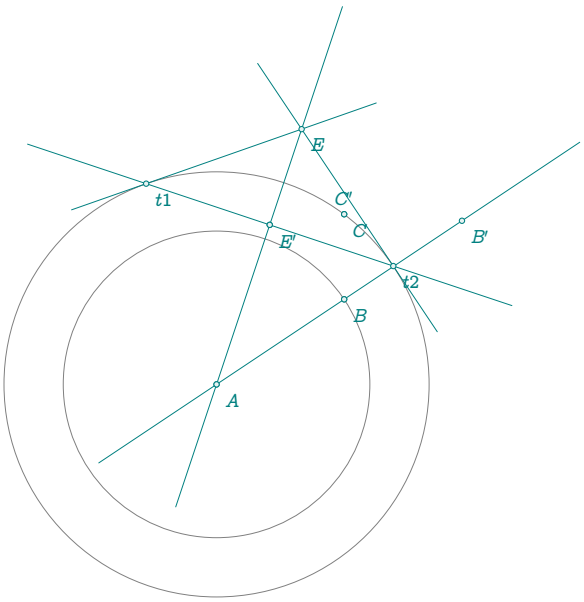
```
\directlua{%
init_elements ()
   scale             = 2
   z.a               = point:  new (0,0)
   z.b               = point:  new (5,0)
   z.c               = point:  new (2,3)
   T                 = triangle: new (z.a,z.b,z.c)
   z.O               = T.circumcenter
   z.o,z.p           = get_points (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)
```

```
}
\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}
```



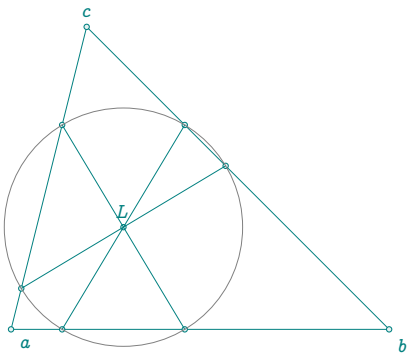
24.52 Inversion

```
 \directlua{%
init_elements ()
  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 )
}

\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}
```



24.53 Antiparallel through Lemoine point

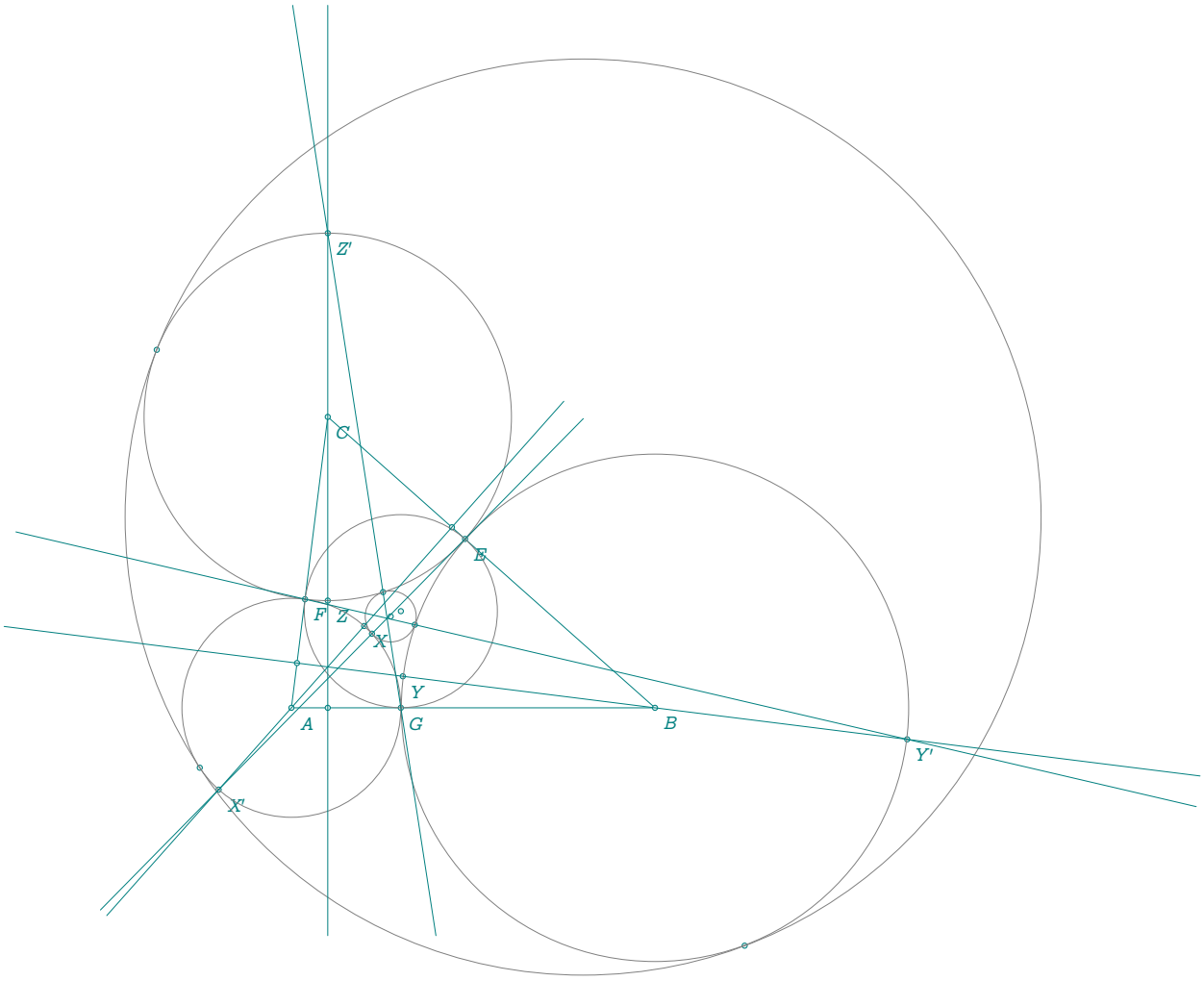
```
\directlua{%
init_elements ()
   z.a         = point:  new (0,0)
   z.b         = point:  new (5,0)
   z.c         = point:  new (1,4)
   T           = triangle: new (z.a,z.b,z.c)
   z.L         = T : lemoine_point ()
   L.anti      = T : antiparallel (z.L,0)
   z.x_0,z.x_1 = get_points (L.anti)
   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)
}

\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}
```



