Euclidean geometry with tkz-elements and tkz-euclide

Alain Matthes

Abstract

tkz-elements [2][1] is based on Lua and LuaLATEX to perform calculations and obtain point coordinates in the plane. These coordinates are then transmitted to a package that can plot them. Currently, plotting is accomplished with Ti*k*Z or tkz-euclide, but MetaPost is also a viable option.

This paper demonstrates how tkz-elements can be utilized for tasks requiring mathematical computations. With it, not only can you create Euclidean geometry figures, but you can also conduct calculations within your document.

1 Introduction

The aim of the tkz-euclide [3] package is to provide a tool that would facilitate the construction of Euclidean geometric figures, with a key focus on being suitable for individuals who think mathematically, and even better, geometrically. tkz-euclide is built on top of PGF and its associated front-end Ti*k*Z. As a result, the calculations rely on TEX. To aid TEX in performing certain calculations, auxiliary packages are necessary. However, this approach can be challenging to program, slow in execution, and sometimes lacks accuracy.

An extension of TEX, LuaTEX, has been developed, enriching TEX with the programming language Lua, which is fast, light and easy to program. tkz-elements is an attempt to use Lua's capabilities to enhance tkz-euclide.

The final section of this paper explains the basics of drawing objects with tkz-euclide.

2 What are the foundations of tkz-elements?

2.1 Structure

The package mainly comprises two environments: the tkzelements environment and the tikzpicture environment. The former utilizes Lua-created functions to acquire point coordinates, while the latter employs tkz-euclide to draw figures. I have a preference for tkz-euclide, as it includes all the fundamental figures.

An important aspect is the relationship between the two environments. The coordinates of the points are stored in the only data structure available in Lua: a table z (z being a common reference to the affixes of complex numbers). This table is global, and its data is only cleared when a new tkzelements environment is initiated. At the start of the tikzpicture environment, the tkzGetNodes macro retrieves the coordinates and generates nodes whose names are those of the z table keys.[2]

Following the tkzelements environment, you can obtain results that can be incorporated into your document (an advantage of a figure source within your document), by using the \tkzUseLua command. The definition of this macro is
\directlua{tex.print(tostring(#1))}.

Let's look at the following example:[3]

```
% !TEX TS-program = lualatex
\documentclass{article}
\usepackage{tkz-euclide,tkz-elements}
\begin{document}
\begin{tkzelements} -- part elements
  z.A   = point : new (1,1)
  z.B   = point : new (3,2)
  C.AB  = circle : new (z.A,z.B)
  z.C   = C.AB : point (1/6)
  T.ABC = triangle : new (z.A,z.B,z.C)
\end{tkzelements}
\begin{tikzpicture}% part tikz
  \tkzGetNodes
  \tkzDrawPolygon(A,B,C)
  \tkzDrawCircle(A,B)
  \tkzDrawPoints(A,B,C)
  \tkzLabelPoints(A)
  \tkzLabelPoints[above](B,C)
\end{tikzpicture}

The length of AC is \tkzDN[4]{\tkzUseLua{%
                      length(z.C,z.A)}}
  Affix of $C$: \tkzUseLua{z.C}
\end{document}
```

Figure 1: Sample program

The result is shown in fig. 2.

The macro tkzDN serves as a formatting tool for numerical results.

Now, let's consider whether the triangle is equilateral. If the ifthen package has been loaded, this can be done with:

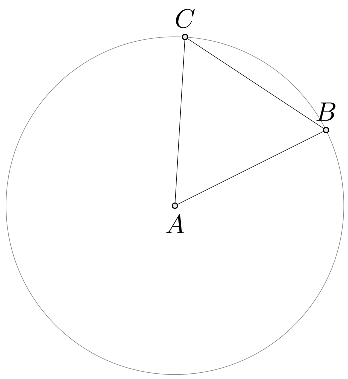
```
\ifthenelse{\equal{\tkzUseLua{%
  T.ABC : check_equilateral ()}}{true}}{%
  The triangle ABC is equilateral}{%
  The triangle ABC is not equilateral}
```

[1] The current version is 2.00 and is required to compile the examples in this paper.

[2] The table type implements associative arrays. An associative array is an array that can be indexed with numbers, strings, or any other value; that is, they store a set of key/value pairs.

[3] "-- part elements" is a comment in Lua; "% part tikz" is a comment in LATEX.

doi.org/10.47397/tb/45-1/tb139matthes-euclid



The length of AC is 2.2361
Affix of C: 1.13+3.23i

Figure 2: Result of sample program fig-1

which, for our example, outputs:
The triangle ABC is equilateral

2.2 Tools

2.2.1 Complex numbers

Our primary aim was precision in calculations, and since programming with Lua is much easier than in TEX, we considered utilizing mathematical tools better suited to geometry instead of basic arithmetic operations like addition and subtraction. The initial concept was to incorporate complex numbers.

A complex number, denoted as z, can be represented by an ordered pair $(\mathcal{R}e(z), \mathcal{I}m(z))$ of real numbers, which can be interpreted as coordinates of a point in a two-dimensional space such as the Euclidean plane. This plane is commonly referred to as the complex plane or the Argand plane (Fig. 4). To create a point object, we specify its two coordinates and its name (future node name); for example: `z.A= point : new (2,3)`. What happens here? An object of type `point` is created, consisting of attributes and methods stored in the table (associative array) `z`.

The key `A` is associated with the data. The `tostring` method has been adapted to display the affix corresponding to the point. That is,
`tex.print(tostring(z.A))` outputs `2+3i`.

Point objects behave similarly to the affixes that represent them. Hence, we can manipulate them with the same operations. Here's an example: adding two points means obtaining another point whose affix is the sum of the affixes of the previous points.

Let's consider a second point:
`z.B = point : new (2,-1)`
Then `z.C = z.A+z.B` has affix `4+2i`; analogously, `z.D = z.A*z.B` has affix `7+4i`.

Let's check: `\tkzUseLua{z.A*z.B}` computes: 7+4.00i.

Refer to the documentation for a comprehensive list of all methods available. Some are more significant than others, one being the complex conjugate: `z.B = z.A : conj()`, which can alternatively be expressed as `z.B = point.conj (z.A)`.

It's important to note that two operations have been repurposed from their conventional meanings: "`..`", typically represents concatenation but here denotes scalar product, and "`^`", usually signifies exponentiation but here denotes the determinant.[4]
`z.A .. z.B = (z.A * z.B : conj()).re` $= 1$
`z.A ^ z.B = (z.A : conj() * z.B).im` $= -8$

2.2.2 Barycenter

Another useful tool is the barycenter, which is utilized numerous times in our diagrams. Here are two examples demonstrating the advantages of combining complex numbers and barycenters:

- Obtaining the incenter in a triangle defined by its three vertices (a,b,c):

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

  `point.abs` is a method which gives the modulus of a complex number.

- Obtaining the orthocenter:

```
function ortho_center_ (a,b,c)
  local ka = math.tan (get_angle_ (a,b,c))
  local kb = math.tan (get_angle_ (b,c,a))
  local kc = math.tan (get_angle_ (c,a,b))
  return barycenter_ ({a,ka},{b,kb},{c,kc})
end
```

  `get_angle_` is an internal macro in the package that produces a normalized angle defined by three complex numbers.

2.2.3 Objects — OOP

Finally, while the package's internal functions are classically programmed using Lua, user functions are based on object-oriented programming principles. Users manipulate points, lines, circles, triangles, etc., all of which are objects from specific classes. Currently, `tkz-elements` utilizes the following classes: point, line, circle, triangle, ellipse, quadrilateral, square, rectangle, parallelogram, regular (polygon) and vector (matrix will be added soon).

An object (or instance) of the class `point` has both state and behavior, defined by the class. The

[4] Here we consider `z.A` and `z.B` as the vectors $\overrightarrow{OA}$ and $\overrightarrow{OB}$ with O as the origin of the plane.

state is characterized by attributes, while behavior is determined by methods. The structure of the `object` class is shown in fig. 3; here, all the attributes are listed but only a few of the available methods are displayed. See [2], section "Class point" for the complete definition.[5]

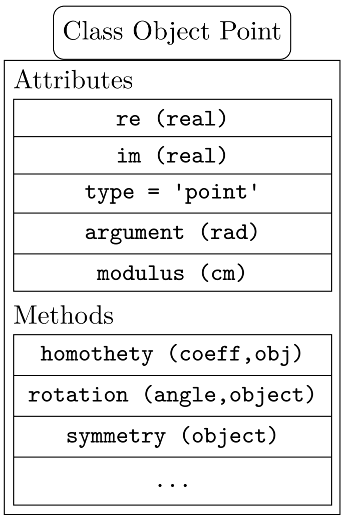


Figure 3: The Point object

We can access the instance's attributes as follows to obtain the real part (the point's abscissa): `z.A.re`.

We can already benefit from the use of LuaLaTeX. To obtain figure 4, the point A has been defined as follows `z.A = point : new (2,3)`. Therefore, we can use the attributes of this point. The modulus of z_A is 3.60555 . This value is obtained as follows: `\tkzUseLua{z.A.modulus}`

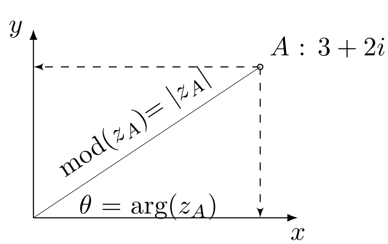


Figure 4: Argand diagram

Other classes possess their unique attributes and methods. We recommend consulting the documentation. In the remainder of this article, we'll utilize examples to elucidate specific attributes and methods. It's not feasible to cover all the documentation in this article, so we'll employ examples to illustrate certain attributes and methods. Refer to [2], sections "Class line", "Class circle", etc.