24.54 Soddy circle without function

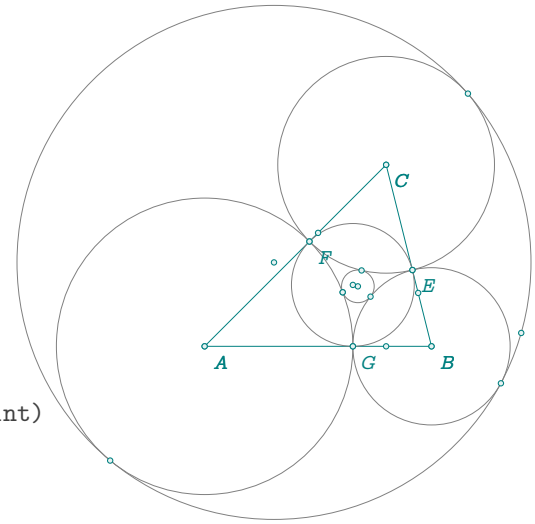
```
\directlua{%
init_elements ()
z.A = point : new ( 0  , 0  )
z.B = point : new ( 5  , 0  )
z.C = point : new ( 0.5 ,  4  )
T.ABC = triangle : new ( z.A,z.B,z.C )
z.I = T.ABC.incenter
z.E,z.F,z.G = T.ABC : projection (z.I)
C.ins = circle : new (z.I,z.E)
T.orthic = T.ABC : orthic ()
z.Ha,z.Hb,z.Hc = 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)
}

\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}
```



24.55 Soddy circle with function

```
  \directlua{%
init_elements ()
  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)
  }
\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)
\end{tikzpicture}
```

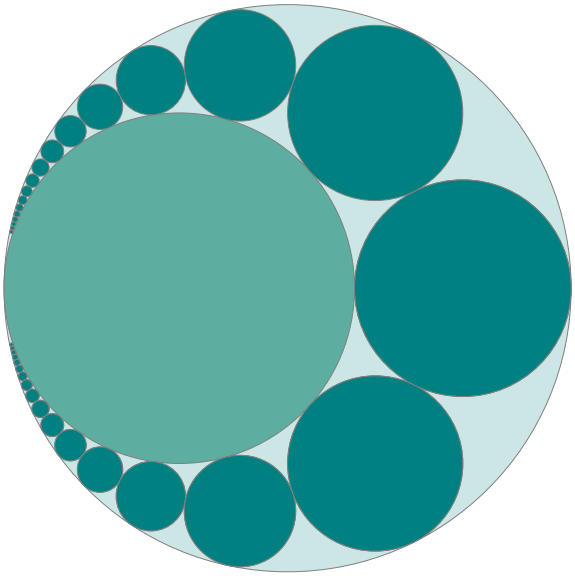


24.56 Pappus chain

```
\directlua{%
init_elements ()
  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)
```

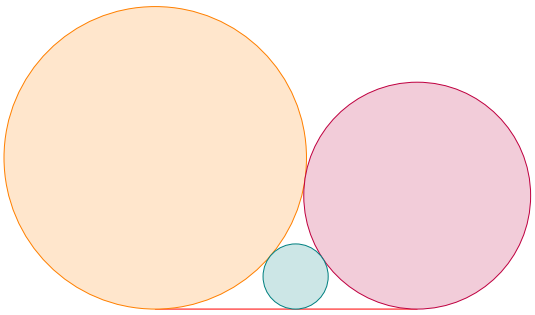
```
     z["J"..i.."p"], z["H"..i.."p"]   = C.AC : inversion (z["J"..i],z["H"..i])
     L.AJ        = line : new (z.A,z["J"..i])
     C.JH        = circle: new ( z["J"..i] ,  z["H"..i])
     z["S"..i], z["T"..i]             = intersection (L.AJ,C.JH)
     z["S"..i.."p"], z["T"..i.."p"]   = C.AC : inversion (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
}
```

```
\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}
```