[5] It's recommended to have the package documentation at hand while reading this paper.

3 Small examples

Let's examine two brief examples. While they don't require high-precision calculations, they will demonstrate how to create a figure and utilize objects.

3.1 Alternate angles

```
\documentclass{article}
\usepackage{tkz-euclide}
\usepackage{tkz-elements}
\begin{document}
  \begin{tkzelements}
    scale = .8
     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.AD  = T.ABC : bisector ()
     z.D   = L.AD.pb
     L.LLC = T.ABC.ab : ll_from (z.C)
     z.E   = intersection (L.AD,L.LLC)
  \end{tkzelements}
\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawPolygon(A,B,C)
   \tkzDrawLine(C,E)
   \tkzDrawSegment(A,E)
   \tkzMarkAngles[mark=|](B,A,D D,A,C)
   \tkzMarkAngles[mark=|](C,E,D)
   \tkzDrawPoints(A,...,E)
   \tkzLabelPoints(A,B)
   \tkzLabelPoints[above](C,D,E)
   \tkzMarkSegments[mark=s||](A,C C,E)
\end{tikzpicture}
\end{document}
```

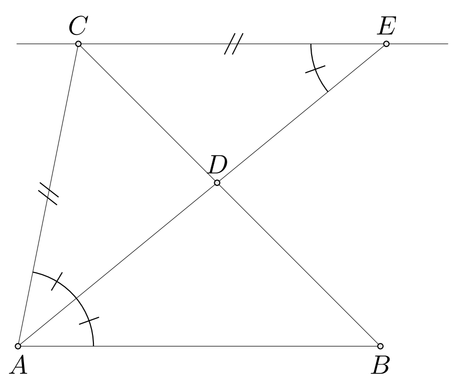


Figure 5: Alternate angles

First, we create three points, then a triangle named `T.ABC`. Subsequently, we define the bisector emanating from vertex A.

`L.AD = T.ABC : bisector ()`: The bisector is defined by two points: the vertex A and the foot D on the opposite side. For the bisector from B you need to use `L.BE = T.ABC : bisector (1)`, where 1 is for the next point of the triangle.

To find the intersection of the bisector with a line parallel to the (AB) line at C, we'd have to name this line, but this is already done in the triangle's attributes: `T.ABC.ab` represents the triangle line defined by the first and second vertices. Finally, `z.E = intersection (L.AD,L.LLC)` gives the last point.

The `tkz-euclide` section gives an overview of the possibilities the package provides to mark segments and angles.

3.2 An Apollonius circle

Given k a positive real number other than 1, and A and B two points in the plane, the set of points M verifying $MA/MB = k$ is a so-called Apollonius circle. In the example below, k is defined by $k = EA/EB$.

```
\begin{tkzelements}
  scale = .5
  z.A   = point : new (0,0)
  z.B   = point : new (6.5,0)
  z.E   = point : new (7,4)
  T.EAB = triangle : new (z.E,z.A,z.B)
  EA    = length (z.E,z.A)
  EB    = length (z.E,z.B)
  C.OE  = T.EAB.bc : apollonius (EA/EB)
  L.bis = T.EAB : bisector ()
  z.C   = L.bis.pb
  z.O   = C.OE.center
  z.D   = C.OE : antipode (z.C)
  z.F   = T.EAB.ab : point (-0.5)
\end{tkzelements}
```

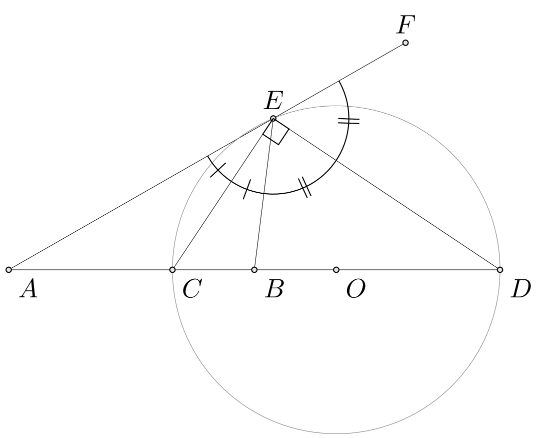


Figure 6: Apollonius $MA/MB = k$

- We define the triangle after defining three points: `T.EAB = triangle : new (z.E,z.A,z.B)`
- The length EA is determined with `length(z.E,z.A)`
- `T.EAB.bc` represents the straight line (AB) b for the second point and c for the third. Find the circle defined by these two points and the ratio EA/EB. It is called `C.EC` because its center will be O and it passes through E.
- We get the circle with `T.EAB.bc : apollonius (EA/EB)`
- Next, we look for the bisector of the angle $\widehat{AEB}$. It intersects the opposite side at point C of $C_{(O,E)}$.

  In `T.EAB : bisector ()`, the first point designates the vertex. The bisector is defined by the vertex and the intersection with the opposite side; `L.bis.pb` designates the second point.
- In `z.O = C.OE.center`, `center` is a circle attribute, then the "antipode" method is used to obtain the diametrically opposite point `z.D   = C.OE : antipode (z.C)`.
- Finally we need a point F to mark an angle in `tkz-euclide`.

4 Harmonic mean of two numbers

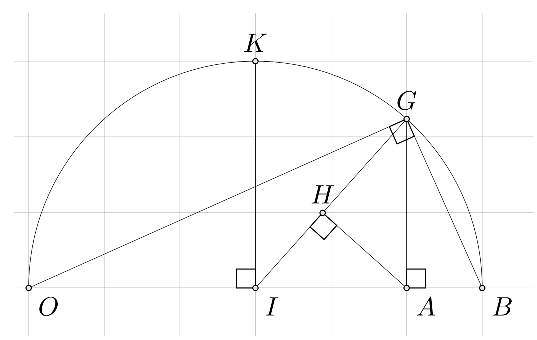


Figure 7: Means of two numbers

For two numbers a and b, such as $OA = a$ and $AB = b$, here are the definitions and geometric representations of three means:

Mean	Definition	Segment
Arithmetic	$\mathcal{A} = \dfrac{a+b}{2}$	IK
Geometric	$\mathcal{G} = \sqrt{ab}$	AG
Harmonic	$\mathcal{H} = \dfrac{2ab}{a+b} = \dfrac{\mathcal{G}^2}{\mathcal{A}}$	HG

```
\begin{tkzelements}
  local a = 5
  local b = 1
  z.O    = point : new (0,0)
  z.A    = point : new (a,0)
  z.B    = z.A + b
  L.OB   = line : new (z.O,z.B)
```

```
  z.I    = L.OB.mid
  C.IO   = circle : new (z.I,z.O)
  L.orth = L.OB : ortho_from (z.A)
  z.K    = C.IO.north
  z.G,z.Gp = intersection (L.orth,C.IO)
  L.IG   = line : new (z.I,z.G)
  z.H    = L.IG : projection (z.A)
\end{tkzelements}
```

Tracing with `tkz-euclide`:

```
\begin{tikzpicture}[gridded]
  \tkzGetNodes
  \tkzDrawSegments(I,G A,H O,B)
  \tkzDrawSegments(O,G G,B I,K A,G)
  \tkzDrawArc(I,B)(O)
  \tkzLabelPoints[below right](O,A,B,I)
  \tkzLabelPoints[above](H,K,G)
  \tkzMarkRightAngles(O,I,K B,A,G)
  \tkzMarkRightAngles(A,H,I O,G,B)
  \tkzDrawPoints(O,A,B,G,K,H,I)
\end{tikzpicture}
```

Some explanations:

- `z.B = z.A + b`
  Adding points means adding their corresponding affixes. `z.A` is represented in this equation by the affix, so it's possible to add a real or complex number to it. We have $OB = a + b$.
- `L.OB = line : new  z.O,z.B)`: create a `line` object with key `OB`. Then, in `z.I = L.OB.mid`, `mid` is an attribute of the `line` object giving the midpoint of the segment defined by the two points characterizing the line.
- `C.IO = circle : new (z.I,z.O)`: create a `circle` object with key `IO`.
- In `z.K = C.IO.north`, the `north` attribute of a circle is used.
- This is followed by an intersection: `intersection (L.orth,C.IO)`
  The arguments are objects, given in no particular order. Depending on the object types, the function selects the correct algorithm.

  The two points of intersection will be G and G' (`Gp` in Lua for the moment).
- `projection` is a method of the `line` object.

Let's check some data:

- The coordinates of G are (5 ; 2.2361) with `\tkzUseLua{z.G.re}` ; `\tkzUseLua{z.G.im}`
- The coordinates of H are (3.8889 ; 0.9938)
- The harmonic mean is the length of GH = 2.2361 , i.e., $\sqrt{5}$ with `\tkzUseLua{length(z.G,z.A)}`

```
\begin{tkzelements}
  scale =.8
  dofile ("means_b.lua")
\end{tkzelements}
```

It's good practice to place the Lua code in an external file. This approach makes it easier to correct and reuse, and it helps avoid errors when using special characters like the % symbol.