24.57 Three Circles

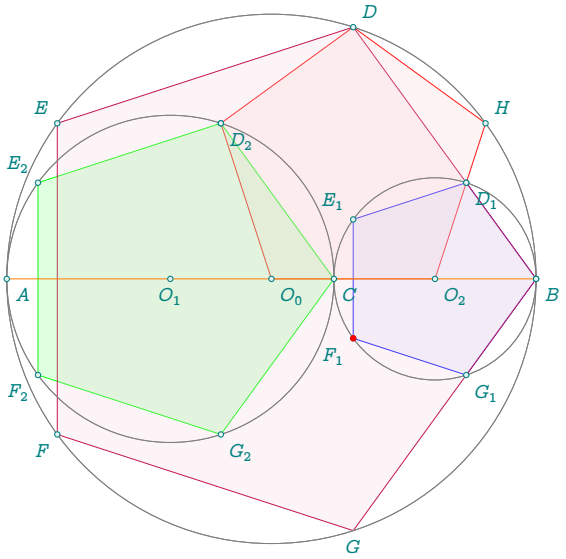
```
\directlua{%
init_elements ()
function threecircles(c1,r1,c2,r2,c3,h1,h3,h2)
   local xk = math.sqrt (r1*r2)
   local cx = (2*r1*math.sqrt(r2))/(math.sqrt(r1)+math.sqrt(r2))
   local cy = (r1*r2)/(math.sqrt(r1)+math.sqrt(r2))^2
   z[c2] = point : new ( 2*xk , r2 )
   z[h2] = point : new (2*xk,0)
   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")
}
```



```
\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}
```

24.58 p Pentagons in a golden arbelos

```
\directlua{%
init_elements ()
  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.O2B    = circle: new ( z.O_2, z.B)
  z.M_0    = C.O1C:external_similitude(C.O2B)
  L.O0C    = line:new(z.O_0,z.C)
  T.golden = L.O0C : golden ()
  z.L      = T.golden.pc
  L.O0L    = line:new(z.O_0,z.L)
  z.D      = intersection (L.O0L,C.O0B)
  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.O0B)
  C.BD     = circle:new (z.B,z.D)
  C.DB     = circle:new (z.D,z.B)
  _,z.G    = intersection (C.BD,C.O0B)
  z.E      = intersection (C.DB,C.O0B)
  C.GB     = circle:new (z.G,z.B)
  _,z.F    = intersection (C.GB,C.O0B)
  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_0 : homothety (kk,z.D_1,z.E_1,z.F_1,z.G_1)
}
```




```
\begin{tikzpicture}[scale=.8]
\tkzGetNodes
\tkzDrawPolygon[red](O_2,O_0,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_0,B O_1,C O_2,B)
\tkzFillPolygon[fill=red!20,opacity=.20](O_2,O_0,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_0,B O_1,C O_2,B)
\tkzDrawSegments[new](A,B)
\tkzDrawPoints(A,B,C,O_0,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_0,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}
```

Index

25 Cheat_sheet

r denotes a real number, cx complex number, d a positive real number, n an integer, an an angle, b a boolean, s a character string, pt a point, t a table, m a matrix, 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, M a matrix, 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

Functions table(1)

new	-> pt
polar	-> pt
polar_deg	-> pt

Methods table(29)

+ - * /	(pt,pt)	-> pt
..	(pt,pt)	-> r
^	(pt,pt)	-> r
=		-> b
tostring		-> s

Methods table(2) table(30)

conj	-> pt
abs	-> r
mod	-> d
norm	-> d
arg	-> d
get	-> r,r
sqrt	-> 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
print()	-> s

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
report (r,pt)	-> pt
colinear_at (pt,k)	-> 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

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
conway_points ()	-> pts
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
conway_circle ()             -> C
pedal_circle ()              -> C
cevian_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
pedal (pt)                   -> 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)

```
type                         -> s
norm                         -> d
slope                        -> r
mtx                          -> M
```

Methods table(24)

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

matrix

Attributes table(25)

```
set                      ->  t
rows                     ->  n
cols                     ->  n
type                     ->  s
det                      ->  r
```

Functions table(27)

```
new                      ->  m
square                   ->  m
htm                      ->  m
vector                   ->  m
```

Metamethods table(26)

```
+ - *  (m,m)              -> m
^ (m,n)                  -> m
=                        -> b
tostring                 -> s
```

Method table(27)

```
print                    -> s
get                      -> r/cx
inverse                  -> m
adjugate                 -> m
transpose                -> m
is_diagonal              -> b
is_orthogonal            -> b
homogenization           -> m
htm_apply                -> m
```

Misc.

Attributes table(28)

```
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
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
solve_quadratic(cx,cx,cx) -> cx,cx
\tkzUseLua{v}            -> s
```

Macros

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