Figure 9 illustrates how to obtain half the harmonic mean and, importantly, demonstrates that this method is independent of the distance d.

```
z.A   = point : new (0,6)
z.B   = point : new (6,4)
z.Bp  = point : new (8,4)
z.I   = point : new (0,0)
z.J   = point : new (6,0)
z.Jp  = point : new (8,0)
L.AJ  = line : new (z.A,z.J)
L.IJ  = line : new (z.I,z.J)
L.BI  = line : new (z.B,z.I)
z.C   = intersection (L.AJ,L.BI)
z.K   = L.IJ : projection (z.C)
L.AJp = line : new (z.A,z.Jp)
L.BpI = line : new (z.Bp,z.I)
z.Cp  = intersection (L.AJp,L.BpI)
z.Kp  = L.IJ : projection (z.Cp)
```

Figure 8: File `means_b.lua`

```
\begin{tikzpicture}
  \tkzSetUpPoint[size=8]
  \tkzGetNodes
  \tkzDrawSegments[dashed](A,J B,I I,J)
  \tkzDrawSegments[dashed](A,J' B',I)
  \tkzDrawPoints[gray,size = 8](A,I,C,K,B,J)
  \tkzDrawPoints[black,size = 8](C',K',B',J')
  \tkzSetUpLine[ultra thick]
  \tkzDrawSegments[black](C',K' B',J')
  \tkzDrawSegments[gray](C,K A,I B,J)
\end{tikzpicture}
```

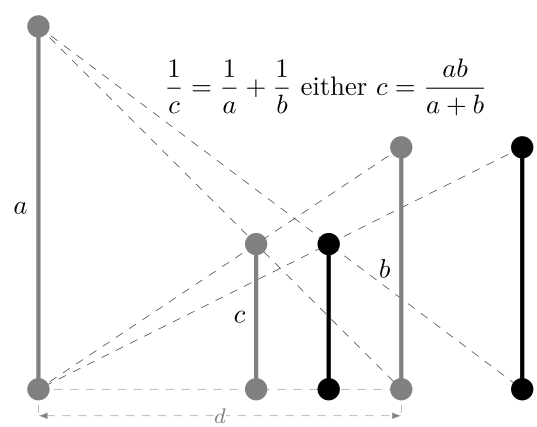


Figure 9: Half of harmonic mean

5 THE Apollonius circle

The circle that touches all three excircles of a triangle and encompasses them is commonly referred to as

"THE" Apollonius circle. Our approach is from the fourth definition given in [4], due to Kimberling [1, p. 102].

The objective here is to determine the external tangent circle to the three exinscribed circles of a triangle. While this problem is mathematically challenging, the idea is to demonstrate that the package offers some highly useful capabilities for experienced geometers.

The approach involves determining the inner tangent circle, and then transforming this inner circle into an outer circle, also tangent to the exinscribed circles. The result is shown in fig. 10.

The Lua code is created in an external file, `apollonius.lua`, shown in fig. 11.

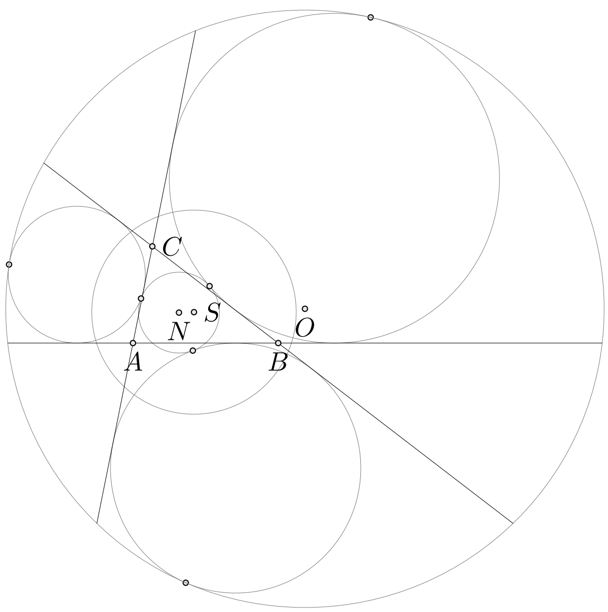


Figure 10: THE Apollonius circle

```
scale       = .32
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
T.feuerbach = T.ABC : feuerbach ()
T.excentral = T.ABC : excentral ()
z.Ea,z.Eb,z.Ec = get_points (T.feuerbach)
z.Ja,z.Jb,z.Jc = get_points (T.excentral)
z.S         = T.ABC.spiekercenter
C.JaEa      = circle : new (z.Ja,z.Ea)
r_ortho     = math.sqrt (C.JaEa : power (z.S))
C.ortho     = circle : radius (z.S,r_ortho)
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)
```

Figure 11: File `apollonius.lua`

5.1 Code analysis

- A `triangle` object is created: `T.ABC`, then we utilize its attributes and methods linked to the `triangle` class.
- For example, `z.N` refers to the Euler center or the center of the nine-point circle. Additionally, `T.feuerbach` is a triangle created using a method. Its vertices are the points of contact of the Euler circle with the exinscribed circles.
- Then, to draw them, we'll need the points that define the vertices of `T.feuerbach`. This is the role of `get_points (T.feuerbach)`.

  `get_points` is a function that retrieves the points (attributes) required to create the object. In this case, these are the vertices of the triangle Ea, Eb, Ec. The circle with center N passes through these points.
- The same procedure is used to recover the centers of the exinscribed circles (Ja, Jb, Jc).
- On a more technical note, the radical axes of the three exinscribed circles intersect at a point called the "radical center", which is none other than the Spieker center. This point is known to the package as one of the attributes of the triangle: `z.S = T.ABC.spiekercenter`.

  It's also possible to directly request the radical center. The radical center has the same power with respect to the three circles. This allows for determining the radius of a circle that will be orthogonal to the three exinscribed circles. The radius is
  `r_ortho = math.sqrt (C.JaEa : power (z.S))`.

  Calculate the power of point S with respect to one of the three circles, then take the square root of the result.
- The circle "ortho" can be defined as
  `C.ortho = circle : radius(z.S,r_ortho)`.

  All that remains is to utilize this circle to perform an inversion of the Euler circle, which will give the Apollonius circle
  `C.apo = C.ortho : inversion(C.euler)`.

  We then retrieve the center
  `z.O = C.apo.center` (center is an attribute for a circle) and the three points of contact with the exinscribed circles. These are images of the inverted contact points of the nine-point circle or Euler circle.

```
\begin{tikzpicture}
  \tkzGetNodes
  \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)
  \tkzLabelPoints(O,N,A,B)
  \tkzLabelPoints[right](S,C)
\end{tikzpicture}
```

6 Kissing circles

6.1 The problem

Given three circles tangent to each other and to a straight line, the problem is to express the radius of the middle circle in terms of the radii of the other two. This problem was presented as a Japanese temple problem on a tablet from 1824 in the Gunma Prefecture (MathWorld). [5]

While not overly complicated, the construction and justification with ruler and compass are interesting. The desired output is shown in fig. 12.

The first step is to create a function to obtain the centers of the three circles, and then to determine the projections of these centers onto the common tangent of the three circles.

We call the function responsible for doing this `kissing` (fig. 13). In the following example, A, B and C represent the centers of the circles, 4 and 3 the radii of the two given circles, and E, F and G the projections of the centers.

Additionally, the function defines several useful objects such as straight lines `L.AB`, `L.EF`, and circles `C.AE`, `C.BF` and `C.CH`.

It's worth noting that the function uses the normal syntax `L[c1..c2]` instead of the "syntactic sugar" `L.name`. While the function's logic is not overly complex, attention to syntax is essential for proper execution.

```
\begin{tkzelements}
  dofile ("kissing.lua")
\end{tkzelements}
```

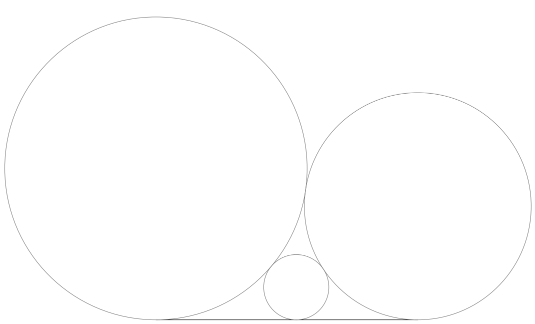


Figure 12: Three tangent circles

```
function kissing(c1,r1,c2,r2,c3,h1,h3,h2)
  local xk  = math.sqrt (r1*r2)
  local de  = math.sqrt (r1) + math.sqrt (r2)
  local cx  = (2*r1*math.sqrt(r2))/de
  local cy  = (r1*r2)/(de^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[c1..c2] = line : new (z[c1],z[c2])
  L[h1..h2] = line : new (z[h1],z[h2])
  z[c3]     = point : new (cx,cy)
  z[h3]     = L[h1..h2] : projection (z[c3])
  C[c1..h1] = circle : new (z[c1],z[h1])
  C[c2..h2] = circle : new (z[c2],z[h2])
  C[c3..h3] = circle : new (z[c3],z[h3])
end
```

Figure 13: The "kissing" function code

```
\begin{tkzelements}
  scale = .5
  kissing ("A",4,"B",3,"C","E","G","F")
  L.AE  = line : new (z.A,z.E)
  z.H   = L.AE : projection (z.B)
\end{tkzelements}
```

Now the code for the Ti*k*Z part:

```
\begin{tikzpicture}
  \tkzGetNodes
  \tkzDrawSegment(E,F)
  \tkzDrawCircles(A,E B,F C,G)
\end{tikzpicture}
```

6.2 Construction with an inversion

The diagram for a construction with an inversion is shown in fig. 14.

```
\begin{tkzelements}
   scale    = .92
   dofile ("kissing.lua")
   kissing ("A",4,"B",2,"C","E","G","F")
   z.X      = intersection (C.AE,C.CG)
   z.Y      = intersection (C.BF,C.CG)
   z.T      = intersection (L.AB,C.AE)
   z.H      = L.EF : projection (z.T)
   z.O      = midpoint (z.T,z.H)
   C.TH     = circle : new (z.T,z.H)
   z.x,z.xp = intersection (C.AE,C.TH)
   z.y,z.yp = intersection (C.BF,C.TH)
   z.x,z.xp = intersection (C.AE,C.TH)
   if z.x.re < z.xp.re then else
     z.x,z.xp = swap (z.x,z.xp) end
   z.y,z.yp = intersection (C.BF,C.TH)
   if z.y.re < z.yp.re then else
     z.y,z.yp = swap (z.y,z.yp) end
   L.OS     = L.AB : ortho_from (z.O)
   C.O      = circle : new (z.O,z.H)
   _,z.S    = intersection (L.OS,C.O)
   z.W      = z.S : symmetry (z.O)
```

```
  z.Np     = z.W : symmetry (z.S)
  z.Ep,z.Fp,
  z.N      = C.TH : inversion (z.E,z.F,z.Np)
  z.Xp,z.Yp= C.TH : inversion (z.X,z.Y)
  T.EFN    = triangle : new (z.E,z.F,z.N)
  T.EFNp   = triangle : new (z.E,z.F,z.Np)
  z.I      = T.EFN .circumcenter
  z.Ip     = T.EFNp.circumcenter
  z.Bn     = C.BF.north
  z.Fp     = z.Bn : symmetry (z.F)
\end{tkzelements}
```

6.2.1 Lua code analysis

After calling `kissing`, several points are defined such as A, B, …, G. Additionally, circles `C.AE`, `C.BF`, `C.CG` and lines `L.AB` and `L.EF` are defined.

- We designate as X, Y and T the contact points between the three circles. These points are obtained through intersections, for example: `z.X = intersection (C.AE,C.CG)`.
- H is obtained by projecting T onto the line (EF): `z.H = L.EF : projection (z.T)`.
- To maintain consistent notation, a test is conducted to ensure that x and y are closest to the line (EF). Depending on the results, the points x, x' and y, y' may be exchanged.
- `L.OS = L.AB : ortho_from (z.O)` is defined as the orthogonal line to (AB) passing through O.
- The method `symmetry` attached to points is utilized to determine point W, which is the symmetric of O with respect to S. This is obtained with: `z.W = z.S : symmetry (z.O)`.
- Finally, points E', F' and N are obtained by inversion with respect to the circle with center T passing through H. This circle is denoted `C.TH` and the transformations of the points are obtained with:
  `z.Ep,z.Fp,`
  `z.N = C.TH : inversion (z.E,z.F,z.Np)`.

  Note the use of the letter `p` in the point names, which indicates the "prime" when converting points to nodes.
- The remaining steps involve using attributes and methods which we've already discussed.

```
\begin{tikzpicture}
  \tkzGetNodes
  \tkzDrawSegments(E,F A,B E,A F,B A,C Bn,E)
  \tkzDrawSegments[lightgray](T,X' T,Y' T,N')
  \tkzDrawCircles(B,F T,H)
  \tkzDrawCircles[](C,G)
  \tkzDrawCircle[](O,H)
  \tkzDrawCircle[](W,S)
  \tkzDrawArc[delta=10](A,E)(x')
```

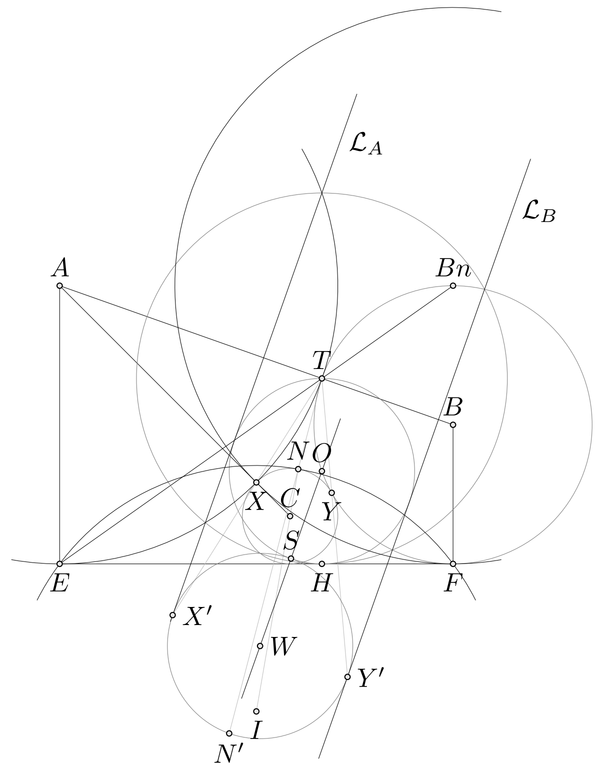


Figure 14: Method with inversion

```
  \tkzDrawArc[delta=10](I,F)(E)
  \tkzDrawArc[delta=10](Bn,F')(F)
  \tkzDrawLines[add=.3 and 0.3](x,x' O,W)
  \tkzDrawLines[add=.8 and 0.5](y,y')
  \tkzDrawPoints(A,B,E,F,T,S,W,C,H,X,Y)
   \tkzDrawPoints(X',Y',N',N,Bn,O)
  \tkzLabelPoints(E,F,H,X,Y,N')
  \tkzLabelPoints[right](X',Y',W)
  \tkzLabelPoints[above](S,Bn,N,A,B,O,T,C)
  \tkzLabelLine[pos = 1.15,right]%
     (x,x'){$\mathcal{L}_A$}
  \tkzLabelLine[pos = 1.3,right]%
     (y,y'){$\mathcal{L}_B$}
\end{tikzpicture}
```

7 Drawing with `tkz-euclide`

If you're utilizing `tkz-elements` and intend to use Ti*k*Z, the macro `\tkzGetNodes` is essential. It generates nodes from points defined in the `tkzelements` environment.

7.1 A few basics

1. Drawing: The role of `tkz-euclide` is minimized in drawing simple Euclidean geometry objects.

- Points: `\tkzDrawPoints(A,B,C)`
- Segments: `\tkzDrawSegements(A,B C,D)`
- Lines: `\tkzDrawLines(A,B C,D)`

- Circles: `\tkzDrawCircles(A,B C,D)`[6]
- Polygons: `\tkzDrawPolygons(A,B,C D,E,F)`[7]
- Ellipse: `\tkzDrawLuaEllipse(C,A,B)`[8]

  You can define the styles of objects globally or use a style locally. For example:
  `\tkzDrawPoints[style](A,B,C)`.

2. Marking: Additionally, you have the option to mark segments or angles.
   - `\tkzMarkSegments[s|](A,B C,D)`
   - `\tkzMarkArc(O,A)(B)`
   - `\tkzMarkAngles(A,B,C)`
   - `\tlkzMarkRightAngles(A,B,C)`
3. Labeling:
   - `\tkzLabelPoints(A,B,C)`
   - `\tkzLabelSegments(A,B C,D)`
   - `\tkzLabelAngle(A,B,C){$\alpha$}`
   - `\tkzLabelCircle(O,A)(60){$C(O,A)$}`

7.2 Styling

The `tkz-euclide` package includes a configuration file `tkz-euclide.cfg` containing all style definitions, which can be duplicated and modified as needed. Let's explore the methods for changing point styles; the principle will be identical for other objects.

7.2.1 Styling the points

Points: To draw points A, B and C, you can use `\tkzDrawPoints(A,B,C)`. This is the same as Ti*k*Z. In `tkz-euclide`, points are represented as Ti*k*Z `coordinates`.

Here are some additional details on styling points in `tkz-euclide`:

- Setting global point size: You can set the global point size for the entire figure or document.
  `\tkzSetUpPoint[size=.8pt]`

  You can also change this size locally when needed. In some cases, you may need to use a group or a scope for local modification.
- Creating local styles: You can create local styles by customizing the style name. For example:
  `\tikzset{step 1/.style={cyan,thin}}` and
  `\tikzset{step 2/.style={red,thick}}`
  which you can use in this way:
  `\tkzDrawPoints[step 1](A,B)` and
  `\tkzDrawPoints[step 2](C)`
- Combining general and specific styles: You can define a general style and then create adaptations from it. For example:
  `\tkzSetUpPoint[size=.8pt]`
- Modifying predefined styles: It's possible to modify predefined styles directly:
  `\tikzset{point style/.style={...}}`
- Retaining and modifying predefined styles: You can retain part of a predefined style and add to or modify it as needed.
  `\tikzset{point style/.append style={}}`
- Finally, you can create your own local style from a global style as follows:

```
\tikzset{new/.style={point style/
.append style={minimum size=8 pt,
fill=green}}}
```

  This allows you to build upon a global style and make specific modifications for local use.

7.2.2 Styling other objects

Besides `point style`, you can look at, modify, etc., these other styles:

- `line style`
- `circle style`
- `compass style`
- `arc style`
- `vector style`

[6] center A through B
[7] triangle ABC
[8] C = center, A = vertex, B = covertex

References

[1] C. Kimberling. Triangle centers and central triangles. *Congressus Numerantium*, 129:1–295, 1998.

[2] A. Matthes. tkz-elements 2.00c, 2024. `ctan.org/pkg/tkz-elements`

[3] A. Matthes. tkz-euclide 5.06c, 2024. `ctan.org/pkg/tkz-euclide`

[4] E.W. Weisstein. Apollonius circle. From MathWorld — A Wolfram Web Resource, n.d. `mathworld.wolfram.com/ApolloniusCircle.html`

[5] E.W. Weisstein. Tangent circles. From MathWorld — A Wolfram Web Resource, n.d. `mathworld.wolfram.com/TangentCircles.html`

⋄ Alain Matthes
5 rue de Valence
Paris V, 75005
France
`alain (dot) matthes (at) mac (dot) com`
`https://altermundus.fr`